Web Services

Tecnologias de Middleware 2004/2005

José Mocito

Universidade de Lisboa

11 de Novembro, 2004

Parte I

Web Services

Porquê os Web Services?

- Importantes em cenários de integração entre negócios (B2B - *business to business*)
- Necessidade de automatização de processos nos serviços fornecidos
- Necessidade de "expor" os serviços de forma normalizada para permitir o seu uso pelo maior número de aplicações possíveis

Porquê Integrar?

- Sistemas na vida real são complexos:
  - *E-commerce*, finanças e banca, serviços de saúde...
- Sistemas complexos não podem ser construídos numa única aplicação
- Sistemas complexos requerem:
  - Aplicações distribuídas
  - Interoperabilidade
  - Transparência da localização
  - Facilidade de programação

Limitações do Middleware Convencional

- Não existe um sítio óbvio onde colocar o *middleware*
- Operações entre organizações podem demorar muito tempo
  - Alguns protocolos convencionais (ex: 2PC) não se podem aplicar
- Relações de confiança entre organizações potencialmente fracas
  - Autenticação de mensagens
  - Cifra de mensagens
  - Restrições nas operações permitidas

Integração antes dos WS

- Companhias de *brokering* que facilitam a integração desempenhando funções análogas às do *middleware* convencional
  - Ariba
  - CommerceOne
- EDIFACT - define as diferentes partes de uma mensagem e como organizar o seu conteúdo.
  - Também define um conjunto de tipos de mensagens normalizadas.

CORBA e Web Services

- CORBA, tecnologia de escolha para aplicações distribuídas
  - Numerosas histórias de sucesso
  - Norma bem aceite e activa
  - Usado na maioria das aplicações críticas
- Web Services, a tecnologia nova e emergente
  - "Hype" sem precedentes
  - Suportado por empresas importantes (IBM, Microsoft, SUN)
  - Impulsionado pelo "hype" do XML
- Assuntos relevantes
  - Como é que as duas tecnologias se comparam?
  - Como escolher qual utilizar?
  - Convergência entre ambas

Definição de Web Services

- Aplicação acessível a outras através da Web
- **UDDI** - "self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces".
- **W3C** - "a software application indentified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols".

Paradigma da Orientação ao Serviço

- **Serviço:** procedimento, método ou objecto com uma interface estável e pública que pode ser invocada por clientes
- Invocação realizada por um programa
- Serviço invocável através da *Web* e entre organizações

Protocolos de *Middleware*

- Modificados para trabalhar ponto-a-ponto e entre organizações
  - Protocolos de interacção e coordenação têm de ter em conta as restrições impostas pelas relações de confiança (tipicamente fracas) entre organizações
- Protocolos trabalham num ambiente descentralizado e entre domínios de confiança

Normalização

- Linguagens e protocolos utilizados são normalizados e de uso generalizado
- Tecnologias *Web* são utilizadas de forma generalizada e são muito bem sucedidas na interacção entre pessoas e aplicações

Tecnologias de Web Services

- Descrição de serviços
- Descoberta de serviços
- Interacção com serviços
- Composição de serviços

Descrição de Serviços

- Função análoga às IDLs
  - Nos WS é necessário incluir informação de contexto relativa ao serviço
- Cinco aspectos envolvidos
  - Linguagem base comum (XML)
  - Inferfaces (WSDL)
  - Protocolos de negócio (WSCL e BPEL)
  - Propriedades e semântica (UDDI)
  - "Verticais" (RosettaNet)

Descoberta de Serviços

- As descrições dos serviços têm de ser disponibilizadas a quem se mostrar interessado
- Descrições de serviços são guardadas num directório
- Directório permite:
  - Registar novos serviços
  - Pesquisar por e localizar serviços
- A descoberta de serviços pode ser realizada em:
  - Tempo de implementação
  - Tempo de execução
- Necessárias APIs e protocolos para clientes interagirem com o serviço de directório
  - UDDI

Interacção com Serviços

- Para interagir com os diversos serviços foram desenvolvidas diversas normas que actuam a diversos níveis
  - **Transporte.** Rede de comunicação é escondida por um protocolo de transporte (HTTP)
  - **Mensagens.** Normalização do formato e empacotamento de mensagens (SOAP)
  - **Infraestrutura de protocolos.** Permite a execução de *meta-protocolos* que facilitam e coordenam a execução dos protocolos de negócio (*WS-Coordination*)
  - **Protocolos de middleware.** Implementam protocolos ponto-a-ponto (*horizontais*) que concretizam propriedades de comunicação complexas (*WS-Transaction*)

Composição de Web Services

- Web Service simples - implementado usando o sistema local
- Web Service composto - implementação de um Web Service à custa de chamadas a outros Web Services
- Diferença entre os dois é totalmente transparente para o cliente
- Composição de serviços tem o potencial para construir serviços complexos a partir de outros mais simples
- BPEL é a linguagem de composição de serviços mais emergente

Web Services Resumido

- Integra aplicações baseadas em Web usando XML, SOAP, WSDL e UDDI
- HTTP (HyperText Trasfer Protocol) usado para transportar a informação
- XML (eXtended Markup Language) usado para descrever dados
- SOAP (Simple Object Access Protocol) usado para normalizar a troca de informação
- WSDL (Web Services Definition Language) usado para descrever os serviços disponíveis
- UDDI (Universal Description, Discovery and Integration) usado para listar os serviços disponíveis

CORBA vs. Web Services

	CORBA	**Web Services**
Type System	IDL (static + runtime checks)	XML Schemas (runtime checks only)
Transfer Syntax	CDR (binary)	XML (UTF)
State	Stateful	Stateless
Registry	Interface repository Implementation repository	UDDI/WSDL
Service Discovery	CORBA naming/ trading service	UDDI
Security	CORBA security service	HTTPS, XML signature
Firewall Tunneling	Work in progress	Over HTTP

CORBA vs. Web Services

CORBA stack	**WS stack**
IDL	WSDL
CORBA Services	UDDI
CORBA stubs/skeletons	SOAP messages
CORBA binary encoding	XML UTF encoding
GIOP/IIOP	HTTP
TCP/IP	TCP/IP

Qual utilizar?

- Interfaces *Web*
  - XML é o modelo de dados para a *Web* $\longrightarrow$ WS
- Arquitectura segura com *firewalls*
  - HTTP é habitualmente aceite por *firewalls* $\longrightarrow$ WS
- Estado
  - Estado capturado pelas instâncias de objectos $\longrightarrow$ CORBA
  - $+$ Serviços de suporte à persistência e transacções do CORBA

Qual utilizar? (cont.)

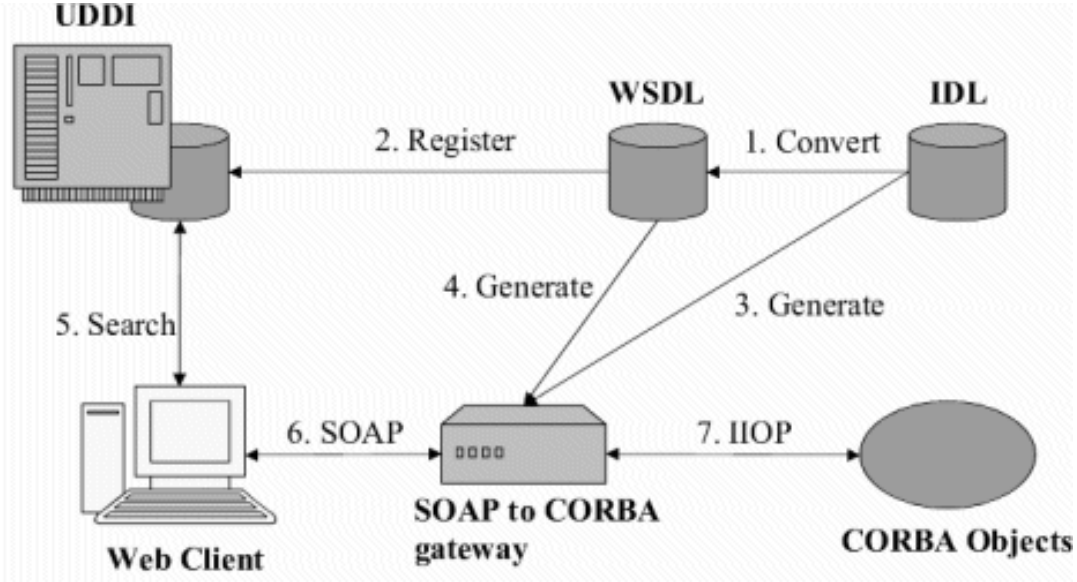
- Ambientes móveis
  - Ambientes desconectados favorecem protocolos sem estado
  - SOAP tem a noção de encaminhamento de mensagens → WS
- Clientes ligeiros
  - CORBA requer bibliotecas do ORB (tudo ou nada)
  - WS requerem apenas suporte ao envio e recepção de mensagens → WS
- Proxies/Filtros
  - Novos filtros/funcionalidades implicam alterações ao ORB
  - SOAP suporta *proxies* (reescrita de mensagens) → WS
- Desempenho
  - CORBA mais maduro + codificação binária → CORBA
  - É possível codificar as mensagens dos WS em formatos binários (ex: usar filtro de compressão)

Interoperabilidade WS↔CORBA

- Usar os Web Services para expor o serviço e utilizar uma infraestrutura CORBA para implementá-lo
  - Muitas empresas têm grandes investimentos feitos em infraestruturas de integração baseadas em CORBA
  - Os Web Services podem assim funcionar como o *middleware* integrador de outros *middleware*

Gateway SOAP-CORBA

- Pedido SOAP
  1. Pedido SOAP analisado
  2. *Gateway* procura descrição IDL do serviço CORBA
  3. *Gateway* procura descrição WSDL do pedido SOAP
  4. Pedido dinâmico CORBA é construído e envido para o servidor
  5. A resposta SOAP é construída a partir da resposta CORBA



Conclusões

- Os Web Services tentam substituir o CORBA mas apresentam um subconjunto limitado daquilo que o CORBA oferece:
  - Não suporta transacções, persistência, segurança, balanceamento de carga...
- Web Services e CORBA são complementares
  - CORBA-SOAP e SOAP-CORBA *gateways*
  - Mapeamento automático entre IDL e WSDL

Parte II

Implementações Passo-a-Passo

Orbacus

www.orbacus.com

- Orbacus é um ORB (*Object Request Broker*)
- É compatível com especificação *Common Object Request Broker Architecture* (CORBA).
- Possui tradutores de IDL para C++, Java, HTML e RTF
- Java vai ser utilizado para exemplificar a implementação de um serviço

Serviço em CORBA passo-a-passo

1. Criar o ficheiro .idl contendo as definições IDL da interface da aplicação
2. Traduzir o código no ficheiro .idl para a linguagem de programação escolhida (gerar *stubs*)
3. Implementar o servidor
4. Implementar o cliente
5. Compilar as classes de implementação e as classes geradas pelo tradutor
6. Correr a aplicação

Definir a interface

1. Criar o ficheiro .idl contendo as definições IDL da interface da aplicação

```
interface Hello
{
  void say_hello();
};
```

Traduzir IDL

2. Traduzir o código no ficheiro .idl para Java (gerar *stubs*)

```
jidl --package hello Hello.idl
```

- São geradas as seguintes classes:
  - `Hello.java`
  - `HelloHelper.java`
  - `HelloHolder.java`
  - `HelloOperations.java`
  - `HelloPOA.java`
  - `_HelloStub.java`

Implementar o servidor

3a. Implementar a classe `Hello` no servidor

```
package hello;

public class Hello_impl extends HelloPOA {
  public void say_hello() {
    System.out.println(``Hello World!'');
  }
}
```

Implementar o servidor

3b. Implementar a classe `Server` que contém os métodos `main()` e `run()` do servidor

```
package hello;
public class Server {
  public static void main(Sring[] args) {
    // ..Substituir classes CORBA do Java por
    // classes correspondentes do Orbacus
    int status = 0;
    org.omg.CORBA.ORB orb = null;
    try {
      orb = orb.omg.CORBA.ORB.init(args,props);
      status = run(orb);
    } catch (Exception ex) {status = 1}
    if (orb != null)
      try { orb.destroy(); }
      catch (Exception ex) {status = 1}
    System.exit(status);
  }
```

Implementar o servidor (cont.)

```
package hello;
  static int run(org.omg.CORBA.ORB orb)
    throws org.omg.CORBA.UserException {
    //..Obter referência ao POAManager
    Hello_impl helloImpl = new Hello_impl();
    Hello hello ) helloImpl._this(orb);
    try {
      String ref = orb.object_to_string(hello);
      String refFile = ``Hello.ref'';
      PrintWriter out = new PrintWriter(
        new FileOutputStream(refFile));
      out.println(ref);
      out.close();
    } catch(IOException ex) {return 1;}
    manager.activate();
    orb.run();
    return 0;
  }
}
```

Implementar o cliente

4. Implementar o cliente

```
package hello;
public class {
  public static void main(String args[]) {
    //..Igual ao servidor
  }
  static int run(org.omg.CORBA.ORB orb) {
    or.omg.CORBA.Object obj = null;
    try {
      String refFile = ``Hello.ref'';
      java.io.BufferedReader in =
        new java.io.BufferedReader(
          new java.io.FileReader(refFile));
      String ref = in.readLine();
      obj = orb.string_to_object(ref);
    } catch (java.io.IOException ex) {...}
    Hello hello = HelloHelper.narrow(obj);
    hello.say_hello();
    return 0;}}
```

Compilar classes e Correr Aplicação

5. Compilar as classes de implementação e as classes geradas pelo tradutor *IDL-to-Java*

```
javac hello/*.java
```

6. Correr aplicação
   a. Correr o servidor:

```
java hello.Server
```

   b. Correr o cliente:

```
java hello.Client
```

Mono

www.mono-project.com

- Plataforma de desenvolvimento *open source* baseada na *.NET framework*, com suporte para diversos ambientes de execução (Linux, Mac OS X, Windows...)
- Implementação .NET segundo as normas da ECMA para o C# e *The Common Language Infrastructure*
- **XSP.** Servidor *web* simples, escrito em C#, e que pode ser utilizado para correr aplicações ASP.NET
  - ASP.NET usado para desenvolver Web Services, entre outras funcionalidades

Web Service em Mono (.NET) passo-a-passo

1. Implementar o serviço do lado do servidor
2. Colocar implementação no directório raíz do servidor XSP
3. Gerar o *proxy* do cliente
4. Implementar a aplicação cliente
5. Compilar e executar a aplicação cliente

Implementar o servidor

1. Implementar o serviço do lado do servidor

```
<%@ WebService Language=''C#''
Class=``DirectoryLister'' %>
using System.IO;
using System.Web.Services;
[WebService(Description=''Provides a listing
  of directory contents.'')]
public class DirectoryLister : WebService {
  [WebMethod(Description=``Lists the
    Contents of the given directory.'')]
  public string [] ListDirectory(string path){
    return
     Directory.GetFileSystemEntries(path);
  }
}
```

Gerar o *Proxy* do Cliente

3. Gerar o *proxy* do cliente

```
wsdl -o:DirectoryListerProxy.cs \
  http://localhost:8080/DirectoryLister.asmx
```

- É gerado o código do *proxy* do cliente a partir de uma descrição do serviço em WSDL. Este código concretiza a interface com o Web Service de forma transparente para o cliente

Implementar o cliente

4. Implementar a aplicação cliente

```
using System;
public class DirectoryListerClient {
  public static void Main(string [] args) {
    string path = args[0];
    DirectoryLister lister =
      new DirectoryLister();
    foreach (string fileSystemEntry in
    lister.ListDirectory(path)) {
      Console.WriteLine(fileSystemEntry);
    }
  }
}
```

Compilar e Executar a Aplicação

5a. Compilar a aplicação cliente:

```
mcs -r:System.Web.Services
 DirectoryListerClient.cs
 DirectoryListerProxy.cs
 -out:DirectoryListerClient.exe
```

5b. Executar a aplicação cliente:

```
mono Directory ListerClient.exe /usr
   /usr/X11R6
   /usr/bin
   /usr/dict
   /usr/doc
       .
       .
       .
```