



alessandro ricci

aricci@deis.unibo.it

DEIS - Università di Bologna

Sede di Cesena

2002/01

Middleware/Infrastructures for the Interaction-Centric era

seminar overview

- Interaction: paradigm shift
 - some perspectives on the impact
- Client/server middleware (overview and comparisons)
 - CORBA
 - HTTP/CGI
 - HTTP/Servlets
 - RMI
- Service Oriented Architectures/Infrastructures
 - JINI
 - JavaSpaces Service
 - WebServices
 - Peer-to-Peer Infrastructure
 - JXTA

seminar overview

- main sources of seminar pictures / slides:
 - for Client/Server middleware:
 - “*Client/Server Programming with Java and CORBA*”
Robert Orfali and Dan Harkey – Addison Wesley (*)
 - online documentation
 - <http://www.java.sun.com>
 - www.jini.org
 - ...
 - articles:
 - <http://www.onjava.com>
 - <http://www.javaworld.com>
 - <http://www.computer.org>
 - <http://www.acm.org>

paradigm shift

Computation (...-1980)

- algorithms
- mdTs, lambda calculus
- functions, procedures
- focus on *the computer*

paradigm shift

- pervasiveness / interdisciplinarity (OSs, Programming Languages, AI, HCI, ...)
- new metaphors, concepts
- new mathematical models
- middleware explosion

Interaction (1980s-1990s)

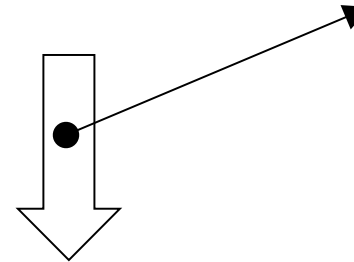
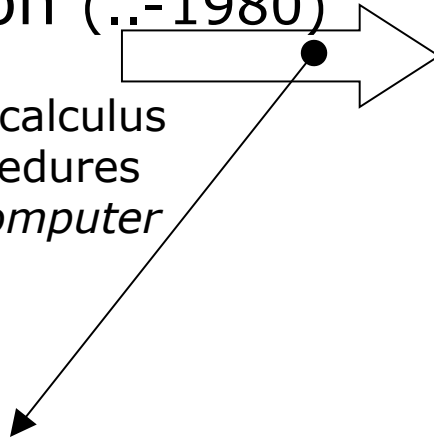
- communication
- mobility
- state
- space, time
- environment
- focus on the network

Engineering the interactions

- methodologies
- *infrastructures*

Coordination (1990s-...)

- systems
- services
- cooperation
- organisation
- Information Society



paradigm shift impact

- some perspectives related to the infrastructure issues:
 - Client/Server architecture evolution
 - Architecture evolution: from Client/Server to Service Oriented Architecture
 - Models of Communication evolution
 - Software Engineering/Organisation evolution

evolution: client/server

- from DBase Server (SQL)
 - 2-tier
 - User interface on the client
 - DataBase on the server
- to Application Servers
 - 3-Tier: middle-tier providing services
 - concurrency and transactions management (OTM)
 - n-Tier
 - each level → different level of services
 - scalability
- to Extensible Application Servers
 - frameworks on the server side
 - EJB, CORBA Component Model (CCM)

evolution: architectures

- from Client-Server
 - two coupled roles: client - server
 - Socket, RPC, RMI, HTTP/CGI, Servlet, RMI, CORBA
- to Service Oriented Architectures (SOA)
 - 3 roles: client, service provider, service registrar
 - space-uncoupling
 - asking for a service, not a specific server
 - Toward pervasive / mobile computing
 - JINI, UPnP, JES
 - Web Services (SOAP, WSDL,...)
 - PeerToPeer (JXTA,...)

evolution: Models of communication

- from point-to-point communication
 - time and space coupling
 - Socket, RPC, Object-based (RMI, CORBA), Web-based
- to
 - time uncoupled interaction
 - JMS (Java Message Service)
 - event-oriented services
 - CORBA event services
 - space uncoupled interaction
 - Service Oriented Architectures
 - WebServices, P2P
- to time/space uncoupled interaction
 - Sun JavaSpaces
 - IBM TSpaces

evolution: Software organisation (1)

- from procedures
 - RPC
- to objects
 - RMI, CORBA
- to components
 - *system perspective* to objects
 - Component Container as first class role
 - enabling and managing interaction among components
 - From aggregation/composition to ensemble/system
 - mediated interaction – coordination (glue)
 - services
 - example: EJB, CORBA Component Model (CCM)

evolution: Software organisation (2)

- from objects
- to services
 - often ***autonomous***
 - Embedding state & control
 - often ***situated***
 - Sensing and acting upon an environment
 - often ***pro-active***
 - not only reactive
 - often with ***high-level Communication Language***
- to system of services
 - *societies*

research context:

Agents and MultiAgent systems

Middleware and Infrastructures

- *middleware* concept
 - *the medium* enabling the interaction among participant
 - communication
 - interoperability
 - the *glue* among objects/components/services
- *infrastructure* concept
 - the medium is alive
 - runtime, instant deployability
 - basic shared services

seminar overview

- Interaction: paradigm shift
 - some perspectives on the impact
- Client/server middleware (overview and comparisons)
 - CORBA
 - HTTP/CGI
 - HTTP/Servlets
 - RMI
- Service Oriented Architectures/Infrastructures
 - JINI
 - JavaSpaces Service
 - WebServices
 - Peer-to-Peer Infrastructure
 - JXTA

Client-server middleware

- overview/comparison of some important approaches
 - CORBA
 - taken as reference middleware for comparisons
 - HTTP/CGI
 - HTTP/Servlet
 - RMI

CORBA - overview

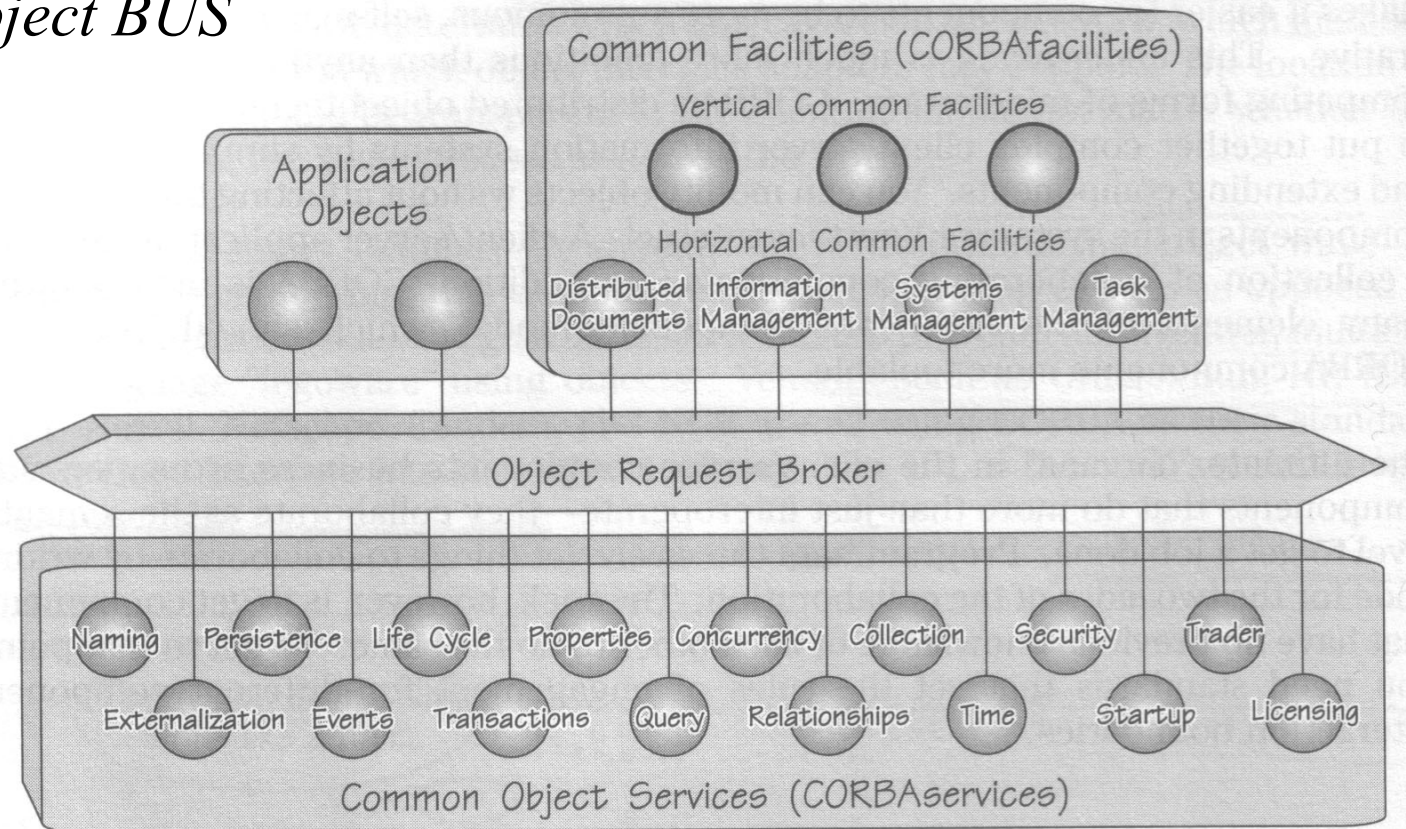
- middleware / infrastructure for Distributed Object computing
 - product of the OMG (Object Management Group) (800 company)
 - enable interaction and inter-operation among objects coming from different programming languages and models (heterogeneity), supporting also non OO programming languages (es: C)
 - providing services on top of interoperability
- ***Object*** as unifying metaphor for bringing existing applications to the network
 - foundation for component based future
 - the entire system is self describing

CORBA – overview

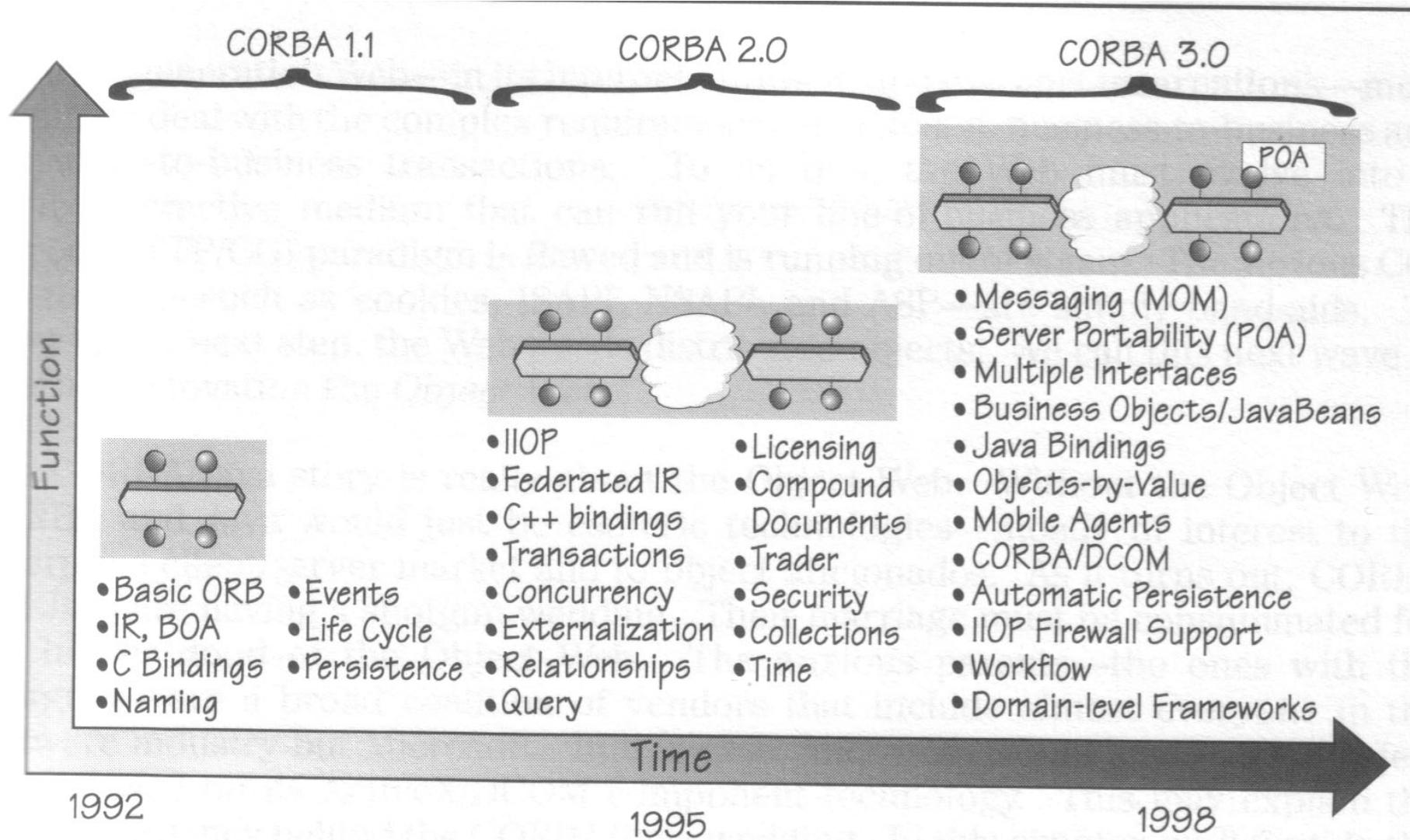
- clear separation of object specification and implementation:
 - Objects' Interfaces are described using the neutral **CORBA IDL** (Interface Definition Language)
 - define component's boundaries and contractual interfaces with clients
 - objects are packaged as binary components
 - heterogeneity (programming languages, OS,...)

CORBA – overview

- ORB (Object Request Broker) as the *medium* enabling interaction / interoperation
 - *Object BUS*

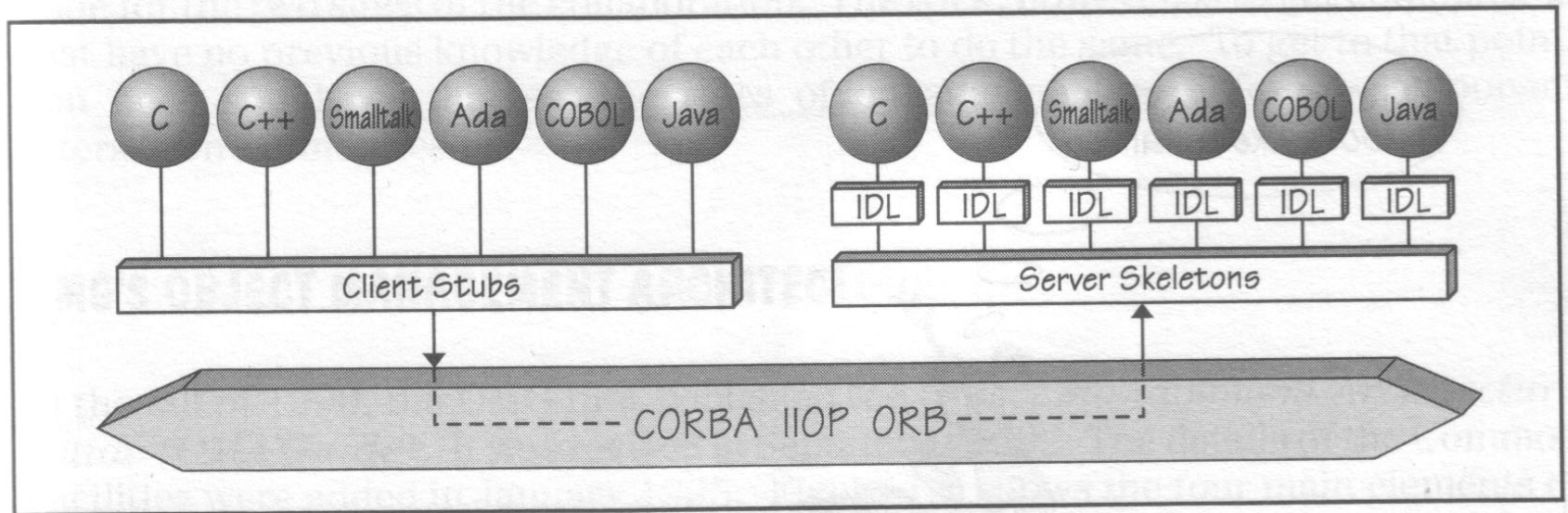


CORBA – evolution



CORBA – overview

- GOAL: to *IDL-ize* all client/server middleware and all components living on the ORB



CORBA – IDL syntax

```
module <identifier> ←
```

Defines a
naming context

```
{  
  <type declarations>;  
  <constant declarations>;  
  <exception declarations>;
```

```
    interface <identifier> [:<inheritance>] ←
```

Defines a
CORBA class

```
    {  
      <type declarations>;  
      <constant declarations>;  
      <attribute declarations>;  
      <exception declarations>;
```

```
      [<op_type>]<identifier>(<parameters>) ←  
      [raises exception][<context>;
```

Defines a
method

```
      .  
      .
```

```
      [<op_type>]<identifier>(<parameters>) ←  
      [raises exception][<context>;
```

Defines a
method

```
      .  
      .
```

```
    }
```

```
    interface <identifier> [:<inheritance>] ←
```

Defines a
CORBA class

```
    .  
    .
```

```
}
```

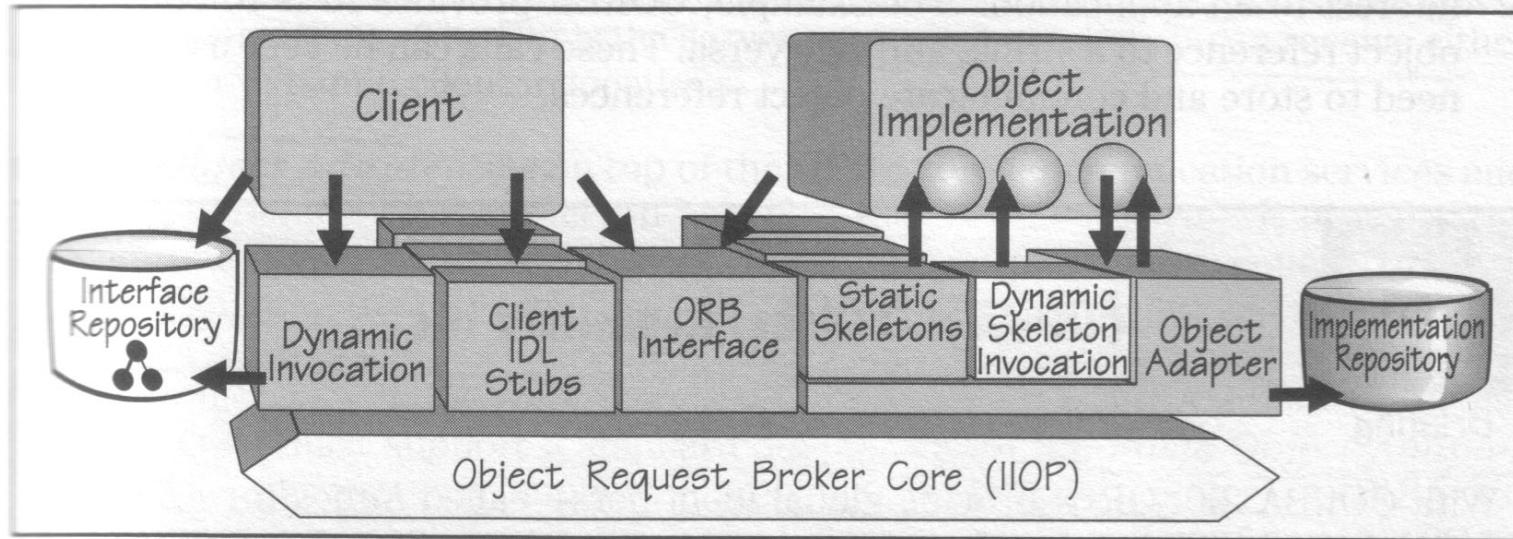

CORBA – IDL/Java mapping

CORBA IDL	Java
General Constructs	
module	package
user exceptions	Java class—extends org.omg.CORBA.UserException which in turn extends java.lang.Exception
system exceptions	Java class org.omg.CORBA.SystemException —extends java.lang.RuntimeException
in parameters	normal Java parameters
out and inout parameters	Java Holder classes (IDL generated and then instantiated by client side)
typecasts (or narrow)	Java Helper classes
attributes	Java overloaded accessor and modifier methods with the same name as the attribute
Primitive Types	
const	public static final field
boolean, TRUE, FALSE	boolean, true, false

CORBA – IDL/Java mapping

CORBA IDL	Java
Primitive Types (continued)	
char, wchar	char
octet	byte
string, wstring	java.lang.String
short, unsigned short	short
long, unsigned long	int
long long, unsigned long long	long
float	float
double	double
Constructed Types	
interface	interface
sequence	array
array	array
struct	Java class with same name as struct type (includes instance variables and constructors for each field)
enum	Java class with same name as enum type
union	Java class with same name as union type class with get/set methods for fields
typedef	Java does not have a typedef construct. Mapped to simple IDL types or user-defined IDL types.
any	Java class org.omg.CORBA.Any (the ORB is the factory)
Server-Side Mapping	
server implementation using inheritance	Write an implementation class that extends the IDL-generated <interface_name>ImplBase class
server implementation using delegation (i.e., Tie)	IDL-generated <tie_name>Tie class that delegates calls to the implementation class you provide. The implementation class implements one or more IDL-defined interfaces (each represented by an IDL-generated <interface_name>Operations interface).
POA root servant class	Java class org.omg.CORBA.portable.ObjectImpl

CORBA – ORB architecture



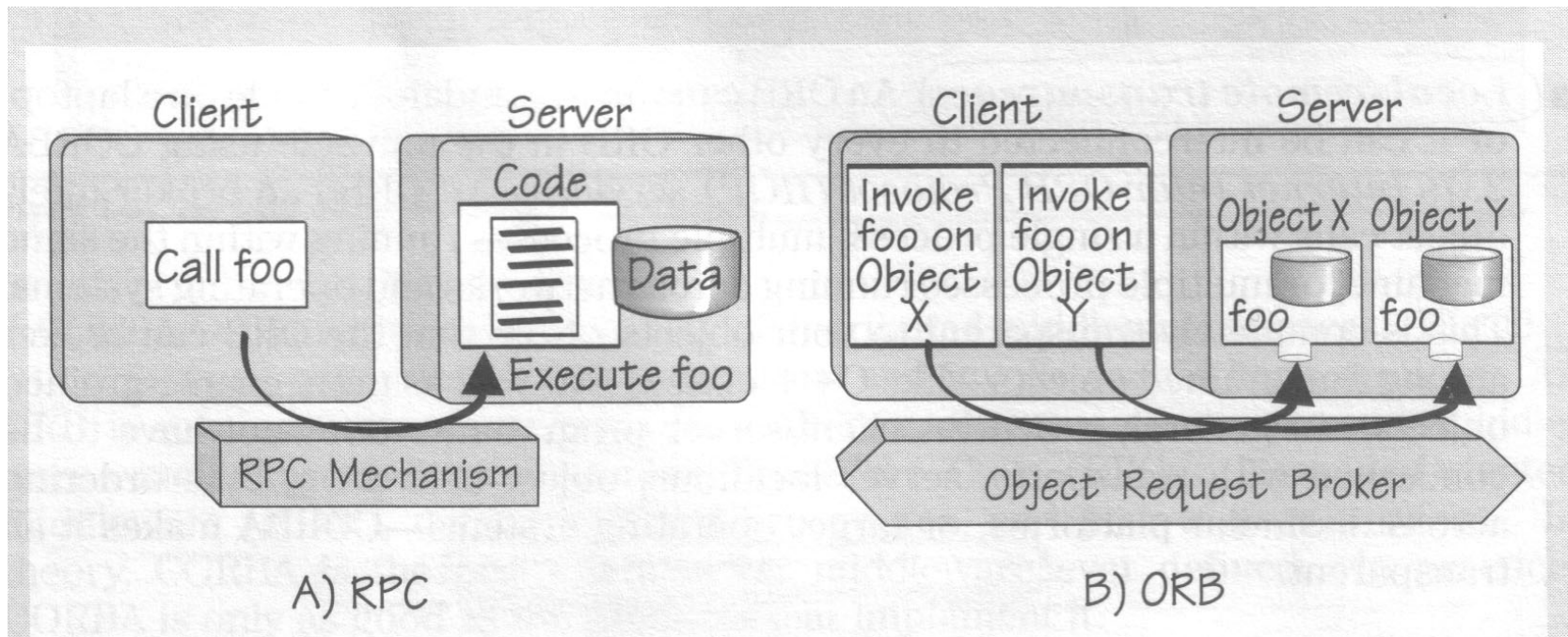
- Client Side
 - Client IDL Stubs
 - Dynamic Invocation Interface (DII)
 - Interface Repository
- Server Side
 - Object Adapter
 - Server IDL Stubs (Skeletons)
 - Dynamic Skeleton Interface (DSI)
 - Implementation Repository
- Both
 - ORB Interface (general purpose local services)

CORBA – ORB architecture

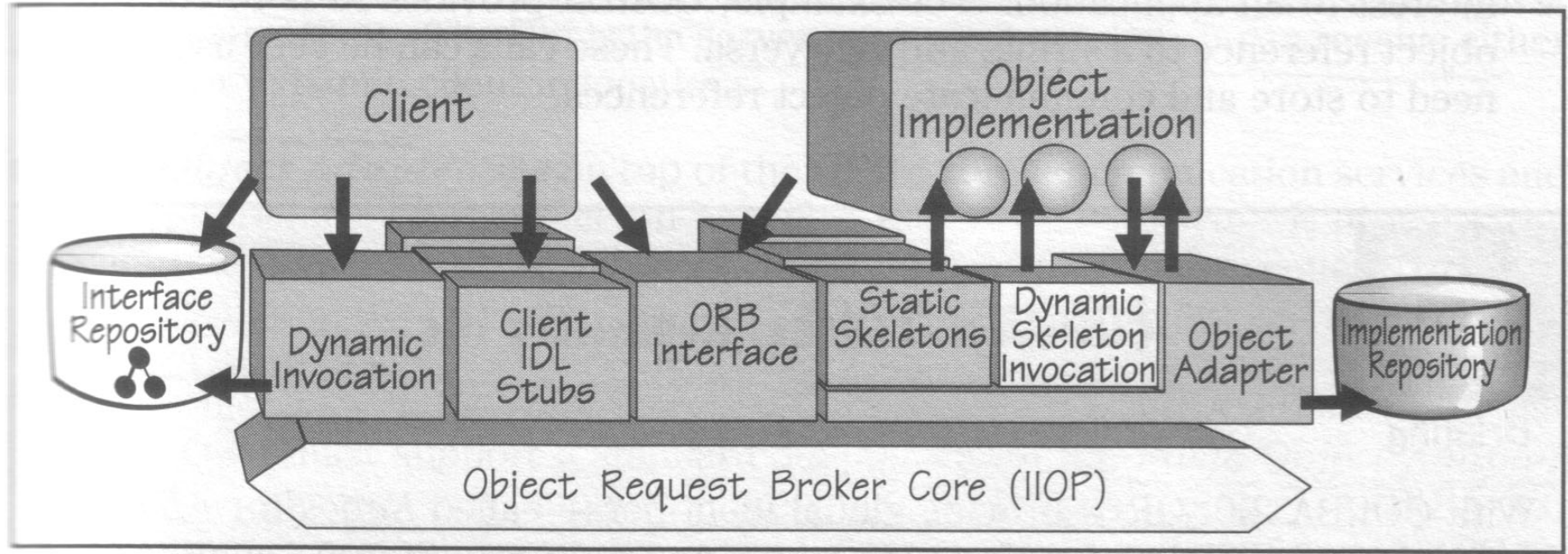
- Lets objects discover each at runtime and invoke each other's services
 - static and dynamic method invocations
 - polymorphic messaging (ORB vs RPC)
 - self-describing systems
- local/remote *transparency*
 - developers not concerned with transports, server locations, object activations, ...
- coexistence with existing systems
 - CORBA IDL to encapsulate existing applications
- CORBA 2.0 → standards for interoperability among ORBS (IIOP - Internet Inter-ORB Protocol - services)

CORBA – ORB architecture

- ORB vs RPC: bringing OO features to the network
 - encapsulation
 - polymorphism

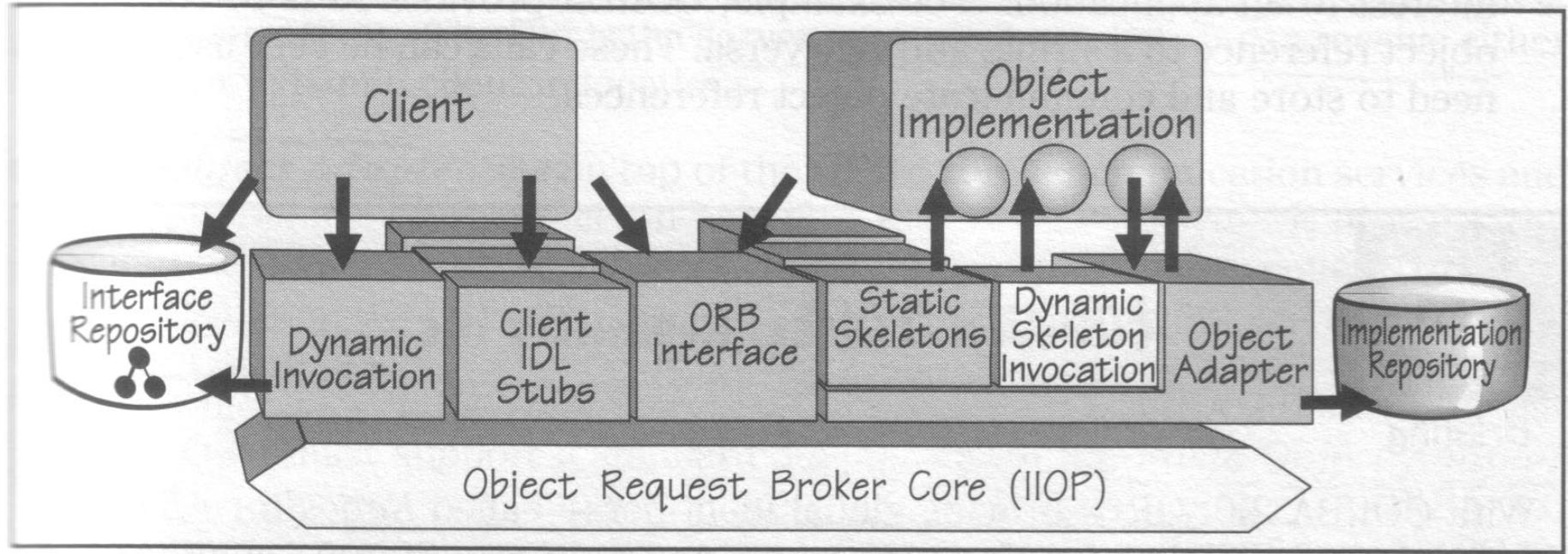


CORBA – ORB components: client side



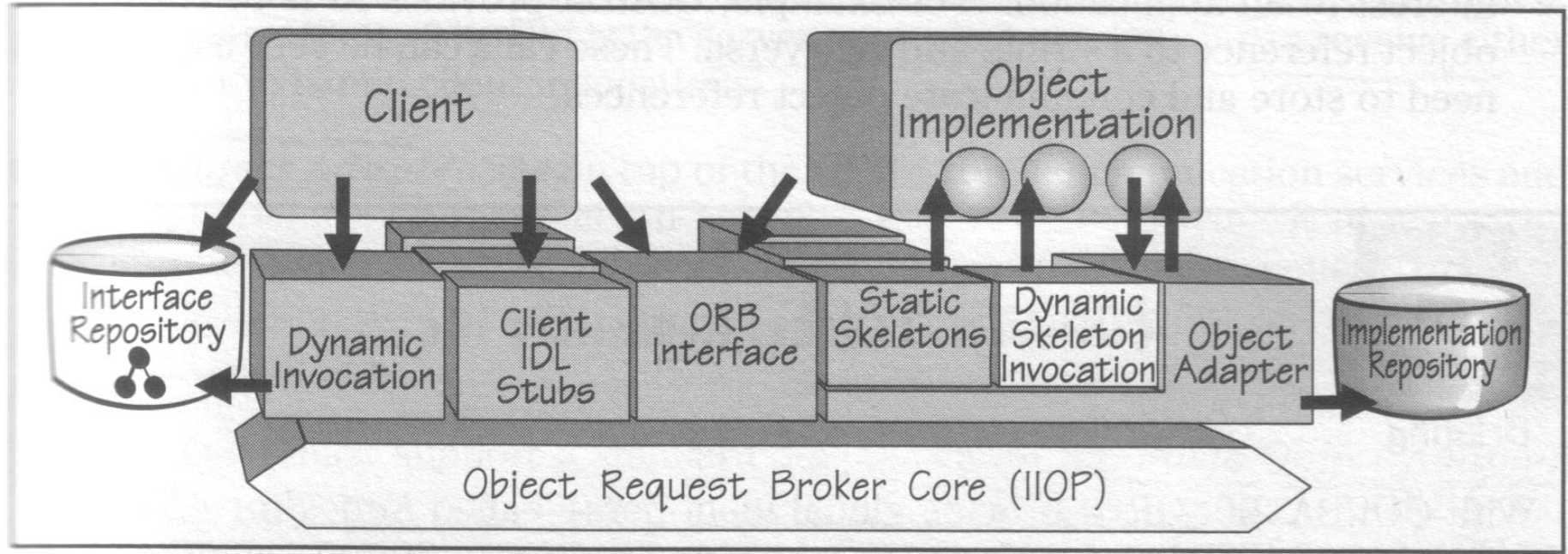
- Client IDL Stubs
 - provide static interfaces to object services
 - also called *Proxy*

CORBA – ORB components: client side



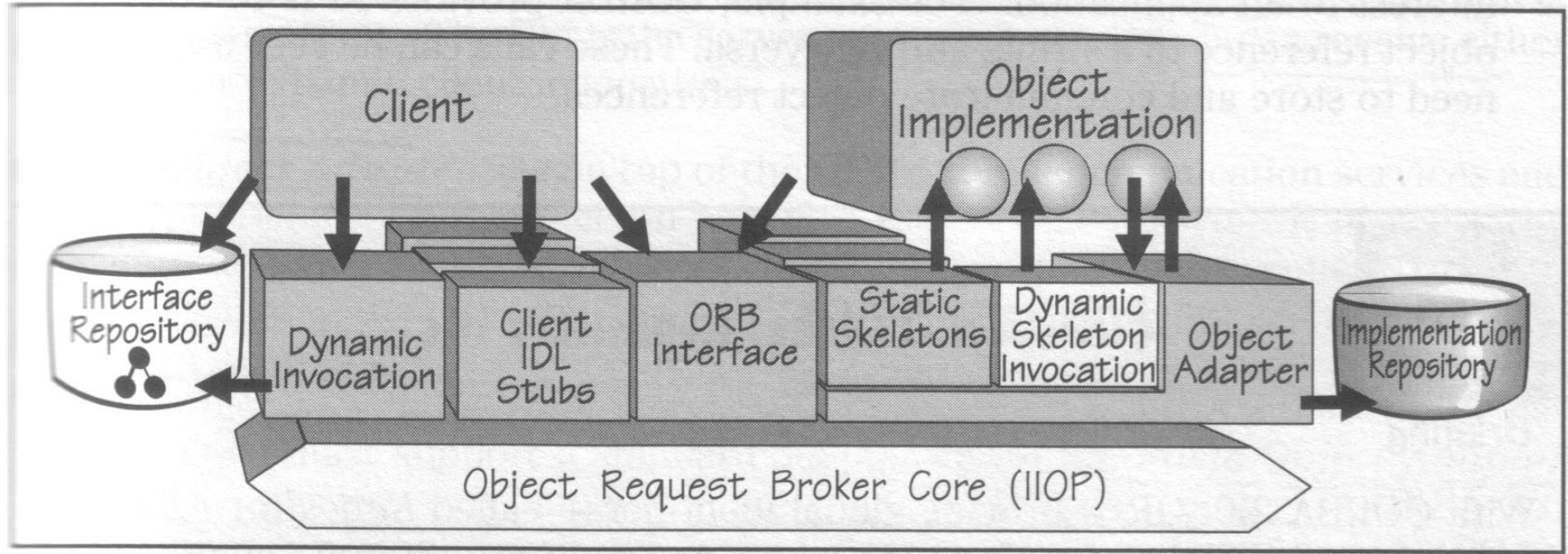
- Dynamic Invocation Interface (DII)
 - lets clients discover methods to be invoked at runtime

CORBA – ORB components: client side



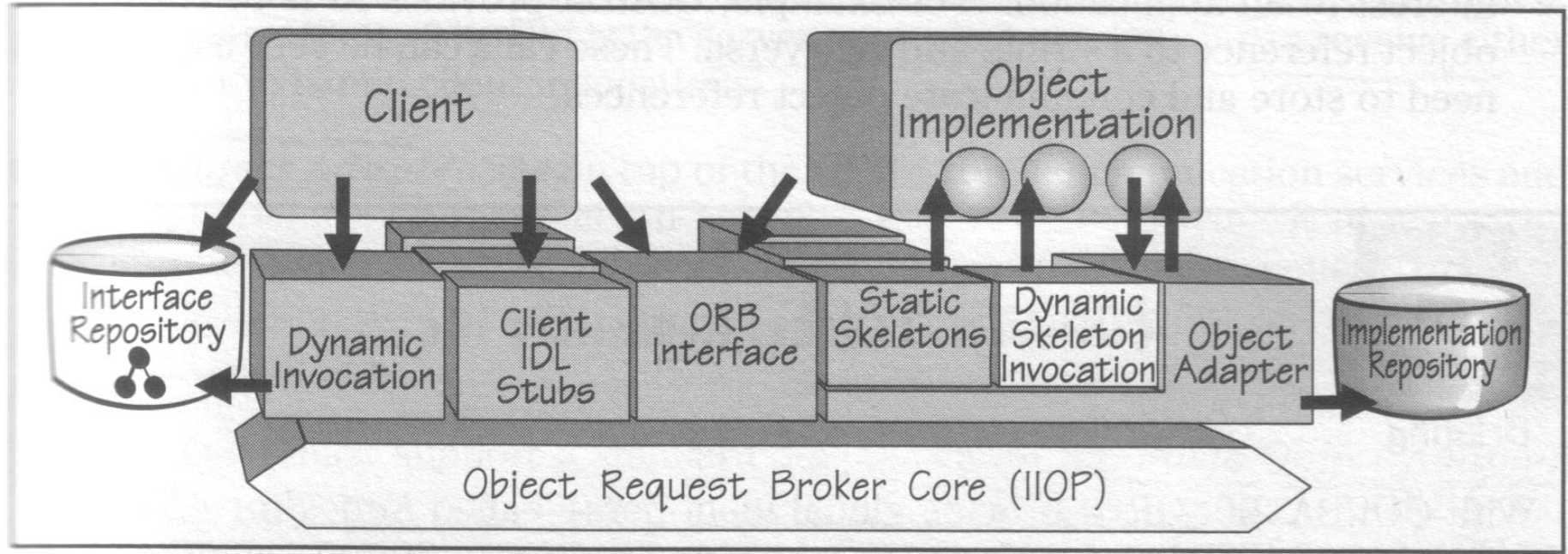
- Interface Repository API
 - allows objects on the bus to obtain and modify the descriptions (IDL) of all registered component interfaces
 - it's like a runtime distributed dbase, dynamic repository for ORBs
 - focus on METADATA: component with self-describing interfaces: ORB too is a self-describing BUS

CORBA – ORB components: server side



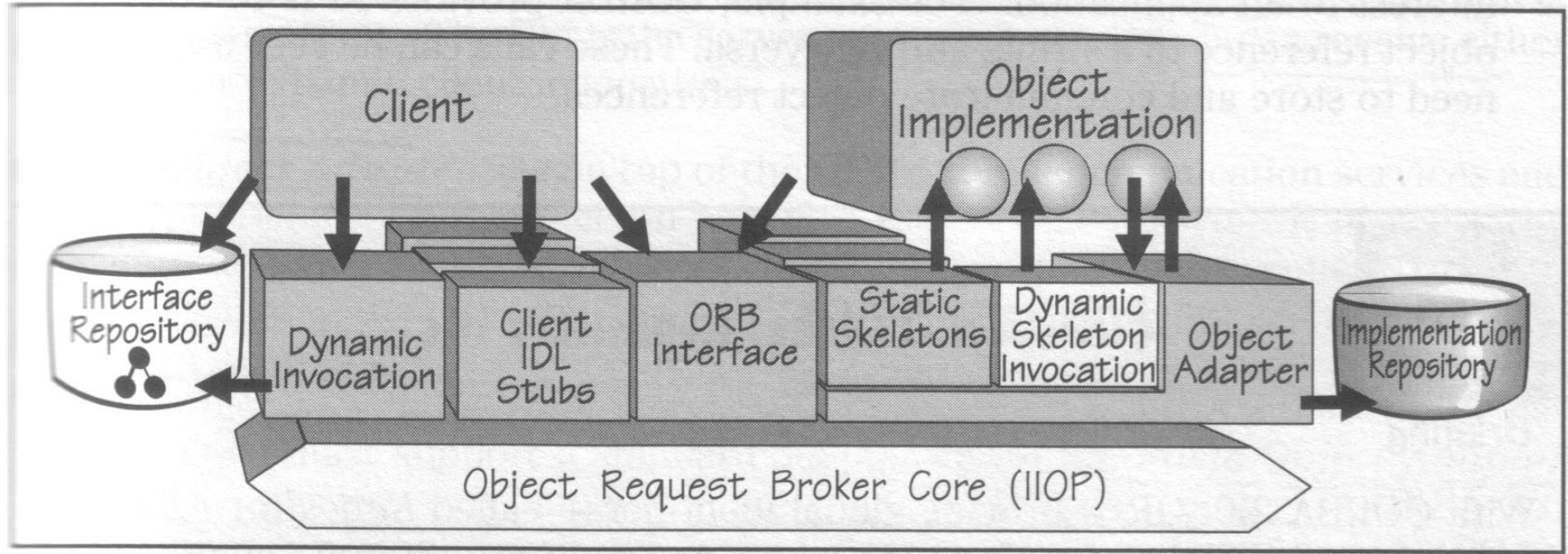
- Object Adapter (OA) (1)
 - accepts requests for service on the behalf of the server's objects
 - run-time environment for instantiating server objects, passing request to them, assigning them an ID (CORBA object reference)
 - registers the classes it supports and their run-time instances with the Implementation Repository

CORBA – ORB components: server side



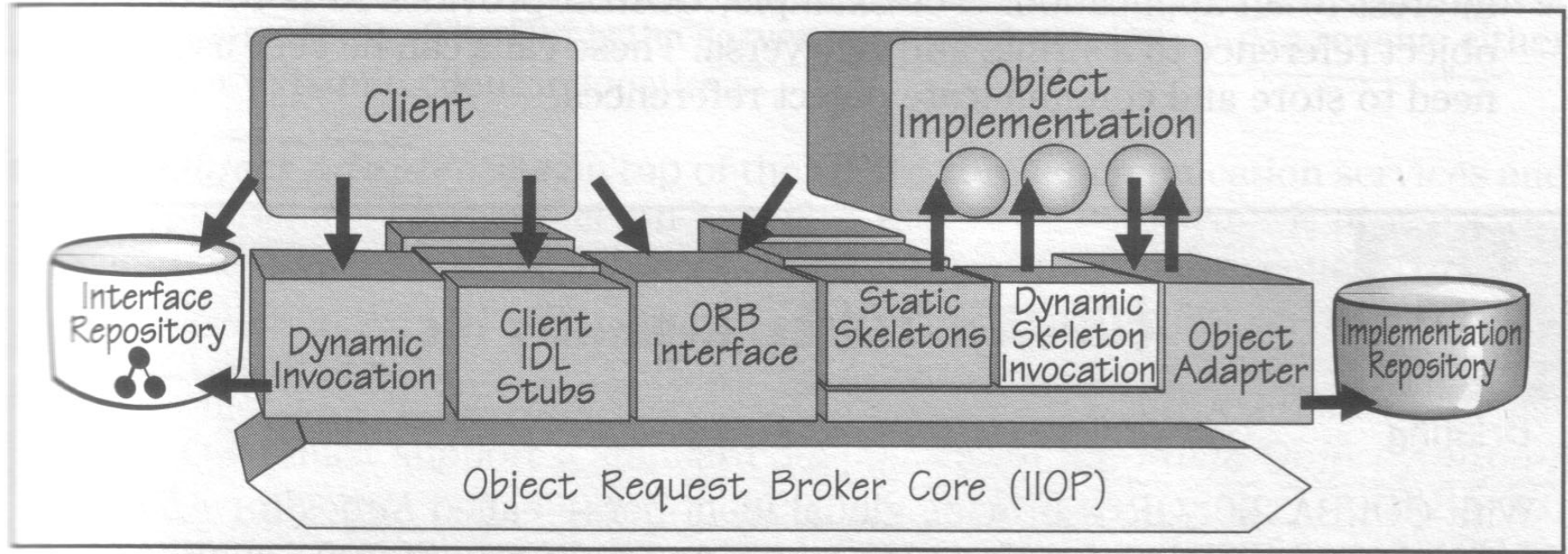
- Object Adapter (OA) (2)
 - multiple OAs can be supported by a server
 - BOA - Basic OA (CORBA 1.1) – basic services spec
 - POA - Portable OA (CORBA 2.0): portable BOA

CORBA – ORB components: server side



- Server Side Stubs (*Skeletons*)
 - provides static interfaces to each service exported by a server

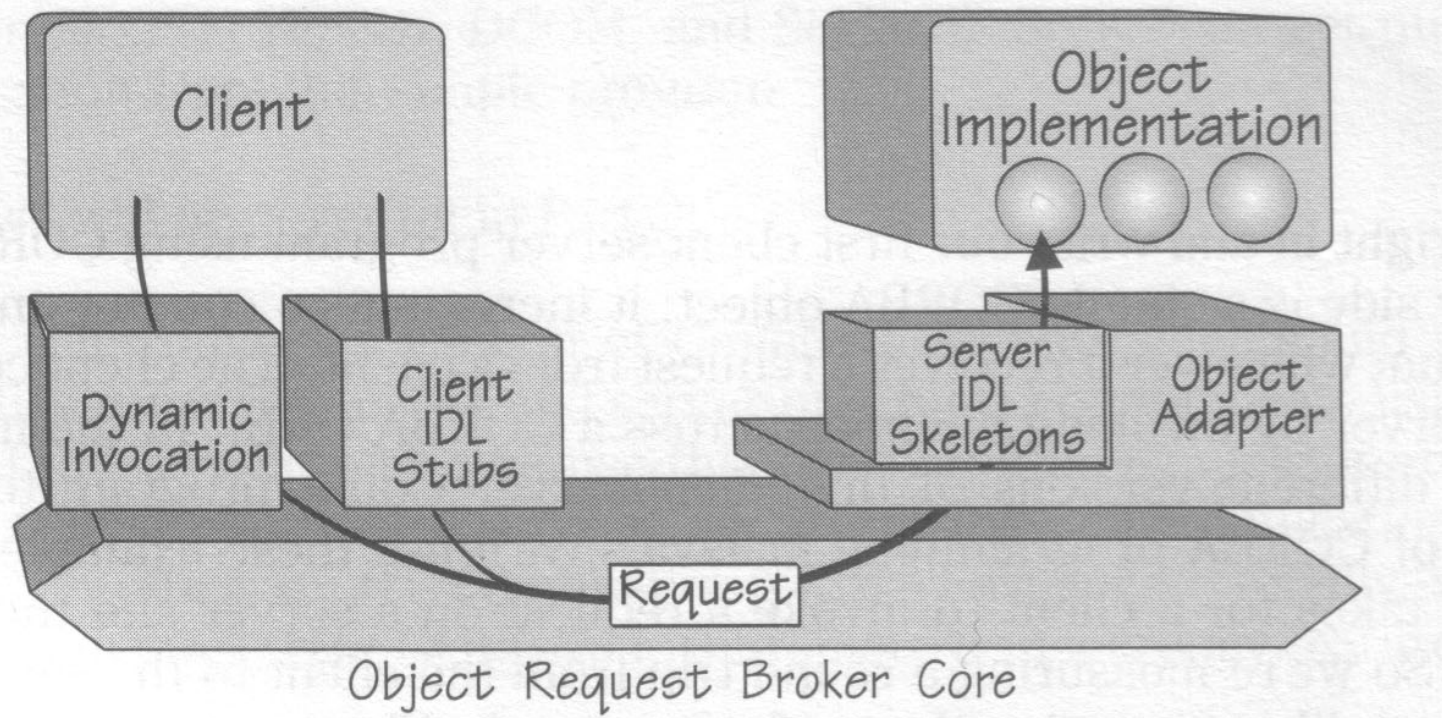
CORBA – ORB components: server side



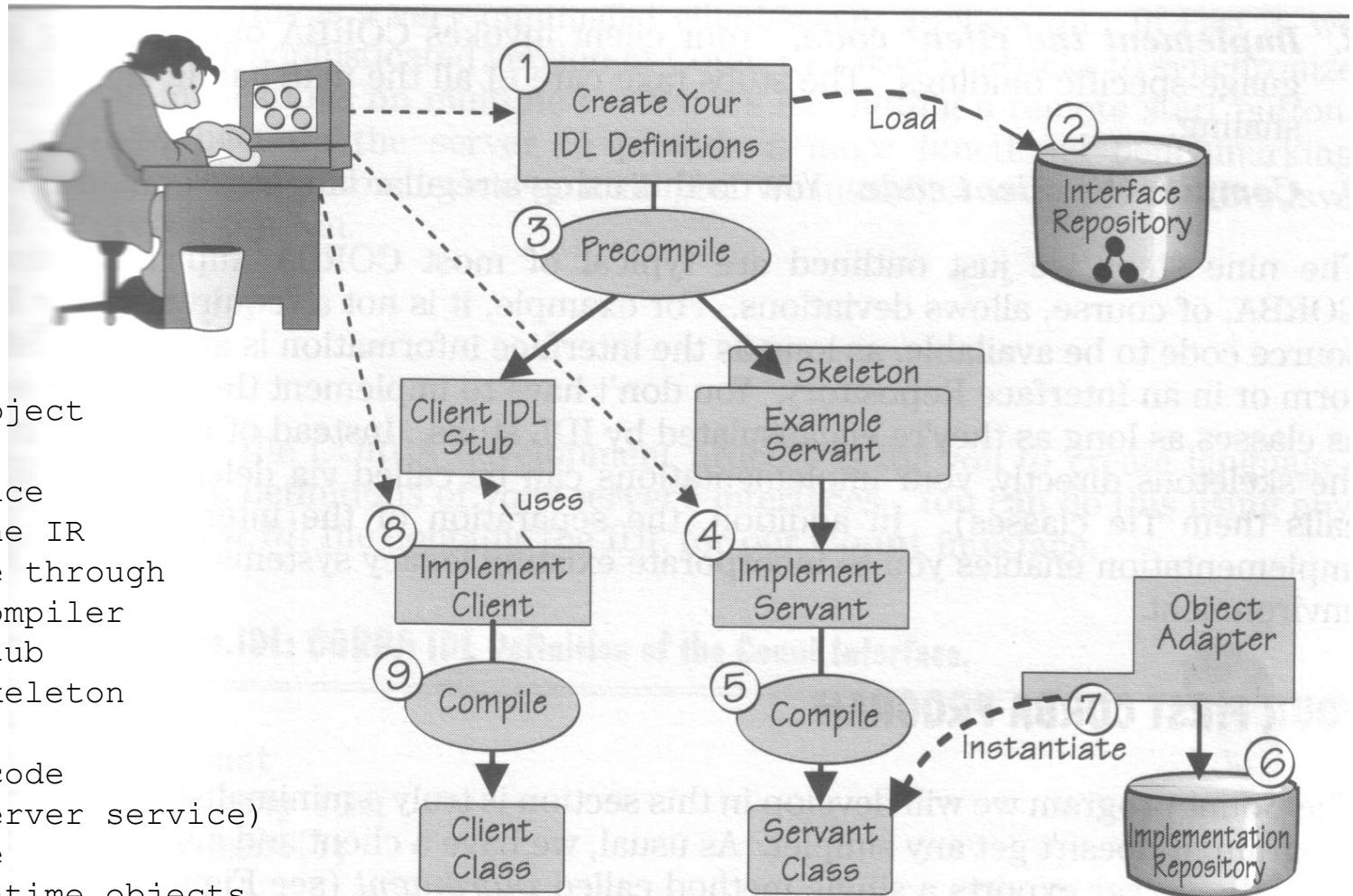
- Implementation Repository
 - maintain information about implementation of objects on the bus

CORBA – static & dynamic requests

- both static (via stubs) and dynamic (via reflection) request supported
 - run-time flexibility
 - support for open / dynamic / changing environments



CORBA – development process (static)

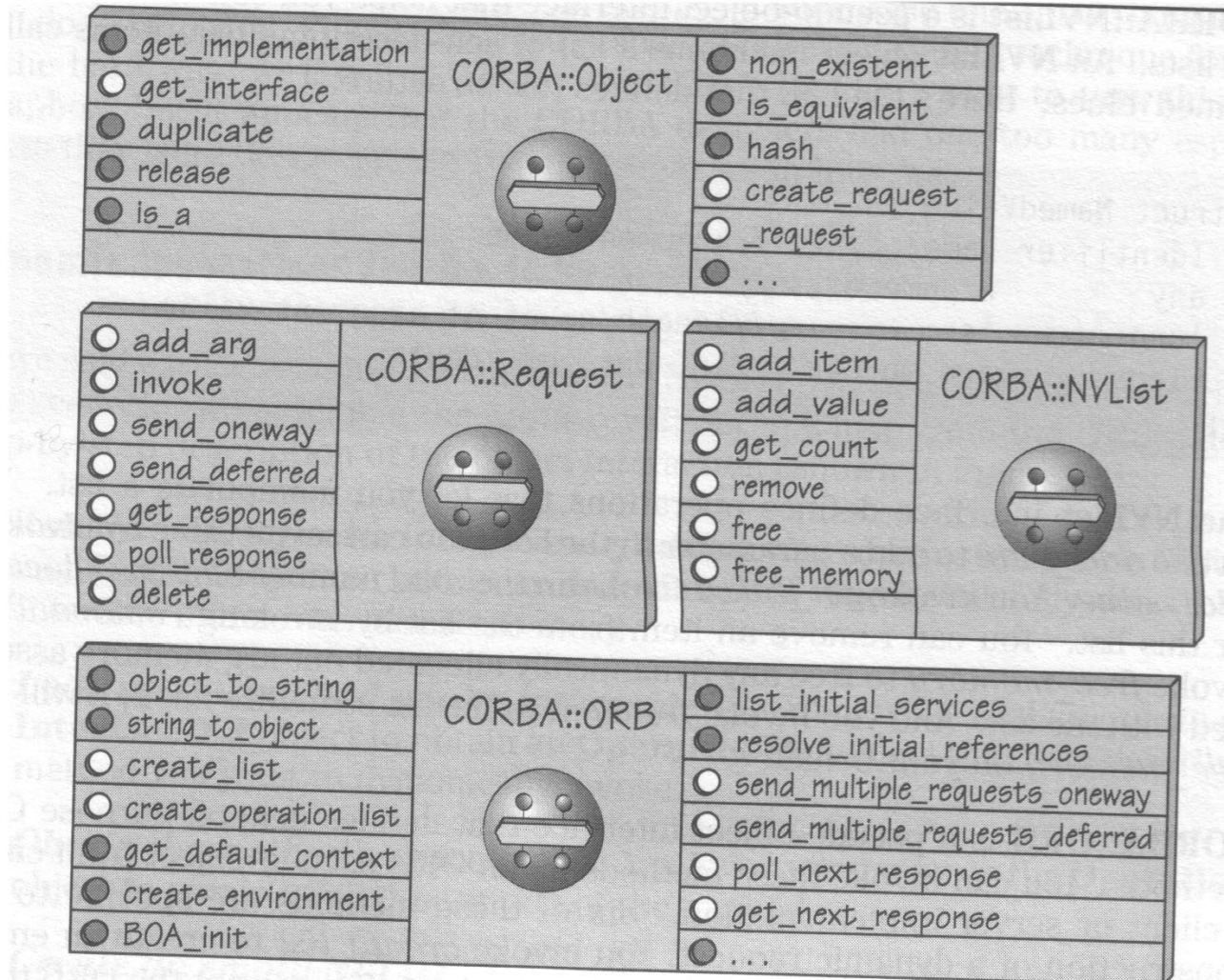


Define server object interface (IDL)
Bind the interface definition to the IR
Run the IDL file through a language precompiler

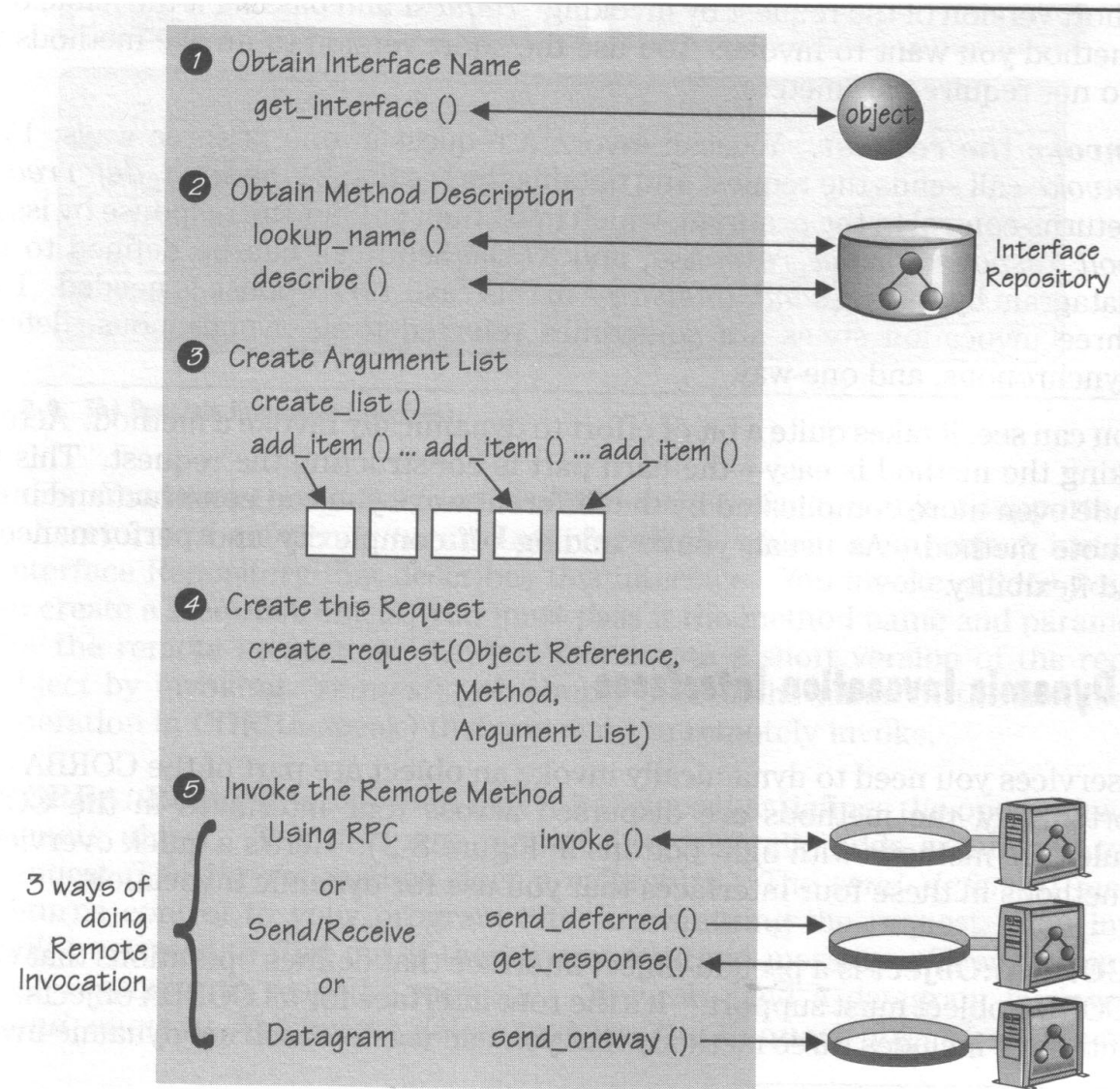
- client stub
- server skeleton

Add the servant implementation code (implementing server service)
Compile the code
Register the runtime objects with the Implementation Repository
Instantiate the Objects on the server
Implement the client code
Compile client code

CORBA – deploying Reflection



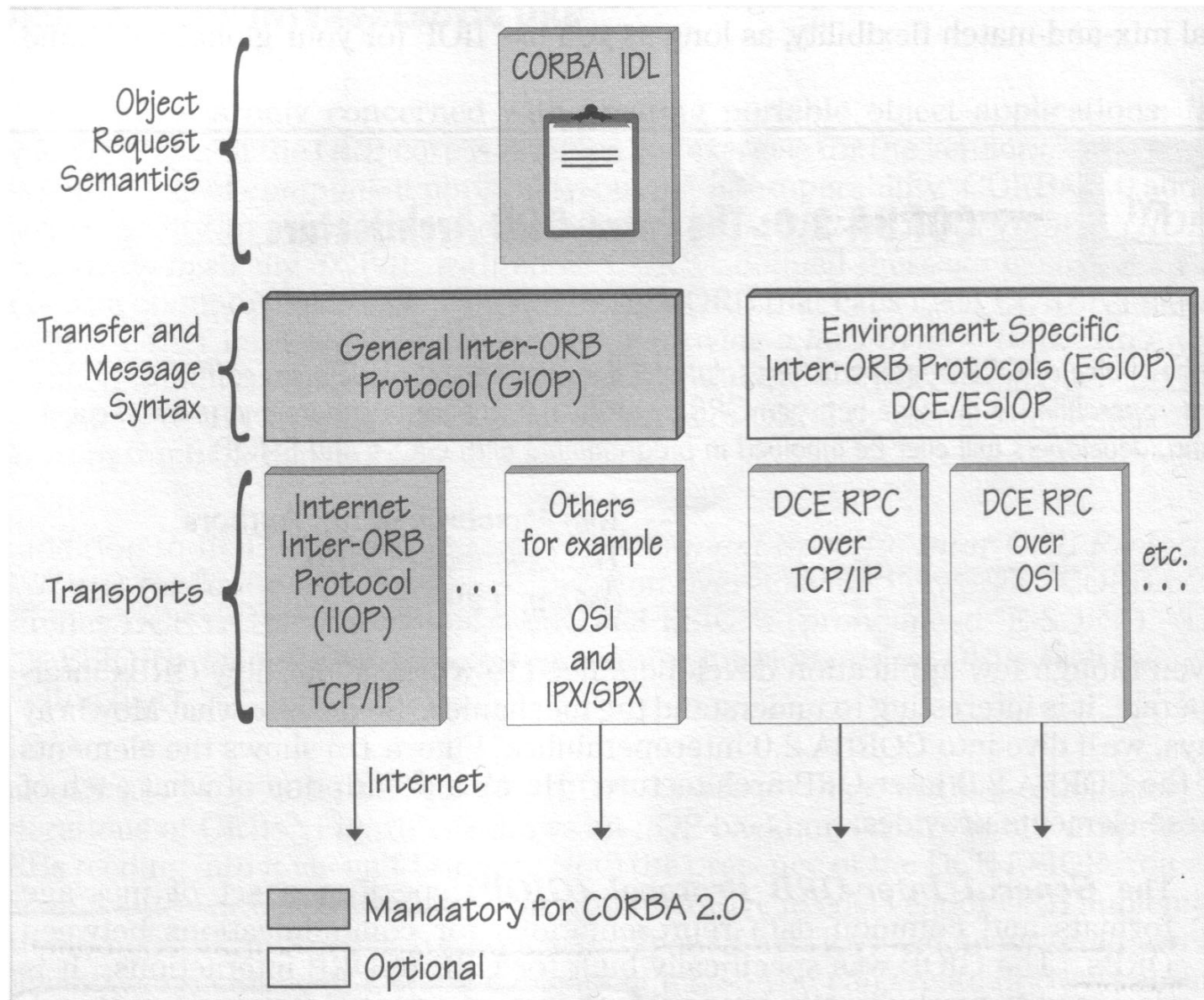
CORBA – dynamic invocation scenario



CORBA – Inter-ORB architecture

- Enabling interaction/interoperation among objects of different ORBs (CORBA 2.0)
 - GIOP (General Inter-ORB protocol)
 - specifies a set of message formats and common data representations for communication between ORBs
 - IOR (Interoperable Object Reference), to identify an Object Reference across ORBs
 - IIOP (Internet Inter-ORB protocol)
 - specifies how GIOP messages are exchanged over a TCP/IP network
 - ENSIOP (Environment specific Inter-ORB protocol)
 - Interoperation over specific networks

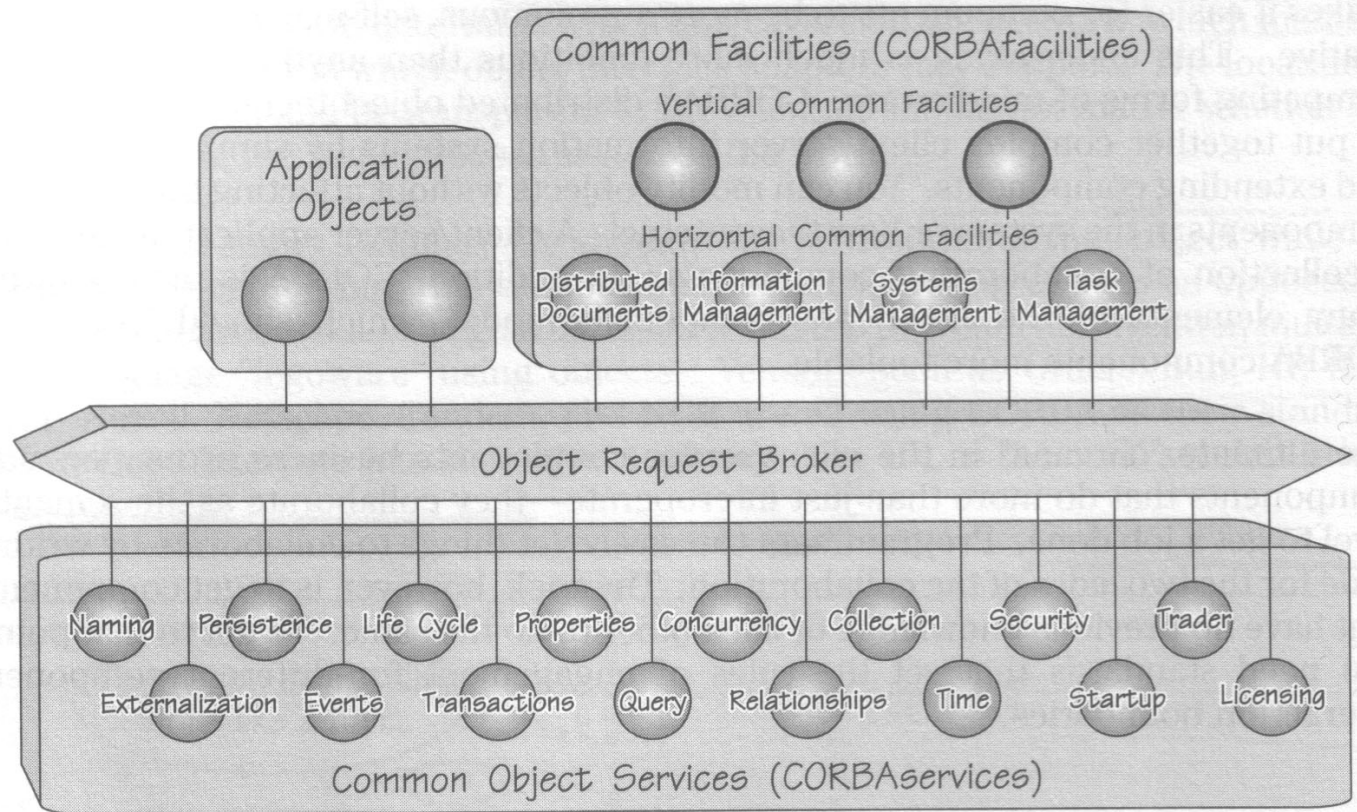
CORBA – Inter-ORB architecture



CORBA - services

- CORBA services

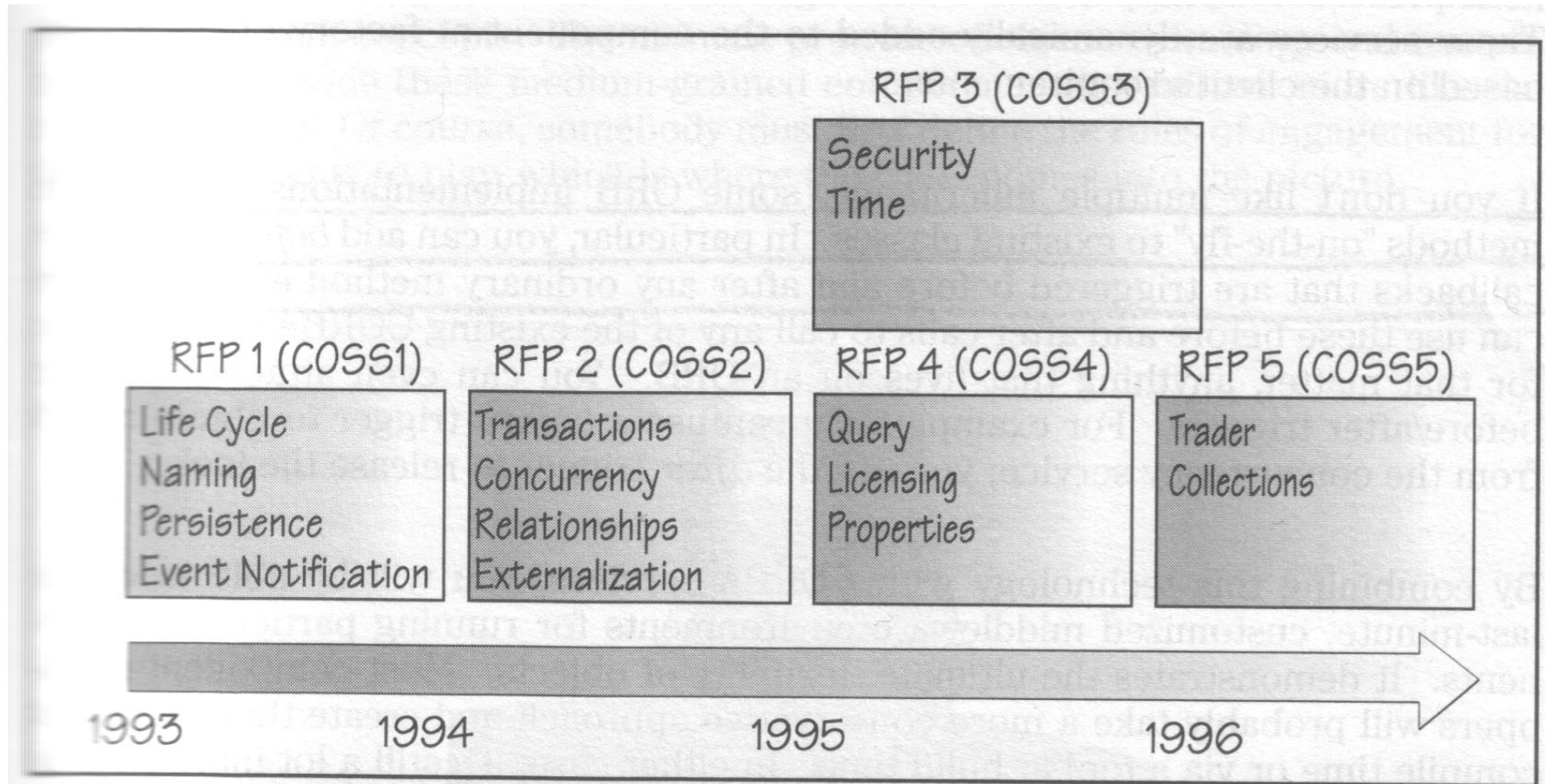
- collection of system level services packages with IDL interfaces
- beyond interaction and inter-operation, toward coordination, cooperation



CORBA – service overview

- Life Cycle
 - creating, copying, moving deleting objects on the bus
- Concurrency
 - provides mechanism (locks,...) for managing concurrent access to object
- Naming
 - White pages (locating object on the bus by name)
- Security
- Event
 - Publish/subscribe pattern for objects (event-channels)
- Persistence
 - Storing component on dbase (ODBMS, RDBMS,...)
- Transaction
 - 2-Phase commit protocol for transaction coordination activities
- Trader
 - Yellow pages
- Query (OQL)
- Time
-

CORBA – services evolution



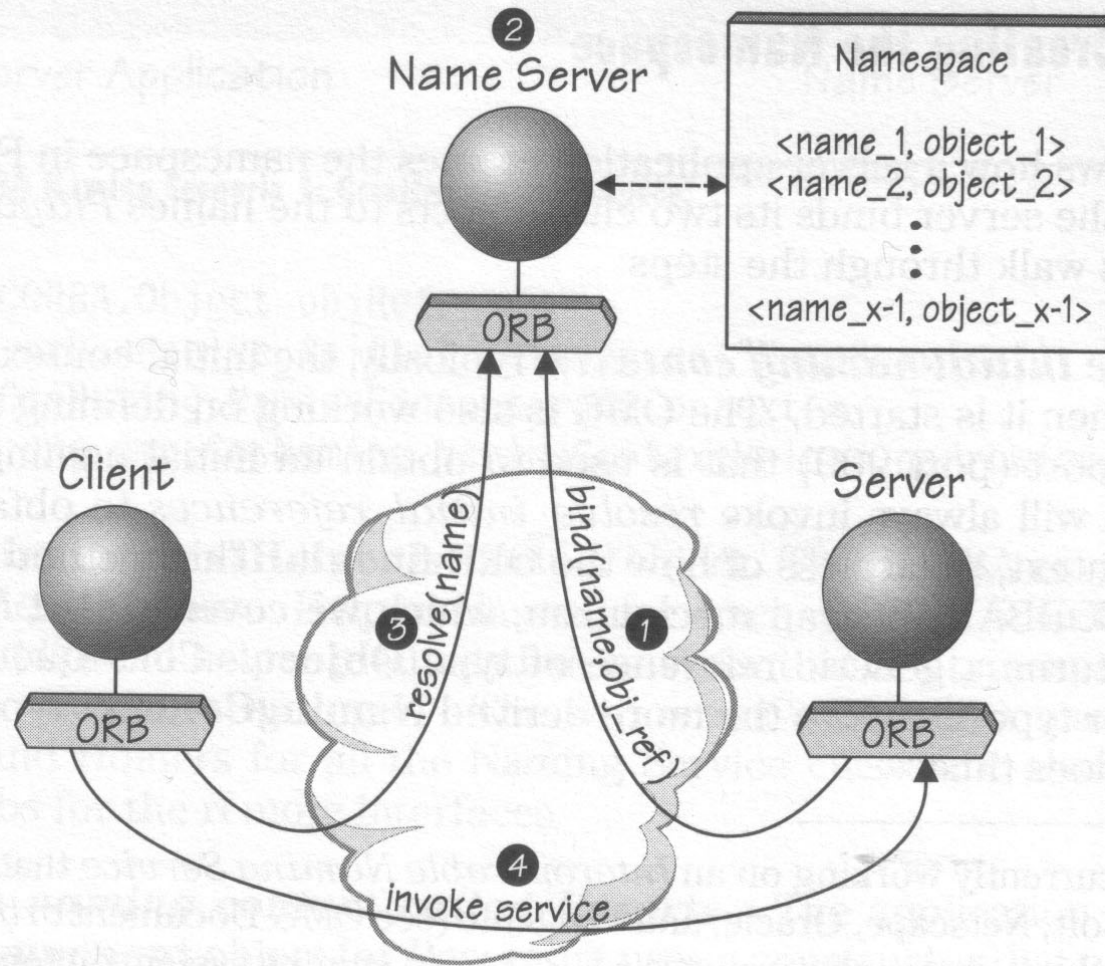
CORBA – Naming Service

- *White Pages service* for Objects on the bus
- allows components on the bus to locate other components by name, mapping human names to object references
- support for naming hierarchies, navigable by clients
- allows objects to be bound to existing network directories / naming contexts, such as X.500, LDAP, Sun NIS+

CORBA – Naming Service: concepts

- **Name** structure
 - identifier (name string)
 - kind (description string)
- Name **Context** = name space where the object's name is unique
 - Context can be nested
 - compound name: list of contexts + name
 - es: {"Ingegneria", "Lab-01", "Printers", "MyLaser"}
- **To Resolve** a name: find the object associated with a name in a given context
- **To Bind** a name: create a name-to-object association (*binding*) for a given context

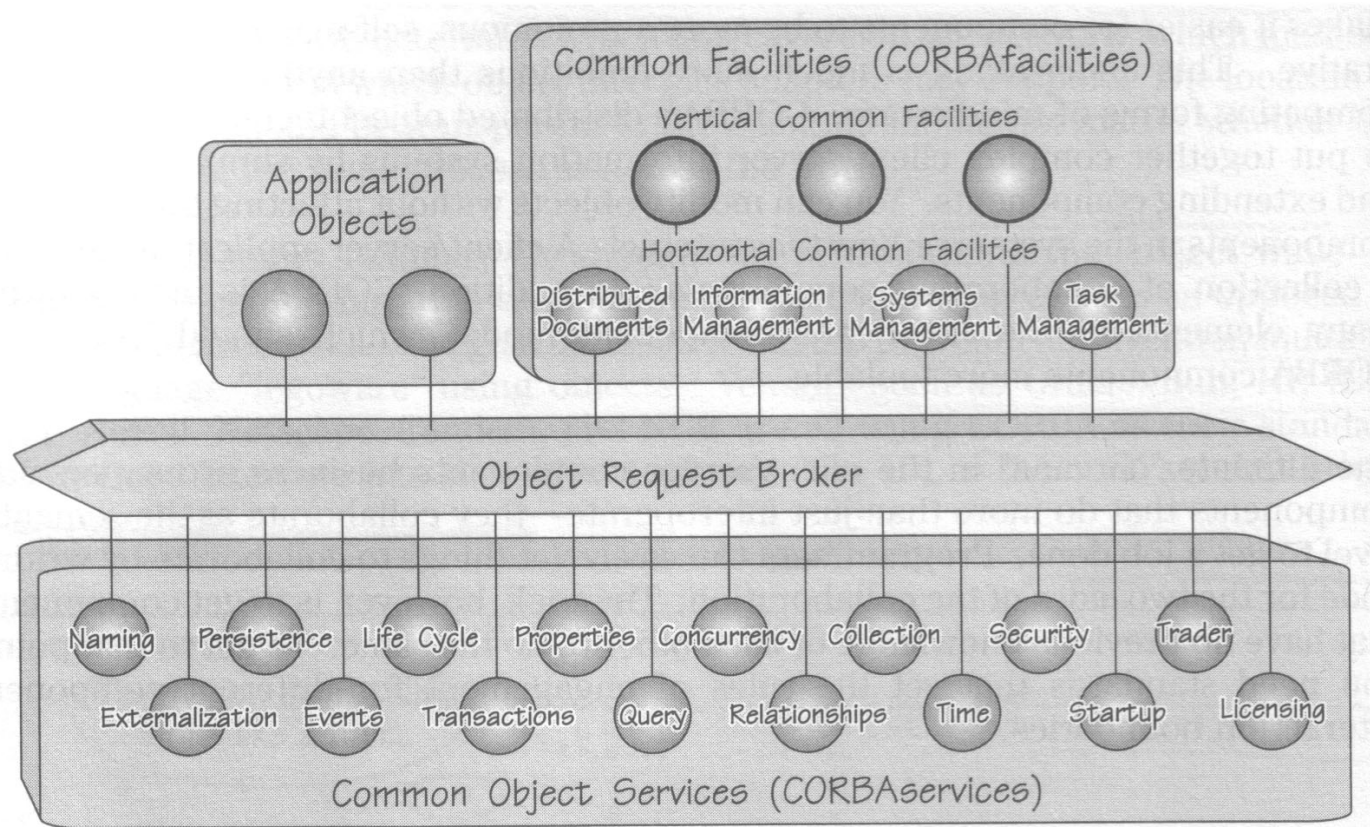
CORBA – Naming Service: scenario



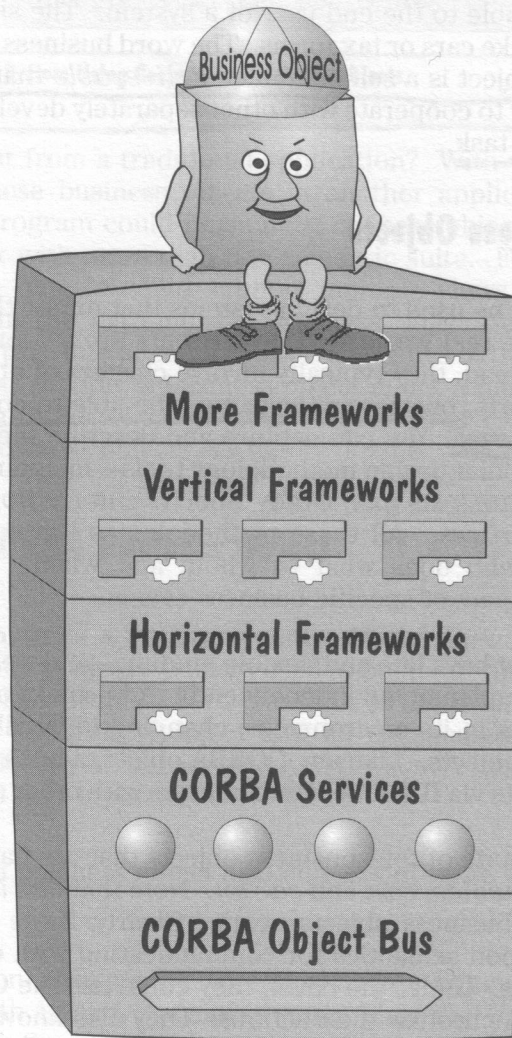
CORBA - frameworks

- CORBAfacilities

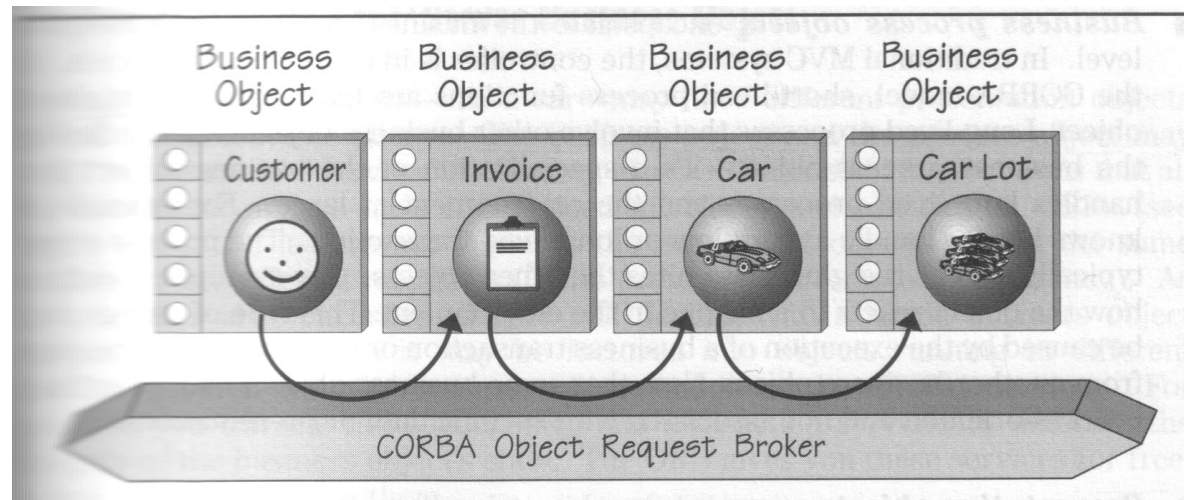
- collections of IDL-defined *frameworks* that provides services of direct use to application objects
- next step in the semantic hierarchy
 - Workflow
 - Internationalisation
 - Licensing
 - ...



CORBA – Business Objects

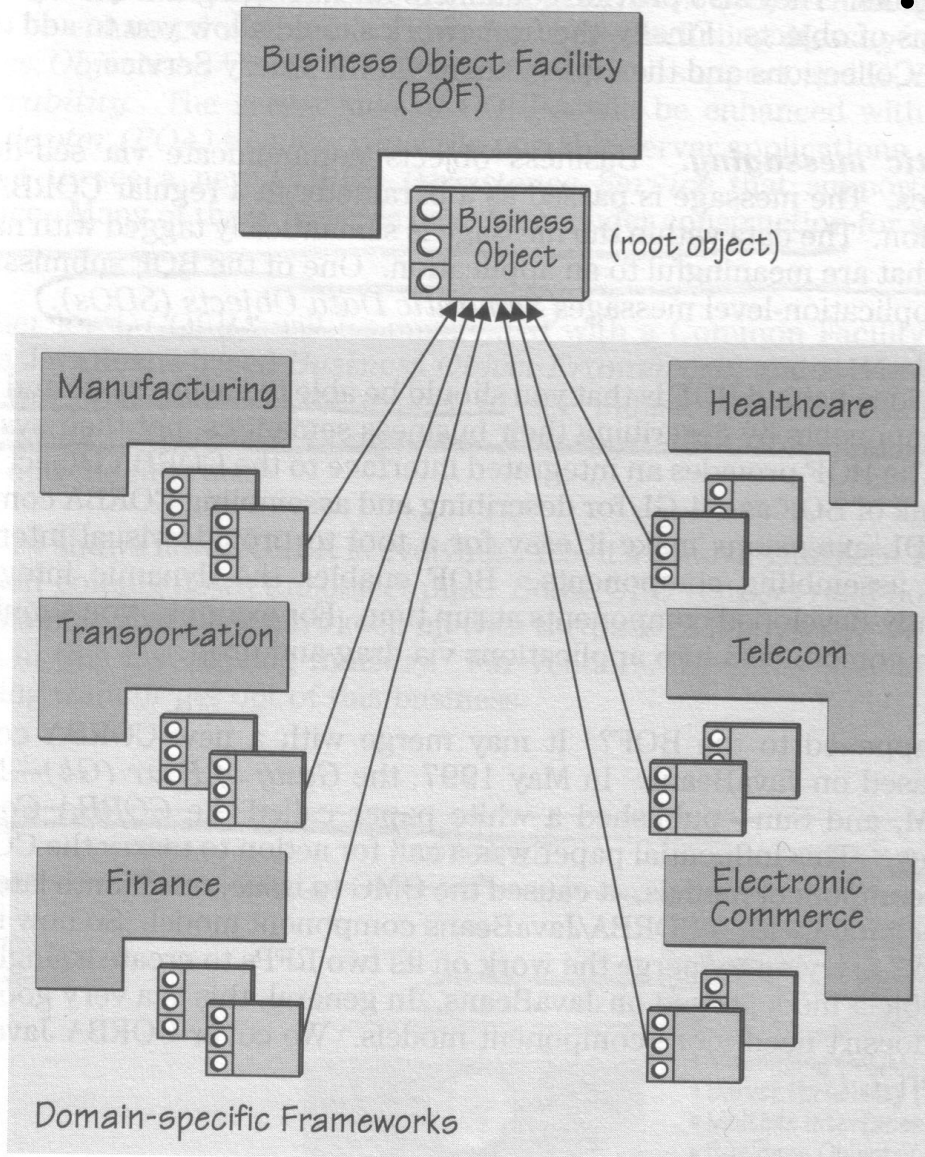


- Representing application-independent concepts such as customer, cars
- sharing semantics objects among different applications

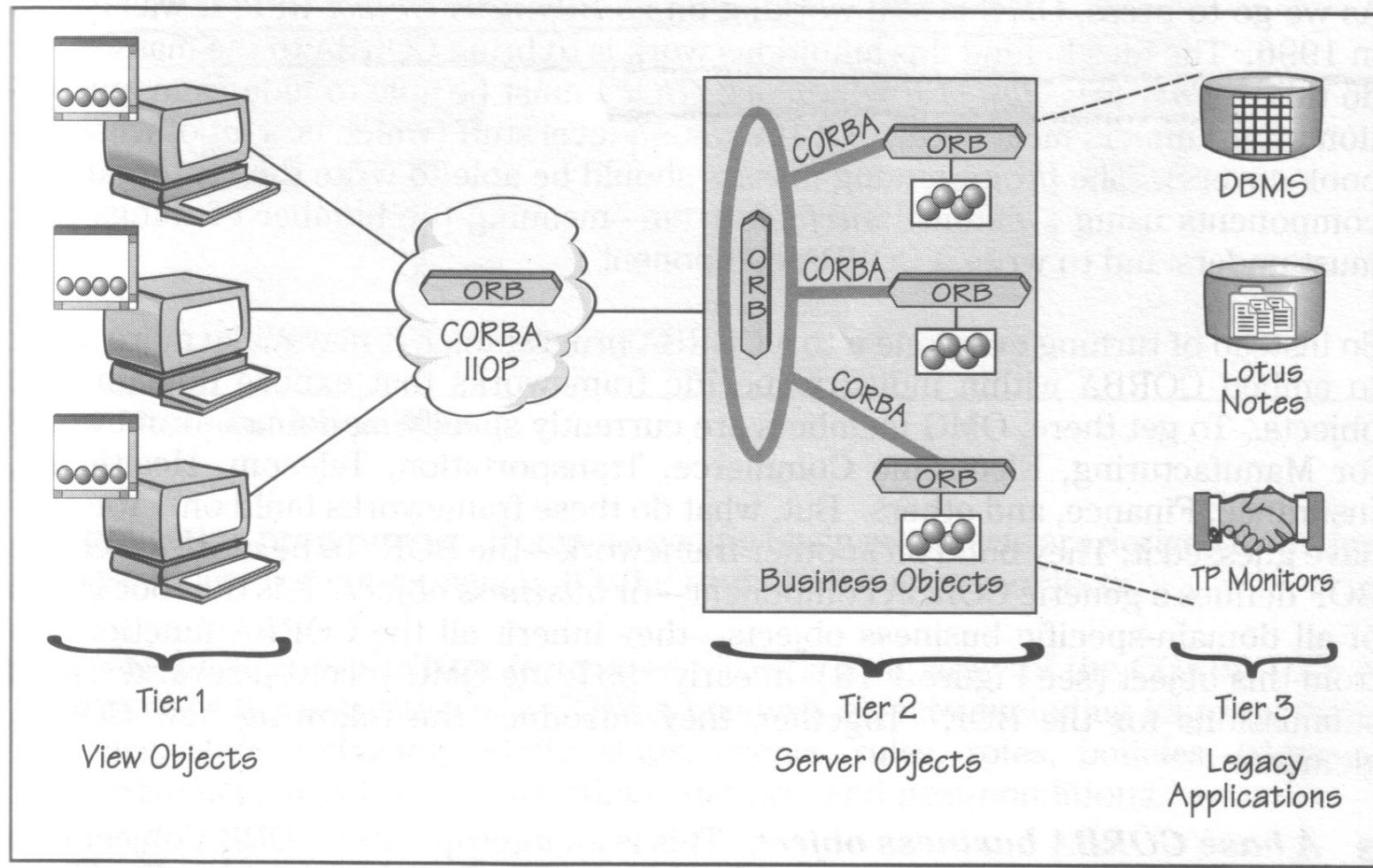


CORBA - Frameworks

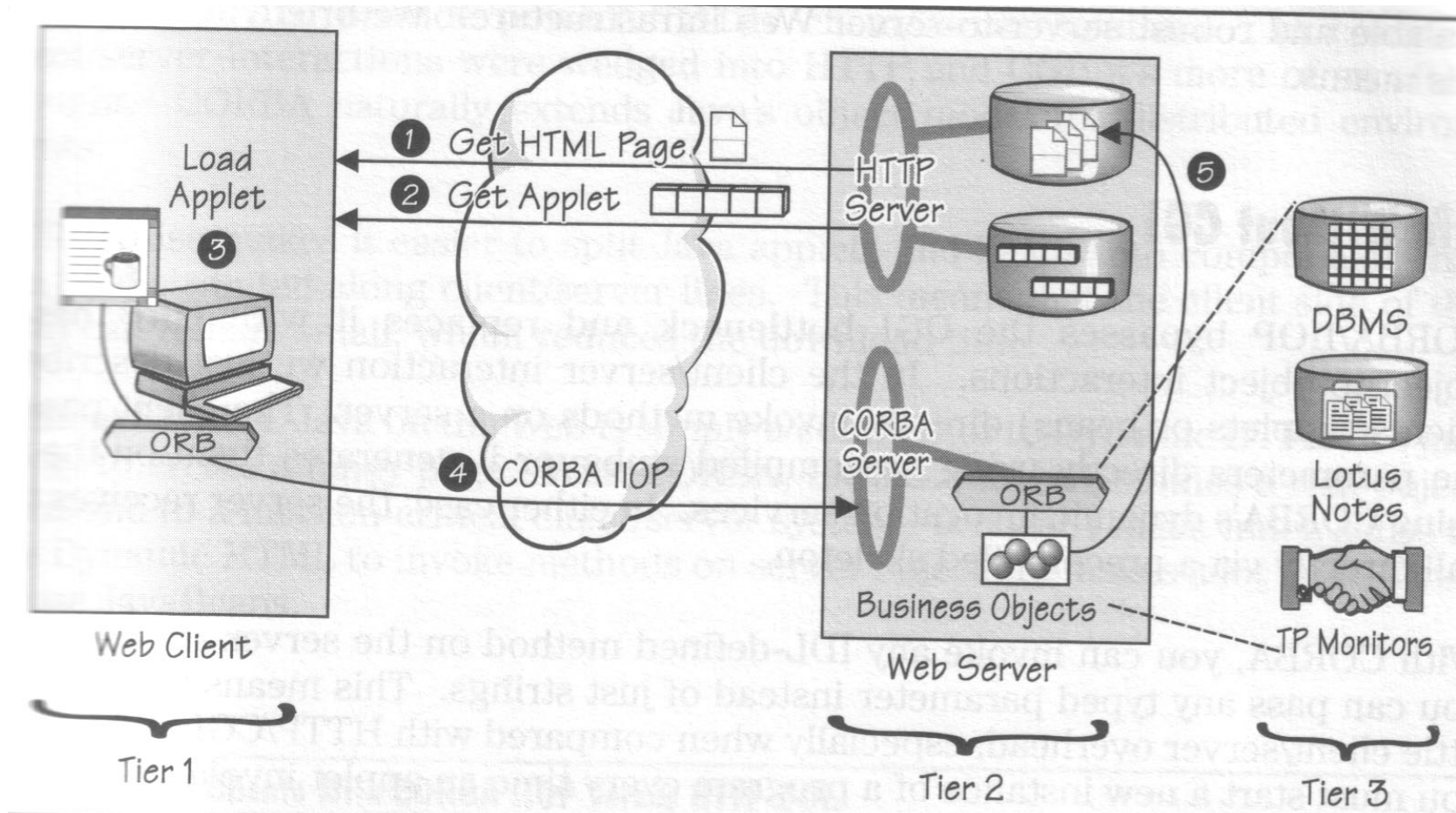
- BOF (Business Object Facility)
 - semantic frameworks where business objects live and interact



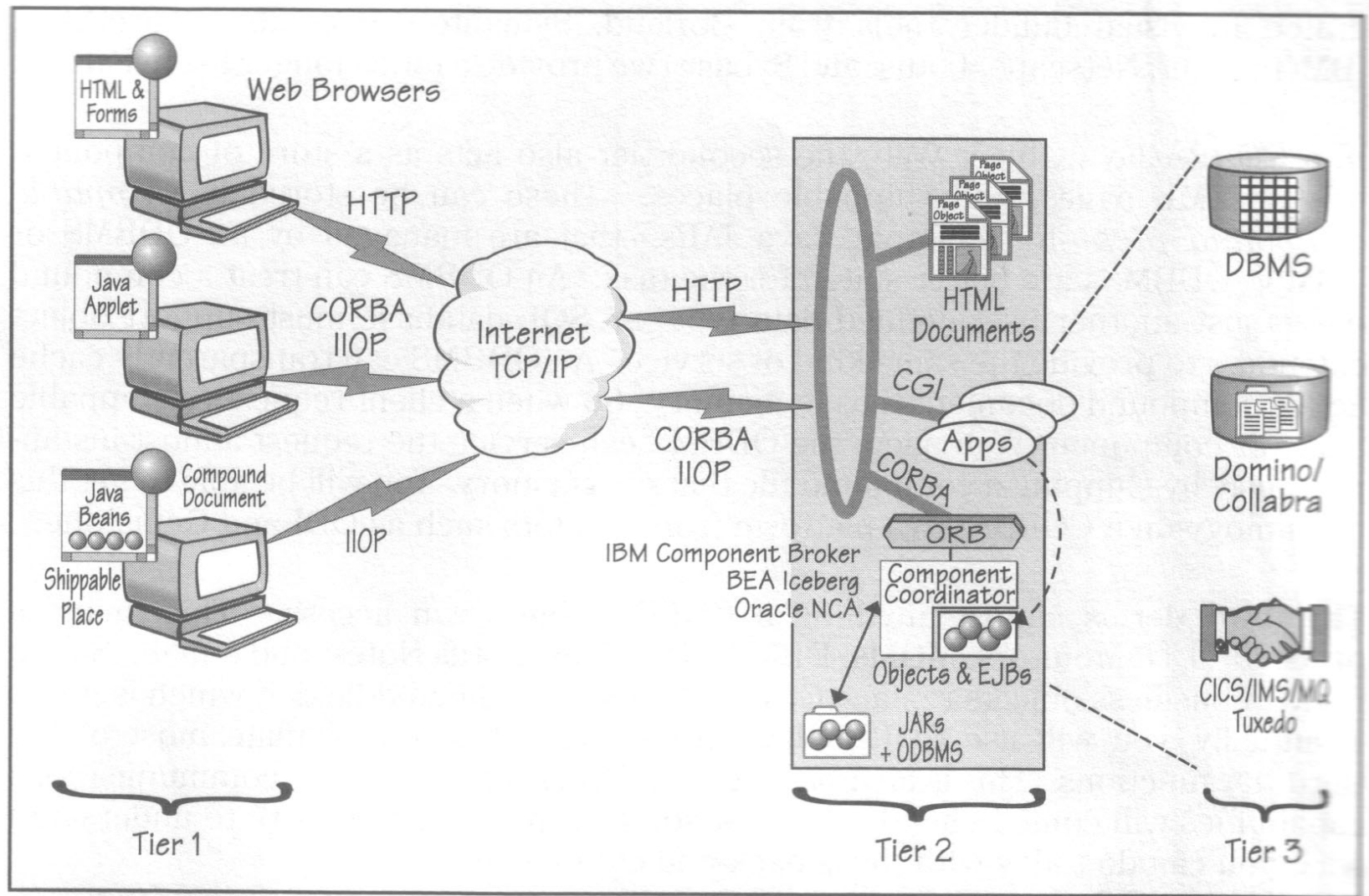
CORBA – 3 Tier architecture



CORBA - 3 Tier web architecture



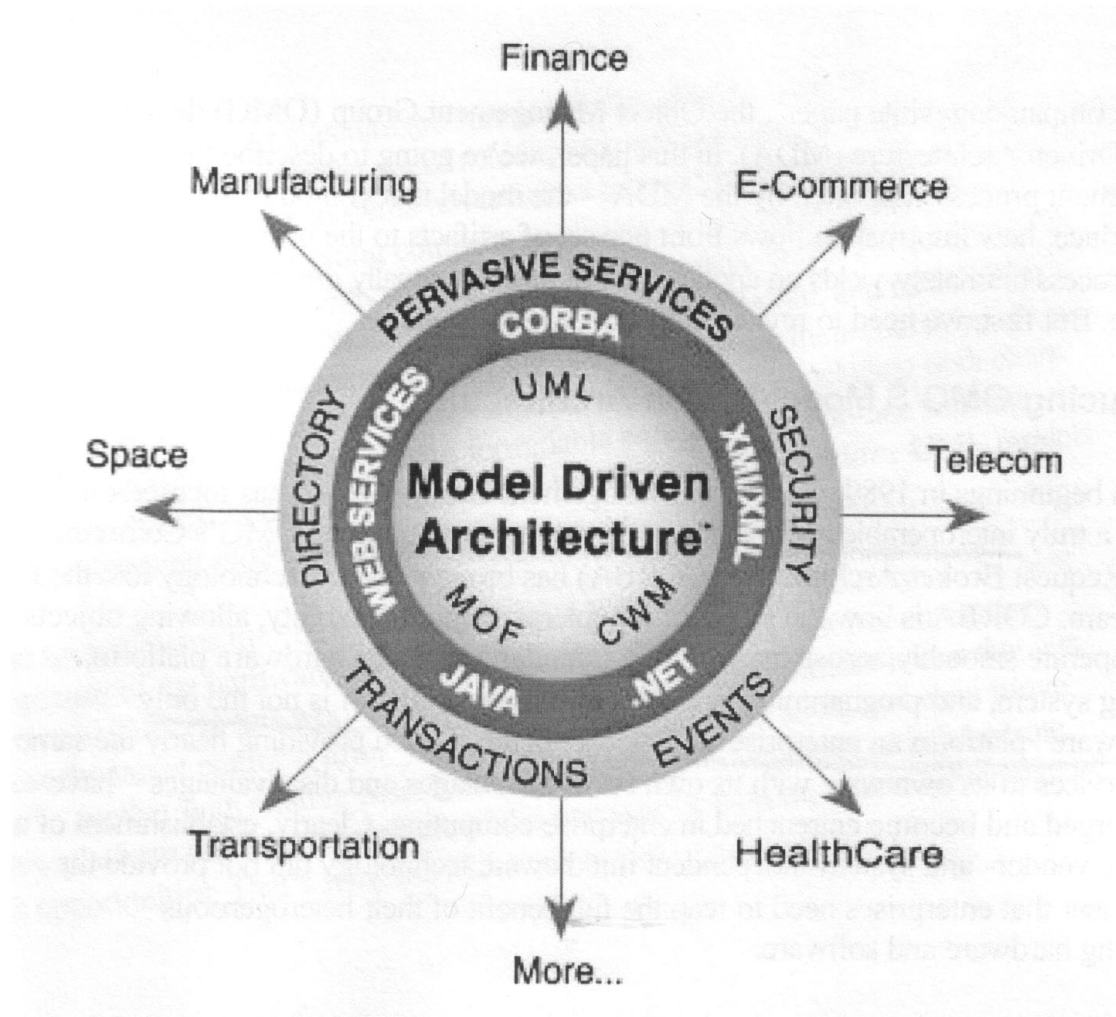
CORBA – 3 Tier ObjectWeb architecture



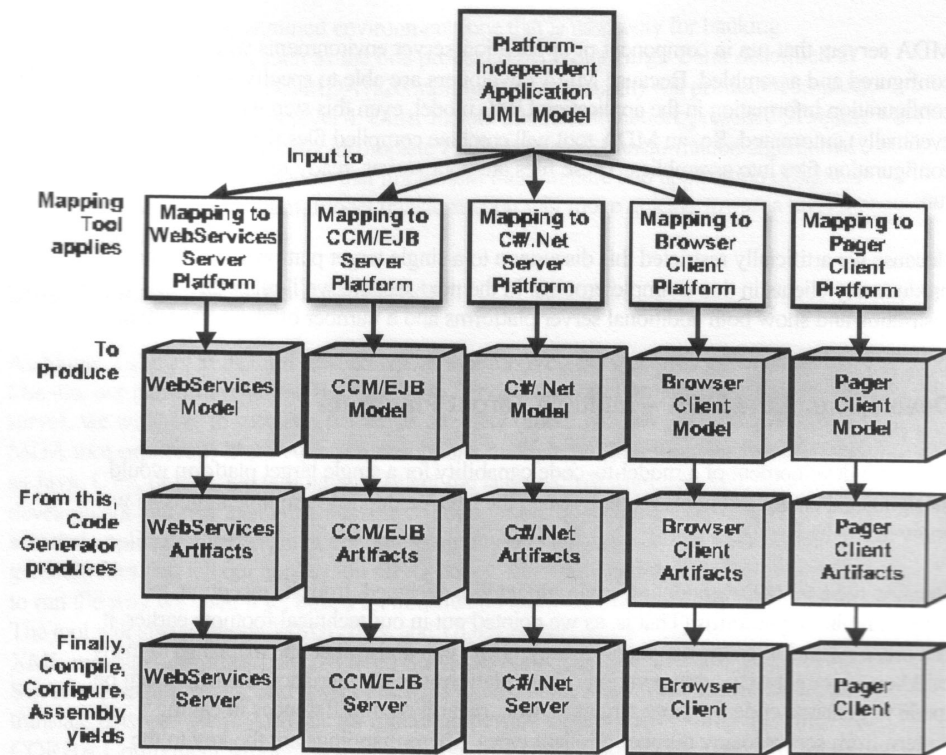
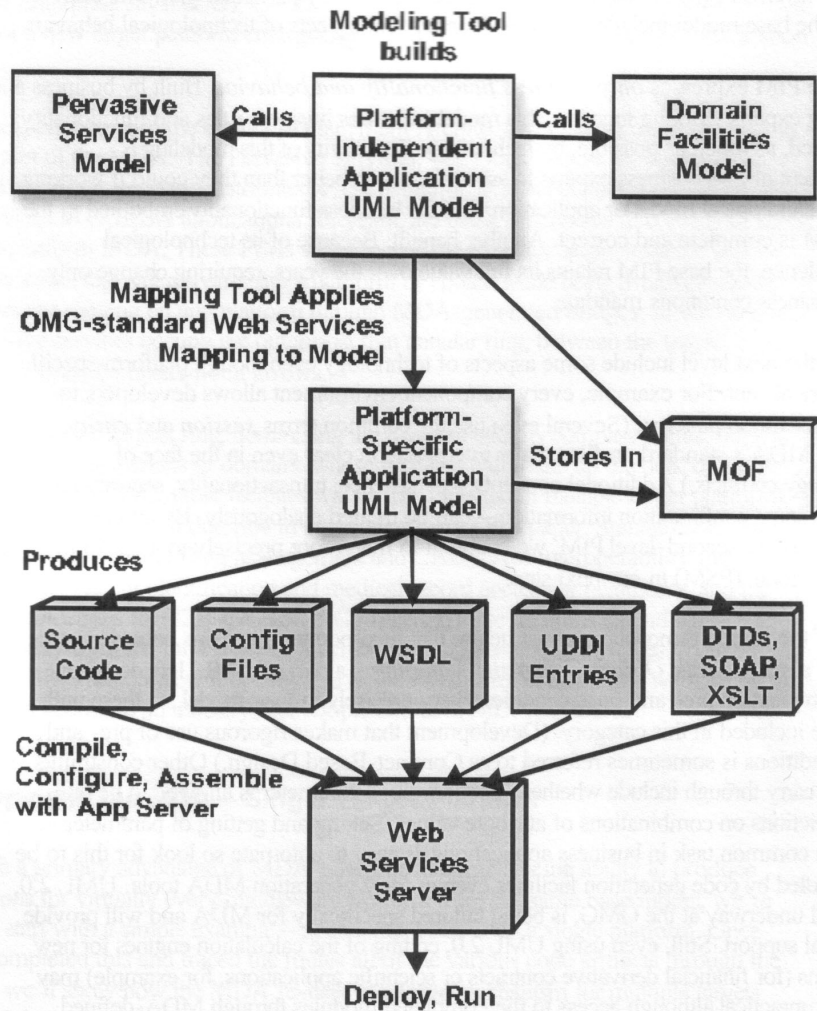
CORBA – latest(?) news

- CCM: CORBA Component Model
 - EJB model and integration
- OMG MDA (Model Driven Architecture)
 - Integration of corporate computing environments
 - Standardisation upward from application implementation to the level of *application design*
 - standards (UML, XML, ...)
 - Semantics

CORBA – MDA overview



CORBA – MDA scenario



CORBA – example (the Count)

- tasting CORBA in Java
 - CORBA (2.3) from Java (JDK 1.3)
 - tool `idlj`
 - mapping IDL to Java
 - stub and skeleton creation
 - infrastructure runtime: `tnameserv`
 - news in JDK 1.4
- naive example: Count
 - Interacting with a distributed count object
 - Classic interface: reset, increment

CORBA – Count IDL

```
// Count.idl
```

```
module examples {
```

```
    interface Count{
```

```
        void reset(in long value);
```

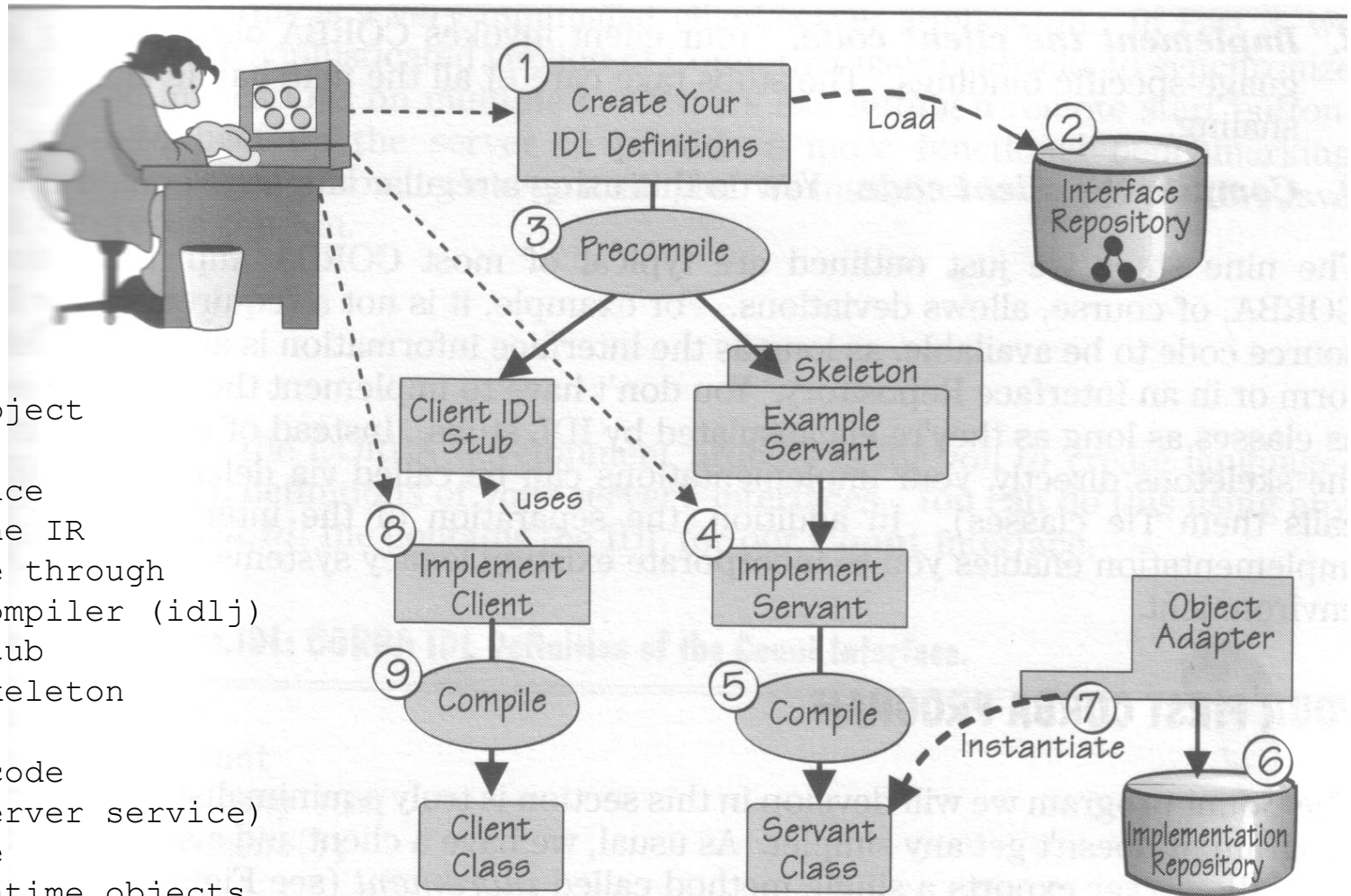
```
        long increment();
```

```
        long getValue();
```

```
    };
```

```
};
```

CORBA – development process (static)



Define server object
interface (IDL)

Bind the interface
definition to the IR

Run the IDL file through
a language precompiler (idlj)
- client stub
- server skeleton

Add the servant
implementation code
(implementing server service)

Compile the code

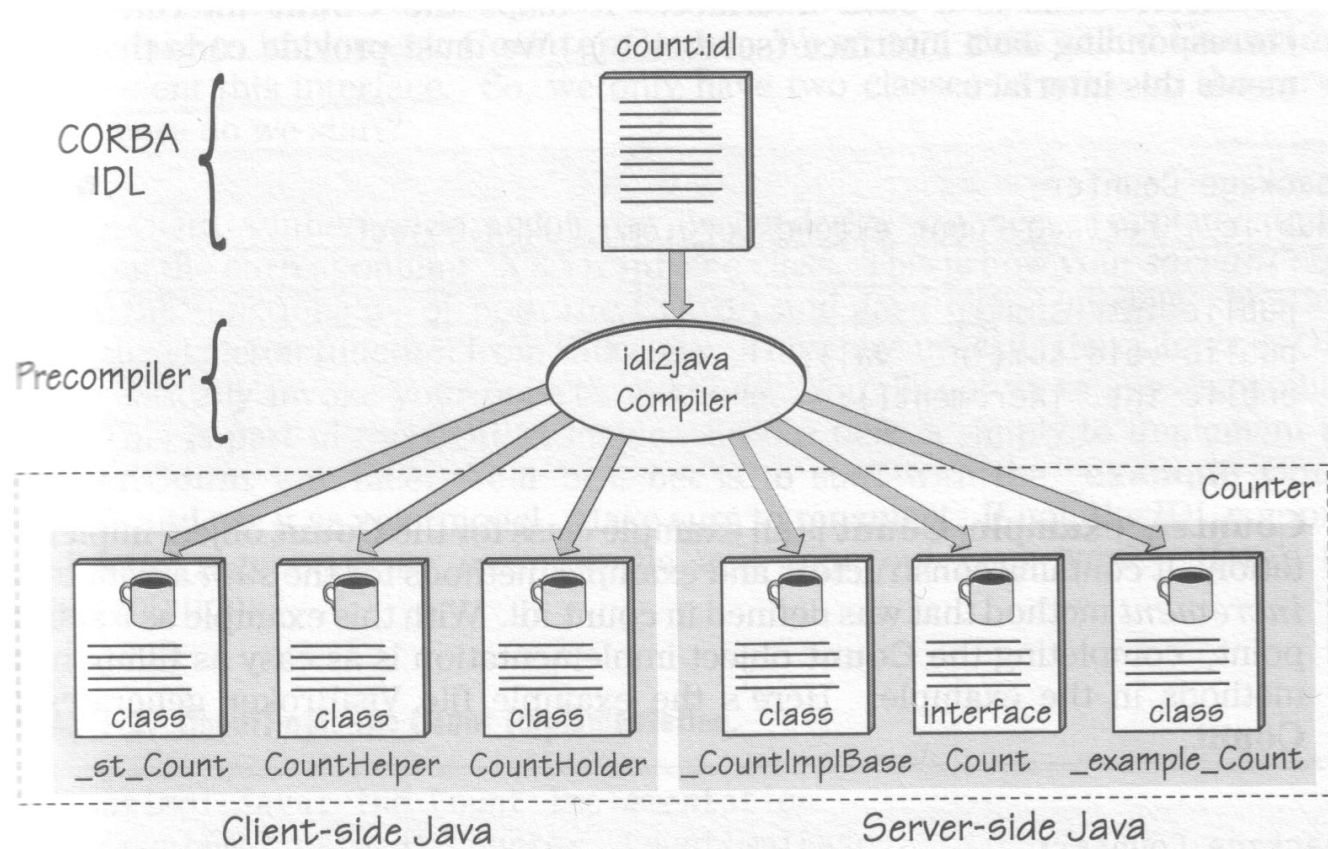
Register the runtime objects
with the Implementation Repository

Instantiate the Objects on the server

Implement the client code

Compile client code

CORBA – example: compiling IDL



CORBA – example: server

```
package examples;

import org.omg.CosNaming.*;

// CountPortableImpl.java: The Count Implementation
class CountImpl extends _CountImplBase {
    private int sum;

    CountImpl() {
        super();
        System.out.println("Count Object Created");
        sum = 0;
    }

    public synchronized int getSum() {
        return sum;
    }
}
```

CORBA – example: server

...

```
public synchronized void reset(int val){  
    sum = val;  
}
```

```
public synchronized int increment(){  
    sum++;  
    return sum;  
}
```

```
}
```

CORBA – example: server

// CountServer.java: The Count Server main program

```
import org.omg.CosNaming.*;
```

```
class CountServer {
```

```
    static public void main(String[] args){
```

```
        try {
```

```
            // Initialize the ORB
```

```
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
```

```
            // Create the Count object
```

```
            examples.CountImpl count = new examples.CountImpl();
```

```
            // Export the newly create object
```

```
            orb.connect(count);
```

```
            // Get a reference to the Naming service
```

```
            org.omg.CORBA.Object nameServiceObj =
```

```
                orb.resolve_initial_references ("NameService");
```

```
            org.omg.CosNaming.NamingContext nameService =
```

```
                org.omg.CosNaming.NamingContextHelper.narrow (nameServiceObj);
```

CORBA – example: server

...

// bind the Count object in the Naming service

```
NameComponent[] countName = {new NameComponent("myCount", "")};
```

```
nameService.rebind(countName, count);
```

// wait forever for current thread to die

```
Thread.currentThread().join();
```

```
} catch(Exception e){
```

```
    System.err.println(e);
```

```
}
```

```
}
```

```
}
```


CORBA – example: client

```
// CountClient.java  Static Client

import org.omg.CosNaming.*;

class CountClient {

    public static void main(String args[]){

        try {

            // Initialize the ORB

            System.out.println("Initializing the ORB");

            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

            // Get a reference to the Naming service

            org.omg.CORBA.Object nameServiceObj =

                orb.resolve_initial_references("NameService");

            org.omg.CosNaming.NamingContext nameService =

                org.omg.CosNaming.NamingContextHelper.narrow(nameServiceObj);

            ...
        }
    }
}
```

CORBA – example: client

...

// resolve the Count object in the Naming service

```
NameComponent[] countName = {new NameComponent("myCount", "")};
```

```
examples.Count counter =
```

```
    examples.CountHelper.narrow(nameService.resolve(countName));
```

// Set sum to initial value of 0

```
System.out.println("Setting sum to 0");
```

```
counter.reset((int)0);
```

// Calculate Start time

```
long startTime = System.currentTimeMillis();
```

// Increment 1000 times

```
System.out.println("Incrementing");
```

```
for (int i = 0 ; i < 1000 ; i++ ){
```

```
    counter.increment();
```

```
}
```

...

CORBA – example: client

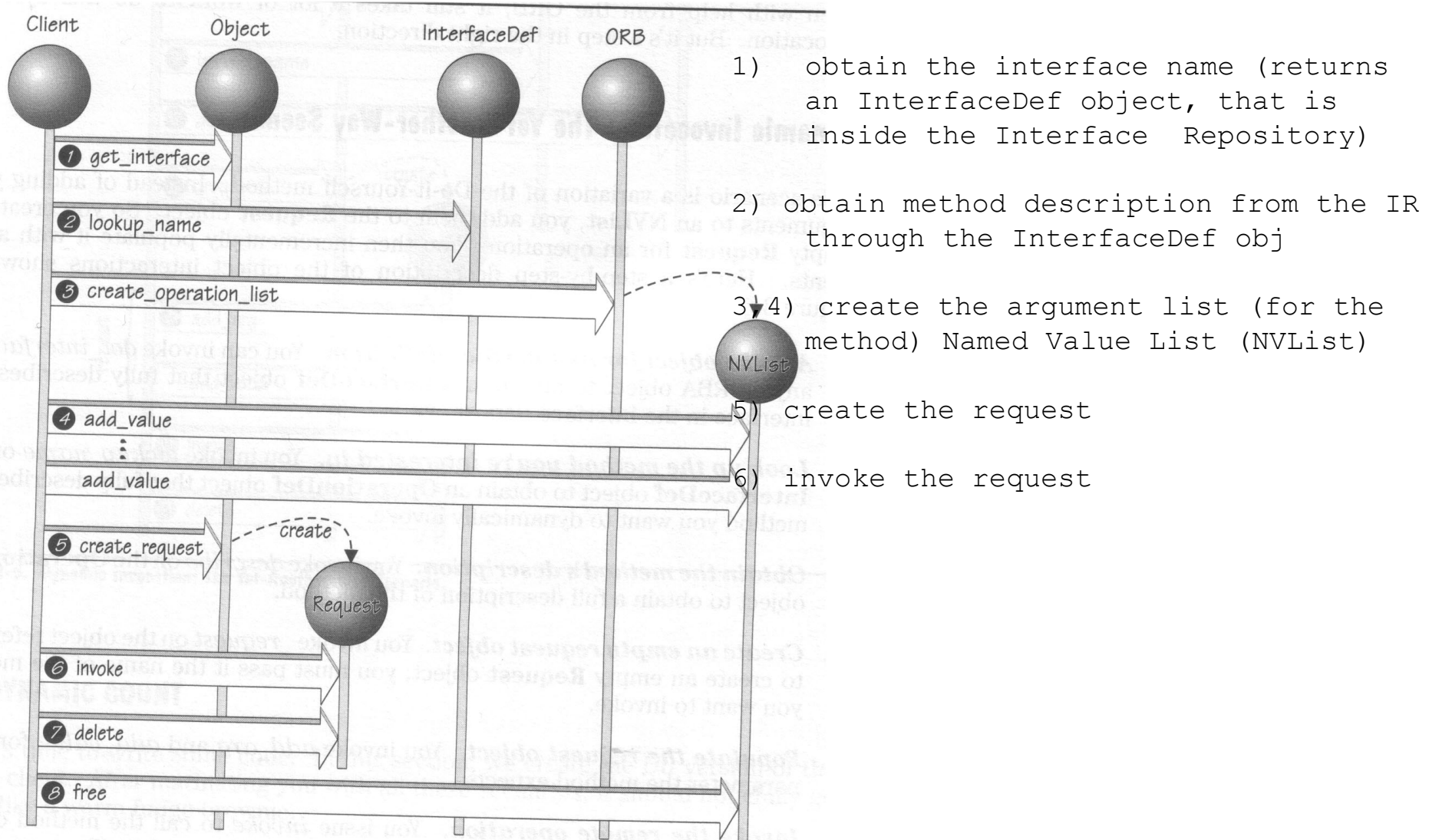
```
...  
// Calculate stop time; print out statistics  
long stopTime = System.currentTimeMillis();  
System.out.println("Avg Ping = "  
                    + ((stopTime - startTime)/1000f) + " msecs");  
System.out.println("Sum = " + counter.sum());  
} catch (Exception e){  
}  
}  
}
```

CORBA – using the DII

- dynamic version: no stubs, no skeletons
- CORBA objects are introspective → given their reference, we can use introspective services to know about interfaces, methods,...

- 1) obtain the interface name
returns an InterfaceDef object, that is inside the Interface Repository
- 2) obtain method description from the IR through the InterfaceDef obj
- 3) create the argument list (for the method)
Named Value List (NVList)
- 4) create the request
- 5) invoke the request

CORBA – using the DII



CORBA – example: access with DII

```
// CountClientDII.java  Dynamic Client
```

```
import org.omg.CosNaming.*;
```

```
class CountClientDII {
```

```
    public static void main(String args[]){
```

```
        boolean loop_all = false;
```

```
        long startTime, stopTime;
```

```
        try {
```

```
            // Initialize the ORB.
```

```
            System.out.println("Initializing the ORB");
```

```
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
```

```
            // Get a reference to the Naming service
```

```
            org.omg.CORBA.Object nameServiceObj =
```

```
                orb.resolve_initial_references("NameService");
```

```
            org.omg.CosNaming.NamingContext nameService =
```

```
                org.omg.CosNaming.NamingContextHelper.narrow(nameServiceObj);
```

CORBA – example: access with DII

```
// resolve the Count object reference
```

```
NameComponent[] countName = {new NameComponent("myCount", "")};
```

```
examples.Count counter =
```

```
    examples.CountHelper.narrow(nameService.resolve(countName));
```

```
//get interface for Count object
```

```
org.omg.CORBA.InterfaceDef CountInterface = counter._get_interface();
```

```
// describe interface for Count object
```

```
org.omg.CORBA.InterfaceDefPackage.FullInterfaceDescription intDesc =
```

```
    CountInterface.describe_interface();
```

```
org.omg.CORBA.Request request = counter._request("increment");
```

...

CORBA – example: access with DII

```
...  
// Calculate Start time  
startTime = System.currentTimeMillis();  
// Increment 1000 times  
for (int i = 0 ; i < 1000 ; i++ ){  
    request.invoke();  
}  
// Calculate stop time; print out statistics  
stopTime = System.currentTimeMillis();  
System.out.println("Avg Ping = "  
                    + ((stopTime - startTime)/1000f)  
                    + " msecs");  
  
System.out.println("Sum = " + counter.sum());  
} catch (Exception e){  
}  
}
```


CORBA – example: results

```
Initializing the ORB
Setting Sum to 0
Incrementing
Avg Ping = 3.625 msec
Sum = 1000
```

Local Static Count Ping (interprocess)		Remote Static Count Ping (interprocess over 10 Mbit/s Ethernet)	
3.7 msec		3.5 msec	
Local Dynamic Count Ping (interprocess, same machine)		Remote Dynamic Count Ping (interprocess, over 10 Mbit/s Ethernet)	
Invoke only		3.9 msec	3.6 msec
Prepare and invoke		61.4 msec	57.8 msec

Usage Pattern	Recommended Invocation Technique
Client invokes server object frequently; server object does not change.	Use static precompiled stubs.
Client invokes server object infrequently.	Use dynamic invocations (DII).
Client discovers server object at run time.	Use dynamic invocations (DII).
Client runs within browser; it discovers new object.	Use downloadable applet and static stubs. Applet becomes the client for this object.

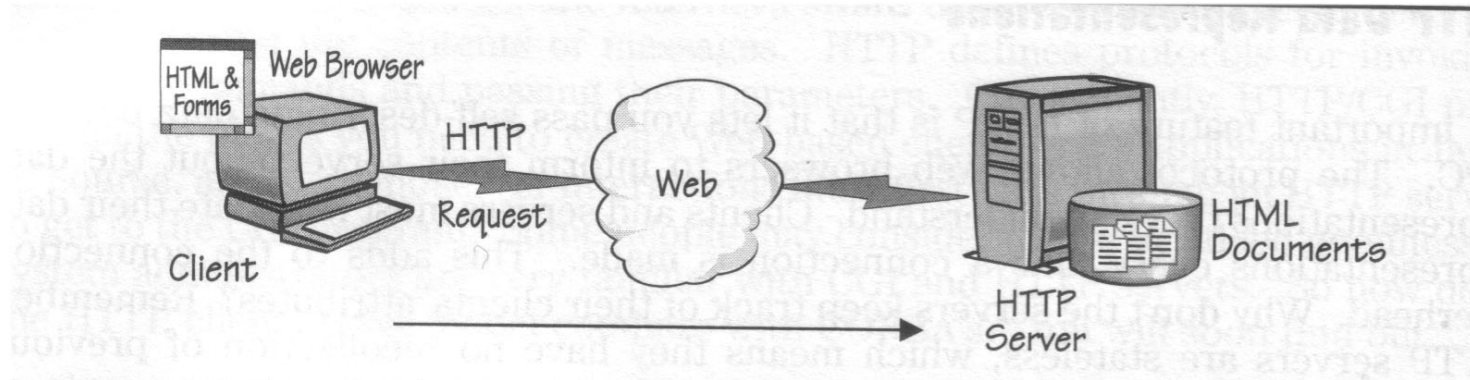
HTTP/CGI - overview

- HTTP = Web's RPC on top of TCP/IP
 - web as a giant URL-based file server
 - focused on documents
- stateless protocol
- new connection for each request

HTTP/CGI – the protocol

- HTTP Request
 - request line (name of the command to be invoked - es: GET)
 - request header fields (additional info, such as RPC arguments)
 - entity body (passing bulk info to the server)
- HTTP Response
 - response header line (info about the response - HTTP versions, status, ...)
 - response header fields (description of doc attributes returned)
 - entity body (doc returned)

HTTP/CGI - request



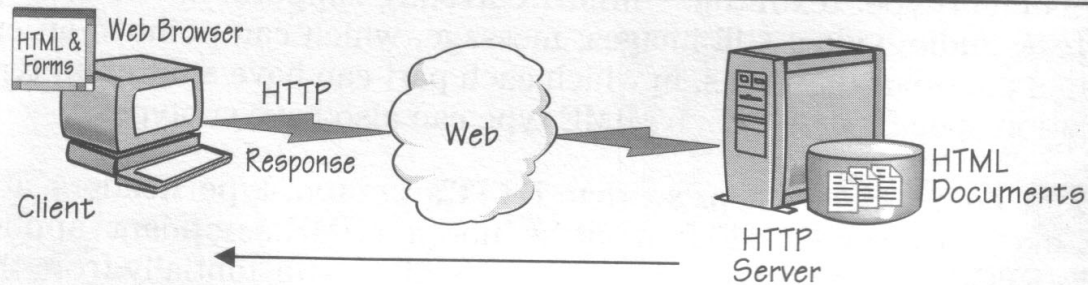
HTTP Request Syntax

```
<method><resource identifier><HTTP version><crlf>      } request line
[<Header> : <value>]<crlf>
:
:                                                         } request
[<Header> : <value>]<crlf>                                } header
blank line      <crlf>                                     } fields
[Entity body]                                           } entity body
```

Example

```
GET /path/file.html HTTP/1.0      } request line
Accept: text/html
Accept: audio/x
User-agent: MacWeb                } request header fields
```

HTTP/CGI - response



HTTP Response Syntax

```
<HTTP Version><result code>[<explanation>]<crlf>  
[<Header> : <value>]<crlf>  
.  
.  
[<Header> : <value>]<crlf>  
blank line      <crlf>  
[Entity body]
```

} response header
} response header fields
} entity body

Example

```
HTTP/1.0 200 OK  
Server: NCSA/1.3  
Mime_version: 1.0  
Content_type: text/html  
Content_length: 2000  
  
<HTML>  
.  
.  
</HTML>
```

} response header
} response header fields
} entity body (i.e., HTML document)

HTTP/CGI – beyond docs

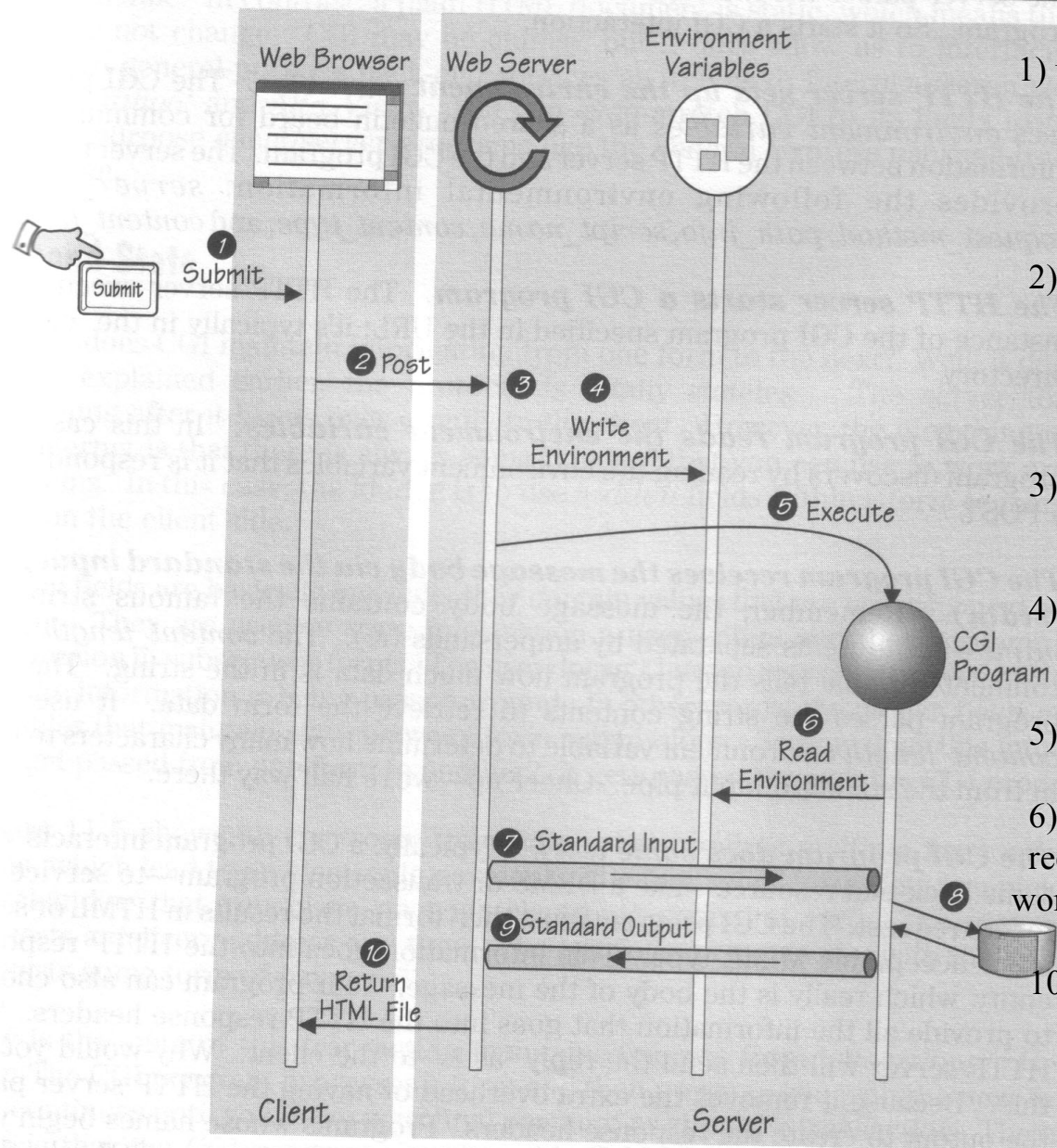
- adding interaction with the server:

Common Gateway Interface protocol

- clients request a program execution
 - web form: HTML page with one or more data entry fields and a mandatory SUBMIT button

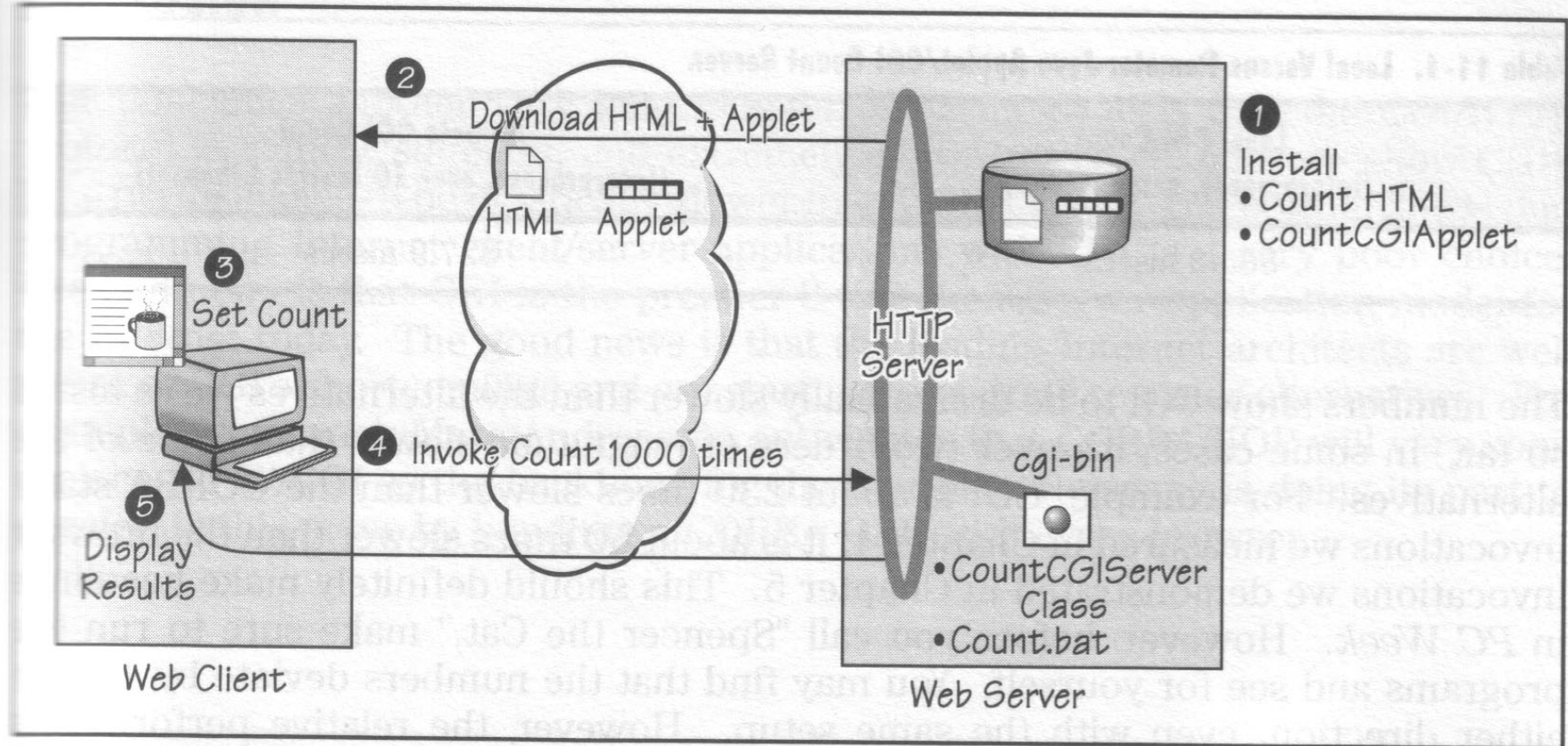
```
<FORM METHOD="POST"
  ACTION="HTTP://www.mylab.org/cgi-bin/sampleform">
```
- the server invokes the programs named in the URL, and using the CGI protocol to pass the method request and its parameters to the back-end program (via stdin)
- the back-end program executes the request and returns the results in HTML format (via stdout) to the Web server
- Client state management: *cookies*

HTTP/CGI - scenario



- 1) user clicks on the form's submit
 - browser collect data within the form, and assemble it into one long string of NAME/VALUE pairs, each separated by an & (spaces inside data are translated into +)
- 2) the browser invoke the POST HTTP method
 - HTTP request
 - POST method
 - entity body: form's data (string name=value&name=value&...)
- 3) HTTP server receives method invocation through a socket connection
 - > POST: start CGI-interaction
- 4) HTTP servers setup environment variables (server_name, request_method, path_info, script_name, content_type...)
- 5) HTTP server starts the CGI program
- 6) CGI program reads the environment variables, 7) receives via STDIN the form's data, 8) does some work, 9) returns the result via STDOUT
- 10) the HTTP sends the results to the Web browser

HTTP/CGI – example (the Count)



HTTP/CGI – example: client code

```
// CountCGIClientApplet.java, Java CGI Applet

import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;

public class CountCGIClientApplet extends java.applet.Applet{
    private TextField countField, pingTimeField;
    private Button runCount;

    public void init(){
        // Create a 2 by 2 grid of widgets.
        setLayout(new GridLayout(2, 2, 10, 10));

        // Add the four widgets, initialize where necessary
        add(new Label("Count"));
        add(countField = new TextField());
        countField.setText("10");
        add(runCount = new Button("Run"));
        add(pingTimeField = new TextField());
        pingTimeField.setEditable(false);
    }
}
```

HTTP/CGI – example: client code

```
public boolean action(Event ev, Object arg)  {
    if(ev.target == runCount) {
        try {
            String sum = "0";

            // get count, initialize start time
            showStatus("Incrementing");
            int count = new Integer(countField.getText()).intValue();
            long startTime = System.currentTimeMillis();

            // Increment count times
            for (int i = 0 ; i < count ; i++ ){
                sum = increment(sum);
            }

            // Calculate stop time; show statistics
            long stopTime = System.currentTimeMillis();
            pingTimeField.setText("Avg Ping = "
                + ((stopTime - startTime) / (float)count) + " msecs");
            showStatus("Sum = " + sum);
        } catch(Exception e){
            showStatus("System Exception" + e);
        }
        return true;
    }
    return false;
}
```

HTTP/CGI – example: client code

```
public String increment(String currentSum){
    String script = "/cgi-bin/Count.bat";
    Socket socket = null;
    String rdata = "";
    String line = "";
    String lastLine = "";

    try {
        socket = new Socket(getCodeBase().getHost(), 80);
        PrintWriter ostream = new PrintWriter(socket.getOutputStream());
        BufferedReader istream = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));

        ostream.print("POST " + script
            + " HTTP/1.0\r\n"
            + "Content-type: text/plain\r\n"
            + "Content-length: "
            + ("increment " + currentSum).length() + "\r\n\r\n");
        ostream.print("increment " + currentSum + "\r\n");
        ostream.flush();

        ...
    }
}
```

HTTP/CGI – example: client code

```
// Read input until input stream is closed by server
try {
    while ((line = istream.readLine()) != null){
        lastLine = line;
    }
}
catch ( SocketException e ){
    istream.close();
    ostream.close();
}
}
catch (Exception e){
    System.out.println("Error " + e);
    if (socket != null)
        try {
            socket.close();
        } catch (IOException ex) {}
}
return lastLine;
}
}
```

HTTP/CGI – example: server code

```
// CountCGIServer.java, Java CGI Server Program

import java.io.*;
import java.util.*;

class CountCGIServer{

    public static void main(String[] args){
        int count;
        String line;

        try {
            // create streams
            PrintWriter ostream = new PrintWriter(System.out);
            BufferedReader istream = new BufferedReader(
                new InputStreamReader(System.in));

            // execute client requests
            line = istream.readLine();
            StringTokenizer tokens = new StringTokenizer(line);
            String myOperation = tokens.nextToken();
            ...
        }
    }
}
```

HTTP/CGI – example: server code

...

// perform increment operation

```
if(myOperation.equals("increment")){  
    String countString = tokens.nextToken();  
    count = Integer.parseInt(countString);  
    count++;  
    ostream.println( Integer.toString(count) );  
    ostream.flush();  
}
```

// close streams

```
istream.close();  
ostream.close();
```

```
}
```

```
catch (Exception e)  
{ System.out.println("Error " + e);  
}
```

```
}
```

```
}
```

HTTP/CGI – example: results

Local CGI Count (interprocess, same machine)	Remote CGI Count (interprocess, over 10 Mbit/s Ethernet)
865.3 msec	827.9 msec

- Pros
 - easy
 - language independent
 - classic languages: C, C++, Java
 - scripting languages: Pearl, Python,
(in particular text-oriented)
 - wide acception
- Cons
 - slow and heavy
 - very low level
 - no idea of interaction engineering:
mechanisms, not systems

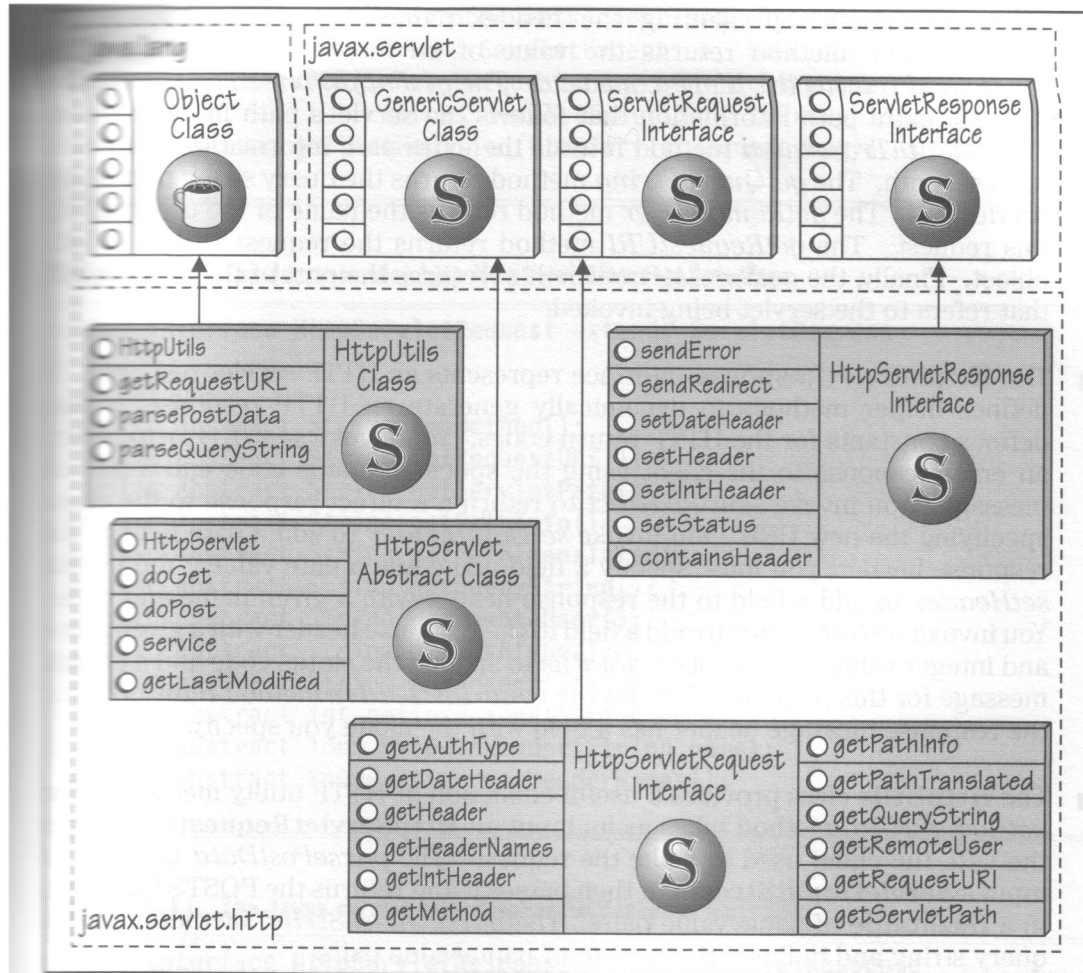
HTTP/CGI – comparisons

Feature	CORBA IIOP	HTTP/CGI
Reliable communications	Yes	Yes
State across invocations	Yes	No
Parameter marshaling	Yes	No
Interface descriptions	Yes (via IDL)	No
Dynamic discovery	Yes (via IR and DII)	No
Parameter data typing	Yes (Language and OS neutral binary format)	No
Performance	Fast (3.6 msec Ping)	Very Slow (827.9 msec Ping)
Security	Yes (via CORBA)	Yes (via SSL or S-HTTP)
Transactions	Yes	No

Servlet - overview

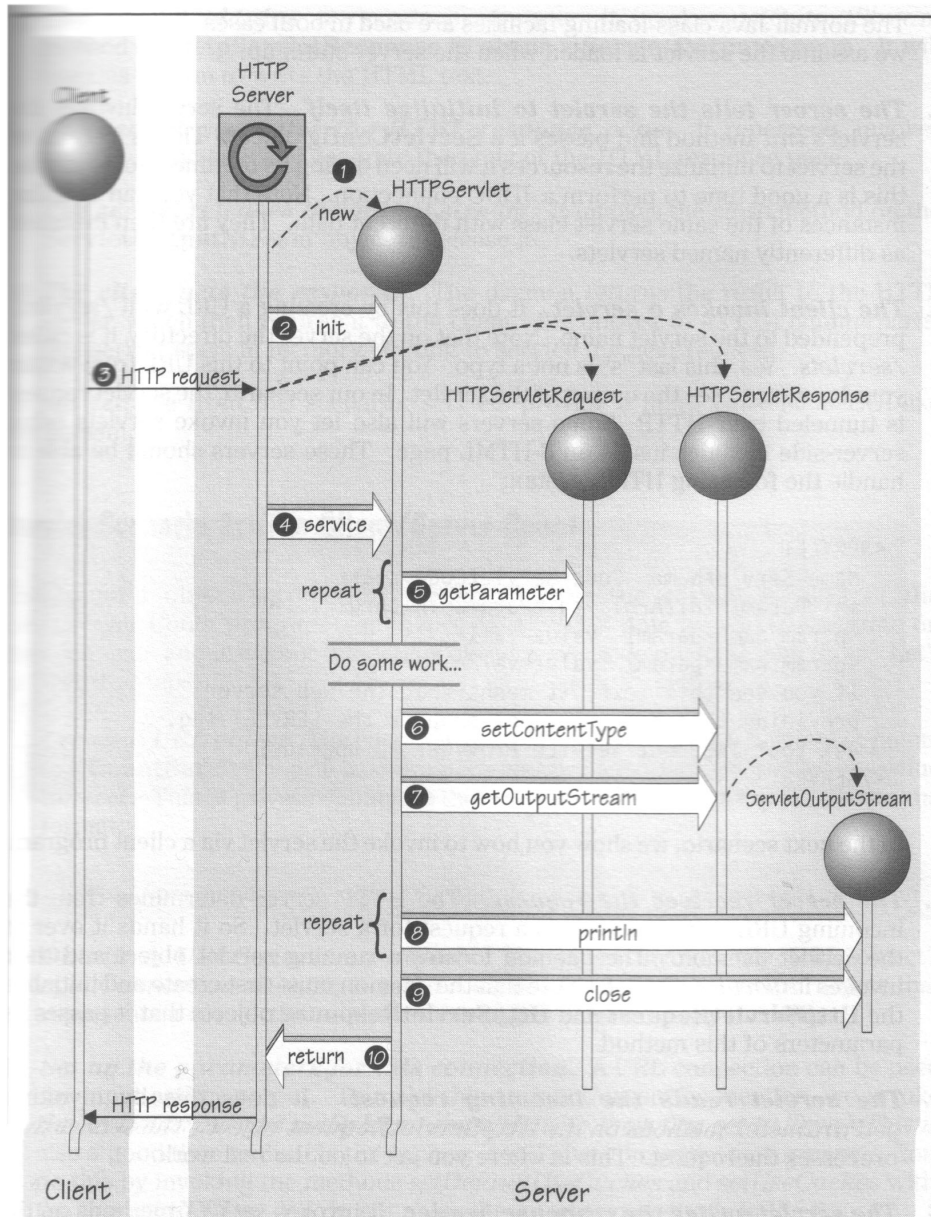
- 1997 JavaSoft shipped Java Web Server:
 - first implementation of a Java-based web server
 - introduces *Servlets* - all-java plug-in architecture for servers
- A servlet is a small piece of Java code that a Web Server loads to handle client requests
 - unlike CGI, servlet code stays resident in memory when request terminates
 - can connect to a DBASE when it is initialised and maintain the connection through more requests
 - can chain a client request to another servlet

Servlet - API



- `javax.servlet.HttpServlet`
 - overriding `init` and `destroy` for lifecycle management
 - overriding `doPost`, `doGet` to handle HTTP POST and GET commands
- `javax.servlet.HttpServletRequest`
 - methods to extract HTTP header info
- `javax.servlet.HttpServletResponse`
 - helper methods to dynamically generate an HTTP response

Servlet - scenario



1) The server creates a new servlet (classes to be placed in servlets directory)

2) The server tells the servlet to initialise itself

3) client invokes a sevlet: URL /servlet/<name>

<SERVLET

Name=ServletName Code=ServletCode.class

initParam1=initArg1 initParam2=initArg2 ...>

<param name=param1 value=val1>

<param name=param2 value=val2>

....

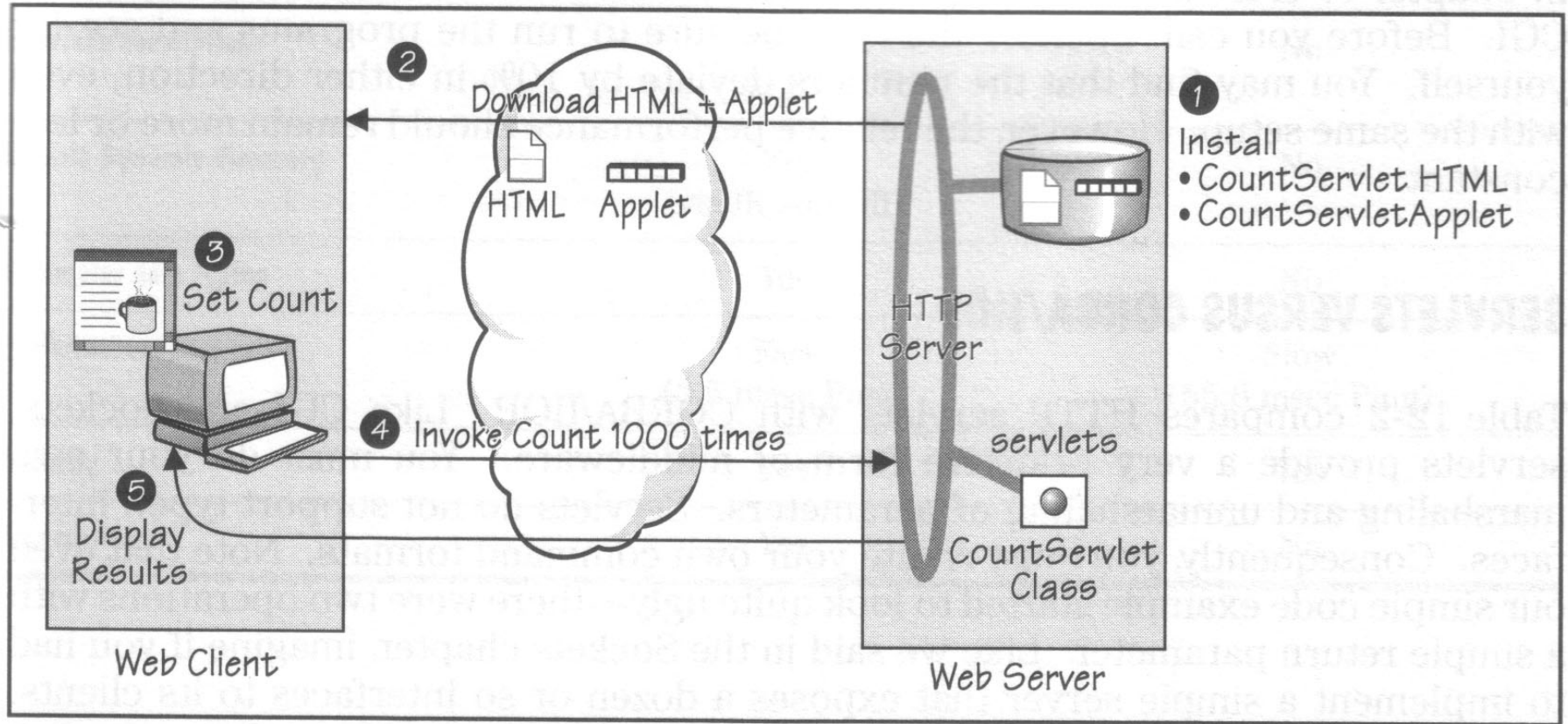
</SERVLET>

4) The server receives the request:, it locates a running servlet object and invokes it SERVICE method

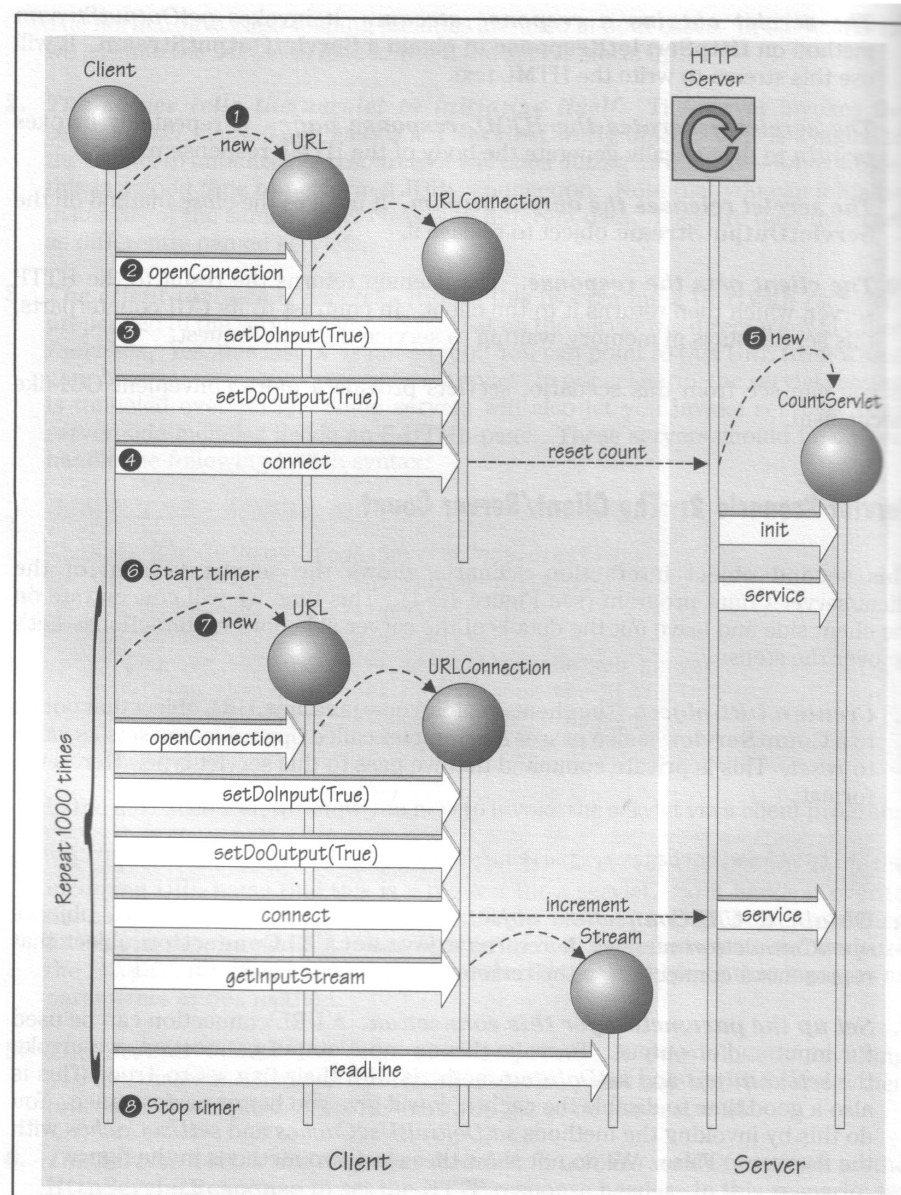
5,6) The servlet reads the incoming request, 6) writes the response header, 7) obtains a response stream, 8) generates the HTML response page (by means of println), 9) releases the output stream (by means of close)

10) The client gets the response

Servlet – example (the Count)



Servlet – example (the Count)



Servlet – example: client

```
// CountServletApplet.java, Java Servlet Applet

import java.awt.*;
import java.io.*;
import java.net.*;

public class CountServletClient{

    public static void main(String args[]){
        URLConnection connect;
        String sum = "0";

        try {
            System.out.println("Resetting count");
            connect = (new URL("http://137.204.191.188" +
                "/servlet/CountServlet?operation=reset")).openConnection();
            connect.setDefaultUseCaches(false); //for future connections
            connect.setUseCaches(false); //for this one
            connect.setDoInput(true);
            connect.setDoOutput(true);
            connect.connect();
            DataInputStream in = new DataInputStream(new BufferedInputStream(
                                                                    connect.getInputStream()));
            String response = in.readLine();
            in.close();

            // get count, initialize start time
            System.out.println("Incrementing");
            long startTime = System.currentTimeMillis();
```

Servlet – example: client

```
...
// Increment count times
for (int i = 0 ; i < 1000; i++ ){
    connect = (new URL("http://137.204.191.188" +
        "/servlet/CountServlet?operation=increment")).openConnection();
    connect.setDefaultUseCaches(false); //for future connections
    connect.setUseCaches(false); //for this one
    connect.setDoInput(true);
    connect.setDoOutput(true);
    connect.connect();

    in = new DataInputStream(new BufferedInputStream(
                                connect.getInputStream()));

    sum = in.readLine();
    in.close();
}

// Calculate stop time; show statistics
long stopTime = System.currentTimeMillis();
System.out.println("Avg Ping = "
    + ((stopTime - startTime) / (float)1000) + " msecs");
System.out.println("Sum = " + sum);
} catch(Exception e){
    System.out.println("System Exception " + e);
}
}
}
```

Servlet – example: server

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CountServlet extends HttpServlet
{
    int count;

    public void init(ServletConfig conf) throws ServletException{
        super.init(conf);
        count = 0;
    }

    ...
}
```


Servlet – example: server

```
...
public void service(HttpServletRequest req, HttpServletResponse res)
                                throws IOException{
    try {
        String myOperation;
        myOperation = req.getParameter("operation");

        // perform increment operation
        if (myOperation.equals("increment")){
            count++;
            ServletOutputStream out = res.getOutputStream();
            res.setContentType("text/html"); //Required for HTTP
            out.println("" + count);
            out.close(); //Close the output stream
        } else {
            if (myOperation.equals("reset")){
                count=0;
                ServletOutputStream out = res.getOutputStream();
                res.setContentType("text/html"); //Required for HTTP
                out.println("" + count);
                out.close(); //Close the output stream
            }
        }
    } catch (Exception e){
        System.out.println("Error " + e);
    }
}
```

Servlet – results

TABLE 12.11 Servlet versus Remote Count Servers.

	Local Servlet Count (interprocess, same machine)	Remote Servlet Count (interprocess, over 10 Mbit/s Ethernet)
Applet-to-Servlet	103.9 msec	92.9 msec
Application-to-Servlet	61.7 msec	55.6 msec

- Pros
 - faster and lighter than CGI
 - client state management easier than CGI
 - easy deployable
 - wide acception
- Cons
 - language dependend (Java)
 - no idea of interaction engineering:
mechanisms, not systems

Servlet – comparisons

cannot not relate HTTP/Servlets.

Feature	CORBA/IIOP	HTTP/Servlets
Reliable communications	Yes	Yes
State across invocations	Yes	Yes (with great difficulty)
Parameter marshaling	Yes	No
Interface descriptions	Yes (via IDL)	No
Remote Dynamic discovery	Yes (via IR and DII)	No
Parameter data typing	Yes	No
Performance	Fast (3.5 msec Ping)	Slow (55.6 msec Ping)
Security	Yes	Yes
Transactions	Yes	No

Servlet – availability and links

- Tomcat web server (Apache project)
 - open source: <http://jakarta.apache.org/tomcat/>
- Java Server Pages (JSP) technology
 - Built on top of Servlets
 - Goal: to simplify the creation and management of dynamic web pages
 - combining HTML code of a web page with pieces of Java code, in the same document
 - benefit: maintain a single document that represents both the page and the Java code that enables it

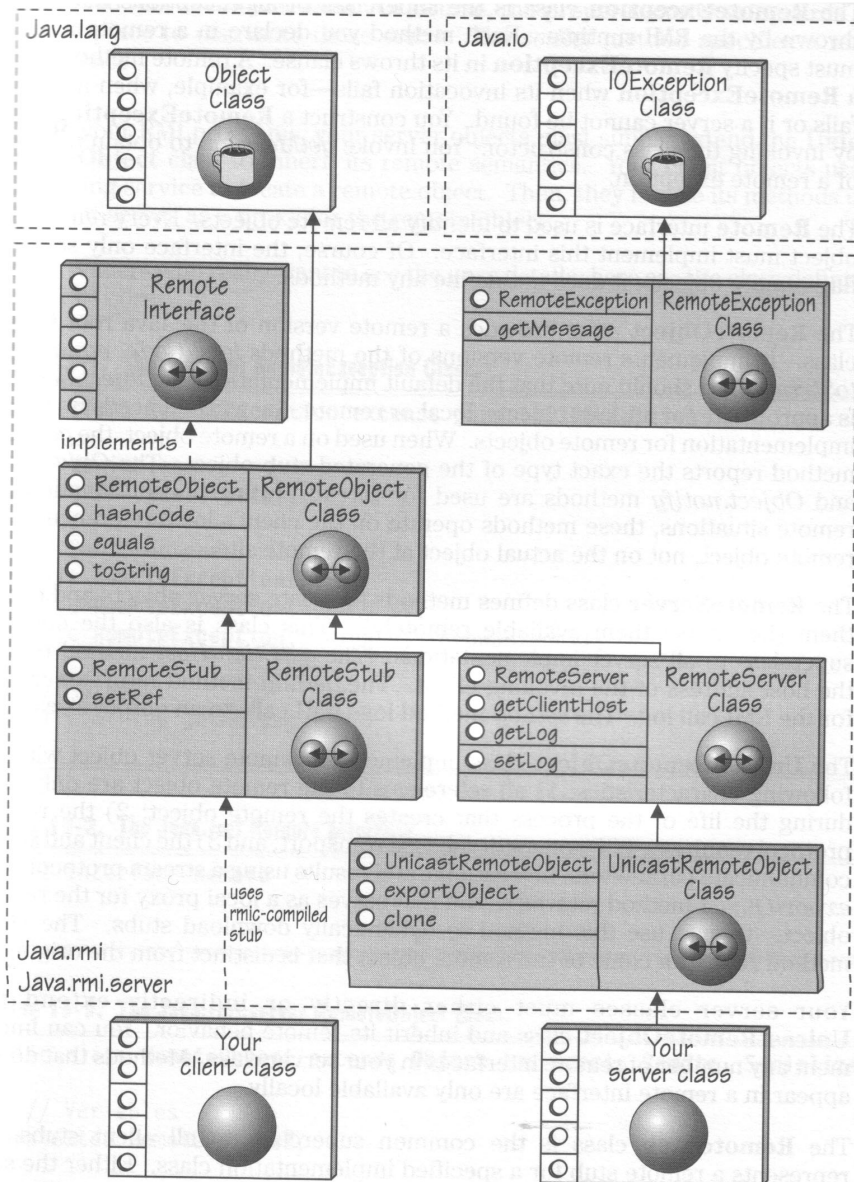
RMI - overview

- Designed to seamlessly support remote method invocations across JVMs
 - from JDK 1.1
 - core (low-level) infrastructure of all major Java distributed architectures
- Integration of a distributed object model in the language
 - Actually not fully transparent
 - Remote Exceptions
 - Different semantics for argument passing
 - objects by-value, using Serialization
 - reference to remote objects
 - Distributed Garbage collector

RMI - overview

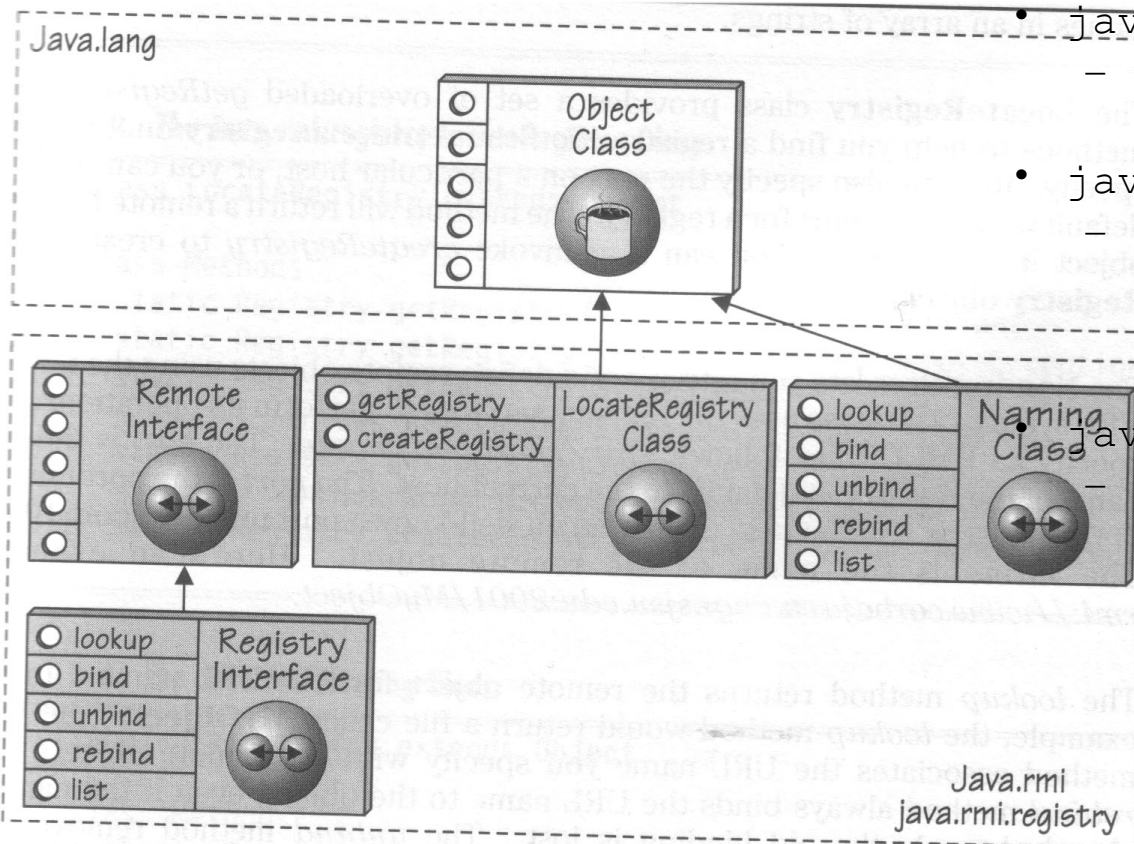
- API: 4 functional categories
 - RMI Core
 - RMI Naming Service
 - RMI Security Exception
 - (RMI marshaling)
- Runtime
 - **rmiregistry** (infrastructure services)
- Tools
 - **rmic** to generate skels and stubs

RMI – Core API



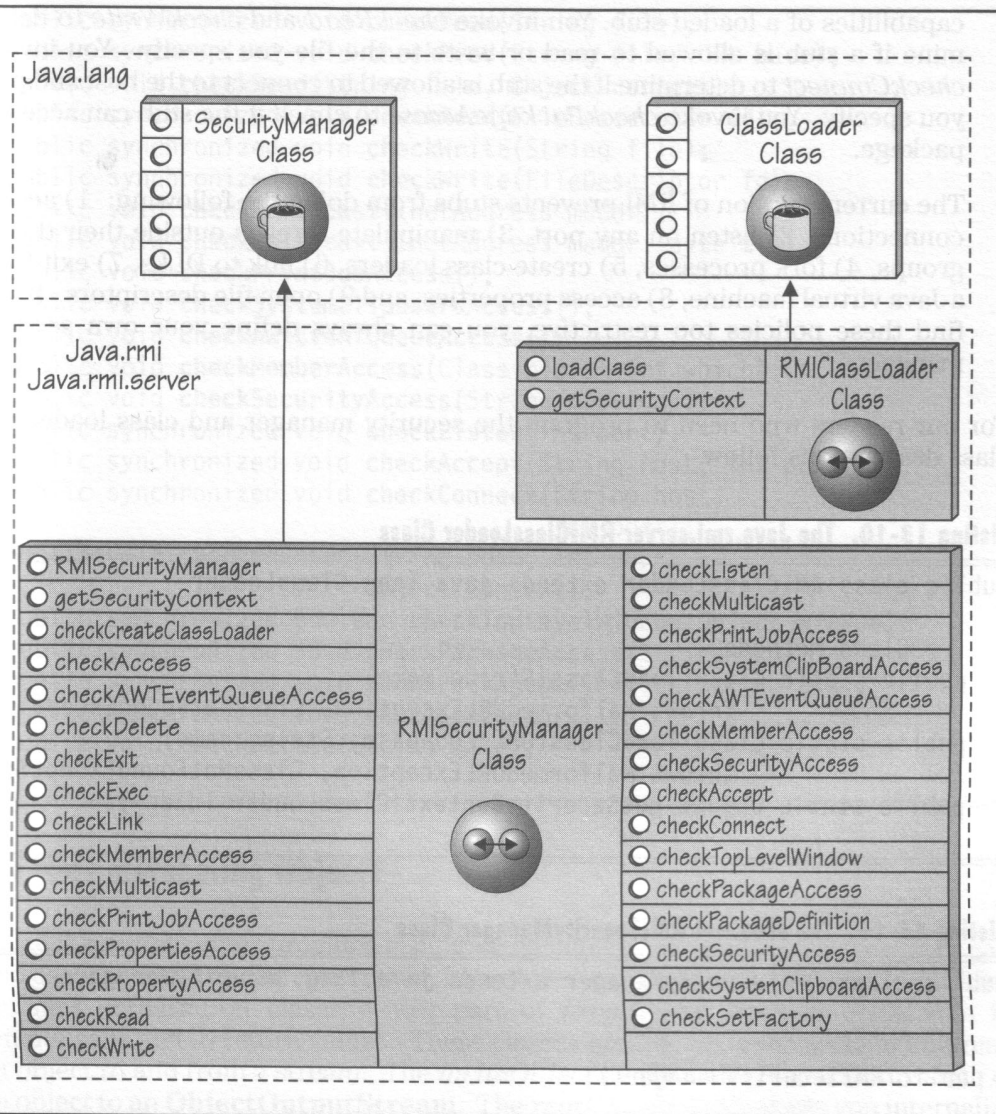
- `java.rmi.Remote` interface
 - flag interface
- `java.rmi.RemoteException` class
 - superclass of all RMI exceptions
 - thrown by every method of a distributed object
- `java.rmi.UnicastRemoteObject`
 - base class for our remote objects
- `java.rmi.RemoteStub` classes
 - client stubs, generated by `rmic` tools

RMI – Naming Service API



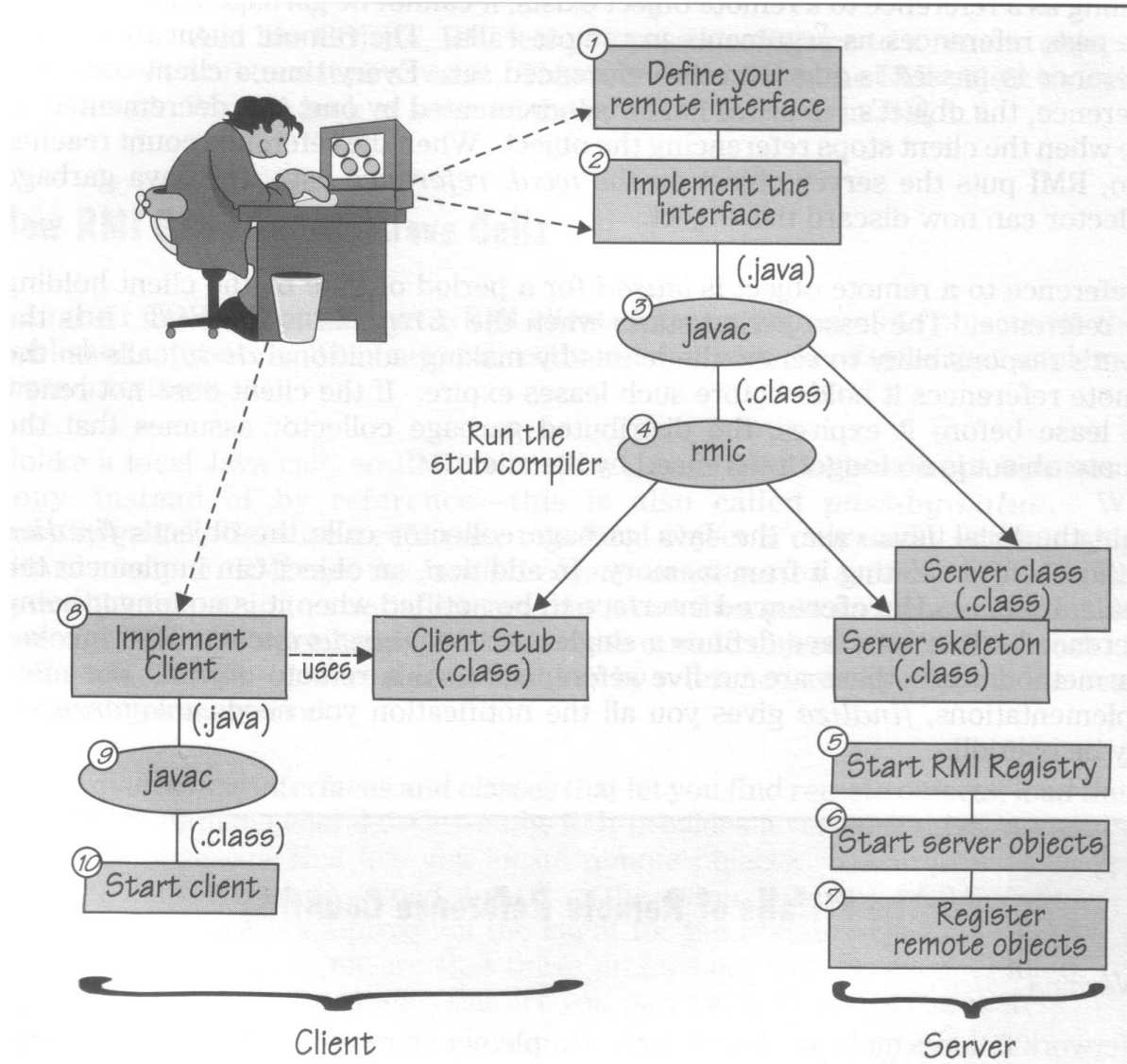
- `java.rmi.Registry` interface
 - define methods to add/remove/change bindings
- `java.rmi.LocateRegistry` class
 - static services to get a Registry object reference
- `java.rmi.Naming` class
 - Define methods to retrieve and define remote objects using the URL syntax:
`rmi://host:port/name`

RMI – Security API

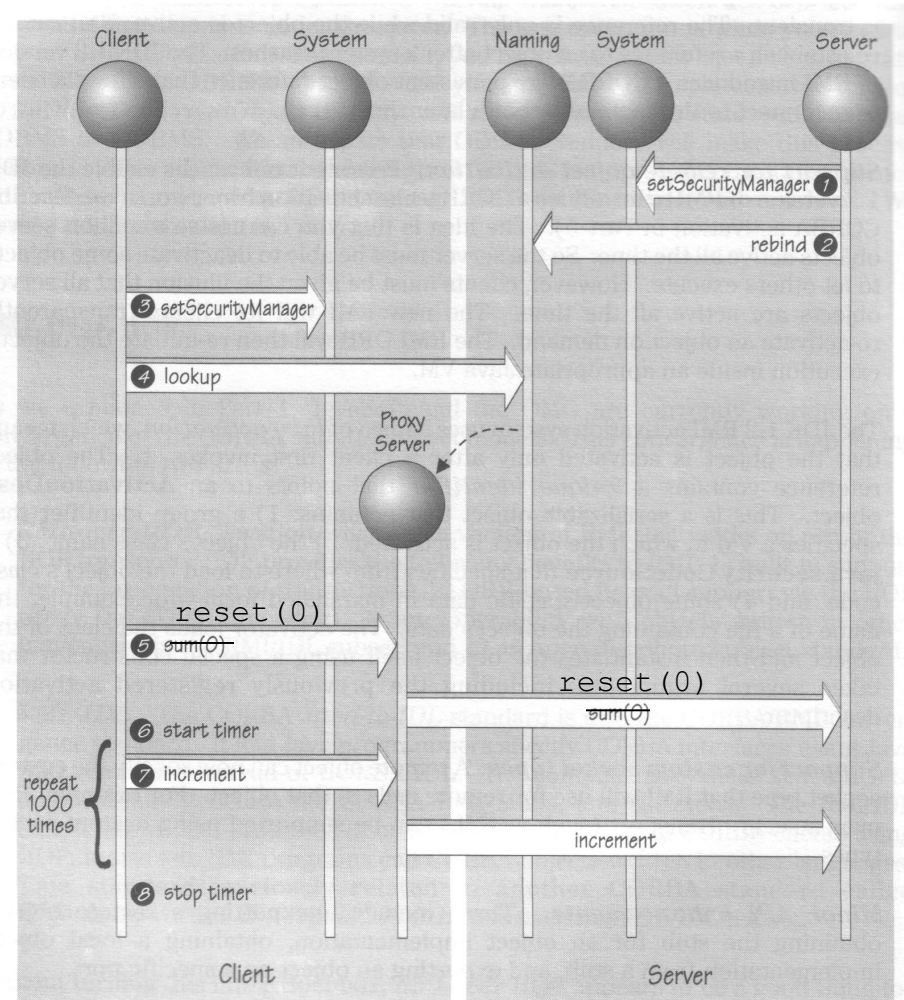
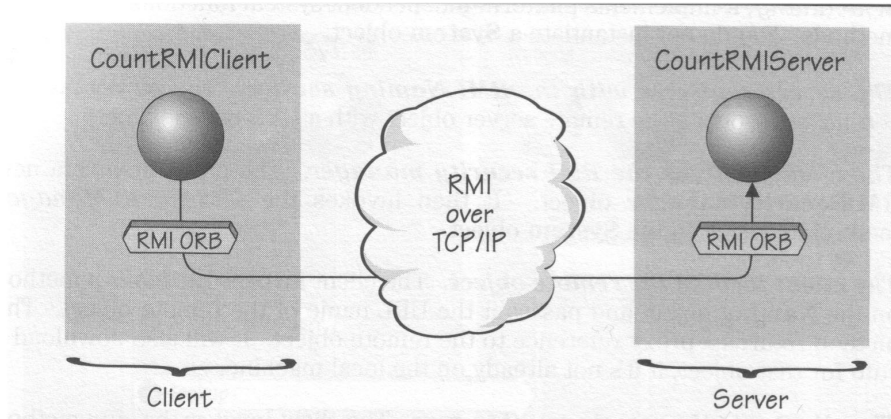


- `java.rmi.RMISecurityManager`
 - responsible for the actions of remotely loaded stubs and classes
 - If no security manager defined → RMI will load stub classes only from local file system
- `java.rmi.RMIClassLoader`
 - used by the RMI runtime to load stubs and skeletons via a URL
 - classes downloaded only if not available locally

RMI – development process



RMI - example (the Count)



RMI – example: CountRMI interface

```
// CountRmi Interface
```

```
public interface CountRMI extends java.rmi.Remote {  
  
    // resets the count  
    void reset() throws java.rmi.RemoteException;  
  
    // increments the count  
    public int increment() throws java.rmi.RemoteException;  
  
    // gets the count value  
    int getValue() throws java.rmi.RemoteException;  
  
}
```

RMI – example: server

```
// CountRMIIImpl.java, CountRMI implementation
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class CountRMIIImpl extends UnicastRemoteObject implements CountRMI {
    private int sum;

    public CountRMIIImpl(String name) throws RemoteException {
        super();
        try {
            // binding name to this object
            java.rmi.Naming.rebind(name, this);
            sum = 0;
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
            e.printStackTrace();
        }
    }

    public int getValue() throws RemoteException {
        return sum;
    }

    public void reset(int val) throws RemoteException {
        sum = val;
    }

    public int increment() throws RemoteException {
        sum++;
        return sum;
    }
}
```

to generate the stub and
the skeleton:

```
rmic CountRMIIImpl
```

RMI – example: server

```
// CountRMIServer.java

import java.rmi.*;
import java.rmi.server.*;

public class CountRMIServer {

    public static void main(String args[]){

        // Create and install the security manager
        System.setSecurityManager(new RMISecurityManager());

        try {
            // Create CountRMIImp1
            CountRMIImp1 myCount = new CountRMIImp1("my CountRMI");

            System.out.println("CountRMI Server ready.");
        } catch (Exception e){
            System.out.println("Exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

RMI – example: client

```
// CountRMIClient.java  RMI Count client

import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;

public class CountRMIClient {
    public static void main(String args[]){
        // Create and install the security manager
        System.setSecurityManager(new RMISecurityManager());

        try {
            CountRMI myCount = (CountRMI) java.rmi.Naming.lookup("rmi://"
                + args[0] + "/" + "my CountRMI");

            // Set Sum to initial value of 0
            System.out.println("Setting Sum to 0");
            myCount.reset(0);

            // Calculate Start time
            long startTime = System.currentTimeMillis();

            // Increment 1000 times
            System.out.println("Incrementing");
            for (int i = 0 ; i < 1000 ; i++ ){
                myCount.increment();
            }...
```

RMI – example: client

...

```
// Calculate stop time; print out statistics
long stopTime = System.currentTimeMillis();
System.out.println("Avg Ping = "
                   + ((stopTime - startTime)/1000f)
                   + " msecs");

System.out.println("Sum = " + myCount.sum());

} catch (Exception e) {
    System.err.println("System Exception" + e);
}
System.exit(0);
}
```


RMI – example: results

Local RMI Count (interprocess, same machine)	Remote RMI Count (interprocess, over 10 Mbit/s Ethernet)
3.1 msec	3.3 msec

- Pros
 - faster and lighter than CGI/Servlets
 - interaction engineering: OO model on the net
 - easy deployment
- Cons
 - security issues using RMI with applets/browsers...
 - language dependend (Java)
- *The* Java low-level middleware to built other high-level infrastructures (ex: JINI).

RMI – some hot news

- RMI-Over-IIOP:
 - enables distributed Web-enabled Java applications to transparently, invoke operations on remote network services using the industry standard IIOP
 - programming to the RMI interfaces, BUT use CORBA IIOP as the underlying transport
 - no need to write CORBA IDL
- Integration with the JNDI Name Service

RMI - comparisons

Feature	RMI-Over-IIOP	RMI over RMP
Parameter marshaling	Yes	Yes
Parameter passing	in, out, and in/out	in
Dynamic stub downloads	Yes (via codebase)	Yes
Dynamic class downloads	Maybe	Yes
Objects can be passed by value	Yes (via Pass-by-Value)	Yes
Garbage collection	Yes (via POA and ORB)	Yes (via language)
Interface descriptions	Yes (via Java interfaces)	Yes (via Java interfaces)
Distributed Dynamic discovery	Yes (via IR)	No
Distributed Dynamic invocations	Yes (via DII)	No

RMI - comparisons

Feature	RMI-Over-IIOP	RMI over RMP
Wire-level security context	Yes (via CORBA Security)	No
SSL-based Security	Yes	Yes (with JDK 1.2)
Wire-level transaction context	Yes (via CORBA OTS)	No
Persistent naming	Yes	No
URL-based naming	Yes (non-volatile)	Yes (volatile)
Persistent object references	Yes	Yes (with JDK 1.2)
Multilingual invocations	Yes (with RMI/IDL subset)	No
Language-neutral wire protocol	Yes (via canonical IIOP data types)	No
Intergalactic scaling	Yes (via IORs, DSI, GUIDs, IIOP, and inter-ORB federations)	No
Standard ORB wire protocol	Yes (IIOP)	No
Enterprise JavaBeans Support	Yes	Unknown

Client/Server middleware – comparisons

Feature	CORBA/IIOP	DCOM	RMI/RMP	HTTP/CGI	Servlets	Sockets
Abstraction level	★★★★	★★★★	★★★★	★★	★★	★
Seamless Java integration	★★★★	★★★★	★★★★	★★	★★	★★
OS platform support	★★★★	★★	★★★★	★★★★	★★★★	★★★★
All-Java implementation	★★★★	★	★★★★	★★	★★★★	★★★★
Typed parameter support	★★★★	★★★★	★★★★	★	★	★
Ease of configuration	★★★	★★★	★★★	★★★	★★★	★★★
Distributed method invocations	★★★★	★★★	★★★	☆	☆	☆
State across invocations	★★★★	★★★	★★★	☆	★★	★★
Dynamic discovery and metadata support	★★★★	★★★	★★	☆	★	☆
Dynamic invocations	★★★★	★★★★	★	☆	☆	☆
Performance (remote Pings)	★★★ 3.5 msecs	★★★ 3.8 msecs	★★★ 3.3 msecs	☆ 827.9 msecs	★ 55.6 msecs	★★★★ 2.1 msecs
Wire-level security	★★★★	★★★★	★★★	★★★	★★★	★★★
Wire-level transactions	★★★★	★★★	☆	☆	☆	☆
Persistent object references	★★★★	★	☆	☆	☆	☆
URL-based naming	★★★★	★★	★★	★★★★	★★★★	★★★
Multilingual object invocations	★★★★	★★★★	☆	★★★	☆	★★★★
Language-neutral wire protocol	★★★★	★★★★	☆	★★★★	★★★★	☆
Intergalactic scaling	★★★★	★★	★	★★	★★	★★★★
Open standard	★★★★	★★	★★	★★★★	★★	★★★★

seminar overview

- Interaction: paradigm shift
 - some perspectives on the impact
- Client/server middleware (overview and comparisons)
 - CORBA
 - HTTP/CGI
 - HTTP/Servlets
 - RMI
- Service Oriented Architectures/Infrastructures
 - JINI
 - JavaSpaces Service
 - WebServices
 - Peer-to-Peer Infrastructure
 - JXTA

Service Oriented Architectures

3 main entities

- *consumers* of the services
- *providers* of the services
- *registrars* of the services

*These entities work in concert to
provide LOOSELY COUPLED
computing paradigm*

Service Oriented Architectures

- JINI
 - example of network-centric computing infrastructure
 - pervasive/ubiquitous computing
- WebServices
 - enterprise computing
- Peer-to-Peer infrastructures

JINI – the vision

- embedding computing
 - disappearing computers
 - computers everywhere
 - the network is the computer
 - from computation device to communicating device
 - from disk-centric to network-centric computing → affecting how we organise and deploy software
- ➔ rethinking computer architecture, given the rising importance of the network and the proliferation of processors (without disk)

JINI – the vision

distributed systems as *federation of services*

- ***Service*** = anything reachable on the network and ready to perform a useful function (hardware, communication channels, humans)
- ***Federation*** = set of services on the network that a client (program, service, user) can use to accomplish a goal
 - no central controlling authority
 - Instead of a central authority, Jini's run time infrastructure merely provides a way for clients and services to find each other
 - a group composed of equal peers

JINI – components

- Infrastructure
 - set of components that enables building a federated Jini system
- Programming model
 - Set of interfaces that enables the construction (and deployment) of reliable services, including those that are part of the infrastructures
- Services
 - entities within the federation

JINI – components

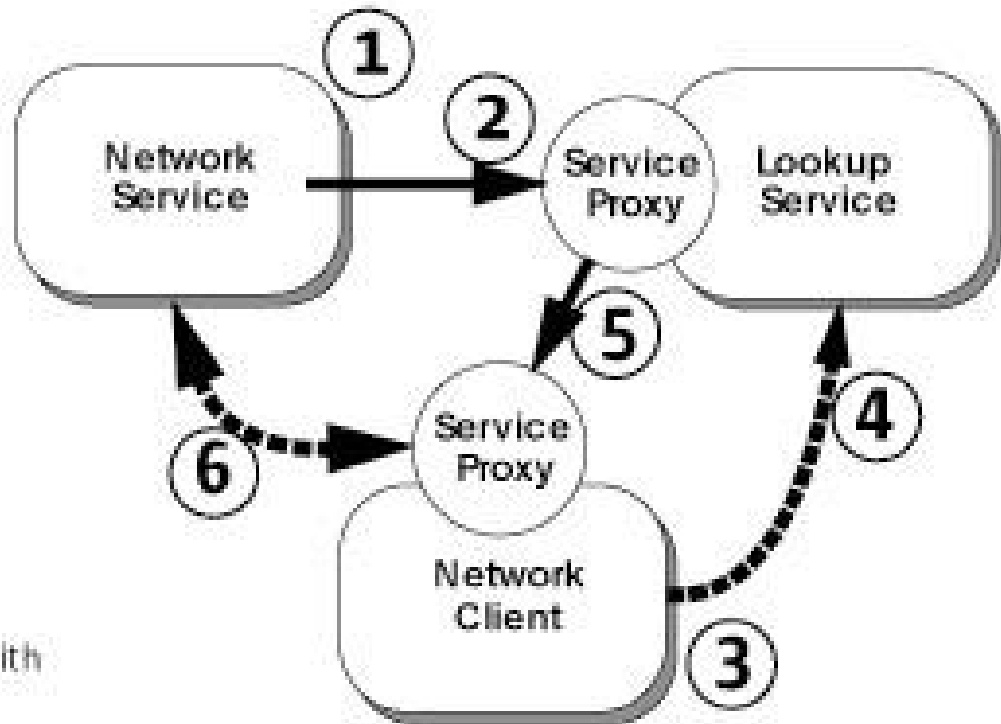
	Infrastructure	Programming Model	Services
Base Java	Java VM RMI Java Security	Java APIs Java Beans™ ...	JNDI Enterprise Beans JTS ...
Java + Jini	Discovery/Join Distributed Security Lookup	Leasing Transactions Events	Printing Transaction Manager JavaSpaces™ Service ...

JINI – run-time infrastructure

- Distributed Security System
 - Integrated with RMI
- Lookup services
 - discovery
 - join
 - lookup
- Service providers
 - Jini-enabled device/software
- Clients

JINI – lookup scenario

- ① **Discover**
Network service discovers available lookup services (LUS)
- ② **Join**
Network service sends service proxy to LUS
- ③ **Discover**
Network client discovers available LUS
- ④ **Lookup**
Network client sends a request to LUS to find desired services
- ⑤ **Receive**
LUS sends registered service proxy to network client
- ⑥ **Use**
Network client interacts directly with network service via service proxy



JINI – discovery service

- discovery service

- enabling clients and services to locate other lookup services
- simple protocol:
 - agents aiming to participate to the federation broadcast a 'presence announcement' by means of a multicast packet onto a well-known port
 - Lookup services monitor the well-known port for the presence of announcement
 - At the arriving of the packet, the lookup inspect the packet, get info about agents location (IP address) and via TCP/IP
 - Using RMI or TCP/IP, lookup sends a *service registrar object* to the agent, for further interaction (join, lookup).

JINI – join service

- join service

- enabling a service to register itself in a lookup service and become part of the federation

- using `register()` service (method) on the service registrar object, passing *SERVICE ITEMS* that provide info about the service provided

- *Service Items* = set of *service objects*

- » can be used by clients to interact with the service

- » can include service ATTRIBUTES (icons, infos,...)

- *Service objects* implement one or more interfaces specifying the interaction with clients

JINI – lookup stage

- lookup service

- enabling a client to query for services in a lookup service

- using lookup() service (method) on the service registrar object, passing a *SERVICE TEMPLATE* that provide info about the service looking for.

- a *Service Template* can include

- » array of Class objects, specifying the Java type(s) of the service object desired by the client

- » serviceID uniquely identifying the service

- Usually: searching an Interface

JINI – built-in services

- Security

- Principal and Access Control List notions

- Leasing

- specifying time in QoS
 - Lease-based access to services
 - *Lease* = time period negotiated by service and users
 - toward services situated in an environment (time, space)

- Distributed Events

- publish/subscribe patterns among services

- Transactions

- wrapping set of operations with one or several (heterogeneous) services in transactions

JINI – features

- abstracting distributed systems from network protocol level to Object Interface Level
 - complete *separation* of interface, implementation, network protocols
 - bringing OOP to the network

JINI – features

- flexibility
 - A service object can grant clients access to the service in many ways.
 - The service object can
 - actually represent the entire service, which is download to the client during lookup and executed locally
 - merely represent a proxy to a remote service provider with a private network protocol, that can be RMI, CORBA, DCOM, TCP/IP as well as proprietary protocols
 - ➔ the same service could be provided with different network protocols
- suitable for open environments
 - dynamic/changing environment
 - *availability/reliability* in unpredictable environment

JINI – technology & availability

- fully Java based technology
 - Serialisation
 - RMI
 - version 1.3
- Specification + Reference Implementation available from Sun
 - spec/docs/books/downloads/examples: <http://www.sun.org/jini>
 - docs/articles/projects/books: <http://www.jini.org>

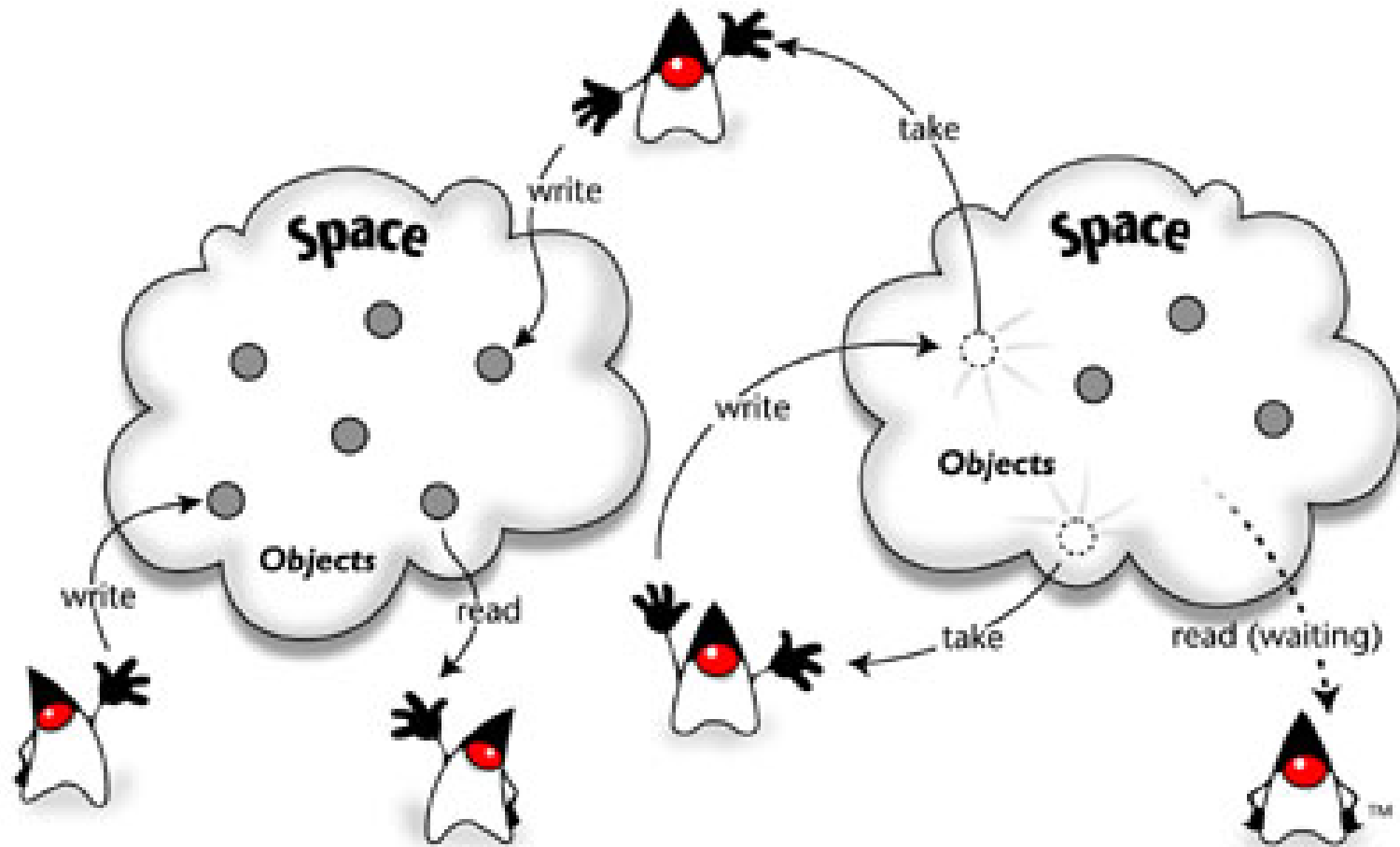
JINI – related technologies

- related technologies (pervasive computing, intelligent homes):
 - JES (Java Embedded Server)
 - <http://www.sun.com/software/embeddedserver/>
 - OSGi (Open Service Gateway Interface)
 - <http://osgi.org/>
 - UPnP (Universal Plug and Play)
 - <http://www.upnp.org/>
 - Openwings project
 - <http://www.openwings.org/>
 - sun Connected Home project
 - <http://www.sun.com/connected-home/>

JavaSpaces - overview

- JINI service
 - also standalone
- Unified service for dynamic communication, coordination, and sharing of objects between Java-based network resources
- Tuple Space Communication/Coordination model
 - Interacting through shared blackboards
 - Generative communication (Linda model)
 - Time and Space uncoupling
- Integrated with JINI builtin services (transactions, leasing, events)

JavaSpaces - scenario



JavaSpaces - benefits

- Orthogonality among computation and interaction/coordination
 - Engineering properties
 - Communication / Interaction as first class issue
- Uniform metaphor for the interaction world
 - concurrency, communication, synchronisations..
- Time/Space uncoupling →
 - natural support for disconnected communication
 - flexibility
 - suitable for open / unpredictable / frequently changing scenarios
- easier to engineer interactions than point-to-point
 - rapid-prototyping / changing / debugging of protocols

JavaSpaces – resource and links

- white papers / specs / docs / downloads:
 - <http://java.sun.com/products/javaspaces/>
- Some Related technologies:
 - IBM Tspaces
 - <http://www.almaden.ibm.com/cs/TSpaces/>
 - LuCe and TuCSoN
 - DEIS, Università di Bologna / Cesena
 - <http://www.lia.deis.unibo.it/research/tucson/>
 - <http://www.lia.deis.unibo.it/research/luce/>
 - MARS
 - DSI, Università di Modena
 - <http://polaris.ing.unimo.it/MOON/MARS/>

Web Services - overview

- examples of Service Oriented Architecture, where a node has a distinct role (consumer, or provider, registrar)
- The real innovation here is that competitors has set aside their differences and agreed on universal approaches to communicate
➔ business integration
- manifestation through standards adoption, that offer the three parties (consumers, providers, registrars) a common language to communicate (interoperate)

Web Services - overview

- first wave: legacy systems with XML-RPC mechanism such as SOAP
- easy routing: servers running web services are well-known hosts, with static IP
- Similar to CORBA but using XML → no binary interface

Web Services – from the industry

What are Web Services?

- ▮ ‘Modular and reusable’ applications that can be
 - Invoked over the web (using a protocol like SOAP)
 - Described using a service description language (like WSDL)
 - Published into a registry (like UDDI) so the service can be searched, bound and invoked
- ▮ Web Services do not imply component based development.
 - In fact a 20 year old TIP or CICS transaction can be wrapped as a web service
- ▮ Web Services need to be composable
- ▮ Can be simple to very complex; free to expensive

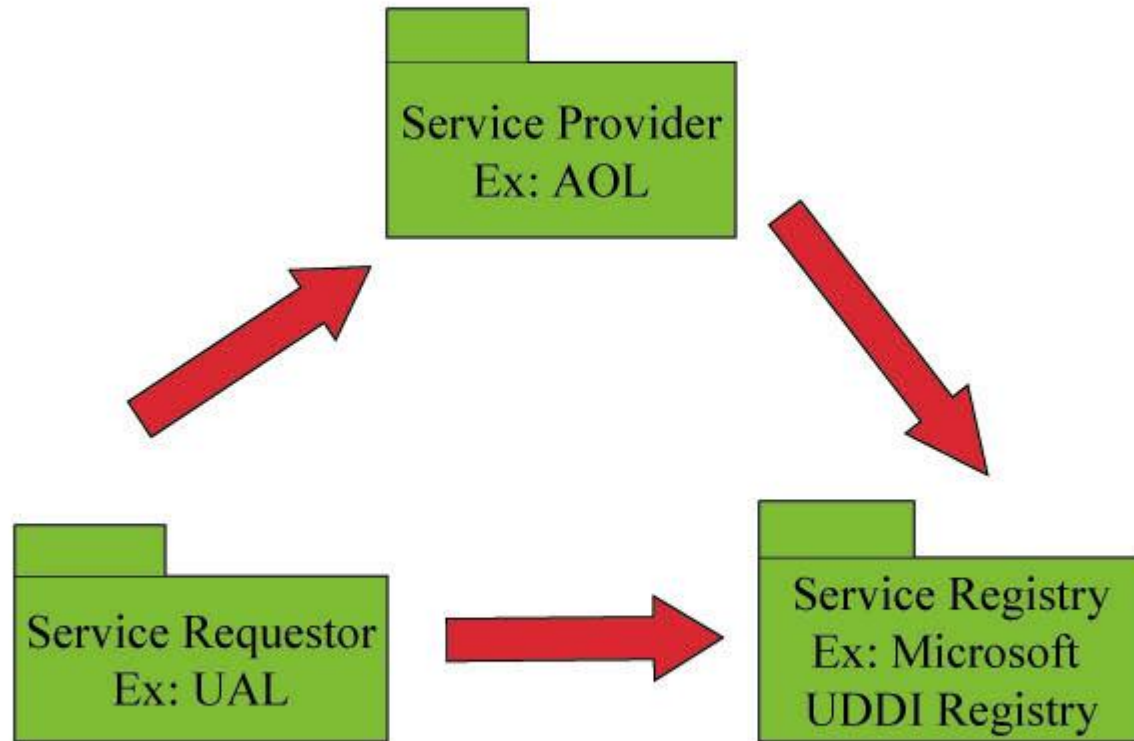
Web Services – from the industry

The Web Services Hype

- ▯ Microsoft brilliantly got the wagon rolling with the .Net vision
- ▯ The early work on SOAP (Simple Object Access Protocol) by Microsoft and Developmentor gradually attracted all the major players
 - IBM jumping on the bandwagon accelerated the momentum
- ▯ The UDDI (Universal Description, Discovery and Integration) efforts by Microsoft, IBM and Ariba gave another strong push
- ▯ Soon WSDL (Web Services Description Language) joined the party
- ▯ SOAP (XML/HTTP), UDDI and WSDL form the core formats and protocols of any Web Services Architecture today
- ▯ All major vendors, middleware platforms and architectures have embraced these technologies

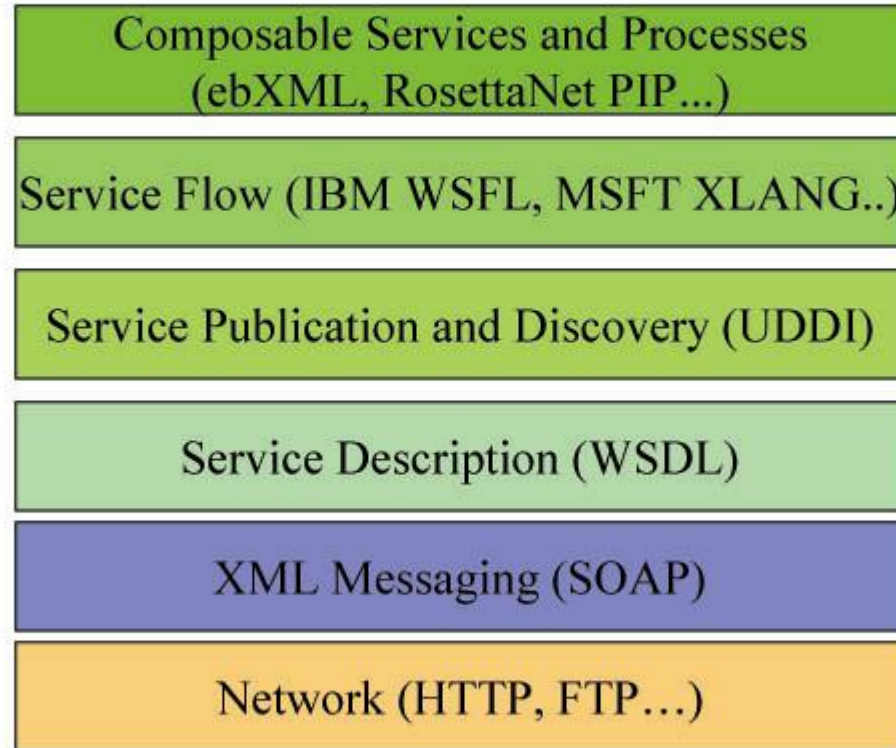
Web Services – from the industry

Web Services : The basic idea



Web Services – from the industry

Web Services : The Stack Evolves



Web Services – from the industry

Web Services : The Stack - 1

- ▮ Network
 - TCP/IP, HTTP, FTP, SMTP..
- ▮ XML Messaging
 - Messages conform to well defined XML Schemas
 - SOAP is the preferred messaging infrastructure
 - Eventually W3C XP will supplant SOAP
- ▮ Service Description
 - These are essentially interface definitions (we called these IDLs before!)
 - WSDL is the interface definition language for web services
 - Implementation information is also captured by WSDL

Web Services – from the industry

Web Services : The Stack - 2

- ▮ Service Publication and Discovery
 - Publish the service definition and implementation metadata
 - Capture additional business oriented service description information (D&B, Phone, URL...)
 - This metadata is published into a UDDI Registry
 - Registry is searchable
 - Service discovery happens dynamically at run time
 - Similar to CORBA Interface Repository and DII; Java Reflection...
 - Services are bound statically at design/development time
- ▮ Service sequencing and flows may extend WSDL

Web Services – from the industry

Web Services : The Stack - 3

- ▮ Service Flows
 - Individual services can be steps in a work flow
 - Flows can be graphically modeled
 - Various notations including UML Activity Diagrams have been used
 - Service composition needed for multi party service integration
- ▮ This area does not have as much consensus as lower level protocols
- ▮ IBM WSFL, MSFT XLANG are among candidate proposals

Web Services – from the industry

Web Services Architecture

Summary

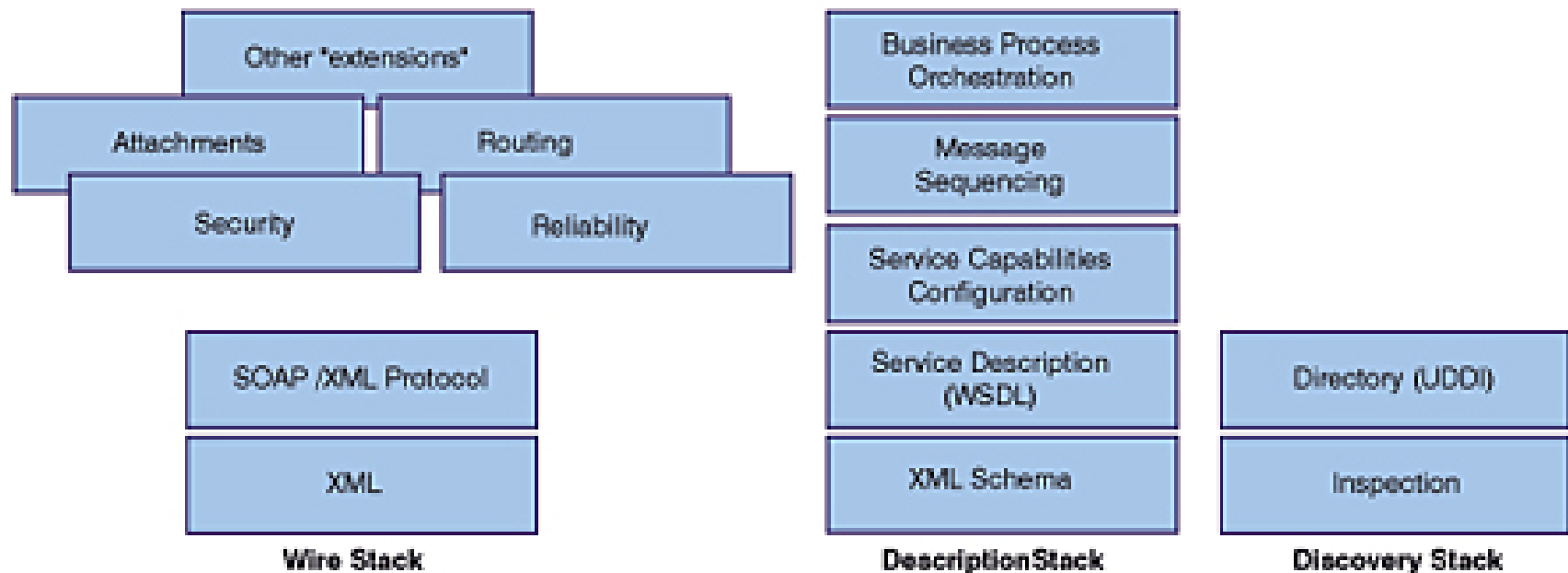
- ▮ Microsoft, IBM, BEA, BowStreet, HP, SUN are providing web services integration into their tool sets
- ▮ Microsoft has focused on the integration and ease of use/deployment the longest with its .Net initiative
- ▮ J2EE is rapidly embracing and extending Web Service formats and protocols
 - JAXR (Java API for XML Registries is an example)
 - IBM, BEA, SUN etc have integrated SOAP/WSDL/UDDI into their J2EE application servers
- ▮ Methodologies, modeling tools and middleware that support the web services life cycle are immature
- ▮ Back to the Integration Problem!

Web Services – from the industry

Solving the Integration Problem *What is needed?*

- ▯ Short Answer is XML and HTTP!
- ▯ Slightly longer answer is SOAP, WSDL, UDDI
- ▯ Even longer answer is SOAP, WSDL, UDDI, WSFL, ebXML, XAML, ACML...
- ▯ The real answer is
 - We need to think and build software like engineers
 - All these acronyms are great, but where is the process, methodology and architecture!
 - We need to integrate the systems we have with the newer technologies
 - etc.

Web Services - standards



Web Services - standards

- XML to represent information
 - no marshaling/unmarshaling
- UDDI to find and bind services
 - defines 3 services
 - White pages (looking by name)
 - Yellow Pages (looking by a topic)
 - Green Pages (looking by characteristics)
 - ebXML: a more complex UDDI
 - after the service is found, it is accessed through SOAP

Web Services - standards

- **SOAP (Simple Object Access Protocol)**
 - protocol to encapsulate a request as a XML payload using standard protocol such as HTTP
 - a SOAP call is packaged as the body of an HTTP request
 - also UDDI services are accessed through SOAP
 - support both async / sync call semantics
- **WSDL to describe the service (in XML)**
 - Similar to an IDL, but
 - IDL: first the interface, then implementation
 - WSDL: specifies the interface to an existing application, a new application, or anywhere in between

Web Services – WSDL example

```
<?xml version="1.0"?>
  <definitions name="StockQuote"
    targetNamespace="http://example.com/stockquote.wsdl"
    xmlns:tns="http://example.com/stockquote.wsdl"
    xmlns:xsd="http://example.com/stockquote.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
      <schema targetNamespace="http://example.com/stockquote.xsd"
        xmlns="http://www.w3.org/2000/10/XMLSchema">
        <element name="TradePriceRequest">
          <complexType>
            <all>
              <element name="tickerSymbol" type="string"/>
            </all>
          </complexType>
        </element>
        <element name="TradePrice">
          <complexType>
            <all>
              <element name="price" type="float"/>
            </all>
          </complexType>
        </element>
      </schema>
    </types>

    ...
```

Web Services – WSDL example

```
... <message name="GetLastTradePriceInput">
  <part name="body" element="xsd1:TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
  <part name="body" element="xsd1:TradePrice"/>
</message>

<portType name="StockQuotePortType">
  <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/>
  </operation>
</portType>

<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>

<service name="StockQuoteService">
  <documentation>My first service</documentation>
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">
    <soap:address location="http://example.com/stockquote"/>
  </port>
</service>
</definitions>
```

Peer-to-Peer Computing

- P2P networking is a network model where, depending on an operation's context, any node can operate as either a server or a client
- example of SOA, where the determination who is a provider, consumer, registrars is *much looser*
 - explosion of Napster, Gnutella → reintroduction of peer-to-peer to the corporate IT settings (Microsoft, Sun,...)
 - not only file sharing, but distributed computing, collaborative works, etc...
- Changes in WHAT the nodes are
 - Internet-connected PCs, which had been formerly relegated to being nothing but clients
- Changes in WHERE the nodes are
 - at the edge of the Internet, cut off from DNS systems, with no fixed IP address

Peer-to-Peer Computing

- *resource-centric* addressing for *unstable environment*
 - taking advantage of resources at the edge of the Internet
- *group* concept

P2P from Sun: JXTA

- generalising P2P functionality
- infrastructure providing a set of simple, small, flexible mechanisms to support P2P computing on any platform, anywhere, at anytime
- 6 different protocols to enable interactions (not interoperation)
 - interoperability must be designed on topo of JXTA
 - JXTA is like TCP/IP for the P2P
- Can run on J2ME (Java micro edition)
 - p2p adds another dimension to the kind of things you can build for wireless devices, like group collaboration, communications without central registration, file sharing on the fly

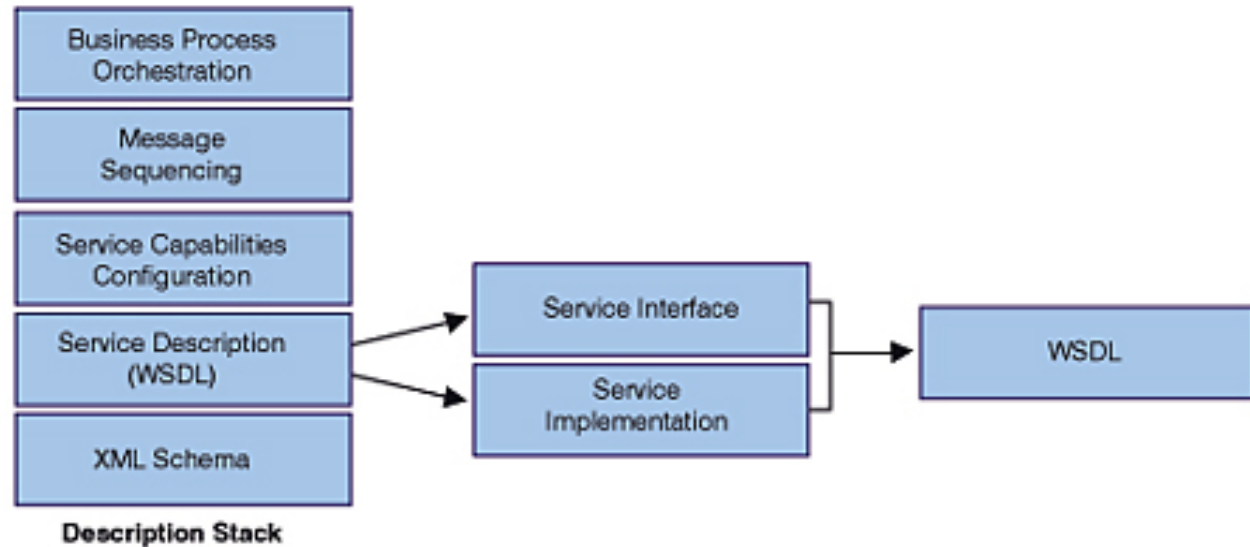
P2P from Microsoft: HailStorm

- set of web services whose data is contained in a set of XML docs and accessed from clients (HS endpoints) through SOAP
- services centered around security (user identity / authentication-centric (myAddress, myProfile, myWallet...)..)
- described as 'massively distributed', 'user centric' BUT ...
 - actually → mega-server
 - largest client-server installation ever conceived
 - because of 'reliability' (!?)

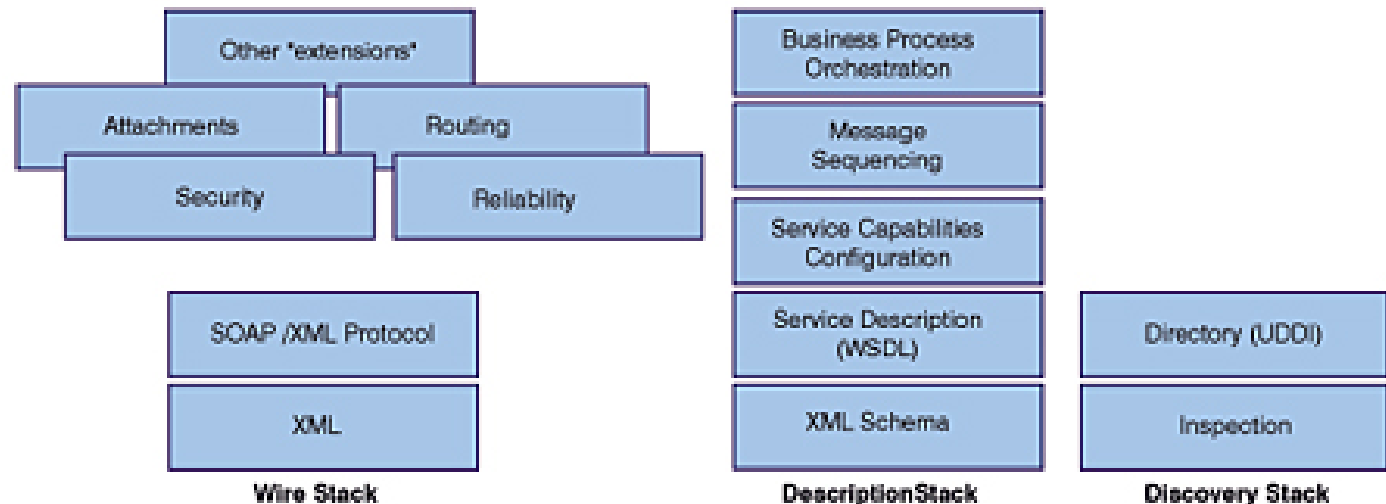
P2P and Web Services

- Both are kind of SOA with different Wire / Discover stack

JXTA Architecture →



WebServices architecture →



P2P and Web Services

- Integration of the architectures

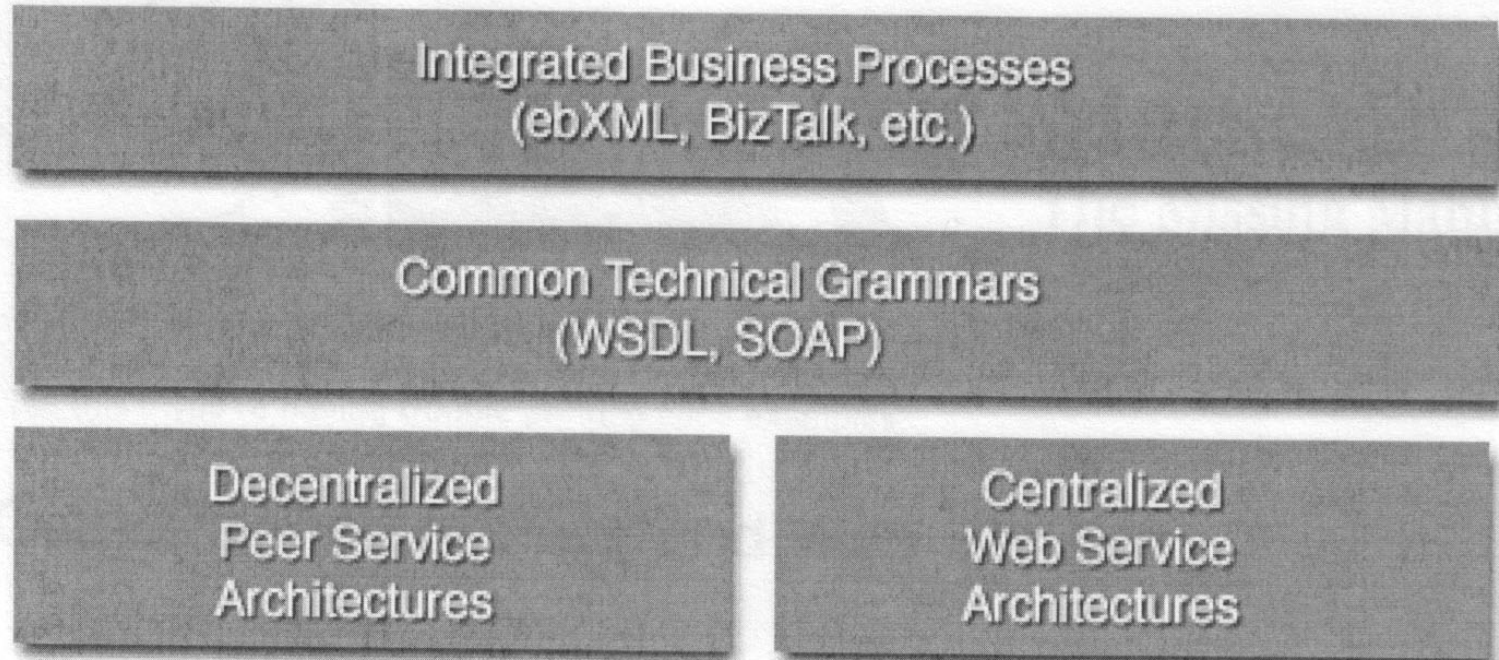


Figure 3. Integration of business processes on combined web/peer service architectures

Other emerging services / infrastructures

- JMS (Java Message Service)
 - Integration with CORBA, EJB
 - Infrastructure Messaging for the enterprise
 - provides a (limited) time uncoupled model of communications
 - <http://java.sun.com/products/jms/>

Middleware salad

Capability	J2EE	COM+	CORBA/OMA	Web Services	OMG MDA*	.Net
Network Layer	TCP/IP	TCP/IP	TCP/IP	TCP/IP	TCP/IP	TCP/IP
Web Protocol	HTTP	HTTP	HTTP	HTTP	HTTP	HTTP
Interface Definition	Java	Microsoft IDL	CORBA IDL	WSDL	IDL/XMI/WSDL	WSDL
Meta Language	XML	XML	XML/XMI	XML	MOF/XML	XML
RPC Mechanism	RMI	DCOM	IIOP	SOAP; XMLP	SOAP; IIOP	SOAP
Registry/Repository	JNDI; LDAP	LDAP; ADSI	Repository	UDDI	MOF;UDDI	UDDI
Process Flow	Proprietary	Proprietary	Proprietary	ebXML; WSFL	UML	Proprietary
Modeling Language	UML	UML	UML	UML	UML	UML

thesis

- Infrastructures for Pervasive Computing
 - exploring infrastructures/services for mobile computing
 - PDA, PocketPCs, ...
 - exploring infrastructures/services for Intelligent Homes and places
- Infrastructures for Cooperative Works
 - workflow management
 - team-work