**Web Services II**

**Objetivos**

* Publicar e pesquisar *Web Services* no servidor de nomes jUDDI
* Ver exemplo de *Web Service* em Java construído com a abordagem *implementation-first*

**Projeto:**

* Modificar o fornecedor da parte 1 para usar UDDI
* Concretizar a operação ping do mediator-ws para contactar todos os fornecedores que encontrar

**UDDI**

Para a publicação e pesquisa de *Web Services* usa-se um servidor de nomes que implementa a norma UDDI (*Universal Description, Discovery and Integration*).

Comparando com as outras tecnologias de chamada remota: no Sun RPC usa-se o rpcbind, no Java RMI usa-se o rmiregistry, nos Web Services usa-se o UDDI.

Mais informação: [UDDI](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/06-ws2/uddi/index.html)

**JAX-R**

Para comunicar com servidores UDDI, existe a biblioteca JAX-R (Java API for XML Registries), que permite publicar, pesquisar e eliminar registos de *Web Services*.

O esquema de dados do UDDI é consideravelmente mais rico do que o habitual num servidor de nomes, com diversas entidades e relações, que permitem guardar informação de negócio sobre a organização e os seus serviços. Como consequência, o JAX-R é uma biblioteca verbosa, que obriga a escrever muitas linhas de código para realizar as tarefas mais habituais de registo e de pesquisa.

Mais informação: [JAX-R](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/06-ws2/jaxr/index.html)

**UDDI Naming**

Para tornar a utilização do UDDI mais simples foi criada a biblioteca UDDINaming que simplifica o esquema de dados para suportar apenas: 1 organização, 1 serviço e 1 implementação.  
Esta biblioteca, cujo código está disponível para consulta e modificação, torna o registo e pesquisa de serviços mais sucinto.

Mais informação: [UDDINaming JavaDoc](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/06-ws2/uddi-naming/apidocs/index.html)

**Exemplo**

O exemplo seguinte é um *Web Service* que se regista no UDDI:

* [Web Service [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/06-ws2/hello-ws_juddi.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/06-ws2/hello-ws_juddi.zip)
* [client [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/06-ws2/hello-ws-cli_juddi.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/06-ws2/hello-ws-cli_juddi.zip)

**Nota:** na aula passada vimos *Web Services* construídos a partir do contrato WSDL, mas os serviços podem também ser construídos a partir de código Java já existente, numa abordagem que se designa por *implementation-first*.  
No exemplo acima segue-se esta abordagem: a interface Hello foi anotada com @WebService e o WSDL é depois gerado automaticamente quando o servidor é lançado.

Seguem-se instruções detalhadas para construir e executar o exemplo.

**jUDDI (*Java UDDI*)**

1. O servidor de nomes a utilizar é o jUDDI alojado na RNL.
   * Para usar basta configurar a propriedade uddi.url com o seguinte valor: http://user:pass@uddi.sd.rnl.tecnico.ulisboa.pt:9090/
   * Os valores de *user* e *pass* para acesso ao UDDI devem ser pedidos ao professor no laboratório.
2. Para que a configuração seja feita para todos os exemplos, pode-se criar um perfil Maven que sobrepõe o valor da propriedade para todos os projetos do utilizador. Para isso, criar um ficheiro [settings.xml](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/06-ws2/settings.xml) na pasta .m2 da *home* do utilizador.

* Em alternativa ao jUDDI da RNL, pode-se usar um jUDDI local:
  + O pacote de instalação pode ser encontrado na lista de [software](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/software/index.html) que se pediu para instalar.
  + Para lançar o servidor, basta executar o seguinte comando na pasta juddi-.../bin:
    - startup.sh (Linux e Mac)
    - startup.bat (Windows)
    - juddi-startup / juddi-shutdown (laboratórios das aulas)

**UDDINaming**

1. Obter o código da biblioteca UDDINaming:
   * [Biblioteca uddi-naming [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/06-ws2/uddi-naming.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/06-ws2/uddi-naming.zip)
2. Instalar o módulo no repositório Maven local:
   * cd uddi-naming
   * mvn install
   * A biblioteca inclui testes de integração (IT - *Integration Tests*)  
     que verificam o funcionamento correto do jUDDI antes de instalar o módulo.
   * Uma vez instalado o módulo no repositório Maven local, a biblioteca pode ser usada como dependência em qualquer pom.xml.

**Servidor JAX-WS**

1. Construir e executar o servidor:
   * cd hello-ws\_juddi
   * mvn compile exec:java  
     O nome da classe a executar e os argumentos estão definidos no pom.xml  
     O servidor deve executar sem erros, disponibilizando o *endpoint address* e registando-se no UDDI.
2. Confirmar que o servidor está à espera de pedidos,  
   consultando o contrato que é gerado automaticamente:
   * <http://localhost:8080/hello-ws/endpoint?wsdl>
   * <http://localhost:8080/hello-ws/endpoint?xsd=1>

**Cliente JAX-WS**

1. Construir o cliente:
   * cd hello-ws-cli\_juddi
   * mvn compile  
     Executa previamente generate-sources onde o cliente obtém o contrato WSDL a partir do servidor e  
     usa a ferramenta wsimport para gerar as classes de invocação do serviço (em target/)
2. Executar o cliente:
   * mvn exec:java  
     O cliente deve executar sem erros, consultando o UDDI para descobrir o endereço do servidor,  
     e fazendo uma invocação remota.

Resumindo:  
primeiro foi configurado o servidor de nomes jUDDI da RNL.  
Depois foi instalada a biblioteca UDDINaming no repositório Maven local, que testa também o funcionamento do servidor jUDDI.  
Em seguida, foi construído e iniciado o servidor, que se regista no jUDDI e fica à espera de pedidos no *endpoint address*.  
Finalmente, o cliente obtém o WSDL a partir do servidor e gera o código de invocação que permite depois fazer invocações remotas.

Visão global do exemplo Hello World: [UML](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/ping/uml_full.png)

**Exercício**

**Segunda parte do Projeto**

Nesta parte devem surgir múltiplos fornecedores e aparece também o serviço mediator-ws que os vai pesquisar e depois consultar. Assim sendo, o UDDI torna-se útil e necessário.

O [ponto de continuação](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/proj/P2_continue.zip) contém novos ficheiros (serão disponibilizados após a primeira entrega). Editar os pom.xml para substituir CXX pelo identificador do grupo.

Pretende-se agora modificar supplier-ws para se registar no UDDI.

1. Adicionar a dependência para a biblioteca UDDINaming:
2. ...
3. <!-- UDDI Naming -->
4. <dependency>
5. <groupId>pt.ulisboa.tecnico.sdis</groupId>
6. <artifactId>uddi-naming</artifactId>
7. <version>1.2</version>
8. </dependency>

...

1. A SupplierApp deverá receber o endereço do UDDI e o nome do serviço como argumento.
2. O SupplierEndpointManager deverá ter um novo construtor que recebe o endereço do UDDI e o nome do serviço, e que efetua o registo (e apaga quando o serviço termina).
3. Feitas as alterações, o servidor deverá iniciar-se, registar-se no UDDI e depois ficar à espera de pedidos.

"Multiplicar" o supplier-ws.

1. A forma mais simples de permitir múltiplas instâncias do fornecedor é parametrizar as propriedades de configuração com um número de instância.
2. Adicionar as seguintes definições ao pom.xml (substituir CXX pelo identificador do grupo):
3. ...
4. <ws.i>1</ws.i>
5. <ws.host>localhost</ws.host>
6. <ws.port>808${ws.i}</ws.port>
7. <ws.url>http://${ws.host}:${ws.port}/supplier-ws/endpoint</ws.url>
9. <ws.name>CXX\_Supplier${ws.i}</ws.name>

...

1. Para lançar um fornecedor:
   * mvn compile
   * mvn exec:java  
     Por omissão, será a instância 1, que fica à escuta de pedidos no porto 8081 e que se regista com o nome CXX\_Supplier1.
2. Para lançar outro fornecedor:
   * mvn exec:java **-Dws.i=2**  
     A instância **2** fica à escuta de pedidos no porto 808**2** e regista-se com o nome CXX\_Supplier**2**.

Pretende-se agora modificar supplier-ws-cli para pesquisar no UDDI.

1. Adicionar a dependência para a biblioteca UDDINaming.
2. O SupplierClient deverá ter um novo construtor que recebe o endereço do UDDI e o nome do serviço a contactar, e que efetua a pesquisa para localizar o servidor.
3. A SupplierClientApp deverá receber o endereço do UDDI e o nome do serviço como argumento.

Vamos construir uma operação simples do servidor mediator-ws que contacta os fornecedores.

1. O mediator-ws vai ser cliente de supplier-ws.  
   Em vez de repetir código, vamos usar o objeto SupplierClient, já desenvolvido e testado.  
   1. Em primeiro lugar, instalar o módulo do cliente do fornecedor:  
      cd supplier-ws-cli  
      mvn install
   2. Depois, acrescentar a dependência Maven no mediator-ws/pom.xml (substituir CXX pelo identificador do grupo):
   3. ...
   4. <dependency>
   5. <groupId>org.komparator.CXX</groupId>
   6. <artifactId>supplier-ws-cli</artifactId>
   7. <version>1.0-SNAPSHOT</version>
   8. </dependency>

...

O sistema em funcionamento terá três processos:

* + - O cliente de testes do mediator;
    - O mediator (que engloba o código dos módulos mediator-ws e supplier-ws-cli);
    - O(s) servidor(es) do(s) fornecedor(es).

[UML](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/proj/uml_deployment.png)

1. Concretizar a operação ping do mediador.
   1. Consultar o UDDI para pesquisar fornecedores.
   2. Criar um SupplierClient para cada fornecedor encontrado.
   3. Chamar a operação ping de cada um.
   4. Juntar as respostas e devolver como resultado.

Finalmente, usar o mediator-ws-cli para chamar o ping que vai tocar o mediador e todos os fornecedores.

* Abrir consola para Fornecedor 1:
  + cd supplier-ws
  + mvn compile exec:java
* Abrir consola para Fornecedor 2:
  + cd supplier-ws
  + mvn compile exec:java -Dws.i=2
* Abrir consola para Mediador:
  + cd mediator-ws
  + mvn compile exec:java
* Finalmente, na consola para o cliente do Mediador:
  + cd mediator-ws-cli
  + mvn compile exec:java  
    O resultado final do ping deverá ser impresso nesta consola.

**O que falta fazer?**

* No mediator-ws:
  + Implementar todas as operações de acordo com o enunciado
  + Fazer um bom tratamento de [exceções](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/01-tools/exceptions/index.html)
  + Não esquecer também que o servidor é multi-tarefa (*thread*) e que os acessos a dados partilhados devem ser devidamente sincronizados
* No mediator-ws-cli:
  + Fazer testes de integração das operações principais
    - Testar casos mais importantes
    - Não esquecer os casos com entradas incorretas: null, "", valores inesperados, etc
  + Os testes de integração correm com mvn verify

**Continuação de bom trabalho!**