SISTEMAS DISTRIBUÍDOS

Capítulo 5 – aula 11

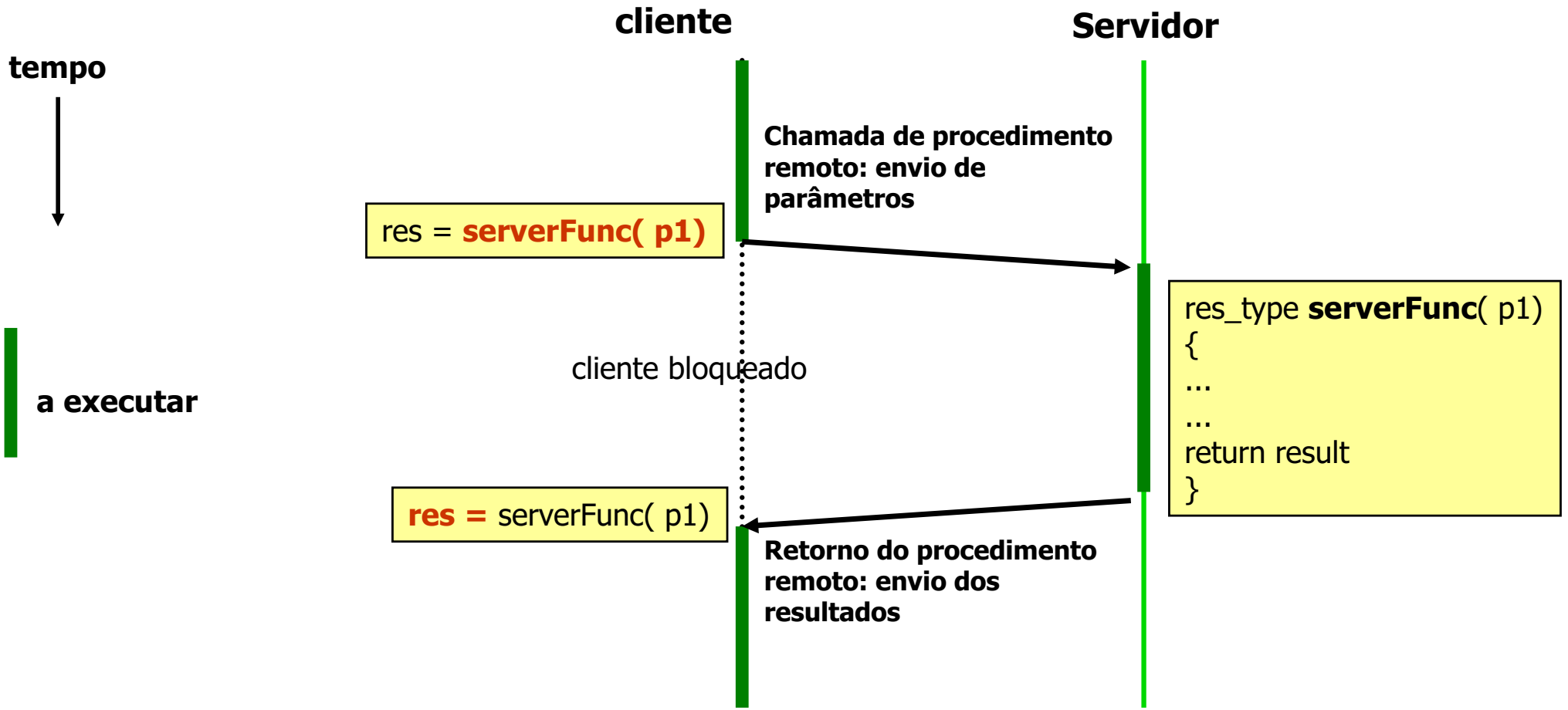
Web services e modelos alternativos de interacção cliente/servidor na internet

NOTA PRÉVIA

A apresentação utiliza algumas das figuras do livro de base do curso

G. Coulouris, J. Dollimore and T. Kindberg,
Distributed Systems - Concepts and Design,
Addison-Wesley, 5th Edition, 2005

NAS ÚLTIMAS AULAS... INVOCAÇÃO REMOTA DE PROCEDIMENTOS



Modelo

Servidor exporta interface com operações que sabe executar

Cliente invoca operações que são executadas remotamente e (normalmente) aguarda pelo resultado

INVOCAÇÃO REMOTA DE PROCEDIMENTOS

Invocação remota de procedimentos/objectos

- Motivação
- Modelo
- Definição de interfaces e método de passagem de parâmetros
- Codificação dos dados
- Mecanismos de ligação (binding)
- Semântica na presença de falhas
- Organização interna do servidor
- Sistemas de objetos distribuídos

ORGANIZAÇÃO DO CAPÍTULO

Web services SOAP

Web services REST

WEB SERVICES: MOTIVAÇÃO

Modelo para acesso a servidores: invocação remota

Desenhado para garantir inter-operabilidade

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It 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 messages, typically conveyed using* ***HTTP*** *with an* ***XML serialization*** *in conjunction with other Web-related standards*

Protocolo: HTTP e HTTPS (eventualmente SMTP)
Referências: USL/URI
Representação dos dados: XML

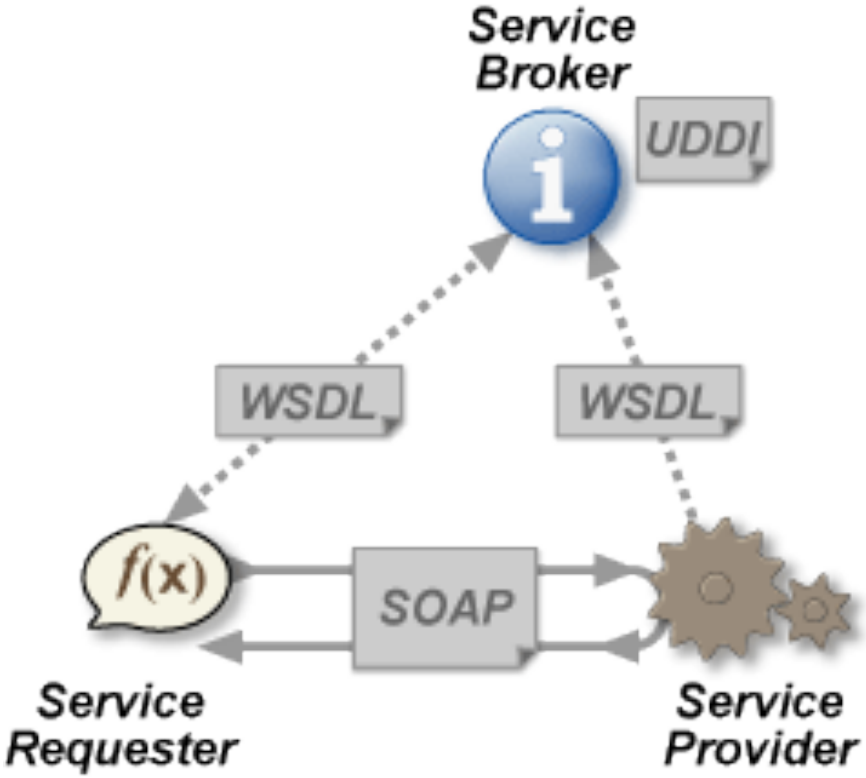
COMPONENTES BÁSICOS

SOAP: protocolo de invocação remota

- Oneway
- Pedido-resposta
- Notificação
- Notificação-resposta

WSDL: linguagem de especificação de serviços

UDDI: serviço de registo



WSDL – IDL PARA *WEB SERVICES*

Define a interface do serviço, indicando quais as operações disponíveis

Define as mensagens trocadas na interacção (e.g. na invocação duma operação, quais as mensagens trocadas)

Permite definir a forma de representação dos dados e a forma de aceder ao serviço

Especificação WSDL bastante verbosa – normalmente criada a partir de interface ou código do servidor

- Em Java e .NET existem ferramentas para criar especificação a partir de interfaces Java
- Sistemas de desenvolvimento possuem *wizards* que simplificam tarefa

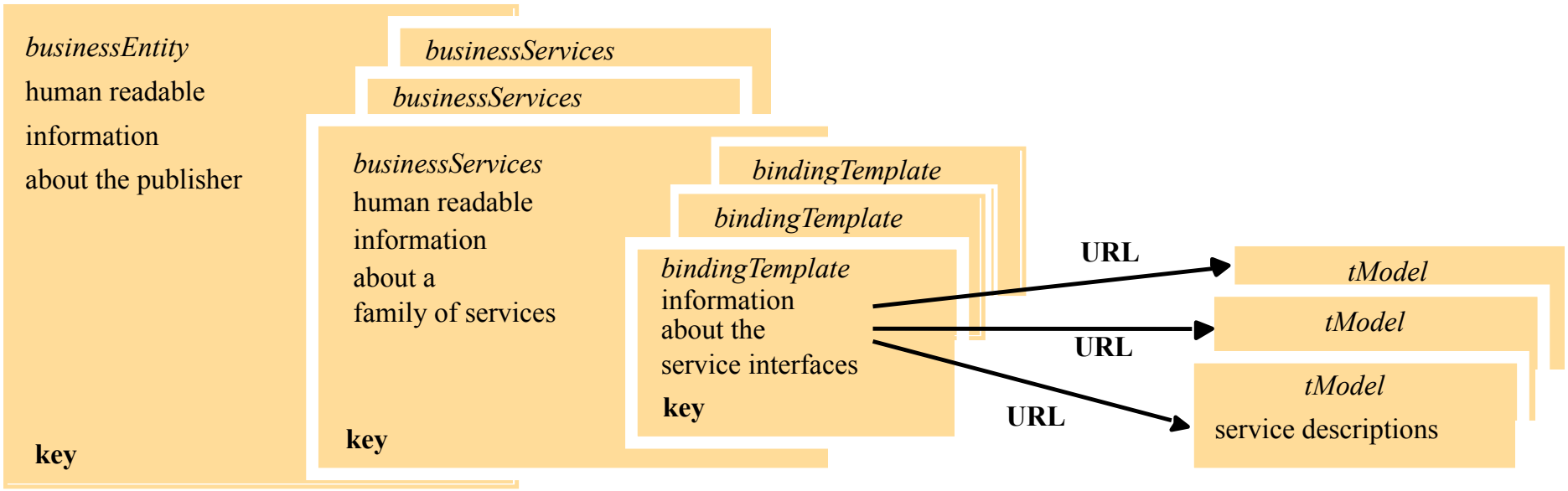
UDDI

Protocolo de ligação ou binding entre o cliente e o servidor

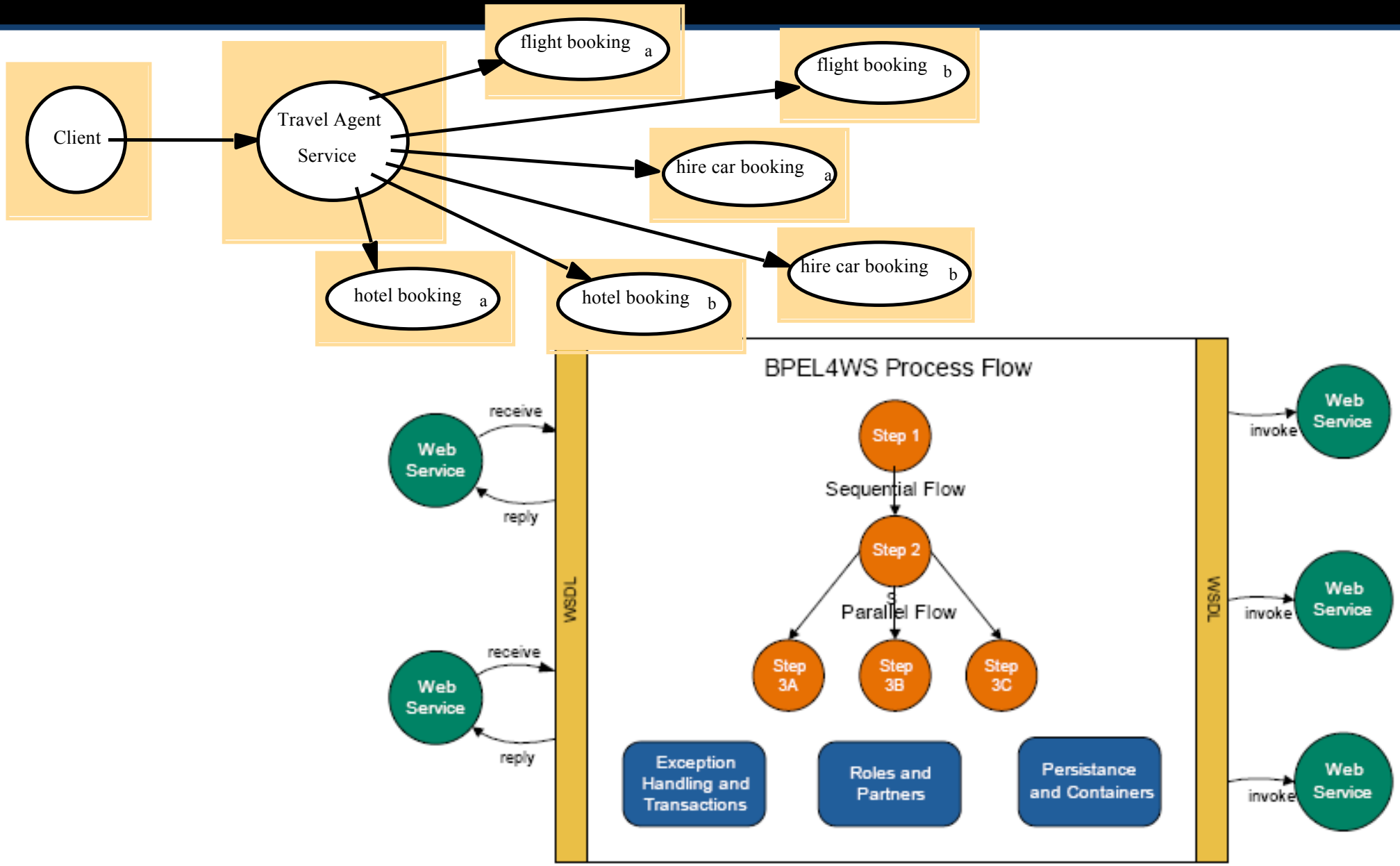
Os fornecedores dos serviços publicam a respectiva interface

O protocolo de inspeção permite verificar se uma dado serviço existe baseado na sua identificação

O UDDI permite encontrar o serviço baseado na sua definição – capability lookup



WS-COORDINATION



WEB SERVICES: RESUMO

Standard na indústria porque apresenta uma solução para integrar diferentes aplicações

Permite:

- Utilização de standards
  - HTTP, XML
- Reutilização de serviços (arquitecturas orientadas para os serviços)
  - WS-Coordination
- Modificação parcial/evolução independente dos serviços
  - WSDL
- Disponibilidade, assincronismo
  - WS-Eventing

WEB SERVICES: UTILIZAÇÕES POSSÍVEIS

Inter-operação entre instituições

- Utilização tende a ser limitada
- Promessas de todos os serviços disponíveis para todos acabaram por não se concretizar

Inter-operação entre sistemas duma mesma instituição

- Utilização com forte implantação

WEB SERVICES: PROBLEMAS

Desempenho:

- Complexidade do XML
- Difícil fazer *caching*: noção de operação + estado

Complexidade

- Necessário utilização de ferramentas de suporte

REST: REPRESENTATIONAL STATE TRANSFER

Aproximação: vê uma aplicação como uma colecção de recursos

- Um recuso é identificado por um URI/URL
- Um URL devolve um documento com a representação do recurso
- Podem-se fazer referências a outros recursos usando *ligações (links)*

Estilo arquitetural, não um sistema de desenvolvimento

Aproximação proposta por Roy Fielding na sua tese de doutroamento

- Não como uma alternativa aos web services, mas como uma forma de aceder a informação

Roy Fielding

UNIVERSITY OF CALIFORNIA,
IRVINE

Architectural Styles and the Design of Network-based Software Architectures

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Roy Thomas Fielding

Dissertation Committee:
Professor Richard N. Taylor, Chair
Professor Mark S. Ackerman
Professor David S. Rosenblum

2000

- Author of HTTP specification
- Co-founder of Apache HTTP server
- Currently working at Adobe

REST: PRINCÍPIOS DE DESENHO

Protocolo cliente/servidor stateless: cada pedido contém toda a informação necessária para ser processado – objectivo: tornar o sistema simples.

Recursos – no sistema tudo são recursos, identificados por um URI/URL.

- Recursos tipicamente armazenados num formato estruturado que suporta hiper-ligações (e.g. JSON, XML)

Interface uniforme: todos os recursos são acedidos por um conjunto de operações bem-definidas. Em HTTP: POST, GET, PUT, DELETE (equiv. criar, ler, actualizar, remover).

Cache: para melhorar desempenho, respostas podem ser etiquetadas como permitindo ou não *caching*.

Mais detalhes e exemplo

Usa HTTP GET, POST, etc., mas

Usa dados bem-tipados (usando schemas XML, json)

Pretende-se disponibilizar um serviço que mantém informação sobre servidor, disponibilizando as seguintes operações:

- Adicionar um servidor
- Remover um servidor
- Modificar um servidor
- Obter informação sobre um servidor, dado o seu id
- Pesquisar informação dum servidor

EXEMPLO: ADICIONAR SERVIDOR

- URL: http://myserver.com/server/35345645
- Método: POST
- Body:

```
{ id: ”35345645”,
name : ”server 1”,
props : [”a”, ”b”, ”c”]
}
```

EXEMPLO: MODIFICAR SERVIDOR

- URL: http://myserver.com/server/35345645
- Método: PUT
- Body:

```
{ id: ”35345645”,
name : ”server 1”,
props : [”a”, ”b”, ”d”]
}
```

EXEMPLO: REMOVER SERVIDOR

- URL: http://myserver.com/server/35345645
- Método: DELETE
- Body:

EXEMPLO: OBTER INFORMAÇÃO DE SERVIDOR

- URL: http://myserver.com/server/35345645
- Método: GET
- Body:
- Reply:

```
{ id: ”35345645”,
name : ”server 1”,
props : [”a”, ”b”, ”d”]
}
```

EXEMPLO: LISTAR SERVIDORES

- URL: http://myserver.com/server
- Método: GET
- Body:
- Reply:

[{ id: ”35345645”,
name : ”server 1”,
props : [”a”, ”b”, ”d”]
}, …
]

EXEMPLO: PESQUISAR SERVIDORES

- URL: http://myserver.com/server?q=a
- Método: GET
- Body:
- Reply:

[{ id: ”35345645”,
name : ”server 1”,
props : [”a”, ”b”, ”d”]
}, …
]

RESUMO

Ideia: sistema que mantém informação sobre utilizadores

SOAP:

- getServer(id)
- addServer(id, srv)
- removeServer(id)
- updateServer (id, srv)
- listServers()
- findServer(query)

REST:

- http://myserver.com/server/{id}
  (GET, POST, DELETE, PUT)
- http://myserver.com/server
- http://myserver.com/server?q=a+b

REST vs. RPCs/Web Services

Nos sistemas de RPCs/Web Services a ênfase é nas operações que podem ser invocadas

Nos sistemas REST, a ênfase é nos recusos, na sua representação e em como estes são afectados por um conjunto de métodos standard

CODIFICAÇÃO DOS DADOS: XML VS. JSON

A informação transmitida nos pedidos e nas respostas é codificada tipicamente em:

- XML
- Json

CODIFICAÇÃO DOS DADOS: XML VS. JSON

```
<Person firstName='John'
lastName='Smith' age='25'>
  <Address streetAddress='21 2nd
Street' city='New York' state='NY'
postalCode='10021' />
  <PhoneNumbers home='212 555-
1234' fax='646 555-4567' />
</Person>
```

(Example from wikipedia)

```
{  "Person”:  {
     "firstName": "John",
     "lastName": "Smith",
     "age": 25,
     "Address”:  {
        "streetAddress":"21 2nd Street",
        "city":"New York",
        "state":"NY",
        "postalCode":"10021"
     },
     "PhoneNumbers”: {
        "home":"212 555-1234",
        "fax":"646 555-4567"
     }
   }
}
```

CODIFICAÇÃO DOS DADOS: XML VS. JSON

A informação transmitida nos pedidos e nas respostas é codificada tipicamente em:

- XML
- Json

Json tem ganho popularidade porque permite manipulação simples em Javascript

REST: PROTOCOLOS

Baseado em protocolos standard

Comunicação

- HTTP (tipicamente)

Identifcação de recursos

- URL/URI

Representação dos recursos

- Json, XML

REST: NOTAS SOLTAS

Tem ganho popularidade

- Muitos serviços web disponibilizados segundo este modelo
- Grandes vantagens: simplicidade e desempenho

Serviços disponibilizados num modelo REST nem sempre aderem completamente aos princípios REST

- Difícil disponibilizar número elevado de métodos

  http://api.flickr.com/services/rest/?method=flickr.photos.search&appid=sdfjkshf

REST: SUPORTE JAVA

Definido em JAX-RS (JSR 311)

Suporte linguístico baseado na utilização de anotações

- Permite definir que um dado URL leva à execução dum dado método
- Permite definir o modo de codificação da resposta
  - JSON – mecanismo leve de codificação de tipos (RFC 4627)
  - XML – usando mecanismo standard de codificação de cobjectos java em XML fornecido pelo JAXB

REST: SUPORTE JAVA (EXEMPLO)

```
@Path("/users/")
public class UserService {

   @GET
   public Response listUsers () {   ......   }

   @GET
   @Path("/user/{id}")
   @Produces("application/json")
   public Response getUser(@PathParam("id") String id) {......   }

   @PUT
   @Path("/user/{id}")
   @Consumes("application/json")
   public Response updateUser (@PathParam("id") String id, User user) {   ......   }

   @POST
   @Path("/add")
   @Consumes("application/json")
   public Response addUser (User user) {   ......   }

   @DELETE
   @Path("/user/{id}")
   public Response deleteUser (@PathParam("id") String id) {   ......   }

   @GET
   @Path("/search/{text}")
   @Produces("application/json")
   public Response searchUsers(@PathParam("text") String text) {......   }
}
```

REST: suporte Java (exemplo)

Servidor por pedido

```
URI baseUri = UriBuilder.fromUri("http://0.0.0.0/").port(9090).build();
ResourceConfig config = new ResourceConfig();
config.register(MessageBoardResource.class);
JdkHttpServerFactory.createHttpServer(baseUri, config);
```

Servidor único

```
URI baseUri = UriBuilder.fromUri("http://0.0.0.0/").port(9090).build();
ResourceConfig config = new ResourceConfig();
config.register(new MessageBoardResource());
JdkHttpServerFactory.createHttpServer(baseUri, config);
```

WEB SERVICES OU REST?

REST:

- Mais simples
- Mais eficiente
- Complicado implementar serviços complexos
- Uso generalizado para disponibilização de serviços na Internet

Web services

- Mais complexo
- Grande suporte da indústria

E.g.:

- http://www.oreillynet.com/pub/wlg/3005

PARA SABER MAIS

G. Coulouris, J. Dollimore and T. Kindberg, Gordon Blair, Distributed Systems - Concepts and Design, Addison-Wesley, 5th Edition

Web services – capítulo 9.1-9.4

REST: http://en.wikipedia.org/wiki/Representational_State_Transfer