

CORBA-III

Prof. David Bakken

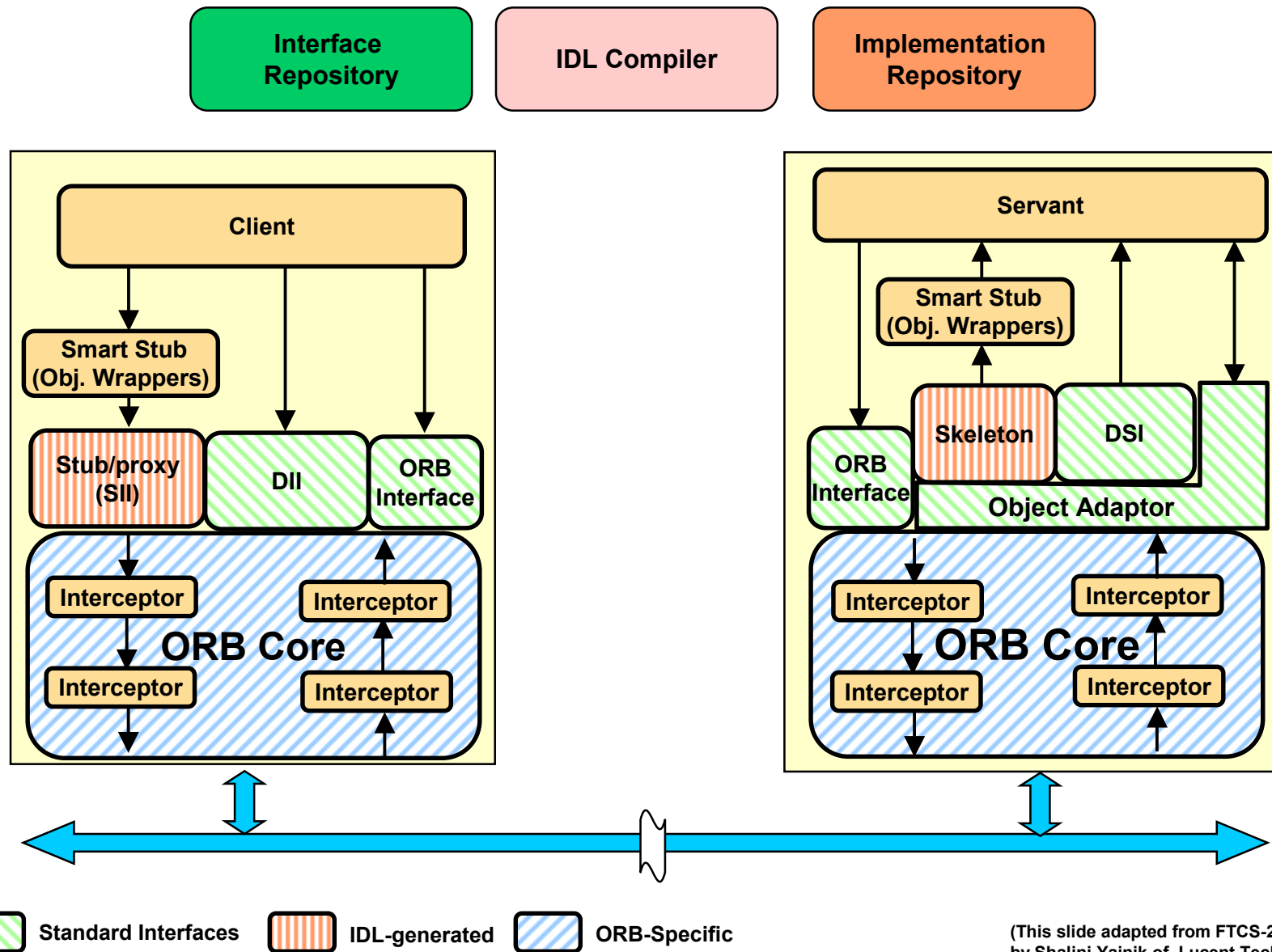
Cpt. S 464/564 Lecture

Oct 11, 2000

Outline

- **Interceptors**
- Smart Stubs (“Object Wrappers” in VisiBroker terminology)
- CORBA Interoperability and GIOP

CORBA Components and System Hooks



(This slide adapted from FTCS-29 Tutorial by Shalini Yajnik of Lucent Technologies)

Interceptors

- Used to plug in additional behavior into the ORB
 - Security
 - Transactions
 - Logging/tracing
- Interceptors give you a callback to your object, which implements a predefined interface
- Two kinds of interceptors
 - Client interceptors: on the client side
 - Server interceptors: on the server side

Kinds of Interceptors

- Q: what might each be used for?
- Client interceptors
 - BindInterceptor: called before and after a bind call goes over the network
 - ClientRequestInterceptor: called before a request goes on the net, and after a reply comes back
- Server interceptors
 - POALifeCycleInterceptor: called every time a POA is created
 - ActiveObjectLifeCycleInterceptor: called whenever an object is added to an Active Object Map
 - ServerRequestInterceptor: called when a request arrives at a server and the reply is being processed
 - IORCreationInterceptor: called whenever a POA creates an object reference
- ServiceResolverInterceptor: used to install a user service that you can dynamically load

Example #1: BindInterceptor Interface

```
public interface BindInterceptor {  
    public IORValue bind(IORValue ior, org.omg.CORBA.object target,  
                        boolean rebind, Closure closure);  
    public IORValue bind_failed(IORValue ior,  
                                org.omg.CORBA.object target, Closure closure);  
    public void bind_succeeded(IORValue ior,  
                              org.omg.CORBA.object target, int Index,  
                              InterceptorManagerControl control,  
                              Closure closure);  
    public void exception_occurred(IORValue ior,  
                                   org.omg.CORBA.object target,  
                                   org.omg.CORBA.Environment env,  
                                   Closure closure);  
}
```

Example #2: ClientRequestInterceptor

```
public interface ClientRequestInterceptor {  
    public void preinvoke_premarshal(org.omg.CORBA.Object target,  
        String operation, ServiceContextListHolder service_context_holder,  
        Closure closure);  
    public void preinvoke_postmarshal(org.omg.CORBA.Object target,  
        OutputStream payload, Closure closure);  
    public void postinvoke(org.omg.CORBA.Object target,  
        ServiceContext[] service_contexts,  
        org.omg.CORBA.Environment env, Closure closure);  
    public void exception_occurred(org.omg.CORBA.Object target,  
        org.omg.CORBA.Environment env, Closure closure);  
}
```

Final Notes on Interceptors

- We will use interceptors in Project #5....
- But lecture CORBA-IV will have to give more details on how to use... they involve implementing and registering factories and then interceptors
- For more info, see chapter in PROG and REF

Outline

- Interceptors
- **Smart Stubs (“Object Wrappers” in VisiBroker terminology)**
- CORBA Interoperability and GIOP

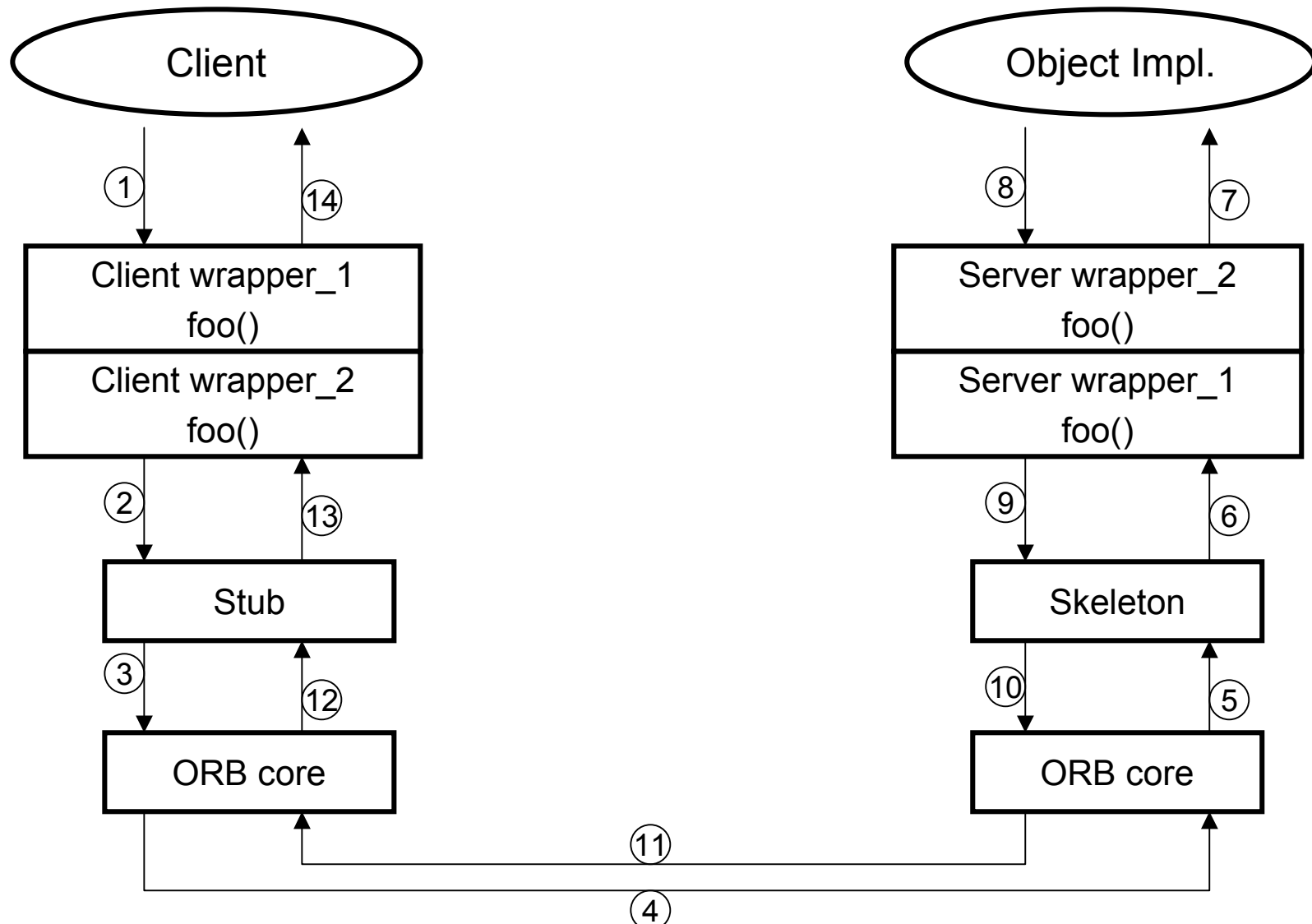
Object Wrappers in VisiBroker

- Object wrappers are a layer above the stub/proxy
- Can be used to implement
 - Logging/tracing
 - Caching (return a value quickly, without going over the network)
 - Timings
- Can be installed on
 - Client side
 - Server side
 - Both
- Can have multiple wrappers installed
- Note: if you stringify a reference to an object for which object wrappers are installed, these will not automatically be installed in another process...
- For more info, see chapter in PROG and REF

Kinds of Object Wrappers

- Typed object wrappers
 - Receives parameters that are passed to stub
 - Can return control to caller without invoking the next wrapper, or the stub, or the remote object
 - Will not be invoked for all operation requests for all objects
- Un-typed object wrappers
 - Does not receive parameters that are passed to stub
 - Can not return control to caller without invoking the next wrapper, or the stub, or the remote object
 - Will be invoked for all operation requests for all objects
- Typed object wrappers are good for
 - Caching
 - Object-specific adaptations
- Un-typed object wrappers are good for
 - Logging/tracing
 - Timing

Typed Wrappers and Chaining, calling foo()



Steps in Creating Typed Object Wrappers

1. Decide which interfaces you want to create a typed object wrapper for
2. Generate wrapper code by using the `-obj_wrapper` flag to the IDL compiler
3. Derive your typed object wrapper class from the class named `<interface_name>ObjectWrapper`, which was generated by the IDL compiler, then implement its methods
4. Modify your application to register the typed object wrapper

Sample Typed Object Wrapper

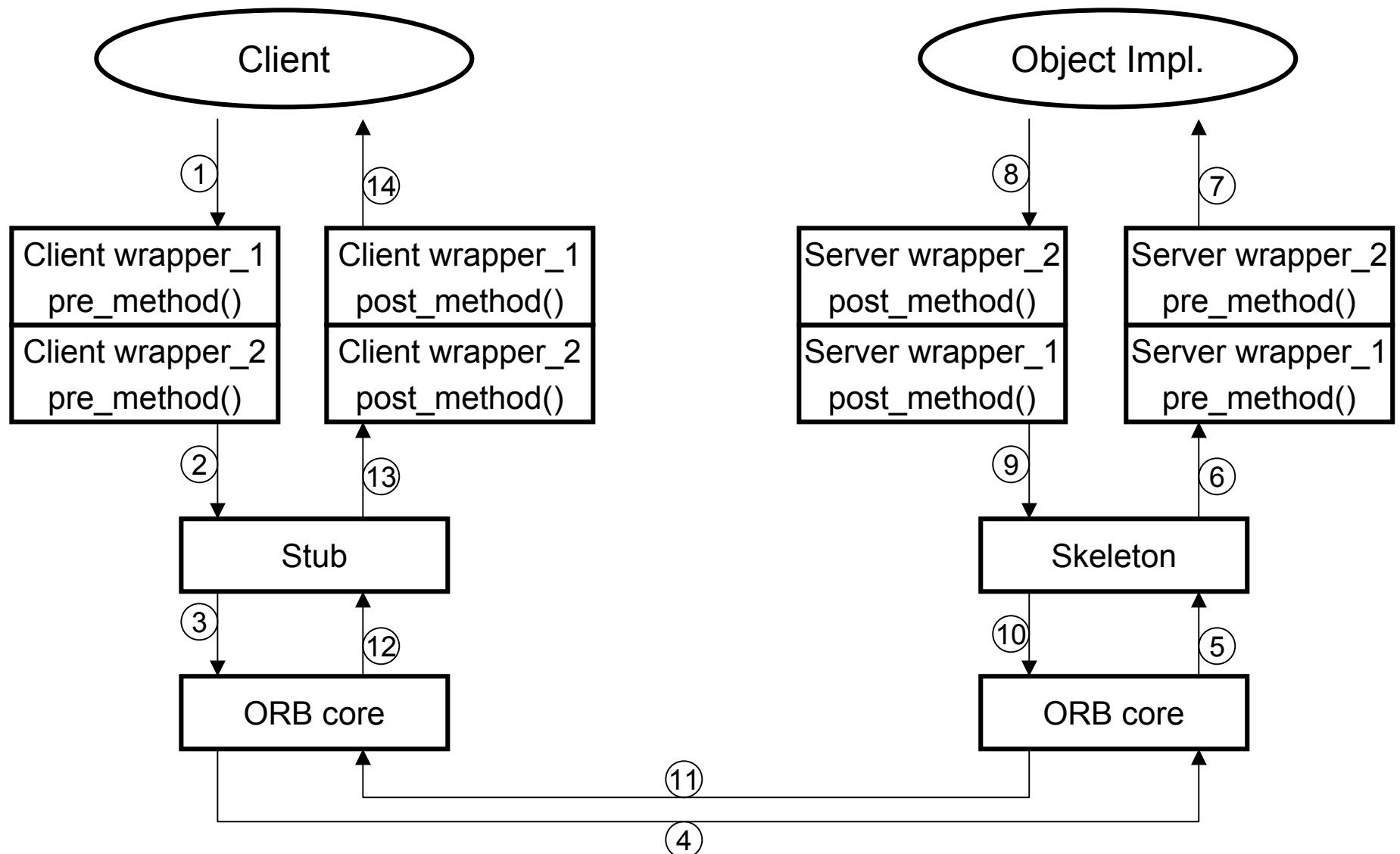
```
package BankWrappers;

public class CachingAccountObjectWrapper extends Bank.AccountObjectWrapper {
    private boolean _initialized = false;
    private float _balance;
    public float balance() {
        System.out.println("+ CachingAccountObjectWrapper: Before calling balance: ");
        try {
            if ( !_initialized ) {
                _balance = super.balance();
                _initialized = true;
            } else {
                System.out.println("+ CachingAccountObjectWrapper: Returning Cached value");
            }
            return _balance;
        }
        finally {
            System.out.println("+ CachingAccountObjectWrapper: After calling balance:");
        }
    }
}
```

Registering a Typed Object Wrapper

```
public class TypedClient {  
  
    public static void main(String[] args) throws Exception {  
        // Initialize the ORB.  
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);  
        doMain (orb, args);  
    }  
  
    public static void doMain(org.omg.CORBA.ORB orb, String[] args) {  
        // Add a typed object wrapper for Account objects  
        Bank.AccountHelper.addClientObjectWrapperClass(orb,  
            BankWrappers.CachingAccountObjectWrapper.class);  
        // Locate an account manager.  
        Bank.AccountManager manager =  
            Bank.AccountManagerHelper.bind(orb, "BankManager");  
  
        // use args[0] as the account name, or a default.  
        ....  
    }  
}
```

Un-typed Wrappers and Chaining



Steps in Using Un-typed Object Wrappers

(For Java; C++ similar)

1. Figure out which interfaces you want to use untyped wrappers with
2. Generate the code using the `-obj_wrapper` flag to the IDL compiler
3. Create an implementation for your un-typed object wrapper factory, which derives from `UntypedObjectWrapperFactory` class (in Java)
 - Has method `create`, called whenever
 - Client binds to an object
 - (Client?) invokes a method on an object
4. Create an implementation for your un-typed object wrapper, which derives from `UntypedObjectWrapperClass`
5. Modify client or server application to access the appropriate type of `ChainUntypedObjectWrapperFactory`.
6. Modify your application to create your un-typed object wrapper factory
7. Use the `ChainUntypedObjectWrapperFactory`'s `add` method to add your factory to the chain

Creating an Un-typed object wrapper factory

```
package UtilityObjectWrappers;

import com.inprise.vbroker.interceptor.*;

public class TimingUntypedObjectWrapperFactory implements
    UntypedObjectWrapperFactory {
    public UntypedObjectWrapper create(org.omg.CORBA.Object target,
        Location loc)
    {
        return new TimingUntypedObjectWrapper();
    }
}
```

Implementing an un-typed object wrapper

```
package UtilityObjectWrappers;
import com.inprise.vbroker.interceptor.*;

public class TimingUntypedObjectWrapper implements UntypedObjectWrapper {

    private long time;

    public void pre_method(String operation, org.omg.CORBA.Object target, Closure closure) {
        System.out.println("Timing: "
            + ((com.inprise.vbroker.CORBA.Object) target)._object_name() + "->" + operation + "()");
        time = System.currentTimeMillis();
    }

    public void post_method(String operation, org.omg.CORBA.Object target,
        org.omg.CORBA.Environment env, Closure closure) {
        long diff = System.currentTimeMillis() - time;
        System.out.println("Timing: Time for call \t"
            + ((com.inprise.vbroker.CORBA.Object) target)._object_name()
            + "->" + operation + "() = " + diff + " ms.");
    }

}
```

Outline

- Interceptors
- Smart Stubs (“Object Wrappers” in VisiBroker terminology)
- **CORBA Interoperability and GIOP**

General Interoperability Protocol (GIOP)

Abstract protocol to allow for interoperability between different vendors' ORBs. To do this, it defines:

- **Interoperable Object Reference (IOR) format (see CORBA-II).**
 - **Inter-ORB message formats**
1. 1. **Request:** from client, sending an invocation. Contains
 - `GIOP.MessageHeader`
 - `GIOP.RequestHeader`
 - `GIOP.RequestBody`
 - 2. **Reply:** from server, responding to Request.
 - 3. **CancelRequest:** from client, telling it to ignore a Request already sent.
 - 4. **LocateRequest:** from client, to find out if a server can service a particular request, or if it has a forwarding IOR to the actual server implementation.
 - 5. **LocateReply:** from server, in response to `LocateRequest`.
 - 6. **CloseConnection:** from server, indicating it is closing the connection.
 - 7. **MessageError:** by both...
 - **Wire protocol (data transfer syntax): Common Data Representation (CDR).**
A coding for all IDL types, structured types, exceptions, object references.
Covers coding into an octet stream, alignment boundaries, how to indicate byte ordering used.

Example of GIOP Format

```
module GIOP {  
    enum MessageType {Request, Reply, CancelRequest, LocateRequest,  
        LocateReply, CloseConnection, MessageError};  
    struct MessageHeader {  
        char magic[4];  
        Version GIOP_version;  
        octet byte_order;  
        octet message_type;  
        unsigned long message_size;  
    };  
    struct RequestHeader {  
        IIOp::ServiceContextList service_context;  
        unsigned long request_id;  
        boolean response_requested;  
        sequence<octet> object_key;  
        string operation;  
        Principal requesting_principal;  
    }  
    ...  
}
```

GIOP Transport Layer Assumptions

- Connection-oriented
 - I.e., transport management deals with opening and closing and using connections
- Connections are like
 - Two roles
 - Clients: open connections to servers
 - Servers: listen for connections
 - Clients and servers may only send a subset of the message types
 - Can be closed in an orderly fashion, or abortive close (both clearly defined)

IIOP

- IIOP is simply GIOP (an abstract protocol, remember) implemented over TCP/IP
- Must be implemented by every ORB
 - Gives a universal way for ORBs to communication
 - A given ORB may implement different transports underneath GIOP, also
- CommunicationID = {IP address, port}