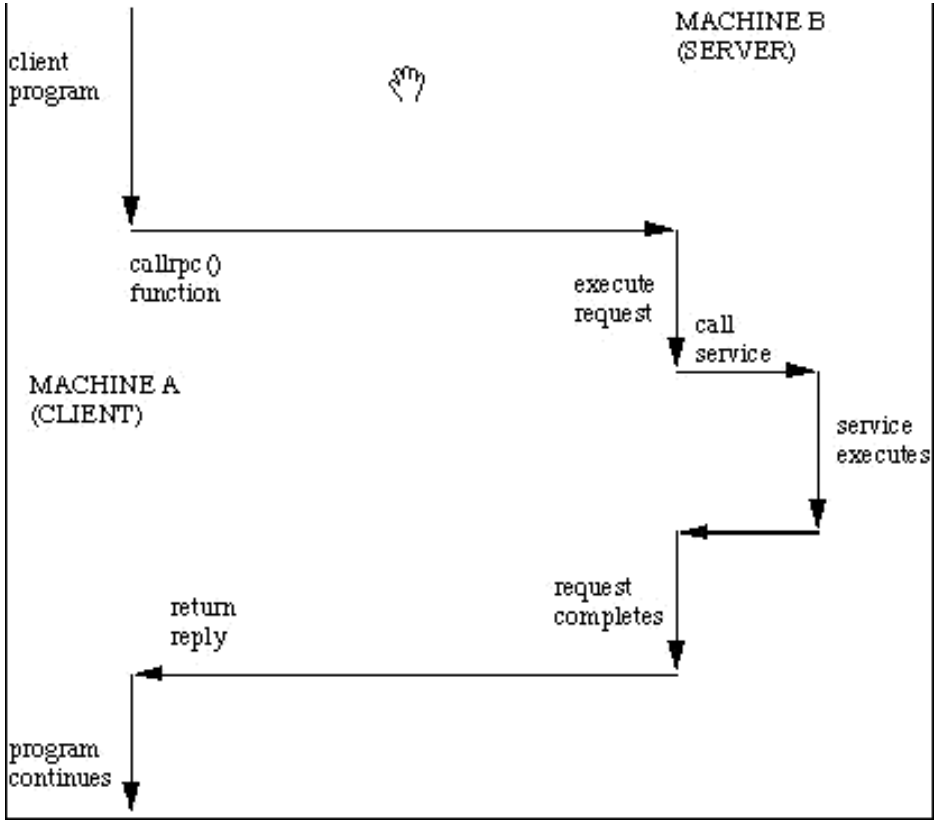
Desenvolvimento de aplicações distribuídas: tópicos adicionais.

- Invocação de procedimento remotos (RPC)
- *Web services / Web API*

Perspectiva conceptual: Implementação do modelo cliente-servidor



- Comunicação através de *sockets*.
- Maior parte do código relativo a comunicações pode ser gerado automaticamente.

Actividades implícitas no paradigma RPC

- Identificar computador com o serviço (*procedure*)
- Identificar o serviço
- Enviar os parâmetros da função e receber o resultado
  - Representação uniforme dos dados de máquina para máquina.

Desafios no intercâmbio de dados

- Máquinas com processadores de diferentes arquitecturas:
  - x86, ARM, MIPS, etc.
- Alguns problemas:
  - *Little endian* vs *big endian*
  - Representação de números de vírgula flutuante (float, double) - IEEE 754 suportado universalmente?
  - ASCII vs UTF
  - Linguagens de programação diferentes
    - Tipos de dados são representados da mesma forma?
    - Tipos de dados têm equivalência noutra linguagem?

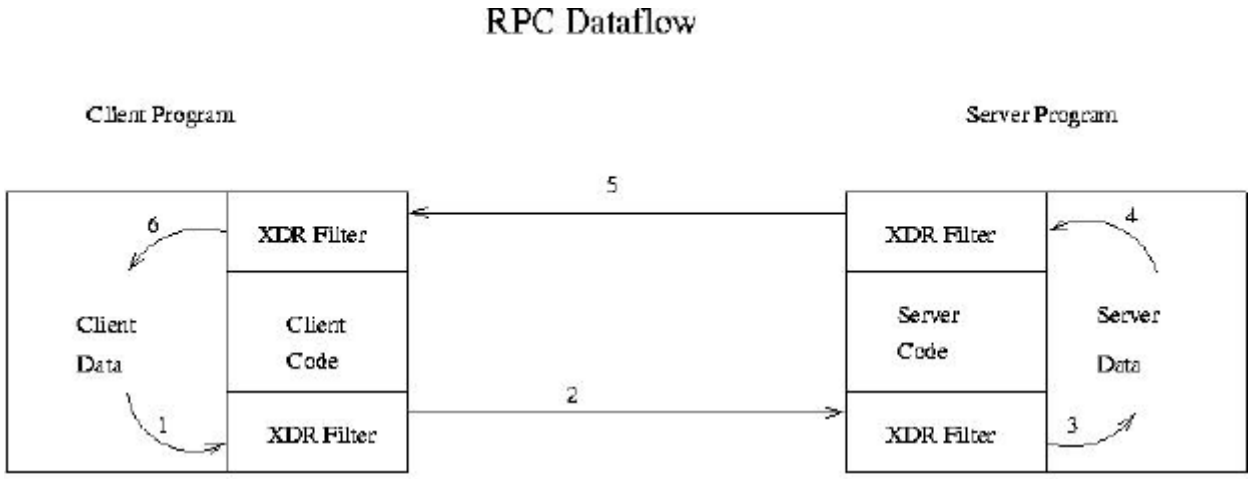
Sistemas RPC “clássicos”

- UNIX:
  - Sistema ONC RPC (inicialmente SUN RPC) - RFC 5531
- Microsoft:
  - Microsoft RPC, baseado no sistema DCE/RPC
    - Raramente usado em UNIX, mesmo havendo suporte para DCE/RPC.

Representação dos dados.

- XDR (External Data Representation)
  - Todas as variáveis com tamanho arredondado para múltiplos de 4 bytes, big-endian. Vírgula flutuante no formato IEEE 754.
  - Standard da Internet Engineering Task Force (IETF).
  - man 3 xdr, RFC4506 (2006)

Mecanismo de invocação de procedimentos remotos (com XDR)



1. Client encodes data through XDR filter.
2. Client passes XDR encoded data accross network to remote host
3. Server decodes data through XDR filter.
4. Server encodes function call result through XDR filter
5. Server pass XDR encoded data accross network back to client
6. Client decodes RPC result through XDR filter and continues processing

Figure 1.

rpcgen

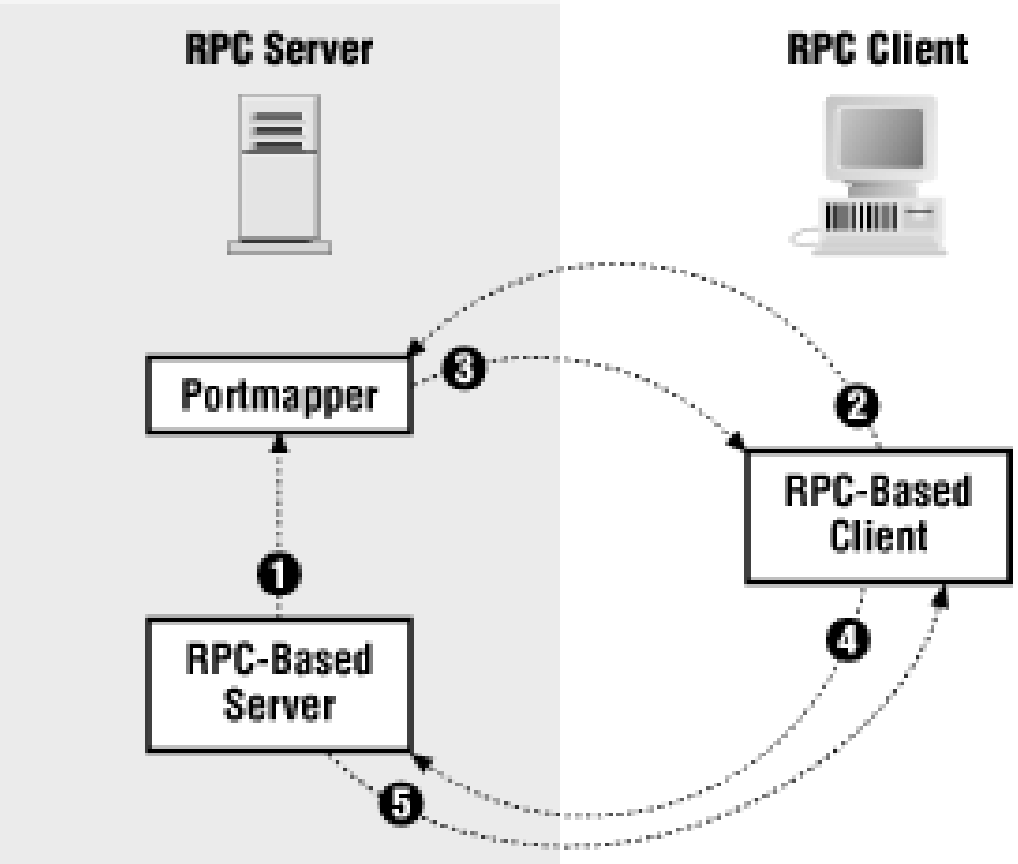
- Pré-compilador para sistemas ONC RPC.
- Geração automática de código C para codificação/descodificação de dados (*stubs*).
  - Usa as funções do standard XDR.
- Interface das funções (tipos de dados) devem ser descritas em *remote procedure call language* (RPCL).

Localização de serviços no sistema RPC: Portmapper

1. *Server registers port with portmapper.*

... later ...

2. *Client contacts portmapper, asks about server.*
3. *Portmapper tells client what port server is using.*
4. *Client contacts server.*
5. *Server responds to client.*



Portmapper

- Programa (mais especificamente, um *daemon*) que converte números de programas RPC no porto a ser utilizado na comunicação.
  - Em Linux chama-se rpcbind ("RPC Locator" no Microsoft Windows)
  - Residente nas máquinas servidoras de procedimentos remotos (no porto 111)
  - Clientes contactam o portmapper para obterem o porto para onde deverão enviar o pedido de execução da função desejada.

Portmapper

- Porquê:
  - As camadas TCP e UDP só permitem 65536 portos.
  - Este valor seria insuficiente para atribuir um valor diferente a cada serviço possível.
- No sistema ONC RPC cada serviço é identificado por um valor 32 bits
  - Em cada servidor, o respectivo portmapper faz o mapeamento entre o número do serviço e número do porto actualmente atribuído a esse serviço, nessa máquina.

ONC RPC – segurança

- *Firewalls* vs *portmapper*
  - Associação porto-serviço pode mudar cada vez que se reinicia o sistema
  - Difícil configurar a *firewall*: que portos deverão estar "abertos"?
    - Solução de compromisso: usar portos fixos para os serviços mais comuns
      - E.g., o serviço nfs baseia-se em RPCs mas usa sempre o mesmo porto (2049)

Outras tecnologias baseadas no conceito RPC

- Java Remote Method Invocation (RMI)
- DCOM => .NET => Windows Communication Foundation.
- Common Object Request Broker Architecture (CORBA)
  - Standard definido pelo Object Management Group (OMG)
  - Suportado em Java

A evolução do conceito RPC

- XML-RPC
  - Usa protocolo HTTP
  - Define formato dos dados. Comandos e parâmetros são enviados em formato XML.
- SOAP ("Simple Object Access Protocol")
  - Baseado no XML-RPC, com mais funcionalidades/complexidade (Microsoft, IBM and W3C).
  - Usado nos "*Web Services*".
- JSON-RPC – codificação das mensagens em JSON
  - JavaScript Object Notation (RFC 7159)

A evolução do conceito RPC: ”novas” tecnologias

- XML
  - Suporte alargado de *software*.
  - Legível por humanos (modo de texto).
  - Independente da plataforma.
  - Pode ser um pouco “pesado” (uso de etiquetas como delimitadores). Mais notório com dados numéricos.
- HTTP
  - Mais facilmente aceite pelos administradores de rede (configuração *firewalls* e *proxies)*
  - Identificação de serviços através de URI

Hypertext Transfer Protocol (HTTP) – RFC 7230-7235

- Protocolo baseado em mensagens de texto (*case-sensitive*).
  - Protocolo de aplicação - principal protocolo usado na *World Wide Web* (WWW) – a "*Web*".
  - Tipicamente usa TCP/IP.
- Protocolo pedido-resposta:
  - Utilização do protocolo, do ponto de vista do cliente:
    - Faz ligação ao servidor (porto 80 é o standard).
    - Envia ao servidor mensagem com pedido
    - Aguarda mensagem de resposta do servidor.
- Tipos de pedido ("Métodos"):
  - Mais usados (HTTP/1.0): GET, POST
  - Outros exemplos (HTTP/1.1): PUT, DELETE

Mensagens HTTP

- Estrutura das mensagens (pedido e resposta):
  - Cabeçalho
  - Linha em branco: *carriage return* + *linefeed* (CR+LF)
  - Corpo (apenas em alguns tipos de mensagens)

Pedidos HTTP

- Cabeçalho.
  - <Método> <URL> **HTTP/**<versão HTTP>
    - Exemplo: POST /app/user.php HTTP/1.0
  - Lista de campos opcionais.
  - No caso do HTTP/1.1 é necessário incluir sempre o seguinte campo: Host: <servidor>
- Corpo da mensagem (opcional).
  - No método POST é geralmente usado para envio de parâmetros (dados, em geral)

Respostas HTTP

- Cabeçalho.
  - HTTP/<versão> <Código do resultado do pedido> <descritivo curto>.
    - Exemplo: HTTP/1.1 200 OK
  - Lista de campos opcionais.
- Corpo da mensagem (opcional).
  - Exemplos:
    - Dados (e.g., no caso das respostas a pedidos do tipo GET)
    - Informação de apoio ao código da resposta

Mensagens HTTP (pedidos e respostas)

- Todas as linhas são terminadas com <CR><LF>
  - (“\r\n” em C).
- O corpo da mensagem, quando existente, é separado do cabeçalho por uma linha em branco.
- Campos frequentemente utilizados nos cabeçalhos das mensagens HTTP:
  - Content-Length: <nbytes>
    - Tamanho do corpo da mensagem, caso exista.
  - Content-Type: text/html | text/plain | octet-stream | application/pdf | **multipart/form-data** | etc.
    - Indicação do tipo de dados no corpo da msg.

Mensagens HTTP (pedidos e respostas)

- Campos frequentemente utilizados nos cabeçalhos das mensagens HTTP (cont.):
  - Connection: close | keep-alive
    - Gestão de ligações persistentes (HTTP/1.1). Usado para indicar se a ligação deve ser fechada imediatamente ou mantida para novos pedidos.
  - Outros:
    - Pedidos: Accept (indicação de formatos de dados aceites), User-Agent, etc.
    - Respostas: Server, Date, Last-Modified, etc.

Exemplo

- Mensagem (pedido) do cliente:

```
GET /index.html HTTP/1.1
Host: www.example.com
```

- Mensagem (resposta) do Servidor:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Date: Mon, 23 May 2005 22:38:34 GMT
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Connection: close

<html>
<head>
<title>An Example Page</title>
</head>
<body>
Hello World, this is a very simple HTML document.
</body>
</html>
```

Pedidos POST

- Em geral, são usados para submeter dados
- Os dados são enviados no corpo da mensagem. Como?
- Possibilidades:
  - Content-Type: application/x-www-form-urlencoded
    - Usado por omissão nos formulários Web.
  - Content-Type: multipart/form-data
    - Usado para enviar ficheiros ou grandes blocos de dados.
  - Content-Type arbitrário (*ad hoc*) – em geral obriga a um maior esforço de programação do lado do servidor.

Pedidos POST

- Exemplo 1:

```
POST /app/vehicle HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 37

mat=123456&prop=John Doe&val=10000.00
```

- Exemplo 2:
  - Formulário HTML:

```
<form action="insert_file.php" method="post"
enctype="multipart/form-data">
  <p><input type="text" name="text" value="mytext">
  <p><input type="file" name="file1">
  <p><button type="submit">Submit</button>
</form>
```

- Exemplo 2 (cont.):
  - Mensagem enviada pelo *browser:*

```
POST / HTTP/1.1
Host: www.example.com
Content-Type: multipart/form-data; boundary=905191404154484
Content-Length: XXX

--905191404154484
Content-Disposition: form-data; name="text"

mytext
--905191404154484
Content-Disposition: form-data; name="file1";
filename="a.html"
Content-Type: text/html

<!DOCTYPE html><title>Content of a.html.</title>

--905191404154484--
```

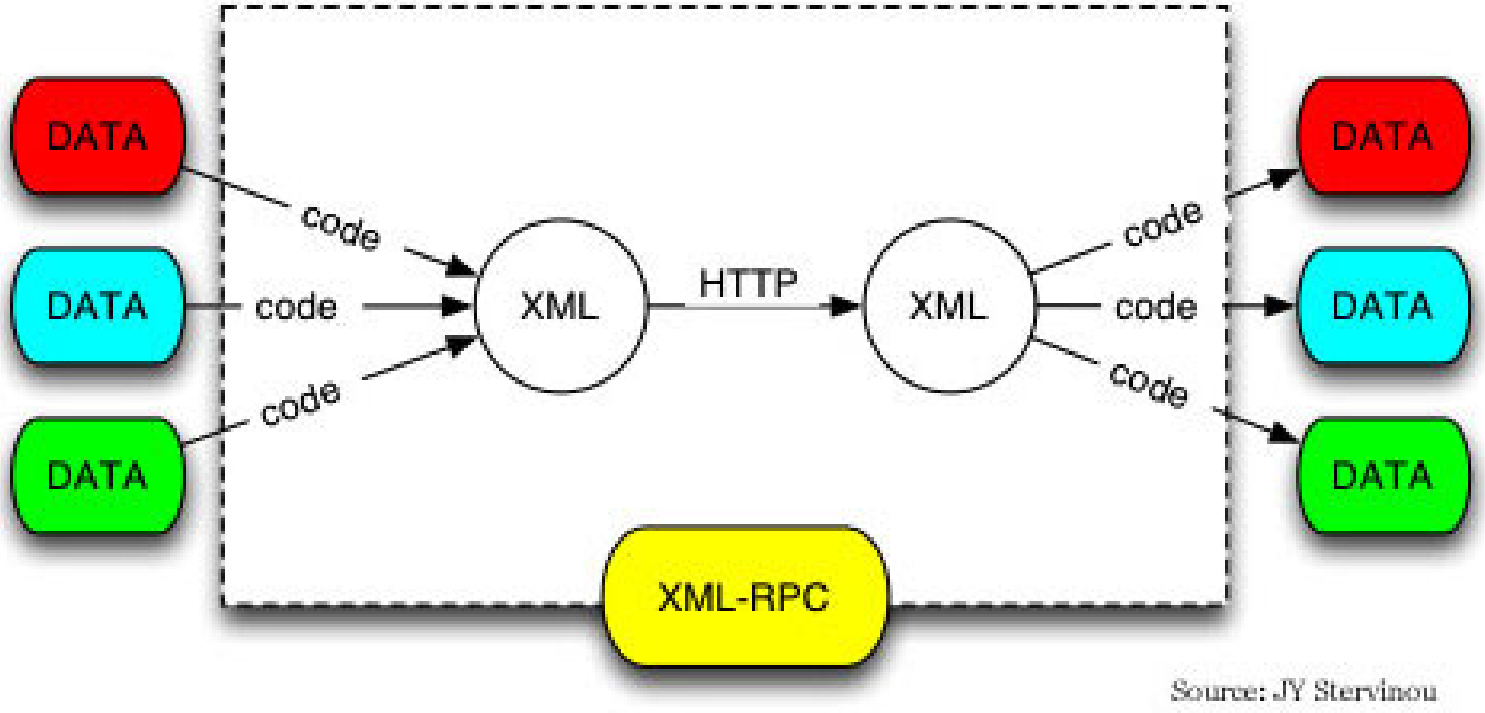
PHP: receção de dados no servidor

- Variáveis (vetores associativos) globais:
  - $_GET – parâmetros passados por *query string*. Exemplo: GET user.php?id=100
  - $_POST – parâmetros de entrada do formulário (exceto ficheiros).
  - $_FILES – descrição de ficheiro(s) recebido(s) (cada elemento é um vetor com vários itens de informação). Dados do(s) ficheiro(s) são guardados em ficheiro(s) temporário(s).
  - $_SERVER – informação adicional sobre o pedido e sobre o estado/configuração do servidor.

HTTP: autenticação básica

- Campo adicional no cabeçalho:
  - WWW-Authenticate: Basic *str_b64*
    - str_b64 =base64("username:password")
- Usar apenas sobre ligação encriptada (HTTPS)
  - A codificação base64 não faz encriptação, foi projetada para permitir uma descodificação eficiente.
  - Possível alternativa: *Digest access authentication*
    - Permite o envio encriptado das credenciais.
    - Mas corpo da mensagem não é encriptado.

XML-RPC


DATA
DATA
DATA
code
code
code
XML
HTTP
XML
code
code
code
DATA
DATA
DATA
XML-RPC
Source: JY Stervinou

Exemplo XML-RPC:
comando com parâmetro inteiro de 4 bytes

```
POST /RPC2 HTTP/1.0
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
 <methodCall>
    <methodName>examples.getStateName</methodName>
    <params>
        <param>
            <value><i4>41</i4></value>
        </param>
    </params>
</methodCall>
```

Serviços Web: definição

- W3C Web Services Architecture Working Group:
  - "*a Web service has an interface described in a **machine-processable format** (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP (Simple Object Access Protocol) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards*"
- Definição mais genérica e "liberal":
  - Funcionalidade disponibilizada através da *Web,* independente de sistemas operativos e linguagens de programação, usando standards abertos como o HTTP e o XML.

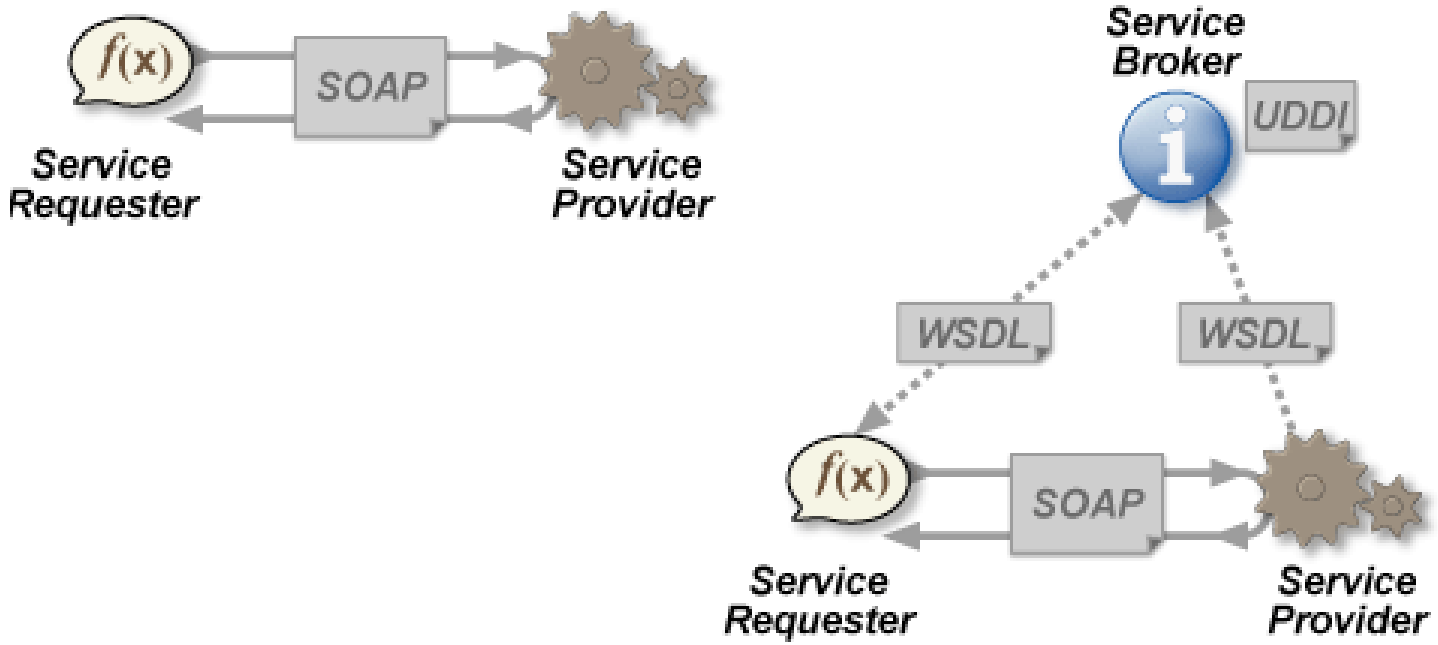
Principais abordagens ao conceito de serviços Web

- Serviços Web (WS) baseados em SOAP
  - Conceito original de WS e considerado por muitos o standard (a começar pelos seus criadores e seguidores...).
    - Designados por "big" na documentação da tecnologia Java.
  - Requerem ficheiros WSDL para descrição de mensagens que podem ser trocadas entre clientes e servidores
- Serviços Web *RESTful* (REST - *Representational State Transfer*) ou Web API
  - Baseados no HTTP, não impõe SOAP/WSDL/UDDI
  - Surgiram como resposta à alegada complexidade dos WS baseados em SOAP

Serviços Web baseados em SOAP

- WSDL – *Web Services Description Language*
  - Como interagir com o *Web Service:*
    - Que tipos de mensagens podem ser trocadas
    - Localização do serviço.
- UDDI - *Universal Description Discovery and Integration*
  - Tecnologia "*Publish and find*"
    - Servidores publicam ficheiros WSDL
    - Clientes podem pesquisar serviços existentes

Serviços Web baseados em SOAP



Serviços *Web* RESTful

- "Estilo arquitetural"
  - Orientação ao "recurso" (objetos)
    - Cada URL representa um recurso.
  - Ações sobre o recurso são realizados com os métodos HTTP:
    - POST – inserção de dados
    - GET – consulta de dados
    - PUT – atualização de dados
      - Recurso tem que ser identificado pelo URL
    - DELETE – eliminação de dados

Tipos de resposta do servidor: códigos HTTP

- Servidor pode usar cabeçalho da resposta HTTP para dar uma informação genérica sobre o sucesso do pedido:
  - 1xx Informational
  - 2xx Success
    - 200 OK, 201 Created
  - 3xx Redirection
  - 4xx Client Error
    - 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
  - 5xx Server Error
    - 500 Internal Server Error, 501 Not Implemented, 503 Service Unavailable

Serviços *Web* REST

- Cada chamada é independente das anteriores.
  - Não existe suporte formal para o conceito de estado (e.g., sessão de um utilizador)
    - Eventuais estados da aplicação (e.g., signed in/signed out) têm que ser implementado de forma *ad hoc* (e.g., usando os métodos tradicionais da programação Web).
- Mecanismos de autenticação e segurança não definidos
  - Possível abordagem: HTTPS + Basic authentication

Serviços *Web* REST: exemplo

- Duas funcionalidades disponíveis:
  - Inserção de registos (matrícula, proprietério, valor)
  - Listagem de registos
    - Formato CSV
    - Formato CSV embebido em HTML
- URL
  - `Inserção: /~sistc/vehicapp/vehicle`
  - `Listagem CSV: /~sistc/vehicapp/vehicles`
  - `Listagem CSV+HTML: /~sistc/vehicapp/vehicles.html`

Serviços *Web* REST: exemplo

- Exemplo de mensagem HTTP para inserção de registo:

```
POST /~sistc/vehicapp/vehicle HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 37

mat=123456&prop=John Doe&val=10000.00
```

- Mensagens HTTP para pedidos de consulta de registos:

```
GET /~sistc/vehicapp/vehicles HTTP/1.0
```

ou

```
GET /~sistc/vehicapp/vehicles.html HTTP/1.0
```

- Em todos os casos, o cliente deverá ligar-se ao porto 80 do servidor e enviar as respetivas mensagens.

Mapeamento de URL através do ficheiro .htaccess (módulo mod_rewrite do Apache)

```
RewriteEngine On
RewriteBase /~sistc/vehicapp/

RewriteCond %{REQUEST_METHOD} =POST
RewriteCond %{REQUEST_URI} =/~sistc/vehicapp/vehicle
RewriteRule ^(.*)$ api.php [QSA,L]

RewriteCond %{REQUEST_METHOD} =GET
RewriteCond %{REQUEST_URI} =/~sistc/vehicapp/vehicles
RewriteRule ^(.*)$ api.php [QSA,L]

RewriteCond %{REQUEST_METHOD} =GET
RewriteCond %{REQUEST_URI} =/~sistc/vehicapp/vehicles.html
RewriteRule ^(.*)$ api.php?format=html [QSA,L]
```

Serviços *Web* REST: exemplo (cont.)

- Implementação do serviço Web em PHP (`api.php`)

```
<?php
switch($_SERVER['REQUEST_METHOD'])
{
case 'GET':
  srvc1_get();
  break;
case 'POST':
  srvc1_post();
  break;
}
```

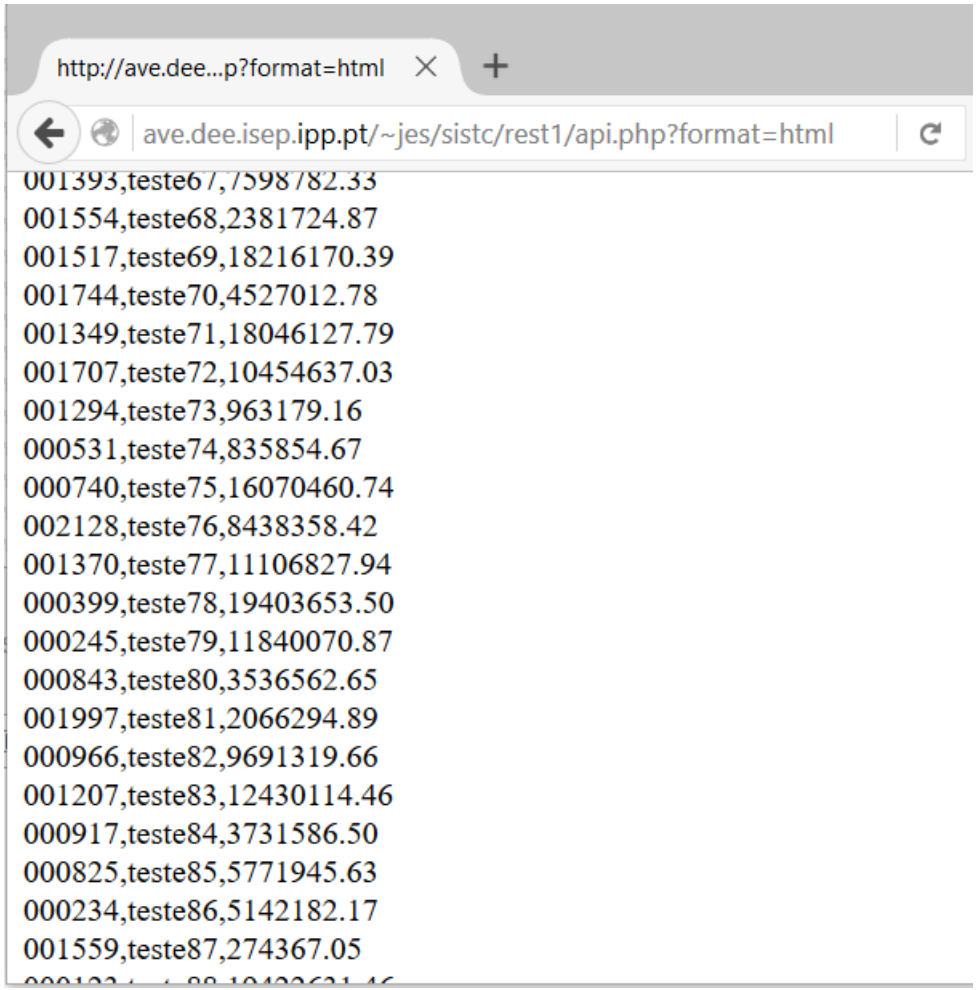
Serviços *Web* REST: exemplo (cont.)

```
function srvc1_post()
{
  //ligação ao servidor BD local, no porto 6050
  $address = gethostbyname('localhost');
  $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
  socket_connect($socket, $address, 6050);
  //envio de pedido de inserção
  socket_write($socket, pack("c",1), 1);

  //envio dos dados para servidor de BD em formato CSV
  $n = socket_write($socket,
    $_POST['mat'].",".$_POST['prop'].",".$_POST['val']."\n");
  //leitura da confirmação da inserção
  $str1 = socket_read($socket, 1);

  //Envio da info. de sucesso/erro através do cabeçalho HTTP
  if($str1 == pack("c",1))
    header("HTTP/1.1 201 Created");
  else
    header("HTTP/1.1 500 Insertion error");
}
```

Serviços *Web* REST: exemplo (cont.)

```
function srvc1_get()
{
  //ligação ao servidor BD local, no porto 6050
  $address = gethostbyname('localhost');
  $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
  socket_connect($socket, $address, 6050);
  //envio de pedido de listagem
  socket_write($socket, pack("c", 2), 1);

  while(true){
    $str1 = socket_read($socket, 4096, PHP_NORMAL_READ);
    if($str1=="")
      exit();
    if($_GET['format']=="html")
      echo $str1."<br>"; //escrita para o cliente
    else
      echo $str1;
  }
}
```

Serviços *Web* REST: exemplo (cont.)



Ficheiros .htaccess: outras funcionalidades

- Permitir o uso de parte do URL para especificar os recursos de forma mais detalhada. Exemplo:
  - Ficheiro .htaccess

    <Files "vehicle.php">

        AcceptPathInfo On

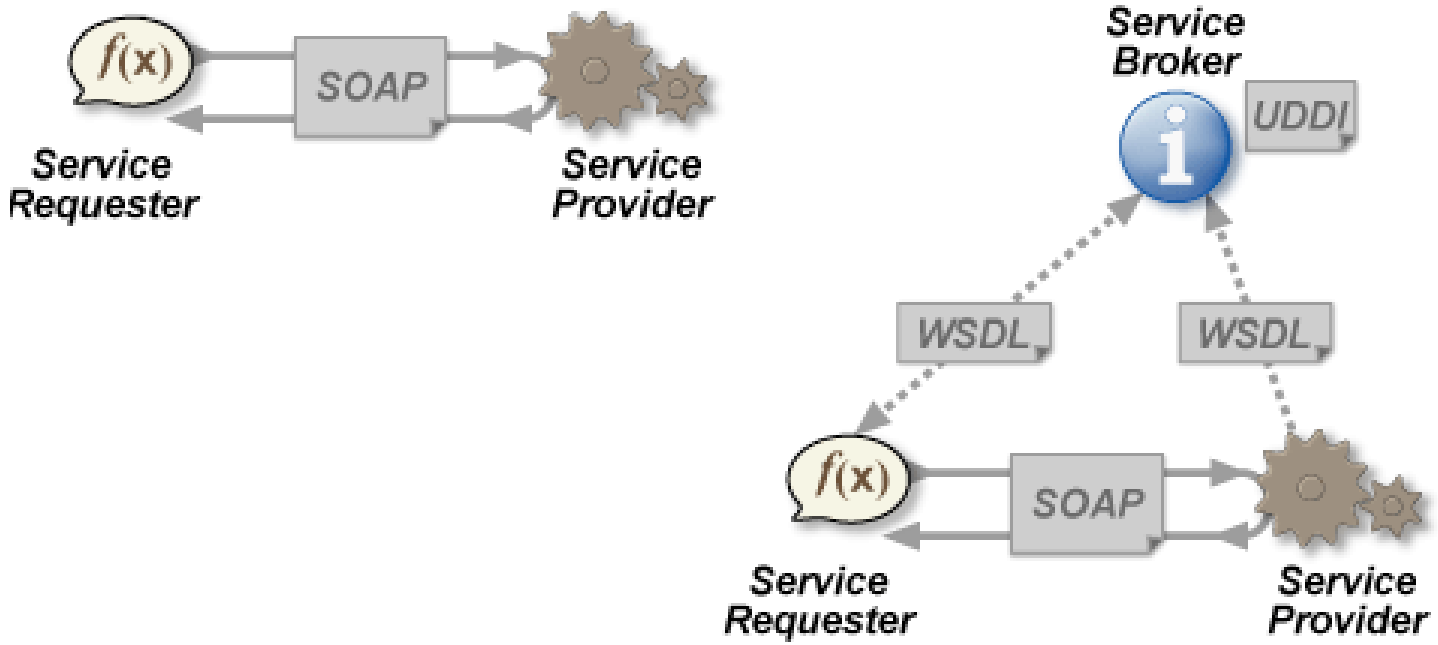
    </Files>
  - URL: http://localhost/vehicles.php/AA11BB/owner
  - Código PHP: $_SERVER['PATH_INFO'] irá conter a string "/AA11BB/owner".

Ficheiros .htaccess: outras funcionalidades

- Ativar todos os métodos do HTTP:

```
<Limit GET POST PUT DELETE HEAD OPTIONS>
   require all granted
</Limit>
```

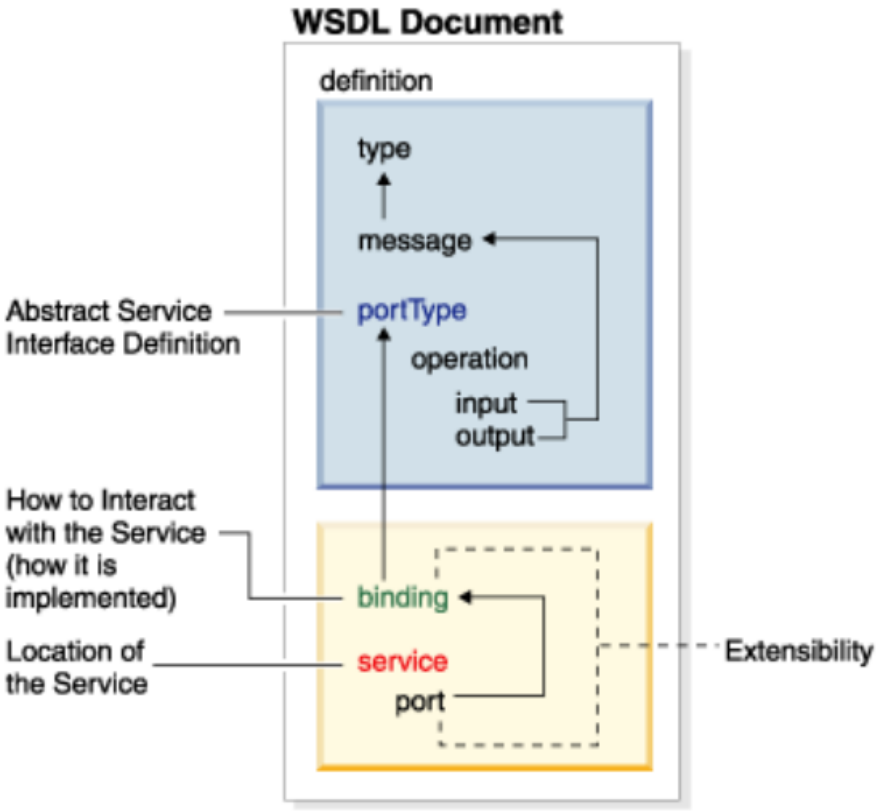
Serviços Web (baseados em SOAP)



Documento WSDL

```
<definitions>

<types>
  data type definitions........
</types>
<message>
  definition of the data being communicated....
</message>
<portType>
  set of operations: <operation> ... </operation>
</portType>
<binding>
  protocol and data format specification....
</binding>
<service>
  associate a binding to a network address
</service>
</definitions>
```

Documento WSDL



Exemplo WSDL

http://www.mkyong.com/webservices/jax-ws/jax-ws-hello-world-example-document-style/

```
<definitions
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://ws.mkyong.com/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://ws.mkyong.com/"
  name="HelloWorldImplService">
<types>
<xsd:schema>
  <xsd:import namespace="http://ws.mkyong.com/"
    schemaLocation =
    "http://localhost:9999/ws/hello?xsd=1">
  </xsd:import>
</xsd:schema>
</types>
```

Exemplo WSDL

http://www.mkyong.com/webservices/jax-ws/jax-ws-hello-world-example-document-style/

```
<message name="getHelloWorldAsString">
  <part name="parameters"
   element="tns:getHelloWorldAsString"></part>
</message>

<message name="getHelloWorldAsStringResponse">
  <part name="parameters"
   element="tns:getHelloWorldAsStringResponse"></part>
</message>

<portType name="HelloWorld">
  <operation name="getHelloWorldAsString">
    <input message="tns:getHelloWorldAsString"></input>
    <output message =
     "tns:getHelloWorldAsStringResponse"></output>
  </operation>
</portType>
```

Exemplo WSDL

http://www.mkyong.com/webservices/jax-ws/jax-ws-hello-world-example-document-style/

```
<binding name="HelloWorldImplPortBinding"
  type="tns:HelloWorld">

  <soap:binding  transport =
   "http://schemas.xmlsoap.org/soap/http"
   style="document">
  </soap:binding>
  <operation name="getHelloWorldAsString">
     <input>
       <soap:body use="literal"></soap:body>
     </input>
     <output>
        <soap:body use="literal"></soap:body>
     </output>
  </operation>
</binding>
```

Exemplo WSDL

http://www.mkyong.com/webservices/jax-ws/jax-ws-hello-world-example-document-style/

```
<service name="HelloWorldImplService">

  <port name="HelloWorldImplPort"
    binding="tns:HelloWorldImplPortBinding">

    <soap:address location =
"http://localhost:9999/ws/hello">
    </soap:address>

  </port>
</service>

</definitions>
```

Mensagem SOAP

```
<?xml version="1.0"?>

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

<soap:Header>
...(optional)
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...(optional)
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

Mensagem SOAP

- Elementos principais são declarados no *namespace* oficial do SOAP:
  - http://www.w3.org/2003/05/soap-envelope/
  - Especificação sujeita a alterações (verificar *site* do W3).

Exemplo de mensagem SOAP

http://www.mkyong.com/webservices/jax-ws/jax-ws-hello-world-example-document-style/

```
POST /ws/hello HTTP/1.1
Content-Type: text/xml; charset=utf-8
Host: localhost:9999
Content-Length: 224

<?xml version="1.0" ?>
<S:Envelope
 xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
      <ns2:getHelloWorldAsString
         xmlns:ns2="http://ws.mkyong.com/">
         <arg0>mkyong</arg0>
      </ns2:getHelloWorldAsString>
  </S:Body>
</S:Envelope
```

Exemplo de mensagem SOAP (resposta)

```
HTTP/1.1 200 OK
Transfer-encoding: chunked
Content-type: text/xml; charset=utf-8

<?xml version="1.0" ?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:getHelloWorldAsStringResponse
       xmlns:ns2="http://ws.mkyong.com/">
       <return>Hello World JAX-WS mkyong</return>
    </ns2:getHelloWorldAsStringResponse>
  </S:Body>
</S:Envelope>
```

Analisadores (*parsers*) XML em PHP

- XML Extension
  - Metodologia *Simple API for XML* (SAX) – leitura sequencial, processamento orientada ao evento (e.g., “open tag”, “close tag”)
- XMLReader - “pull parser”, baseado na biblioteca libxml
  - SAX com controlo de fluxo pelo cliente, em oposição ao modelo *push*/*callback*/*event-oriented.*
- Document Object Model (DOM) - Carregamento de todo o documento numa estrutura de dados tipo árvore (variável PHP).
- SimpleXML – apenas funcionalidades básicas

Referências

- RPC
  - Remote Procedure Call Protocol Specification Version 2, RFC 1831, Sun Microsystems, 1995
  - Network Interfaces Programmer's Guide, Sun Microsystems, 1994
- Serviços Web
  - Stephen Potts and Mike Kopack. Sams Teach Yourself Web Services in 24 Hours (First ed.). Sams, Indianapolis, IN, USA, 2003
  - Leonard Richardson, Sam Ruby, “RESTful Web Services. Web services for the real world”, 2007
  - Leonard Richardson, Mike Amundsen, Sam Ruby, “RESTful Web APIs: Services for a Changing World”, 2013

Referências

- Web Services Architecture, W3C, 2004, https://www.w3.org/TR/ws-arch/
- SOAP, http://www.w3.org/TR/soap/
- WSDL, http://www.w3.org/2002/ws/desc/
- UDDI, http://uddi.xml.org
- XML-RPC, http://www.xmlrpc.com/