CORBA – part 1/2

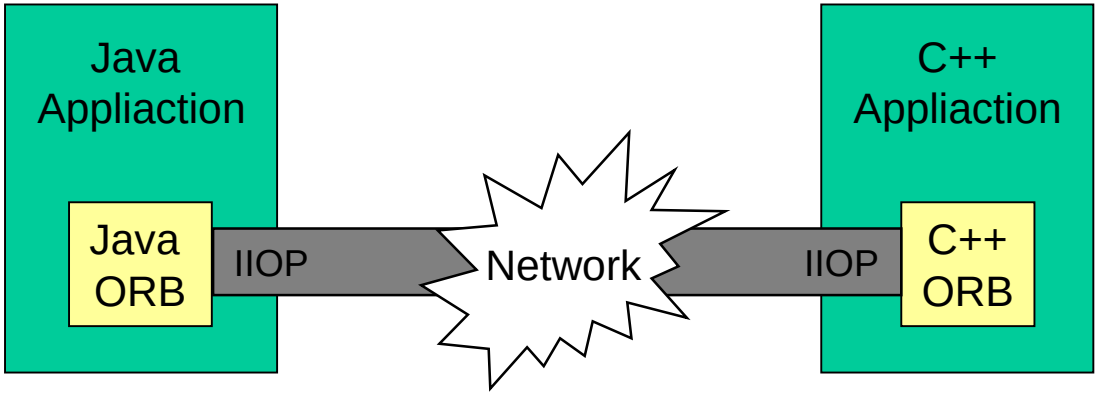
Marcin Jarząb
Paweł Słowikowski
Paweł Rzepa

Agenda

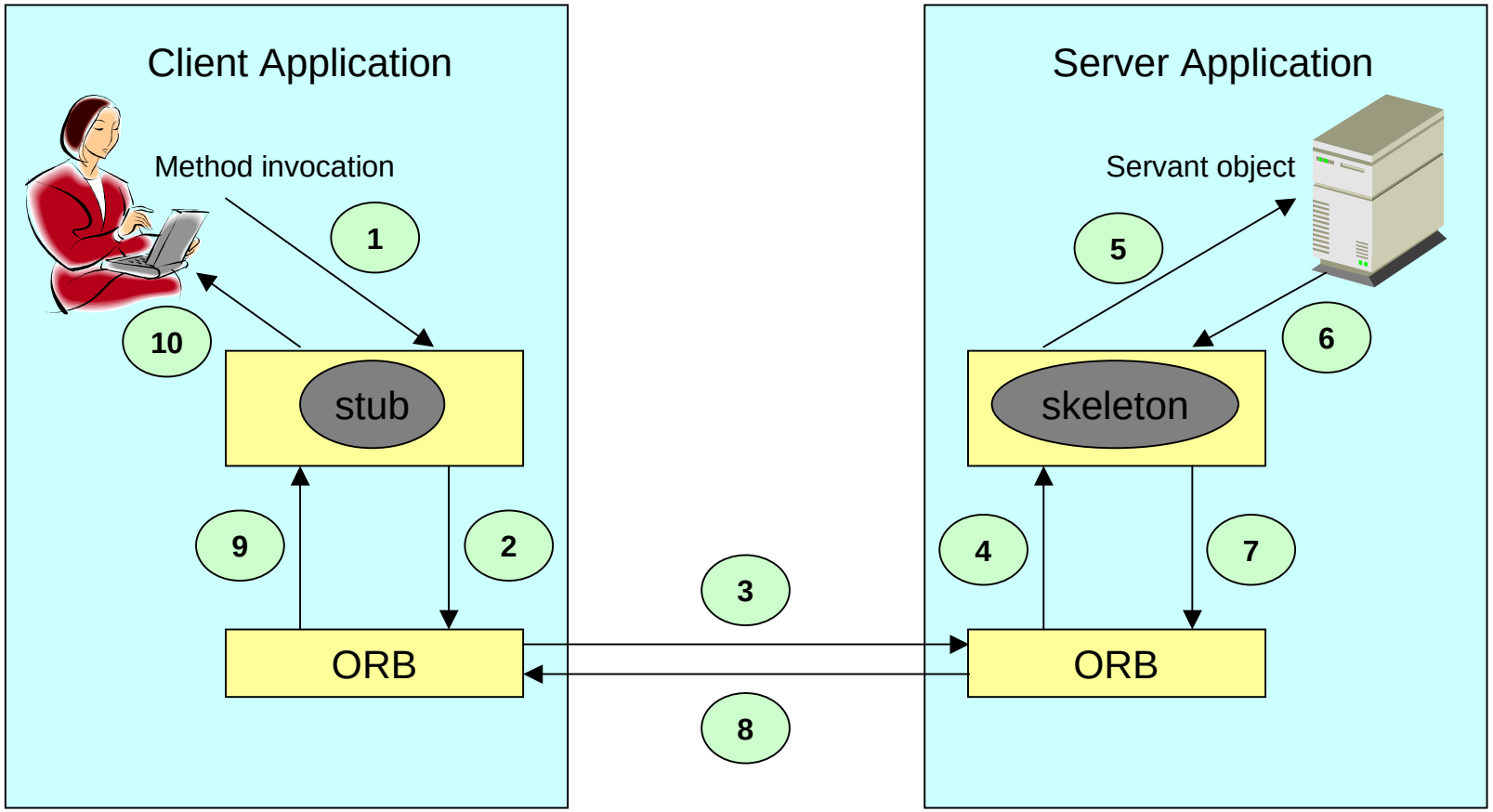
- Describe the CORBA technology
- Explain how to use the Java Interface Definition Language (IDL)
- Define components of a CORBA system
- Explain the function of the Portable Object Adapter (POA)
- Design and develop a sample CORBA application in Java

CORBA Architecture

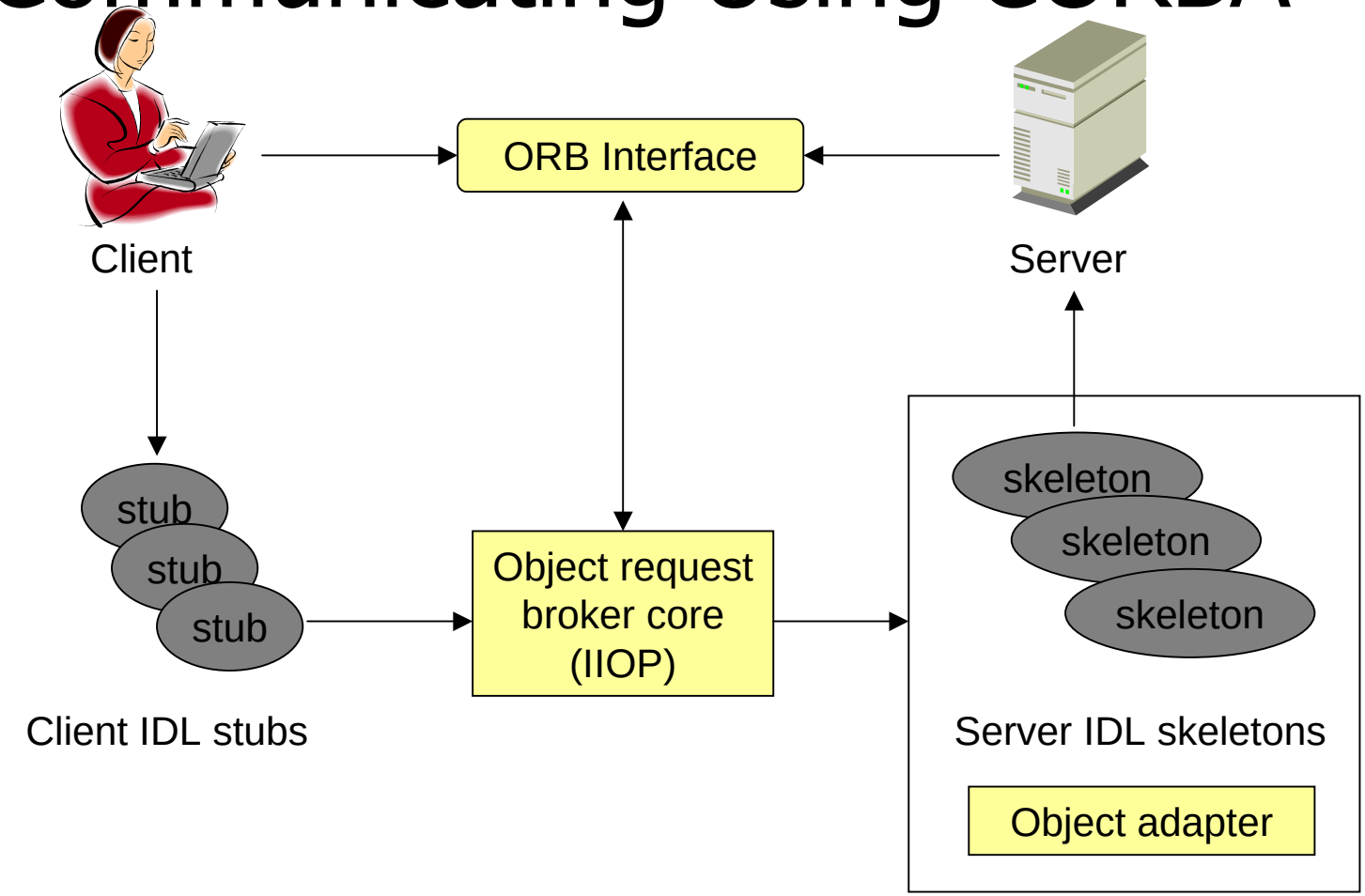
- CORBA is built around the concept of an Object Request Broker (ORB). The Internet Inter-ORB Protocol (IIOP) enables different ORBs to communicate.



Stub and Skeleton

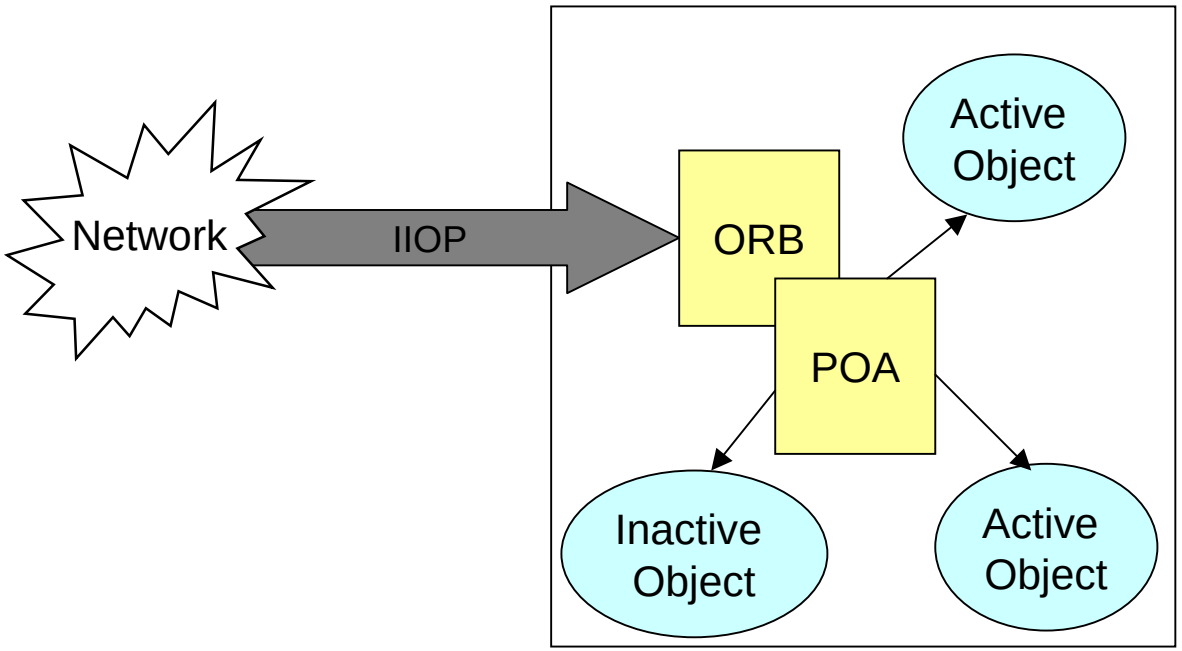


Distributed Applications Communicating Using CORBA



POA Redirecting Requests

- The POA receives incoming network requests and forwards those requests to the appropriate object.

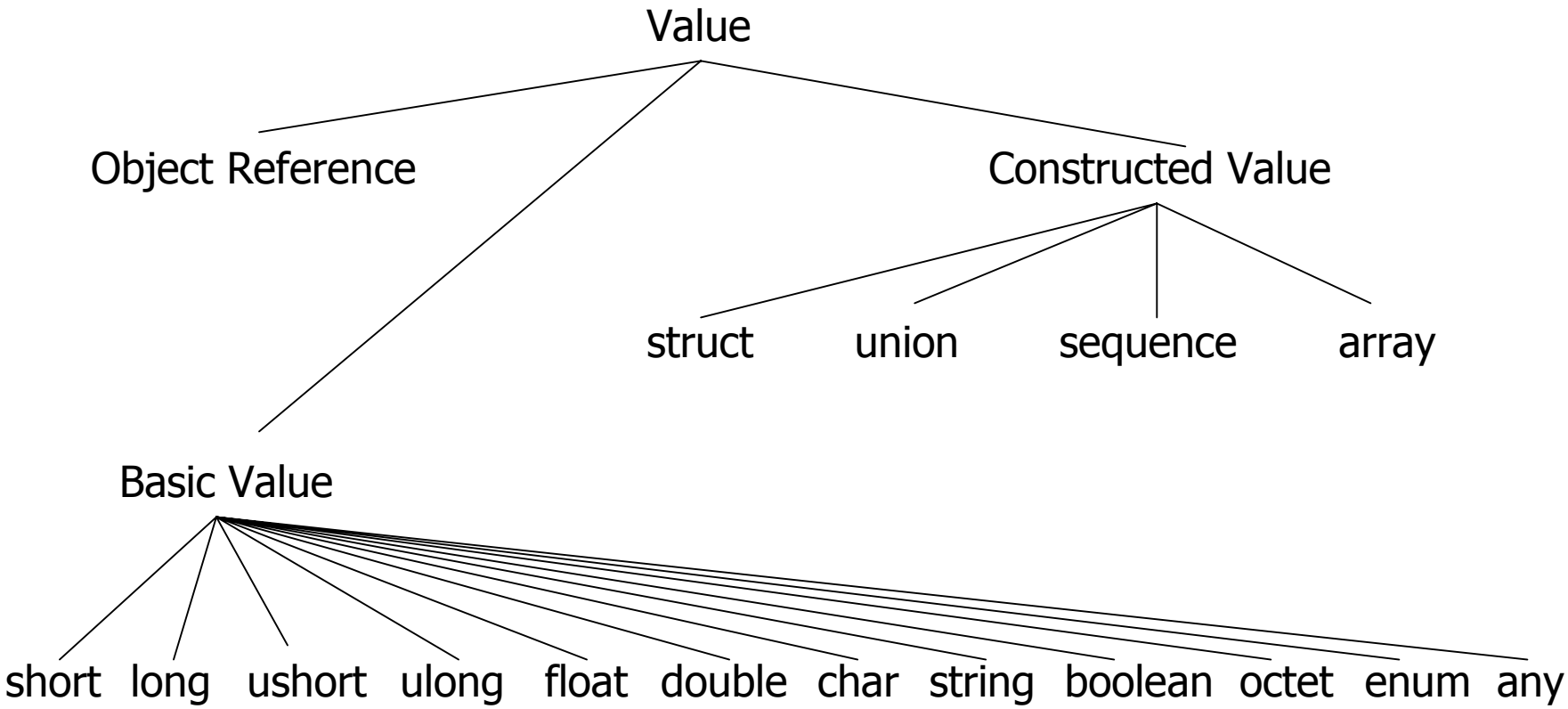


Interface Definition Language

- Defines syntax rules for creating CORBA object interfaces
- Is purely declarative (that is, no loops or flow control)

```
module dictionary{
    enum Language{ polish, english, suahili };
    struct Message{
        Language fromLang;
        Language toLang;
        string text;
    };
    interface Translator{
        string translate( in Message text);
        string translate2( in Language formLang, in Language toLang,
                           in string text );
    };
};
```

IDL – typy danych



IDL - przykład

```
module BankSimple {
  typedef float CashAmount;
  interface Account {
    readonly attribute string owner;
    exception InsufficientFunds { string reason; };
    oneway void deposit(in CashAmount amount);
    void withdraw(in CashAmount amount, out CashAmount
                  remained) raises(InsufficientFunds);
  };
  struct LimitedAccounts {
    unsigned long bankId;
    sequence<Account, 50> accounts;
  };
};
```

Translating IDL Into a Programming Language

- An IDL file is a text file.
- A special compiler is used for translation and generation of source code.
- Translation is typically performed according to standards defined by the OMG.
- Currently several languages have official IDL mappings including C++, Java, Smalltalk, and COBOL.
- Any programming language could potentially be mapped to an IDL and used in a CORBA application.

Choosing an ORB

- Each CORBA application must use an ORB implementation.
- Several vendors provide commercial ORB products with many features and supporting different languages.
- Some common ORB products are:
  - Orbix and ORBacus from IONA
  - Borland Enterprise Server, VisiBroker Edition
  - Java IDL ORB from Sun Microsystems
  - OpenORB (Open Source, Java)
  - JacORB (Open Source, GLGPL Java)
  - omniORB [Open Source, GPL/LGPL] (C++, Python)

Developing a CORBA Application in Java Technology

- Define the object interfaces using IDL.
- Compile the IDL files by using the `idlj` compiler to convert the IDL file into Java programming language code.
- Develop the remote object implementations (servants).
- Develop the server application code.
- Develop the client application code.
- Place all files in the appropriate directories and run the application.

ToUpper IDL File

The following is an example of an IDL file (`corbatest.idl`) for the ToUpper application:

```
module sr{
   interface TextFunctions{
     string toUpper( in string text);
   };
 };
```

ToUpper IDL File

The following is an example of an IDL file (`corbatest.idl`) for the ToUpper application:

```
module sr{
   interface TextFunctions{
     string toUpper( in string text);
   };
};
```

```
package sr;
```

```
public interface TextFunctions
```

```
String toUpper (String text);
```

ToUpper IDL File

```
cp ~rzepa/sr/corba.tar .
```

You can convert this code into Java technology source code using an IDL compiler called `idlj`:

```
idlj -fall corbatest.idl
```

```
idlj -td generated corbatest.idl
```

Compiling this file generates several Java technology source code files that you use to develop the CORBA application.

Usefull option:

```
idlj –pkgTranslate sr org.dsrg.labs.corba.test.generated
corbatest.idl
```

```
make idlj
```

What Is Generated by the `idlj` Compiler?

For each IDL file, for example `corbatest.idl`, the `idlj` compiler creates the following `.java` files:

File Name	Description
`TextFunctions.java`	Java technology interface used by both the client and the server
`TextFunctionsOperations.java`	Java technology interface listing the remote methods
`TextFunctionsPOA.java`	Superclass of the implementation class
`_TextFunctionsStub.java`	Client-side stub class
`TextFunctionsHelper.java`	Abstract class that performs conversion of remote object to correct type using the narrow method
`TextFunctionsHolder.java`	Final class that provides `out` and `inout` parameters to the Java programming language

Creating a Basic Java Technology/CORBA Application

After creating the interface and running the `idlj` compiler, create a servant implementation object.

```
package org.dsrg.labs.corba.test.server;
import org.omg.CORBA.ORB;
//...
import org.dsrg.labs.corba.test.generated.TextFunctionsPOA;
/**
 * Implementation of a POA compatibile servant.
 */
class TextFunctionsImpl extends TextFunctionsPOA
{
    public String toUpper(String text)
    {
        if ( text != null)
        {
          return text.toUpperCase();
        }
        return null;
    }
}
```

Implementing the Server Program

The CORBA server must create an instance of the implementation object and initialize the ORB and POA.

```
// create and initialize the ORB
ORB orb = ORB.init( args, null );
// get reference to rootpoa & activate the POAManager
POA rootpoa = POAHelper.narrow(orb.resolve_initial_references( "RootPOA" ));
rootpoa.the_POAManager().activate();
// create servant
TextFunctionsImpl functionsImpl = new TextFunctionsImpl();
// register it with the POA and get the object reference
org.omg.CORBA.Object ref = rootpoa.servant_to_reference( functionsImpl );

//export object reference to a file
PrintWriter pw = new PrintWriter(new FileWriter(„Test.IOR"));
pw.print(orb.object_to_string(ref));
pw.close();

// wait for invocations from clients
orb.run();
```

Implementing the Server Program

The CORBA server must create an instance of the implementation object and initialize the ORB and POA.

```
// create and initialize the ORB
ORB orb = ORB.init( args, null );
// get reference to rootpoa & activate the POAManager
POA rootpoa = POAHelper.narrow(orb.resolve_initial_references( "RootPOA" ));
rootpoa.the_POAManager().activate();
// create servant
TextFunctionsImpl functionsImpl = new TextFunctionsImpl();
// register it with the POA and get the object reference
org.omg.CORBA.Object ref = rootpoa.servant_to_reference( functionsImpl );

//export object reference to a file
PrintWriter pw = new PrintWriter(new FileWriter(„Test.IOR"));
pw.print(orb.object_to_string(ref));
pw.close();

// wait for invocations from clients
orb.run();
```

Implementing the Server Program

The CORBA server must create an instance of the implementation object and initialize the ORB and POA.

```
// create and initialize the ORB
ORB orb = ORB.init( args, null );
// get reference to rootpoa & activate the POAManager
POA rootpoa = POAHelper.narrow(orb.resolve_initial_references( "RootPOA" ));
rootpoa.the_POAManager().activate();
// create servant
TextFunctionsImpl functionsImpl = new TextFunctionsImpl();
// register it with the POA and get the object reference
org.omg.CORBA.Object ref = rootpoa.servant_to_reference( functionsImpl );

//export object reference to a file
PrintWriter pw = new PrintWriter(new FileWriter(„Test.IOR"));
pw.print(orb.object_to_string(ref));
pw.close();

// wait for invocations from clients
orb.run();
```

Implementing the Server Program

The CORBA server must create an instance of the implementation object and initialize the ORB and POA.

```
// create and initialize the ORB
ORB orb = ORB.init( args, null );
// get reference to rootpoa & activate the POAManager
POA rootpoa = POAHelper.narrow(orb.resolve_initial_references( "RootPOA" ));
rootpoa.the_POAManager().activate();
// create servant
TextFunctionsImpl functionsImpl = new TextFunctionsImpl();
// register it with the POA and get the object reference
org.omg.CORBA.Object ref = rootpoa.servant_to_reference( functionsImpl );

//export object reference to a file
PrintWriter pw = new PrintWriter(new FileWriter(„Test.IOR"));
pw.print(orb.object_to_string(ref));
pw.close();

// wait for invocations from clients
orb.run();
```

Implementing the Server Program

The CORBA server must create an instance of the implementation object and initialize the ORB and POA.

```
// create and initialize the ORB
ORB orb = ORB.init( args, null );
// get reference to rootpoa & activate the POAManager
POA rootpoa = POAHelper.narrow(orb.resolve_initial_references( "RootPOA" ));
rootpoa.the_POAManager().activate();
// create servant
TextFunctionsImpl functionsImpl = new TextFunctionsImpl();
// register it with the POA and get the object reference
org.omg.CORBA.Object ref = rootpoa.servant_to_reference( functionsImpl );

//export object reference to a file
PrintWriter pw = new PrintWriter(new FileWriter(„Test.IOR"));
pw.print(orb.object_to_string(ref));
pw.close();

// wait for invocations from clients
orb.run();
```

Implementing the Client Program

The client application must initialize the ORB, then retrieve the object reference and use it.

```
// create and initialize the ORB
ORB orb = ORB.init( argv, null );
```

```
// get the Object Reference from the file
String ref = new BufferedReader(new FileReader(„Test.IOR")).readLine();
TextFunctions functionsImpl = TextFunctionsHelper.narrow(orb.string_to_object(ref));

// invoke toUpper() on the remote object
String testString = "corba-test-string";
testString = functionsImpl.toUpper( testString );
```

Implementing the Client Program

The client application must initialize the ORB, then retrieve the object reference and use it.

```
// create and initialize the ORB
ORB orb = ORB.init( argv, null );

// get the Object Reference from the file
String ref = new BufferedReader(new FileReader(„Test.IOR")).readLine();
TextFunctions functionsImpl = TextFunctionsHelper.narrow(orb.string_to_object(ref));

// invoke toUpper() on the remote object
String testString = "corba-test-string";
testString = functionsImpl.toUpper( testString );
```

Implementing the Client Program

The client application must initialize the ORB, then retrieve the object reference and use it.

```
// create and initialize the ORB
ORB orb = ORB.init( argv, null );

// get the Object Reference from the file
String ref = new BufferedReader(new FileReader(„Test.IOR")).readLine();
TextFunctions functionsImpl = TextFunctionsHelper.narrow(orb.string_to_object(ref));

// invoke toUpper() on the remote object
String testString = "corba-test-string";
testString = functionsImpl.toUpper( testString );
```

Compiling and launching

- Compile:

```
make javac
```

- Launch:
- SERVER

```
java –classpath classes \
      org.dsrg.labs.corba.test.server.TextFunctionsServer
```

or

```
make run-server
```

- CLIENT

```
java –classpath classes \
      org.dsrg.labs.corba.test.server.TextFunctionsServer
```

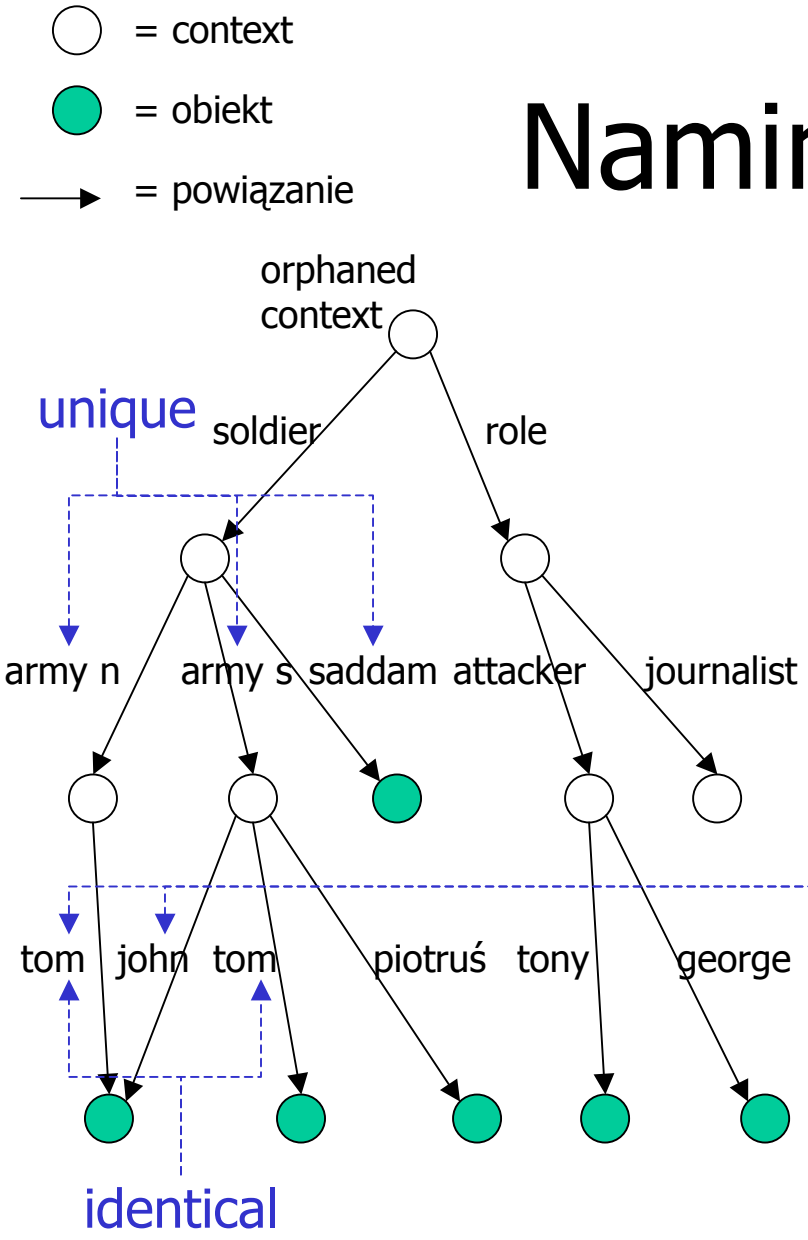
or

```
make run-client
```

Naming Service

Naming Service

- Udostępnia mapowanie z nazw do referencji obiektów
  - Przekazując nazwę otrzymujemy referencję obiektu – nie działa w drugą stronę
- Zalety
  - Klient nie musi używać referencji obiektu – korzysta ze zrozumiałej nazwy
  - Zmieniając referencję obiektu wskazywaną przez określoną nazwę możemy zmienić implementację usługi bez konieczności zmiany kodu źródłowego
  - Rozwiązuje problem wyszukiwania referencji obiektów aplikacji – eliminuje potrzebę skorzystania z pliku, http itd., itp.



Naming graph

1. Nazwy wewnątrz jednego kontekstu są unikalne
2. Ta sama nazwa może się pojawić wielokrotnie ale w różnych kontekstach
3. Obiekt może mieć wiele różnych nazw
4. Unikalnym identyfikatorem obiektu jest ścieżka od kontekstu początkowego do referencji obiektu

Struktura IDL

```
//File: CosNaming.idl
module CosNaming {
  //...
  interface
   NamingContext {
    //...
  };
  //...
};
```

Nazwy obiektów

```
module CosNaming {//...
    typedef string Istring;
    struct NameComponent {
        Istring id;
        Istring kind;
        };
    typedef sequence<NameComponent> Name;
};
```

Mapping do Java?

- **`id, kind`** – dowolnej długości ISO Latin-1 bez ASCII NUL
- **`NameComponent`** – pojedynczy krok w ścieżce; składa się z dwóch **`string`**ów
- **`kind`** – znaczenie podobne do rozszerzenia pliku; można zostawiać puste. *Naming service* w żaden sposób nie interpretuje tych wartości.
- Dwie **`Name`** są równe tylko jeśli wszystkie **`NameComponent`** są identyczne
- **`NameComponent`** są identyczne jeśli posiadają identyczne **`id`** i **`kind`**

Operacje w **NamingContext**

- Tworzenie naming graph
  - Tworzenie, kasowanie kontekstu
  - Dołączanie, odłączanie obiektu lub kontekstu od istniejącego kontekstu
- Wyszukiwanie obiektu zarejestrowanego w NamingService

Interface **NamingContext**

```
interface NamingContext {//...
  NamingContext new_context();
  NamingContext bind_new_context(in Name n) raises(
                  NotFound, CannotProceed,
                  InvalidName, AlreadyBound );
  void destroy () raises (NotEmpty);
  void bind(in Name n, in Object obj) raises (
         NotFound, CannotProceed,
         InvalidName, AlreadyBound);
  void bind_context(in Name n, in NamingContext nc)
         raises(NotFound, CannotProceed,
                InvalidName, AlreadyBound);
};
```

Interface **NamingContext**

```
interface NamingContext {//...
  void rebind(in Name n, in Object obj) raises (
         NotFound, CannotProceed, InvalidName);
  void rebind_context(in Name n, in NamingContext nc)
         raises(NotFound, CannotProceed, InvalidName);
 void unbind(in Name n) raises (
         NotFound, CannotProceed, InvalidName);

// Rozwiązywanie nazw
 Object resolve(in Name n) raises (
         NotFound, CannotProceed, InvalidName);
};
```

Stringified Names

- Jak przekazać nazwę obiektu do klienta?
  - Id1(kind1)/id2(kind2)/id3(kind3)
  - Id1.kind1/id2.kind2/id3.kind3
- Od wersji 1.2 zestandaryzowane
  - Dowolnej długości ISO Latin-1 bez ASCII NUL
  - Znaki '/', '.', '\' mają znaczenie specjalne
    - / - rozdziela **`NameComponents`**
    - **`.`** – rozdziela **`id`** i **`kind`**
    - \ - znak wyłączający znaczenie specjalne znaków '/' i '.'
- Przykład
  - **`a\.id\/a\.kind.a\.id/a.id/a.kind`**

Interoperable Naming Service

```
interface NamingContextExt {
  //...
  typedef string StringName;

  Object resolve_str(in Name n) raises (
         NotFound, CannotProceed, InvalidName);

  Name to_name(in StringName sn) raises (
         InvalidName);

};
```

Uzyskanie Initial Naming Context


```
interface CORBA {//PIDL
  typedef string ObjectId;
  exception InvalidName {};
  Object resolve_initial_references (in ObjectId id)
         raises (InvalidName);
};
```

ObjectId dla *Naming Service* to **'NameService'**

Implementing the Server Program - NS

The CORBA server must create an instance of the implementation object and initialize the ORB and POA.

```
// create and initialize the ORB
ORB orb = ORB.init( args, null );
// get reference to rootpoa & activate the POAManager
POA rootpoa = POAHelper.narrow(orb.resolve_initial_references( "RootPOA" ));
rootpoa.the_POAManager().activate();
// create servant and register it with the ORB
TextFunctionsImpl functionsImpl = new TextFunctionsImpl();
// get object reference from the servant
org.omg.CORBA.Object ref = rootpoa.servant_to_reference( functionsImpl );
TextFunctions href = TextFunctionsHelper.narrow( ref );
// get the root naming context
org.omg.CORBA.Object objRef = orb.resolve_initial_references( "NameService" );
// Use NamingContextExt which is part of the Interoperable
// Naming Service (INS) specification.
NamingContextExt ncRef = NamingContextExtHelper.narrow( objRef );
// bind the Object Reference in Naming
NameComponent path[] = ncRef.to_name(„TextFunctionsServer” );
ncRef.rebind( path, href );
// wait for invocations from clients
orb.run();
```

Implementing the Server Program - NS

The CORBA server must create an instance of the implementation object and initialize the ORB and POA.

```
// create and initialize the ORB
ORB orb = ORB.init( args, null );
// get reference to rootpoa & activate the POAManager
POA rootpoa = POAHelper.narrow(orb.resolve_initial_references( "RootPOA" ));
rootpoa.the_POAManager().activate();
// create servant and register it with the ORB
TextFunctionsImpl functionsImpl = new TextFunctionsImpl();
// get object reference from the servant
org.omg.CORBA.Object ref = rootpoa.servant_to_reference( functionsImpl );
TextFunctions href = TextFunctionsHelper.narrow( ref );
// get the root naming context
org.omg.CORBA.Object objRef = orb.resolve_initial_references( "NameService" );
// Use NamingContextExt which is part of the Interoperable
// Naming Service (INS) specification.
NamingContextExt ncRef = NamingContextExtHelper.narrow( objRef );
// bind the Object Reference in Naming
NameComponent path[] = ncRef.to_name(„TextFunctionsServer” );
ncRef.rebind( path, href );
// wait for invocations from clients
orb.run();
```

Implementing the Client Program - NS

The client application must initialize the ORB and then retrieve the object using NameService.

```
// 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" );
// Use NamingContextExt instead of NamingContext.
// This is part of the Interoperable naming Service.
NamingContextExt ncRef = NamingContextExtHelper.narrow( objRef );

// resolve the Object Reference in Naming
TextFunctions functionsImpl =
    TextFunctionsHelper.narrow( ncRef.resolve_str( „TextFunctionsServer” ) );

// invoke toUpper() on the remote object
String testString = "corba-test-string";
testString = functionsImpl.toUpper( testString );
```

Implementing the Client Program - NS

The client application must initialize the ORB and then retrieve the object using NameService.

```
// 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" );
// Use NamingContextExt instead of NamingContext.
// This is part of the Interoperable naming Service.
NamingContextExt ncRef = NamingContextExtHelper.narrow( objRef );

// resolve the Object Reference in Naming
TextFunctions functionsImpl =
    TextFunctionsHelper.narrow( ncRef.resolve_str( „TextFunctionsServer” ) );

// invoke toUpper() on the remote object
String testString = "corba-test-string";
testString = functionsImpl.toUpper( testString );
```

Implementing the Client Program - NS

The client application must initialize the ORB and then retrieve the object using NameService.

```
// 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" );
// Use NamingContextExt instead of NamingContext.
// This is part of the Interoperable naming Service.
NamingContextExt ncRef = NamingContextExtHelper.narrow( objRef );

// resolve the Object Reference in Naming
TextFunctions functionsImpl =
    TextFunctionsHelper.narrow( ncRef.resolve_str( „TextFunctionsServer” ) );

// invoke toUpper() on the remote object
String testString = "corba-test-string";
testString = functionsImpl.toUpper( testString );
```

Starting the ORB/CORBA Services

You start the Java 2 SDK naming service with the

```
orbd -ORBInitialPort serverport
```

Example

```
orbd -ORBInitialPort 3500
```

or

```
make run-orbd
```
(Zmienić wcześniej nr portu NS_PORT)

Starting the server and client

Server

```
java –classpath classes \
org.dsrg.labs.corba.test.server.TextFunctionsServer \
–ORBInitialPort 3500 –ORBInitialHost localhost
```

albo

`make run-ns-server` (Zmienić wcześniej nr portu NS_PORT)

Client

```
java –classpath classes \
org.dsrg.labs.corba.test.client.TestClient \
–ORBInitialPort 3500 –ORBInitialHost localhost
```

albo

`make run-ns-client` (Zmienić wcześniej nr portu NS_PORT)

Extending server functionality

The following is an example of an IDL file (`corbatest.idl`) for the ToUpper application:

```
module sr{

  enum tmode {TOLOWER, TOUPPER};
  typedef string Message;
  exception NotLetters{};

  interface TextFunctions {
    attribute tmode mode;

    //nrOfChanged: number of chars changed
      Message process(in Message text, out short nrOfChanged)
       raises(NotLetters);
  };
};
```

Zadanie domowe

```
module logs {
   enum LogPriority {LOG_DEBUG, LOG_WARNING,  LOG_PANIC};
   struct LogRecord {
     LogPriority priority;
     long time;
     string message;
   };
  interface LoggerListener {
   void processLogRecord( in LogRecord lr );
  };
  interface Logger {
   exception Disconnected {};
   exception AlreadyAdded {};
   exception NotPresent {};

   void log( in LogRecord rec ) raises (Disconnected);
   void addLoggerListener( in LoggerListener ll, in LogPriority priority) raises (AlreadyAdded);
   void alterLoggerListener(in LoggerListener ll, in LogPriority priority) raises (NotPresent);
   void removeLoggerListener(in LoggerListener ll) raises (NotPresent);
  };
};
```