Programação CORBA

Vale lembrar que CORBA (Common Object Request Broker Architecture) é simplesmente uma especificação desenvolvida pela OMG (Object Management Group). Uma implementação CORBA é conhecida como um ORB (ou Object Request Broker) e há várias implementações como VisiBroker, ORBIX, ORBacus etc. JavaIDL é uma implementação da SUN que vem como um pacote a partir do JDK1.3.

O Desenvolvimento de uma Aplicação CORBA

Os passos fundamentais para o desenvolvimento de aplicações CORBA são:

1. Definição de interface(s) em IDL (Interface Definition Languagem)
2. Mapeamento de interface(s) IDL em Java (ou outras linguagens)
3. Implementação da(s) interface(s)
4. Desenvolvimento de servidor(es) – inclui compilação
5. Desenvolvimento de cliente(s) – inclui compilação
6. Execução do serviço de nomes, do(s) servidro(es) e do(s) cliente(s).

O exemplo a seguir mostra o desenvolvimento de uma aplicação de **transferência de arquivo** em JavaIDL.

Definição da interface

Na definição de uma interface, pense no tipo de operações que o servidor suportará. Na aplicação de transferência de arquivo, o cliente invocará um método para baixar um arquivo. O código abaixo mostra a interface de `FileInterface.`

`FileInterface.idl`

```
interface FileInterface {
    typedef sequence<octet> Data;
    Data downloadFile(in string fileName);
};
```

Uma vez definida a interface, compile usando o compilador `idlj`. Para tanto, antes

Execute o arquivo de lote para configuração do ambiente através do atalho: Start -> Programs -> Linguagens -> Java -> jdk1.3

O compilador `idlj` aceita opções que permitem especificar a geração de stubs de clientes, skeletons de servidores, ou ambos. A opção `-f<lado>` é usada para especificar o que gerar: lado pode ser `client`, `server` ou `all`. Dado que a aplicação executará em duas máquinas distintas, pode-se usar a opção `-fserver` no lado do servidor, e `-fclient` no lado do cliente.

```
prompt> idlj -fserver FileInterface.idl
```

Este comando gera vários arquivos como skeletons, holder e helper, e outros. O arquivo

`_FileInterfaceImplBase` será usado pela classe que implementa a interface.

Implementação da interface

A implementação de uma interface (seus métodos) é chamada de *servant*, e no código abaixo observe que a classe FileServant estende a classe _FileInterfaceImplBase para especificar que este *servant* é um objeto CORBA.

FileServant.java

```
import java.io.*;

public class FileServant extends _FileInterfaceImplBase {
    public byte[] downloadFile(String fileName){
        File file = new File(fileName);
        byte buffer[] = new byte[(int)file.length()];
        try {
            BufferedInputStream input = new
            BufferedInputStream(new FileInputStream(fileName));
            input.read(buffer,0,buffer.length);
            input.close();
        } catch(Exception e) {
            System.out.println("FileServant Error: "+e.getMessage());
            e.printStackTrace();
        }
        System.out.println("Transfering file " + fileName + " ...");
        return(buffer);
    }
}
```

Desenvolvimento do servidor

A classe FileServe, mostrada no código abaixo, implementa um servidor CORBA que faz o seguinte:

1. Inicializa o ORB
2. Cria um objeto FileServant (interface do serviço oferecido)
3. Registra o objeto no serviço de nomes CORBA (COS Naming) – COS = CORBA Object Services
4. Espera por pedidos de cliente

FileServer.java

```
import java.io.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class FileServer {
    public static void main(String args[]) {
        try{
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);
            // create the servant and register it with the ORB
            FileServant fileRef = new FileServant();
            orb.connect(fileRef);
            // get the root naming context; returns a generic CORBA object
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            // cast (narrow down) the reference to the root of the naming
```

```
            // service (a generic CORBA object) to its proper type
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            // Bind the object reference in naming
            NameComponent nc = new NameComponent("FileTransfer", " ");
            NameComponent path[] = {nc};
            ncRef.rebind(path, fileRef);
            System.out.println("Server started....");
            // Wait for invocations from clients
            java.lang.Object sync = new java.lang.Object();
            synchronized(sync){
                sync.wait();
            }
        } catch(Exception e) {
            System.err.println("ERROR: " + e.getMessage());
            e.printStackTrace(System.out);
        }
    }
}
```

Desenvolvimento do cliente

1. Obtém uma referência para o serviço de nomes
2. Usa a referência obtida para acessar o serviço de nomes e encontrar outros serviços
3. Invoca métodos nos serviços encontrados

FileClient.java

```
import java.io.*;
import java.util.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

public class FileClient {
    public static void main(String argv[]) {
        try {
            // create and initialize the ORB
            ORB orb = ORB.init(argv, null);
            // get the root naming context
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent("FileTransfer", " ");
            // Resolve the object reference in naming
            NameComponent path[] = {nc};
            FileInterfaceOperations fileRef =
                FileInterfaceHelper.narrow(ncRef.resolve(path));

            if(argv.length < 1) {
                System.out.println("Usage: java FileClient filename");
            }

            // save the file
            File file = new File(argv[0]);
            byte data[] = fileRef.downloadFile(argv[0]);
            BufferedOutputStream output = new
            BufferedOutputStream(new FileOutputStream(argv[0]));
            output.write(data, 0, data.length);
            output.flush();
            output.close();
            System.out.println("File " + argv[0] + " transfered!");
        } catch(Exception e) {
            System.out.println("FileClient Error: " + e.getMessage());
            e.printStackTrace();
        }
```

```
    }
}
```

Compilação dos Componentes da Aplicação

```
prompt> javac FileServer.java
```

Se o stub do cliente ainda não foi gerado, copie o arquivo FileInterface.idl e compile-o especificando que deseja gerar o stub do cliente:

```
prompt> idlj -fclient FileInterface.idl

prompt> javac FileClient.java
```

Executando a Aplicação

1. Execute o serviço de nomes CORBA. O comando `tnameserv` executa, por *default*, na porta 900, mas pode ser inicializado em outra porta, por exemplo a porta 1500:

   ```
   prompt> tnameserv -ORBInitialPort 1500
   ```

2. Execute o servidor (em *background*, usando `start`), especificando (ou não, no caso da porta *default*) a porta onde se encontra o serviço de nomes:

   ```
   prompt> start java FileServer -ORBInitialPort 1500
   ```

3. Execute o cliente, especificando o nome do arquivo que deseja baixar do servidor da aplicação (este arquivo deve existir no servidor):

   ```
   prompt> java FileClient arquivo.txt -ORBInitialPort 1500
   ```

Observações

- Se o serviço de nomes estiver rodando em uma máquina distinta, esta pode ser especificada usando a opção `ORBInitialHost`, por exemplo:

  ```
  prompt> java FileClient arquivo.txt -ORBInitialHost caruaru -ORBInitialPort 1500
  ```

- Alternativamente, estas opções podem estar no código dos componentes da aplicação. Assim em vez de inicializar o ORB como:

  ```
  ORB orb = ORB.init(argv, null);
  ```

  Inicialize-o da seguinte forma:

  ```
  Properties props = new Properties();
  props.put("org.omg.CORBA.ORBInitialHost", "caruaru");
  props.put("org.omg.CORBA.ORBInitialPort", "1500");
  ORB orb = ORB.init(argv, props);
  ```

- Note que em JavaIDL não há transparência de localização em relação ao servidor de nomes CORBA.

Referência

Qusay H. Mahmoud, "Distributed Java Programming with RMI and CORBA". Janeiro 2002.
URL: http://developer.java.sun.com/developer/technicalArticles/RMI/rmi_corba/