presentation

DAD – Distributed Applications Development
Cristian Toma

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics
www.dice.ase.ro
Agenda for Lecture 9

1. CORBA + DEMO
2. Web Services + DEMO
3. Exchange Ideas
CORBA – Common ORB Architecture
1. CORBA Overview

CORBA – Common Object Request Broker Architecture

OMG created CORBA. CORBA loves JAVA.

1. CORBA Basic Architecture
2. CORBA Basic Flow
3. ORB – Object Request Broker
4. GIOP vs. IIOP, IOR
5. IDL is CORBA “language independency”
6. CORBA Services
7. Products – SUN Java IDL
Curs 9

Partea I – CORBA – Basic Architecture

Client Host Machine

- Client Object
- Stub
- ORB

Server Host Machine

- Remote Object
- Skeleton
- ORB

IIOP
Stubs and Skeletons

- **Stub**
  - lives on client
  - pretends to be remote object

- **Skeleton**
  - lives on server
  - receives requests from stub
  - talks to true remote object
  - delivers response to stub

Similar with RMI 1.1 in concept
Partea I – CORBA – Basic Architecture

Curs 9

Client

ORB

Server

“Object Bus”

request

response
Partea I – CORBA Basic Flow without ORB is like RMI

Curs 9

Client Virtual Machine

Client

Stub

Server Virtual Machine

Remote Object

Skeleton

Server

“Gigel”

Name Server Virtual Machine
Partea I – CORBA Basic Flow without ORB is like RMI

1. Server Creates Remote Object
2. Server Registers Remote Object
Partea I – CORBA Basic Flow without ORB is like RMI

1. Client requests object from Name Server
2. Name Server returns remote reference (and stub gets created)
5. Client invokes stub method
6. Stub talks to skeleton
7. Skeleton invokes remote object method

Partea I – CORBA Basic Flow without ORB is like RMI
OMG does not specify exactly where the ORB is in CORBA.

Depending on which CORBA implementation (product) is used, the ORB may be:

- A set of run-time libraries
- A server machine/process
- Part of the operating system (Spring)
- Part of a web browser (Netscape)
FEATURES:

- Object Request Broker - “Object Bus”
- Handles all communication among objects
- Each host (machine) has its own ORB
- ORBs know how to talk to each other
- ORB also provides basic services to client

RESPONSABILITIES:

- Find the object implementation for the request
- Prepare the object implementation to receive the request
- Communicate the data making up the request
- Retrieve results of request

Note:

- There’s an ORB on the server too, and ORB receives request
- ORB is good if Stub and Skeleton are written in different programming language
• Method invocations
  – Static and Dynamic
  – Remote objects or CORBA services
• High-level language bindings
  – Use your favorite language; ORB translates
• Self-describing
  – Provides metadata for all objects and services
• Local or remote
  – Same API wherever target object lives

• Preserves context
  – Distributed security and transactions

• Coexistence with legacy code
  – Just provide a wrapper object
What is an ORB really?

- Not a separate process
- Library code that executes in-process
- Listens to TCP ports for connections
  - One port per local object
- Opens TCP sockets to other objects
  - N ports per remote machine
The OMG agreed protocol for ORB interoperability is called the General Inter-ORB Protocol (GIOP).

GIOP defines the logical data representation and message formats for communication.

The OMG defines a realization of GIOP that uses TCP/IP as the transport layer. This specialization is called the Internet Inter-ORB Protocol (IIOP).
Interoperability

<table>
<thead>
<tr>
<th>Message type</th>
<th>Originator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Client</td>
<td>Contains an invocation request</td>
</tr>
<tr>
<td>Reply</td>
<td>Server</td>
<td>Contains the response to an invocation</td>
</tr>
<tr>
<td>LocateRequest</td>
<td>Client</td>
<td>Contains a request for the exact location of an object</td>
</tr>
<tr>
<td>LocateReply</td>
<td>Server</td>
<td>Contains location information on an object</td>
</tr>
<tr>
<td>CancelRequest</td>
<td>Client</td>
<td>Indicates client no longer expects a reply</td>
</tr>
<tr>
<td>CloseConnection</td>
<td>Both</td>
<td>Indication that connection will be closed</td>
</tr>
<tr>
<td>MessageError</td>
<td>Both</td>
<td>Contains information on an error</td>
</tr>
<tr>
<td>Fragment</td>
<td>Both</td>
<td>Part (fragment) of a larger message</td>
</tr>
</tbody>
</table>

Message types in GIOP

- General Inter-ORB Protocol (GIOP) assumes a transport protocol that is reliable, connection oriented and support byte stream notion
- Internet Inter-ORB Protocol (IIOP) is GIOP built on TCP
GIOP Messages:

- **Request message** contains a complete marshaled invocation request (object reference, name of the method, input parameters)
  – Each request has request ID

- **Reply message** contains a marshaled return values and output parameters
  – Reply message has a corresponding request ID of the request message
• An IOR (Interoperable Object Reference) is managed internally by the interoperating ORBs.
• An IOR may include:
  – ORB's internal object reference
  – Internet host address
  – Port number
• It is not necessary for an application programmer to know the structure of an IOR.
Curs 9

Partea I – CORBA – ORB

IOR:0000000000000001049444c3a466f7274756e653a312e3000000001000000000000005e00010000000000186d6179666c792e73642e6d6f6e6573642e6d6f6e6173682e6564752e61750000000363a5c6d6179666c792e73642e6d6f6e6173682e6564752e61753a666f7274756e653a466f7274756e65313a3a49523a466f7274756e65

Diagrama:

- Interoperable Object Reference (IOR)
- Tagged Profile
- Repository identifier
- Profile ID
- Profile
- ... ...
- IIOP version
- Host
- Port
- Object key
- Components
- POA identifier
- Object identifier
- Other server-specific information
Pseudo-objects

• The ORB is a pseudo-object
• It works just like a remote object, only it’s local

The Basic Object Adapter (BOA)

• Another pseudo-object
• Helps register objects with the ORB
• Functions
  – Maintain Implementation Repository
  – Generate and interpret object references
  – Activate and deactivate implementation objects
  – Invoke methods via skeletons
Why do you need both an ORB and a BOA?

- Allows vendors to optimize or enhance functionality
  - register many objects *en masse*
  - cache object state elsewhere
- E.g. Object database
Using the BOA

- Slightly different procedure for initializing objects
- Hides name service from you
  - Ask the BOA to register the object
  - Ask the Helper object to bind the object
- Once the object is created, interface is identical
  - Just call methods using normal Java syntax
Object Adapters
Portable Object Adapter – POA

The POA:
• Allows the ORB and Objects to communicate
• Provides many services:
  – Dispatches client calls to server objects
  – Handles incoming client calls
  – Handles registration of servers
  – Instantiates objects at runtime and creates and manages object references
• *POA is a BOA – Basic Object Adapter*
A typical example of the relationship between POA and servants
Different Approaches for POA

Mapping of CORBA object identifiers to servants of the same class
a) The POA supports multiple servants.
b) The POA supports a single servant.
Object adaptor for implementing a specific activation policy

- An object adaptor may be responsible for one or more objects
- Object adaptor is not aware of interfaces implemented on the object it controls
- Object skeleton provides invoke() function for the adaptor to call and pass the message the adaptor receives
- Skeleton does marshall and unmarshall
• Interface Definition Language
• Defines protocol to access objects
• Like a contract
• Well-specified
• Language-independent
Hello.idl

interface Hello {
    string sayHello();
};


From: http://java.sun.com/docs/books/tutorial/idl/hello/index.html
//IDL Sample:
module Calc {
    interface Adder {
        long add(in long x, in long y);
    }
}

• Defines an class called Adder which generates objects with a method called “add”
### IDL vs. Java vs. C++ concepts

<table>
<thead>
<tr>
<th>IDL</th>
<th>Java</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
<td>package</td>
<td>namespace</td>
</tr>
<tr>
<td>interface</td>
<td>interface</td>
<td>abstract</td>
</tr>
<tr>
<td>operation</td>
<td>method</td>
<td>member</td>
</tr>
<tr>
<td>attribute</td>
<td>pair of methods</td>
<td>pair of functions</td>
</tr>
</tbody>
</table>

Partea I – CORBA use IDL for independency
Partea I – CORBA use IDL for independency

IDL Modules

- Map to Java packages
- Unfortunately, it has the root level name of the module
- Clutters up your package hierarchy

IDL Interfaces

- Map to Java interfaces

IDL Operations

- Map to Java methods
IDL Attributes

• Map to pair of functions – like C# do
• IDL
  - string name;
• Java
  - public void name(String val);
  - public String name();
CORBA Services

• APIs for low-level, common tasks
• Life Cycle Service
  – creating, copying, moving, removing objects
• Naming Service
  – Register objects with a name
  – Look up objects by name
Concurrency Control Service
   - Obtain and release exclusive locks

Transaction Service
   - Two-phase commit coordination
   - Supports nested transactions

Persistence Service
   - Storing objects in a variety of databases
     - RDBMS, OODBMS, file systems

Security Service
   - Authentication, ACLs, encryption, etc.

Event Service
   - Uncoupled notifications
• Relationship
• Externalization
• Query
• Licensing
• Properties
• Time
• Trader
• Collection
• ... and so on...

• See what means about CORBA will be never being implemented?
Remember!

- CORBA is a standard by OMG, not an implementation.
- There are many implementations in Java and C/C++:
  - SUN JDK – Java 2 ORB
  - VisiBroker for Java or for C++
  - Orbacus
  - Orbix
  - Visigenic(freely available),
- Depending on the particular CORBA implementation, nonstandardized aspects may be different.
• Should be named “Java CORBA”
  – More than just IDL
  – Full (?) implementation of CORBA in 100% Java
• SUN Java IDL has 3 Parts:
  – ORB
  – Naming Service – COS – CORBA Object Service: tnameserv.exe (Non-persistent) & orbd.exe (Persistent)
  – idltojava & javatoidl compiler – now: idlj.exe
• Ships starting with JDK 1.2
The Java ORB

• 100% Java
• Generic
• Allows Java IDL applications to run either as stand-alone Java applications, or as applets within Java-enabled browsers
• Uses IIOP
The compiler: Transparent API

- JavaIDL turns IDL into direct method calls
- Easy to program
- Clients have no knowledge of implementation
- Highly portable
The compiler idlj: IDL to Java Mapping

- Defined by OMG and implemented here by SUN
- Translates IDL concepts into Java language constructs
- Everything is accessible by writing normal-looking Java code
**The compiler idlj: IDL to Java Type Mapping**

<table>
<thead>
<tr>
<th>IDL Type</th>
<th>Java Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>char / wchar</td>
<td>char</td>
</tr>
<tr>
<td>octet</td>
<td>byte</td>
</tr>
<tr>
<td>short / unsigned short</td>
<td>short</td>
</tr>
<tr>
<td>long / unsigned long</td>
<td>int</td>
</tr>
<tr>
<td>long long / unsigned long long</td>
<td>long</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>string / wstring</td>
<td>String</td>
</tr>
</tbody>
</table>
The compiler: idlj or idltojava

• Development tool provided by SUN
• Automatically generates Java stubs, skeletons, helpers, holders, ... from IDL
• Generates stubs for specific remote interfaces
Stubs – Client Side

• Java objects call stub methods
• Stubs communicate with CORBA objects — and vice versa
• Transparent integration

Skeletons – Server Side

• ORB passes request to skeleton (like a stub)
• Skeleton calls local implementation
DEMO – CORBA Server & Client
Remote Interfaces and Stubs

IDL Interface

Client
Stub
Skeleton
Remote Object (Server)

implements
extends
• Proxy is equivalent to client stub in RPC/RMI; it provides the same object interface as the server object
• Proxy marshalls method invocations into messages and unmarshall the reply messages
• Skeleton is like a server stub in RPC/RMI
Object Model

- ORB provides few services through ORB interface
- Operations to marshall and unmarshall object references
- Getting object reference to an object implementing a specific CORBA service
Section Conclusion

Fact: **DAD core is based on Java RMI**

In few **samples** it is simple to remember: Java RMI Architecture with JRMP protocol analysis in real time plus the core actions for distributed computing and systems:

Picture processing within a RMI Cluster.
DAD Section 4 – Core Middleware Technologies for Distributed Computing / Distributed App Development

Java WS – Web Services
## 2. Java WS Overview

Java WS – Web Services

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Web Service Overview</td>
</tr>
<tr>
<td>2.</td>
<td>XML-RPC</td>
</tr>
<tr>
<td>3.</td>
<td>Web Service WSDL</td>
</tr>
<tr>
<td>4.</td>
<td>Web Service SOAP Req &amp; Resp</td>
</tr>
<tr>
<td>5.</td>
<td><strong>Java Web Service Server – Bottom-Up</strong></td>
</tr>
<tr>
<td>6.</td>
<td>Java Web Service Server – Top-Down</td>
</tr>
<tr>
<td>7.</td>
<td><strong>Java Web Service Client – ADB - Top-Down</strong></td>
</tr>
</tbody>
</table>
Partea II – Web Services Overview

- "a software system designed to support interoperable Machine to Machine interaction over a network." (W3C)
- Remote Procedure Call over HTTP

RPC over HTTP using XML => Web Service
Curs 9

Partea II – Web Services – XML-RPC

POST /xmlrpc1/xmlrpc HTTP/1.1
Content-Type: text/xml
User-Agent: Apache XML RPC 3.0 (Sun HTTP Transport)
Content-Length: 199
Cache-Control: no-cache
Pragma: no-cache
Host: 192.168.1.100:8080
Accept: text/html, image/gif, image/jpeg, */; q=0.2, */*; q=0.2
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>Calculator.add</methodName>
  <params>
    <param><value><i4>2</i4></value></param>
    <param><value><i4>3</i4></value></param>
  </params>
</methodCall>

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml
Content-Length: 189
Date: Thu, 19 Jul 2007 09:15:36 GMT

<?xml version="1.0" encoding="UTF-8"?>
<methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">
  <params>
    <param><value><i4>5</i4></value></param>
  </params>
</methodResponse>
Curs 9

Partea II – Web Services – XML-RPC

- remote procedure calls using HTTP for transport and XML as encoding
- XML-RPC message is an HTTP-POST request
- very simple XML - doesn't use XML namespaces or attributes
- works only with HTTP

```xml
POST /XMLRPCServer/xmlrpc HTTP/1.1
Content-Type: text/xml
User-Agent: Apache XML RPC 3.0 (Jakarta Commons httpclient Transport)
Host: 192.168.1.103:8080
Content-Length: 199

<?xml version="1.0" encoding="UTF-8"?>
<methodCall><methodName>Calculator.add</methodName><params><param><value><i4>2</i4></value></param><param><value><i4>3</i4></value></param></params></methodCall>
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet 2.4; JBoss-4.2.2.GA (build: SVNTag=JBoss_4_2_2_GA
date=200710221139)/Tomcat-5.5
Content-Type: text/xml
Content-Length: 128
Date: Mon, 21 Jan 2008 21:46:57 GMT

<?xml version="1.0" encoding="UTF-8"?>
<methodResponse><params><param><value><i4>5</i4></value></param></params></methodResponse>
```
Web Services Products:

- Apache AXIS2 for Java – course
- SUN Java-WS – integrated in NetBeans 6.0.1 – seminar

WSDL Concept:

- WSDL – Web Service Definition Language
- WSDL = XML document that describes a web service
- WSDL – Specifies the location of the service and the operations (or methods) it exposes
A WSDL describes a web service using the following elements:

- `<binding>` contains communication protocols used by the web service
- `<portType>` defines the operations that can be performed by a web service, and the messages that are involved
- `<message>` defines the in/out messages that are used by the operations ⇔ methods
- `<types>` defines data types used by the messages

A WSDL document can also contain extension elements and a `<service>` element that makes it possible to group together the definitions of several web services.
Partea II – Web Services – SOAP

- SOAP – Simple Object Access Protocol
- simple XML based protocol to let applications exchange information over HTTP
- defines a format for sending messages
- allows you to get around firewalls (firewalls and proxy servers normally block RPC traffic)
- platform and language independent
- transport Protocols: TCP, HTTP, SMTP, and MQ – Message Queues
A SOAP message is an ordinary XML containing the following elements:

- `<Envelope>` which identifies that the XML document is a SOAP message

- `<Header>` which contains application specific information about the SOAP message (such as authentication, payment, etc).

- `<Body>` contains the actual SOAP message

- `<Fault>` contains eventual error messages
The SOAP messaging protocol allows you to send MIME attachments via SOAP messages. WSDL provides a description of these attachments.

SOAP with Attachments API for Java (SAAJ) allows you to do XML messaging from the Java platform.

The SAAJ API conforms to the Simple Object Access Protocol (SOAP) 1.1 and 1.2 specifications and the SOAP with Attachments specification.
The SAAJ API provides the `SOAPMessage` class to represent a SOAP message.

SAAJ API also provides the `SOAPPart` class that is the container for the SOAP-specific portion of a `SOAPMessage` object. All messages are required to have a `SOAPPart`.

A `SOAPPart` object is a MIME part and has the MIME headers `Content-Id`, `Content-Location`, and `Content-Type`.
A SOAP message may include one or more attachment parts in addition to the SOAP part.

- The SAAJ API provides the `AttachmentPart` class to represent an attachment part of a SOAP message.
- An attachment part can contain any kind of content.
Partea II – Web Services - UDDI

- an XML-based registry for businesses worldwide to list themselves on the Internet.
- used in a similar manner to JNDI – to register and locate web services
- is a directory for storing information about web services
- is a directory of web service interfaces described by WSDL
Partea II – Web Services – A Java Architecture

1. Request for location of a web service that does X (UDDI)
2. Response: Server A (UDDI)
3. How to address the web service?
4. Provides web service interface
5. Request operation X (SOAP)
6. Result of operation X (SOAP)
1. set JAVA_HOME, AXIS2_HOME and CATALINA_HOME and all \bin to Path
2. compile java file: javac -cp . package/ServiceName.java
3. create wsdfl file: %AXIS2_HOME%/bin/java2wsdl -cp . -cn package.ServiceName -of ServiceName.wsdl
4. the structure of this service directory will be as follows:
   - ServiceName
     - META-INF
       - services.xml
       - ServiceName.wsdl
     - lib
     - webserv
       - ServiceName.class
5. Create service.xml file (set description for service and parameter name)

6. Copy ServiceName directory or ServiceName.aar to: axis2.war/WEB-INF/services directory in your Servlet engine

7. start the web server

8. Obtain description of the service:
   - http://localhost:8080/axis2/services/ServiceName?wsdl
   - http://localhost:8080/axis2/services/ServiceName?xsd
1. set JAVA_HOME, AXIS2_HOME and CATALINA_HOME and all \bin to Path
2. create WSDL file for service & create the skeleton for the web service using the command: %AXIS2_HOME%/bin/WSDL2Java -uri META-INF/ServiceName.wsdl -p webserv -d adb -s -ss -sd -ssi -o output_directory
3. the following structure of files is generated:
   - resources
     - services.xml
     - ServiceName.wsdl
   - src
     - webserv
       - java files
   - build.xml
4. modify ServiceNameSkeleton.java by adding code to the methods

5. set AXIS2_CASS_PATH and compile java files:
   
   ```
   javac -cp .;%AXIS2_CLASSPATH% package/*.java
   ```

- set up the directory structure:

  ServiceName
  - META-INF
    - services.xml
  - ServiceName.wsdl
  - lib
  - package
    - class files
  - build.xml
7. copy ServiceName directory or ServiceName.aar to: axis2.war/WEB-INF/services directory in your servlet engine and start the server

8. Test the service:
   - http://localhost:8080/axis2/services/ServiceName?wsdl
   - http://localhost:8080/axis2/services/ServiceName?xsd
CLIENT types in AXIS 2:

**Axis2 Databinding Framework (ADB):** ADB is probably the simplest method of generating an Axis2 client. In most cases, all of the pertinent classes are created as inner classes of a main stub class. It is not meant to be a full schema binding application, and has difficulty with structures such as XML Schema element extensions and restrictions.

**XMLBeans:** Unlike ADB, XMLBeans is a fully functional schema compiler, so it doesn't carry the same limitations as ADB. It is, however, a bit more complicated to use than ADB. It generates a huge number of files, and the programming model, while being certainly usable, is not as straightforward as ADB.
CLIENT types in AXIS 2:

JiBX: JiBX is a complete databinding framework that actually provides not only WSDL-to-Java conversion, but also Java-to-XML conversion. In some ways, JiBX provides the best of both worlds. JiBX is extremely flexible, enabling you to choose the classes that represent your entities, but it can be complicated to set up. On the other hand, once it is set up, actually using the generated code is as easy as using ADB.
1. Generate client **ADB** from web service wsdl file using th command:

```
%AXIS2_HOME%/bin/WSDL2Java -uri ServiceName.wsdl -p clientpackage -d adb -s
```

2. compile the stub

```
javac -cp .;%AXIS2_CLASSPATH% clientpackage/ClientStub.java
```

3. Compile and run the client

```
javac -cp .;%AXIS2_CLASSPATH% clientpackage/Client.java
java -cp .;%AXIS2_CLASSPATH% clientpackage/Client.java
```
Section Conclusions

Java WS – Web Services DEMO
Distributed Application Development

Communicate & Exchange Ideas
Questions & Answers!

But wait...

There’s More!
Thanks!

DAD – Distributed Application Development
End of Lecture 9