22. Classical Component Systems – CORBA

Lecturer: Dr. Sebastian Götz

Prof. Dr. Uwe Aßmann

Technische Universität Dresden

Institut für Software- und Multimediatechnik

http://st.inf.tu-dresden.de/teaching/cbse

1. Mai 2017

1. Basics
2. Dynamic Call
3. Traded Call
4. Evaluation according to our criteria list
5. Appendices

Obligatory Reading

- ISC, 3.1-3.3
- Szyperski 2nd edition, Chap 13
- http://java.sun.com/javase/6/docs/technotes/guides/idl/

Literature

- R. Orfali, D. Harkey: Client/Server programming with Java and Corba. Wiley&Sons. easy to read.
- R. Orfali, D. Harkey, J. Edwards: Instant Corba. Addison-Wesley.
- CORBA. Communications of the ACM, Oct. 1998. All articles. Overview on CORBA 3.0.
- CORBA 3.1 specification: http://www.omg.org/spec/CORBA/3.1/
- Jens-Peter Redlich, CORBA 2.0 / Praktische Einführung für C++ und Java. Verlag: Addison-Wesley, 1996. ISBN: 3-8273-1060-1

22.1 Basic Mechanisms

CORBA: Common Object Request Broker Architecture®

- Founding year of the OMG (object management group) 1989
- Goal: plug-and-play components everywhere

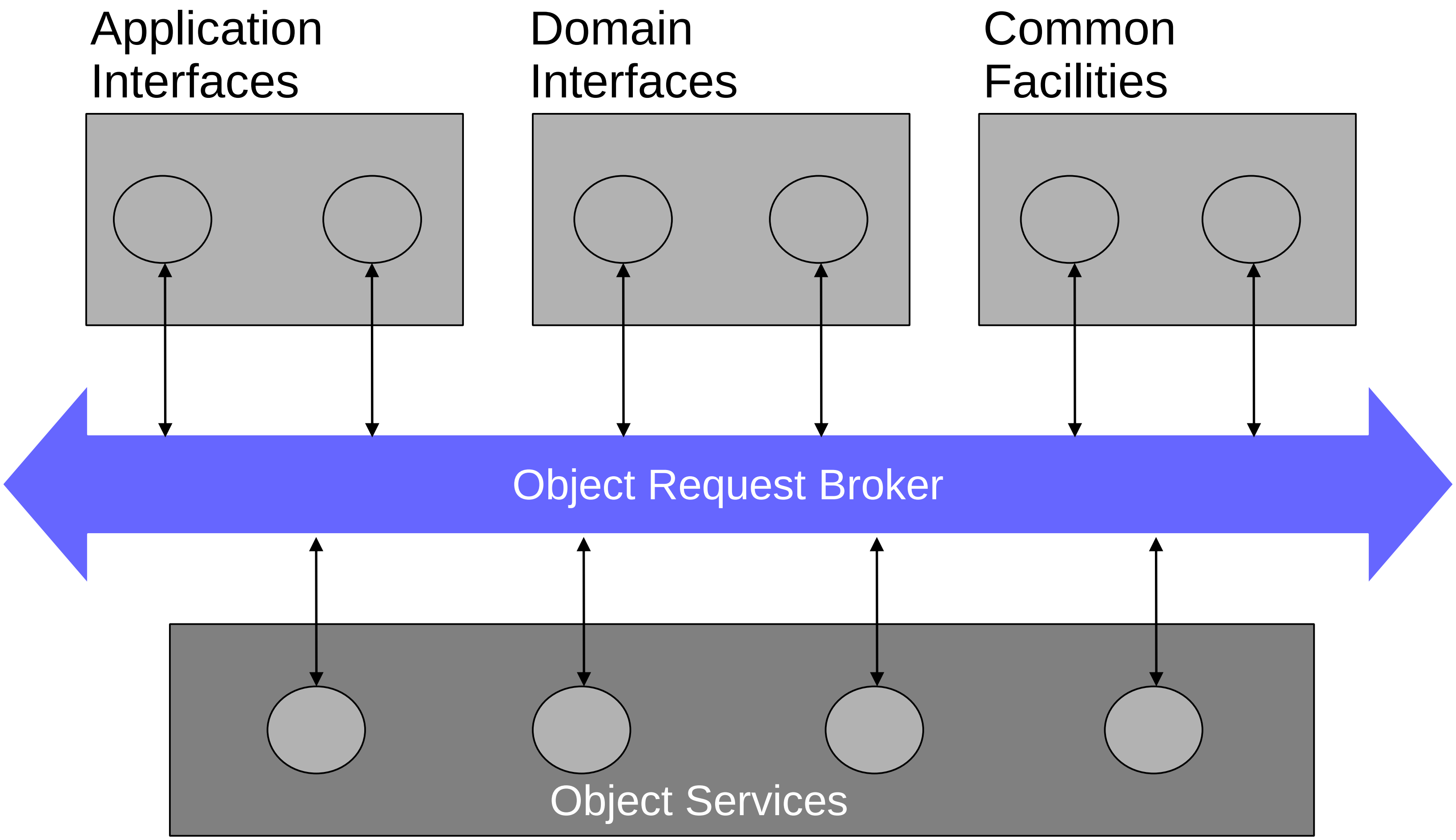
- Corba 1.1 1991 (IDL, ORB, BOA)
- ODMG-93 (Standard for OO-databases)
- Corba 2.0 1995, later 2.2 and 2.4
- Corba 3.0 1999
- Corba is large
  - Object Request Broker – 2000 pages of specification
  - Object Services – 300 pages
  - Common Facilities – 150 pages

Ingredients of CORBA

- Component Model
  - Components are classes and objects, i.e., similar to object-oriented software
    - In CORBA 3.0, the CCM has additionally been introduced
  - Components have more component secrets
    - Language interoperability by uniform interface description
    - Location transparency
    - Name transparency
    - Transparent network protocols
  - Standardization
    - CORBA Services
    - CORBA Facilities
      - Horizontal vs. vertical
- Composition Techniques
  - Adaptation by stubs and skeletons
  - CORBA MOF for metamodelling

OMA (Object Management Architecture)

- A software bus, based on the Mediator (Broker) design pattern
  - Coupled by decorator-connectors



The Top Class CORBA::Object

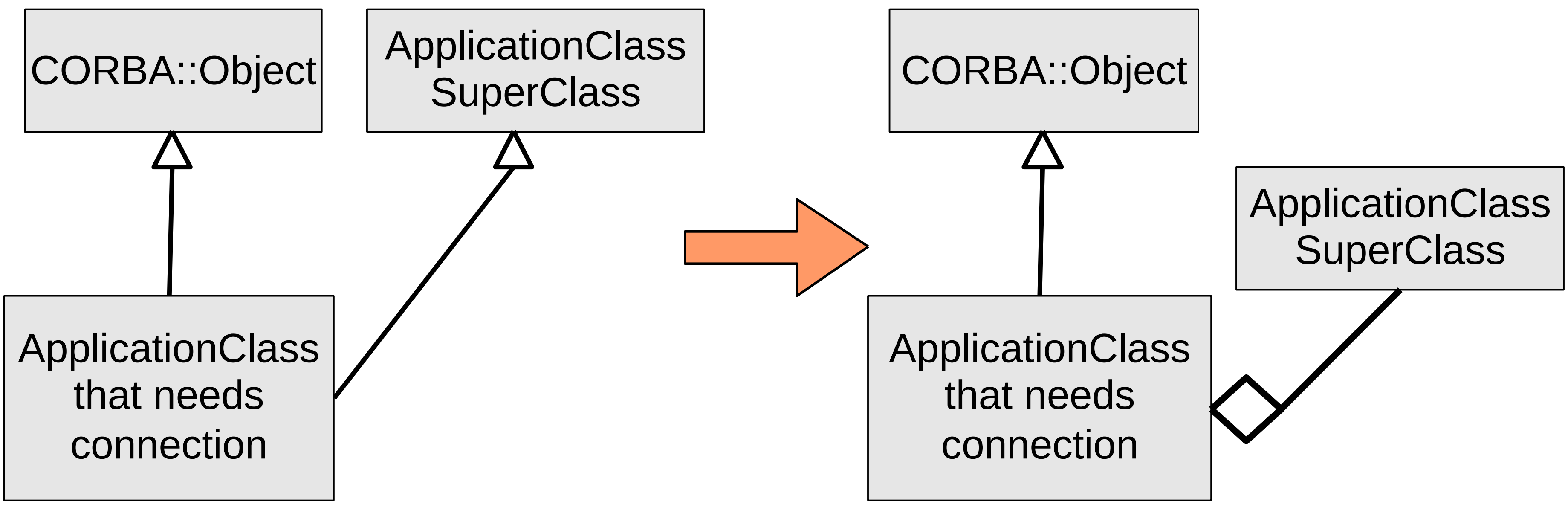
CORBA::Object

get_implementation
get_interface
is_nil
is_a
create_request
duplicate
release
....

- The class CORBA::Object defines a component model
  - The class must be inherited to all objects in the application
- CORBA supports reflection and introspection:
  - get_interface delivers a reference to the entry in the interface repository
  - get_implementation a reference to the implementation
- Reflection works by the interface repository (list_initial_references from the CORBA::ORB interface).

Problem: Multiple Inheritance of CORBA Object

- CORBA::Object includes *code* into a class
- Many languages only offer only single inheritance
  - Application super class must be a delegatee
- Only some languages offer mixin inheritance (mixin layers), such as Scala, C# 4.0, Eiffel



Basic Connections in CORBA

- CORBA composes components with connections
  - Static method call with static stubs and skeletons
    - Local or remote is transparent (compare to EJB!)
  - Polymorphic call
    - Local or remote
  - Event transmission
  - Callback (simplified Observer pattern)
  - Dynamic invocation (DII, request broking, interpreted call, symbolic call)
    - Searching services dynamically in the web (location transparency of a service)
  - Trading
    - Find services in a yellow pages service, based on properties
- Important: CORBA is language-heterogeneous, i.e., offers these services for most of the main-stream languages

22.2 Dynamic Call Connector (with Object Request Broking)

- (Reified or interpreted call)

Dynamic Call Connector (Request Broking)

- CORBA *dynamic call* is a *reified call (interpreted call)*, i.e., a reflective call with a symbolic name and arguments
  - Without knowing that the service exists
  - Services can be dynamically exchanged, brought into play a posteriori
  - Without recompilation of clients, nor regeneration of stubs
  - Binding of names to adresses is dynamic
- Requires descriptions of semantics of service components
  - For identification of services
    - Metadata (descriptive data): catalogues of components (interface repository, implementation repository)
    - Property service (later)
- and a mediator, that looks for services: the ORB

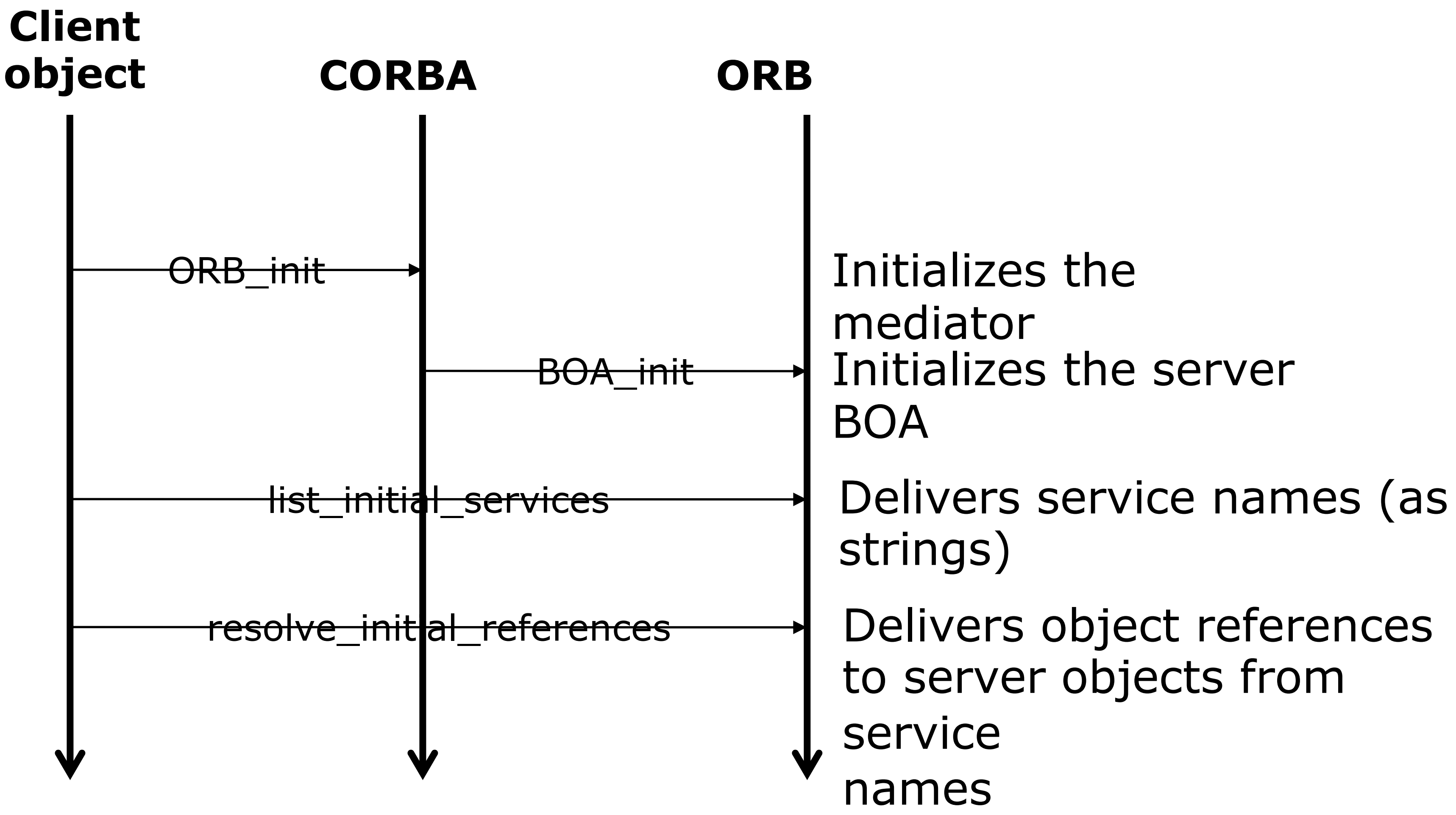
Object Request Broker (ORB)

- For a dynamic call, the ORB must be involved
- The ORB is a *mediator* (design pattern) between client and server
  - Hides the the environment from clients
  - Can talk to other ORBs, also on the web

CORBA::ORB

init
object_to_string
string_to_object
BOA_init
list_initial_services
resolve_initial_references
get_default_context
create_environment
....

ORB Activation



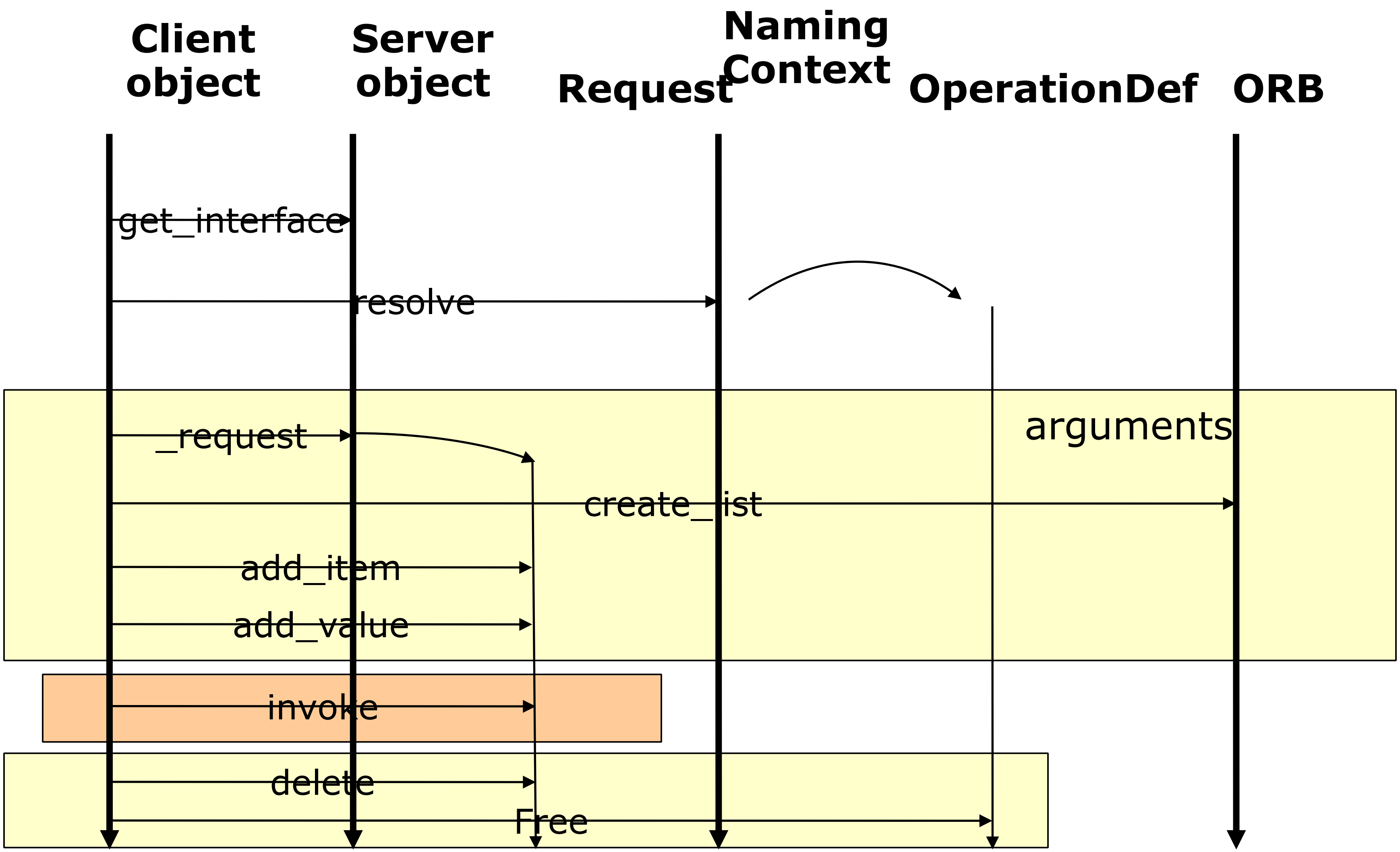
Requesting a Service via the ORB

- Reflective calls
  - Building a call object (Request)
  - Adding arguments
  - Invoking
  - Polling, reading

CORBA::ORB

// dynamic call
create_list
create_operation_list
add_item
add_value
invoke
poll_response
send
get_response
delete
....

Protocol of Dynamic Call (DII)



ORBs

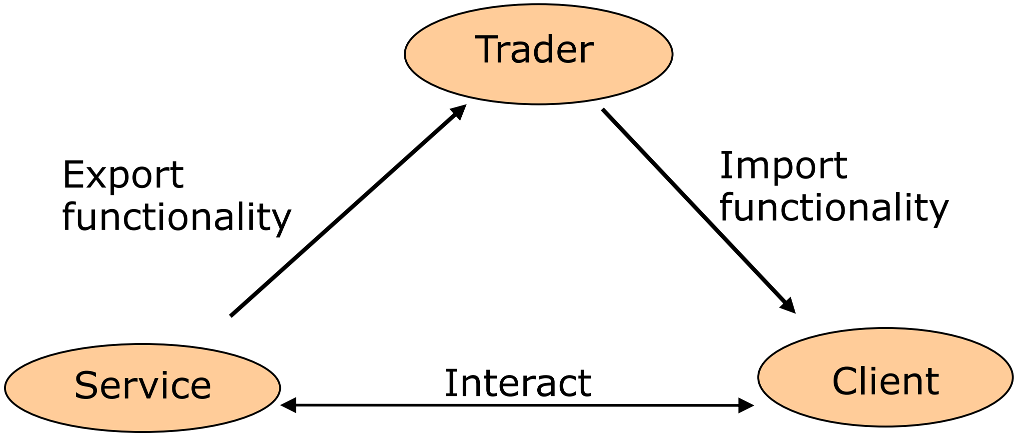
- **Java-based**
  - IBM WebSphere
  - IONA Orbix: In Java, ORBlets possible
  - BEA WebLogic
  - Visibroker (in Netscape)
  - Voyager (ObjectSpace) (with Mobile Agents)
  - free: JacORB, ILU, Jorba, DynaORB
- **C-based**
  - ACE ORB TAO, University Washington (with trader)
  - Linux ORBIT (gnome)
  - Linux MICO
- **Python-based**
  - fnorb
- **http://www.omg.org**

22.3 Trader-Based Call

- The foundation of service-oriented architecture (SOA)

Beyond Dynamic Call: Service Call with the Trader Service

- A service call is a call, not based on naming but on semantic attributes, published properties
  - Requires a yellow page directory of services
- Service-oriented architectures (SOA), requires matchmaking of services
  - The ORB resolves operations still based on naming (with the name service). The trader, however, resolves services without names, only based on properties and policies
- The trader gets offers from servers, containing new services



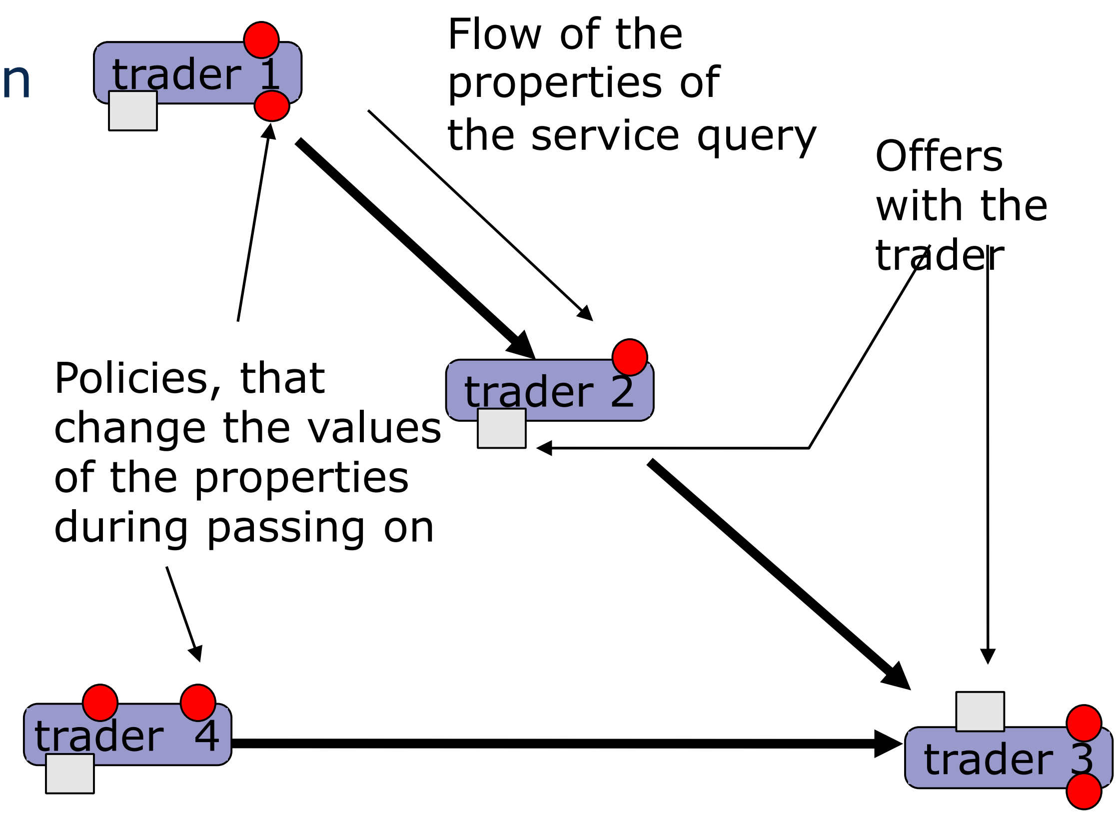
Mediator pattern, mediator lets clients lookup services

Service Offers for Trader

- Service offer (IOR with properties (metadata))
  - Properties describe services
  - Are used by traders to match services to queries
  - *not* facet-based, one-dimensional
- Dynamic property
  - A property can be queried dynamically by the trader of service
  - The service-object can determine the value of a dynamic property anew
- Matching with the standard constraint language
  - Boolean expressions about properties
  - Numeric and string comparisons

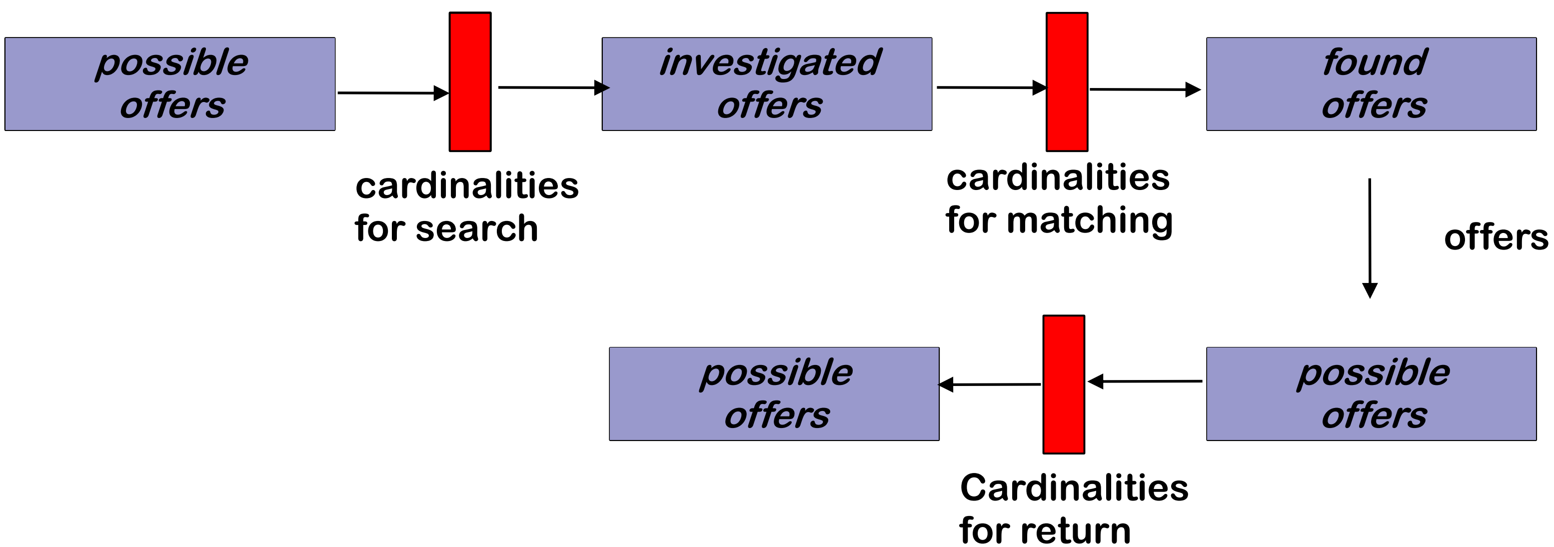
Traders Provide Service Hopping

- If a trader doesn’t find a service, it calls neighbor traders
  - Design pattern Chain of Responsibility
- Graph of traders
  - Links to neighbors via TraderLink
  - TraderLink filters queries and manipulate via policies



Modification of Queries

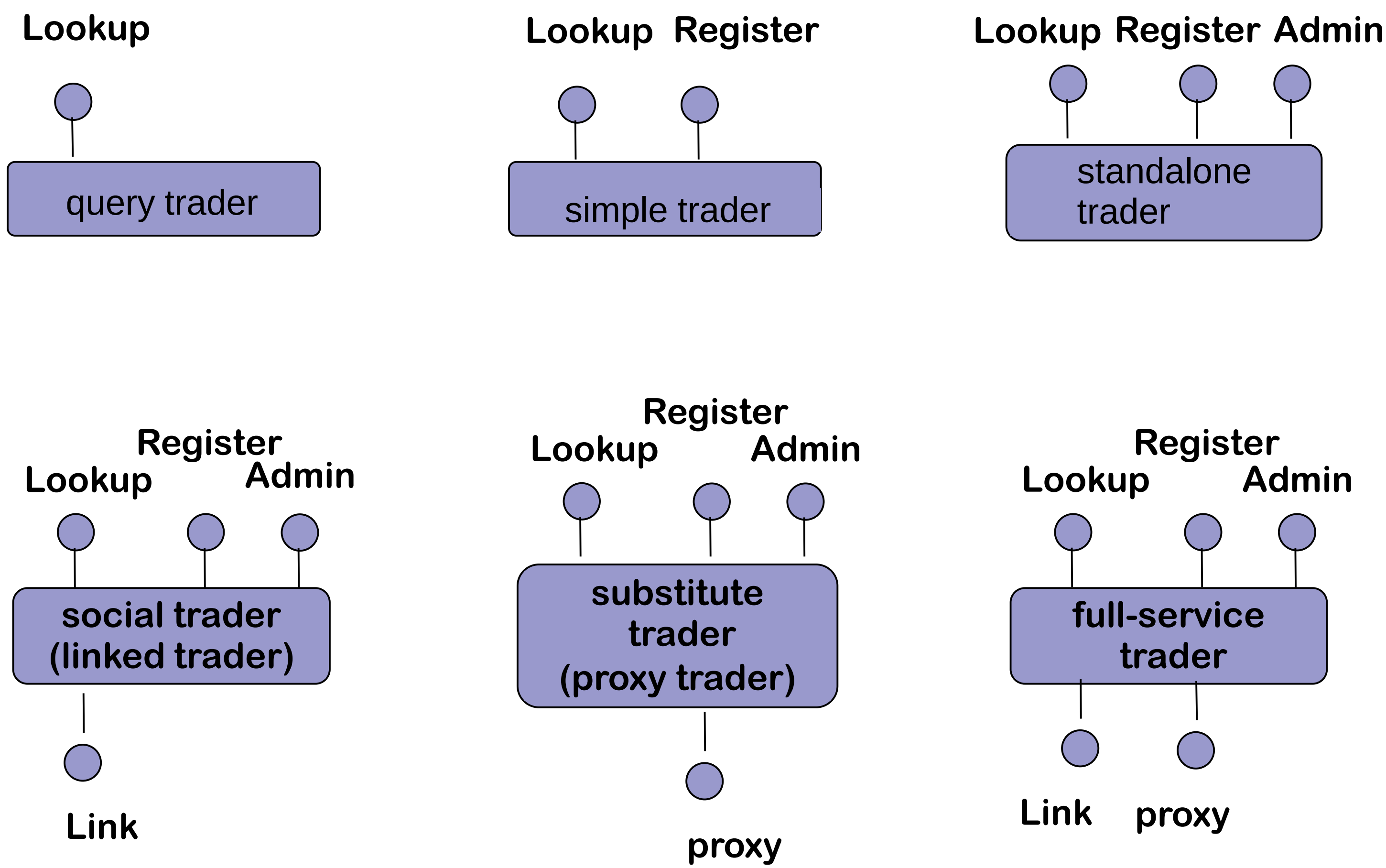
- Policies parameterize the behaviour of the traders and the TraderLinks
  - Filters, i.e., values, modifying the queries:
  - max_search_card: maximum cardinality for the ongoing searches
  - max_match_card: maximum cardinality for matchings
  - max_hop_count: cardinality search depth in the graph



Interfaces Trading Service

- Basic interfaces
  - Lookup (query)
  - Register (for export, retract, import of services)
  - Admin (info about services)
  - Link (construction of trader graph)
- How does a lookup query look like?
  - ```
    Lookup.Query(in ServicetypeName, in Constraint,
                 in PolicySeq, in SpecifiedProperties,
                 in howTo, out OfferSequence, offerIterator)
    ```
- Unfortunately, no faceted matchmaking possible!
    ```

CORBA Trader Types



Corba 3.0

- Provides the well-defined packaging for producing components
  - CORBA Component Model (CCM): similar to EJB
- Message Service MOM: Objects have asynchronous buffered message queues
- Language mappings avoid IDL
  - Generating IDL from language specific type definitions
  - C++2IDL, Java2IDL, …
- XML integration (SOAP messages)
- Scripting (CORBA script), a composition language

22.5 Evaluation of CORBA

- as composition system

Component Model

- Mechanisms for secrets and transparency: very good
  - Interface and Implementation repository
  - Component language hidden (interoperability)
  - Life-time of service hidden
  - Identity of services hidden
  - Location hidden
- No parameterization
- Standardization: quite good!
  - Services, application services are available
  - On the other hand, some standards are FAT
  - Technical vs. application specific vs business components:
  - .. but for business objects, the standards must be extended (vertical facilities) (thats´s where the money is)

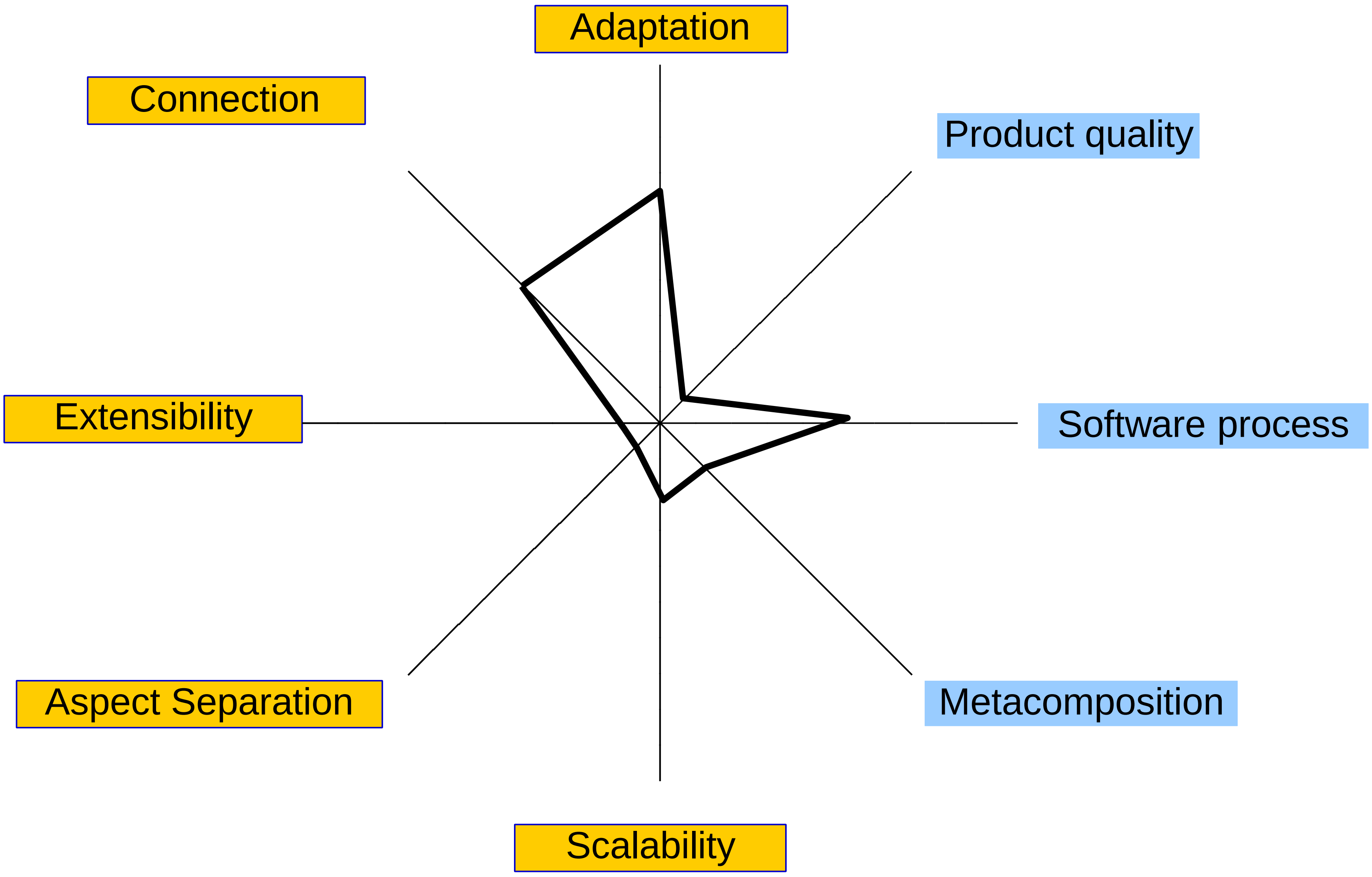
Composition Technique

- Mechanisms for connection
  - Mechanisms for adaptation
    - Stubs, skeletons, server adapters
  - Mechanisms for glueing: marshalling based on IDL
- Mechanisms for aspect separation
  - Multiple interfaces per object
    - Facade classes/objects (design pattern facade)
- Nothing for extensions
- Mechanisms for meta-modeling
  - Interface Repositories with type codes
  - Implementation repositories
  - Dynamic call and traded call are reflective and introspective
- Scalability
  - Connections cannot easily be exchanged (except static local and remote call)

Composition Language

- Weak: CORBA scripting provides a facility to write glue code, but only black-box composition

CORBA


Adaptation
Connection
Product quality
Extensibility
Software process
Aspect Separation
Metacomposition
Scalability

What Have We Learned

- CORBA is big, but universal:
  - The Corba-interfaces are very flexible, work and can be used in practice
  - .. but also complex and fat, may be too flexible
  - If you have to connect to legacy systems, CORBA works
- Corba has the advantage of an open standard
- To increase reuse and interoperability in practice, one has to learn *many* standards
- Trading and dynamic call are future advanced communication mechanisms
- CORBA was probably only the first step, but web services might be taking over

The End

Component-Based Software Engineering (CBSE)

Appendix
Basic Composition Technique of CORBA (Basic CORBA Connections)

- (self study)

Static CORBA Call, Local or Remote

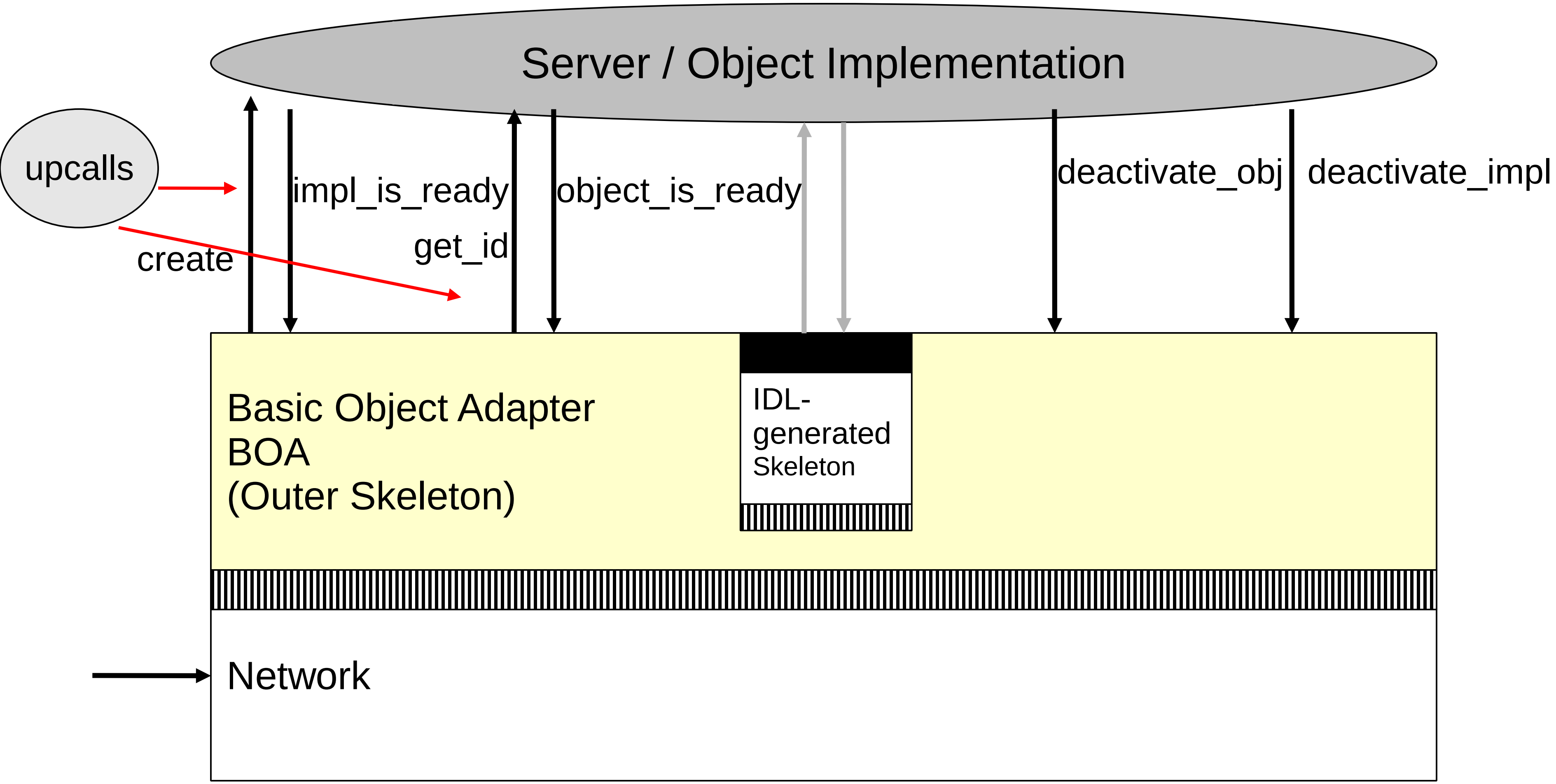
- Advantage: methods of the participants are statically known
  - Indirect call by stub and skeletons, without involvement of an ORB
  - Supports distribution (exchange of local call in one address space to remote call is very easy)
    - Inherit from CORBA class
    - Write an IDL spec
  - No search for service objects, rather fast
  - Better type check, since the compiler knows the involved types
- The call goes through the server object adapter (server decorator)
  - Basic (server) object adapter (BOA)
  - Portable (server) object adapter (POA)
  - This hides the whether the server is transient or persistent

The CORBA Outer Skeleton: Basic Object Adapter BOA

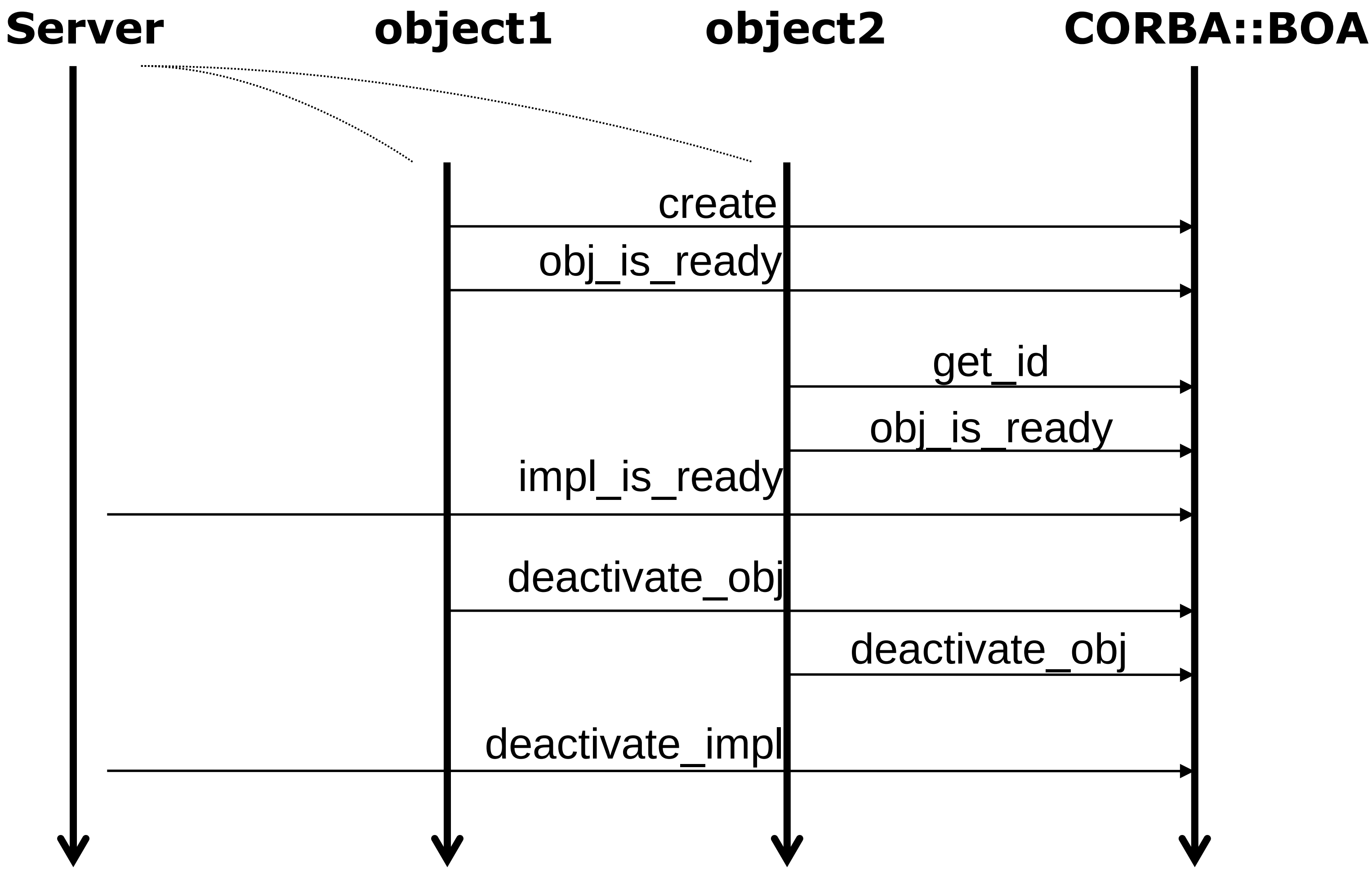
- The BOA is a real adapter (no decorator)
  - The BOA hides the life time of the server object (activation: start, stop)
  - Persistency
- The BOA is implemented in every ORB, for minimal service provision
- The BOA maintains an implementation repository (component registry)
- It supports non-object-oriented code

CORBA::BOA
create **get_id** **dispose** **set_exception** **impl_is_ready** **obj_is_ready** **change_implementation** **deactivate_impl** **deactivate_obj**

Server Site


Server / Object Implementation
upcalls
impl_is_ready
object_is_ready
deactivate_obj
deactivate_impl
get_id
create
Basic Object Adapter
BOA
(Outer Skeleton)
IDL-
generated
Skeleton
Network

Object Activation on the Server through a BOA



Portable Object Adapter POA

CORBA::POA

create_POA
find_POA
create_reference
dispose
set_exception
impl_is_ready
obj_is_ready
change_implementation
activate_object
deactivate_object

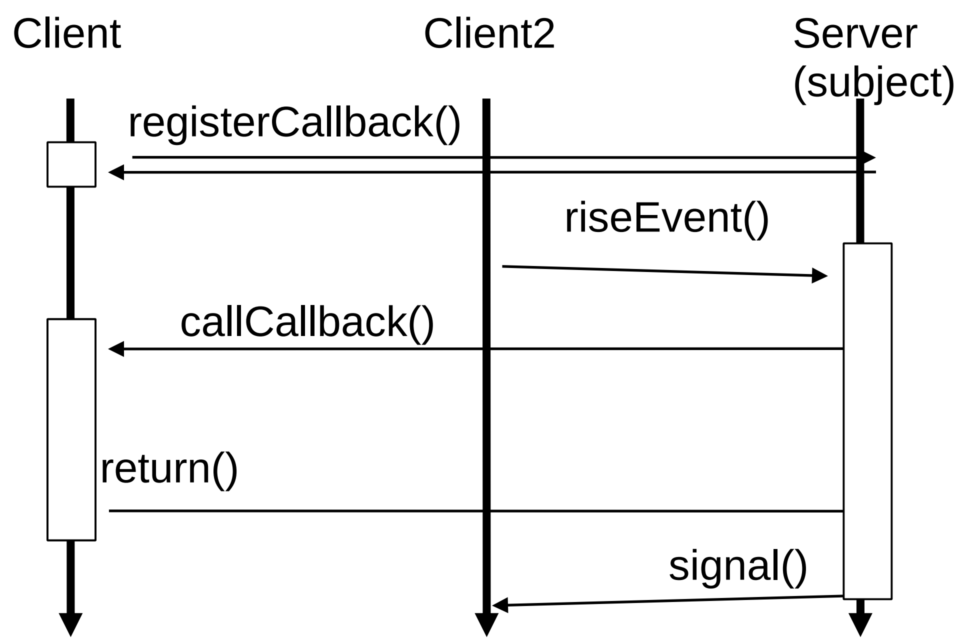
- The POA is a evolution of the BOA in CORBA 3.0
  - One per server, serving many objects
  - Nested POAs possible, with nested name spaces
- User policies for object management
  - User-written instance managers for management of object instances

Object Adapters Support Different Server Life-Time Models

- **Common server process (shared server)**
  - Several objects reside in one process on the server; the BOA initializes them as threads with common address space (common apartment)
    - deactivate_impl, impl_is_ready, obj_is_ready are mapped directly to thread functions
- **Separate server process (unshared server)**
  - For every object an own process
- **Server-per-request (session server)**
  - Every request generates a new process
  - Similar to Session EJB
- **Persistent server**
  - Another application stores the objects (e.g., a data base).
  - The BOA passes on the queries
  - Similar to Entity Bean

Callback Connectors with the Callback Service

- The Callback pattern is a simplified Observer pattern
  - Registration and notification, but not status update
- Callback function registration
  - Register a procedure variable, a closure (procedure variable with arguments), or a reference to an object at the subject, the server
- Callback works for all languages, not only object-oriented ones



Event Connections

- Most flexible way of communication (also called messages)
  - Asynchronous communication
  - Works for every CORBA language
- Receiver models
  - **Unicast**: one receiver
  - **Multicast**: many receivers
  - **Dynamically** varying receivers
- **Push model:** PushConsumer/PushSupplier: object delivers event with push, event is shipped automatically
- **Pull model:** PullSupplier/PullConsumer: object waits for event with pull
  - Synchronous or asynchronous
  - Untyped generic events, or typed by IDL
- **Event channels** as intermediate buffers
  - Channels buffer, filter, and map of pull to push
  - Advantage:
    - Asynchronous Working in the Web (with IIOP and dynamic Call)
    - Attachment of legacy systems interesting for user interfaces, network computing etc.
  - Disadvantage: Very general interface

Appendix Dynamic Call Connector (with Object Request Broking)

- Code example (self study)

Example Dynamic Call in C++


```
// Wow, a complex protocol!!

CORBA::ORB_ptr  orb;
main(int argc, char* argv[]) {
  orb= CORBA::ORB_init(argc,argv, ORBID);
  // alternative description of service
  CosNaming::NamingContext_ptr naming=
     CosNaming::NamingContext::_narrow(

::resolve_initial_references("NameService"));
  CORBA::Object_ptr obj;
  try  {
    obj= naming->resolve(mk_name("dii_smpl"));
  }  catch (CORBA::Exception)  {
    cerr << "not registered" << endl; exit(1); }

// construct arguments
  CORBA::Any val1;  val1 <<= (CORBA::Short) 123;
  CORBA::Any val2;  val2 <<= (CORBA::Short) 0;
  CORBA::Any val3;  val3 <<= (CORBA::Short) 456;
```

```
// Make request (short form)
 CORBA::Request_ptr rq= obj->_request("op");
// Create argument list
 rq->arguments() = orb->create_list();
 rq->arguments()->add_value("arg1",val1,CORBA::ARG_IN);
 rq->arguments()->add_value("arg2",val2,CORBA::ARG_OUT);
 rq->arguments()->add_value("arg3",val3,CORBA::ARG_INOUT);
     // Start request (synchronously)
 cout << "start request" << endl;

 rq->invoke();
     // analyze result
 CORBA::Short rslt ;
 if (*(rq->result()->value()) >>= rslt) {
    // Analyze the out/inout-prameters (arg1 has index 0)
    CORBA::Short _arg2, _arg3;
    *(rq->arguments()->item(1)->value()) >>= _arg2;
    *(rq->arguments()->item(2)->value()) >>= _arg3;
    cout << " arg2= "   << _arg2 << " arg3= " << _arg3
        << " return= " <<  rslt << endl;  }
 else  {
    cout << "result has unexpected type" << endl; }
}
```

DII Invocation in Java (1)


```
// Client.java
// Building Distributed Object Applications with CORBA
// Infowave (Thailand) Co., Ltd.
// http://www.waveman.com
// Jan 1998
public class Client {
   public static void main(String[] args) {
      if (args.length != 2) {
         System.out.println("Usage: vbj Client <carrier-name> <aircraft-name>");
         return;
      }
      String carrierName = args[0];
      String aircraftName = args[1];
      org.omg.CORBA.Object carrier = null;
      org.omg.CORBA.Object aircraft = null;
      org.omg.CORBA.ORB orb = null;
      try {
          orb = org.omg.CORBA.ORB.init(args, null);
      }
      catch (org.omg.CORBA.systemsxception se) {
         System.err.println("ORB init failure " + se);
         System.exit(1);
      }
```

DII Invocation in Java (2)


```
{ // scope
  try {
    carrier = orb.bind("IDL:Ship/AircraftCarrier:1.0",
                       carrierName, null, null);
  } catch (org.omg.CORBA.systemsxception se) {
    System.err.println("ORB init failure " + se);
    System.exit(1);
  }
  org.omg.CORBA.Request request = carrier._request("launch");
  request.add_in_arg().insert_string(aircraftName);
  request.set_return_type(orb.get_priwithive_tc(
                          org.omg.CORBA.TCKind.tk_objref));
  request.invoke();
  aircraft = request.result().value().extract_Object();
}
{ // scope
  org.omg.CORBA.Request request = aircraft._request("codeNumber");
  request.set_return_type(orb.get_priwithive_tc(
                                org.omg.CORBA.TCKind.tk_string));
  request.invoke();
  String designation = request.result().value().extract_string();
  System.out.println("Aircraft " + designation + " is coming your way");
}
```

Server Implementation


```
// Building Distributed Object Applications with CORBA
// Infowave (Thailand) Co., Ltd.
// http://www.waveman.com
// Jan 1998
public class Server {
  public static void main(String[] args) {
    org.omg.CORBA.ORB orb = null;
    try {
      orb = org.omg.CORBA.ORB.init(args, null);
    } catch (org.omg.CORBA.systemsxception se) {
      System.err.println("ORB init failure " + se);
      System.exit(1);
    }

    org.omg.CORBA.BOA boa = null;
    try {
      boa = orb.BOA_init();
    } catch (org.omg.CORBA.systemsxception se) {
      System.err.println("BOA init failure " + se);
      System.exit(1);
    }

    Ship.AircraftCarrier carrier =
       new AircraftCarrierImpl("Nimitz");
try {
  boa.obj_is_ready(carrier);
} catch (org.omg.CORBA.systemsxception se) {
  System.err.println(
"Object Ready failure " + se);
  System.exit(1);
}

System.out.println(
carrier + " ready for launch !!!");

try {
  boa.impl_is_ready();
} catch (org.omg.CORBA.systemsxception se) {
  System.err.println(
"Impl Ready failure " + se);
  System.exit(1);
}
    }
 }
```

Example: Time Server in Java

- On one machine; 2 address spaces (processes)
- Call provides current time
- Contains
  - IDL
  - Server
    - Starts ORB
    - Initializes Service
    - Gives IOR to the output
  - Client
    - Takes IOR
    - Calls service

```
// TestTimeServer.idl

module TestTimeServer{
    interface ObjTimeServer{
        string getTime();
    };
};
```

Service Component


```
// TestTimeServerImpl.java – Server Skeleton
import CORBA.*;
class ObjTestTimeServerImpl extends
TestTimeServer.ObjTimeServer_Skeleton { // generated from IDL
// Variables
// Constructor
// Method (Service) Implementation
public String getTime() throws CORBA.SystemException {
      return "Time: " + currentTime;
}

};
```

Server Implementation


```
// TimeServer_Server.java
import CORBA.*;
public class TimeServer_Server{
public static void main(String[] argv){
      try {
            CORBA.ORB orb = CORBA.ORB.init();
            …
            ObjTestTimeServerImpl obj =
                  new ObjTestTimeServerImpl(…);
            …
            System.out.println(orb.object_to_string(obj));
      }
      catch (CORBA.SystemException e){
            System.err.println(e);
      }
}
};
```

Client Implementation (Simpler Protocol)


```
// TimeServer_Client.java
import CORBA.*;
public class TimeServer_Client{
public static void main(String[] argv){
      try {
            CORBA.ORB orb= CORBA.ORB.init();
            …
            CORBA.object obj = orb.string_to_object(argv[0]);
            …
            TestTimeServer.ObjTimeServer timeServer =
              TestTimeServerImpl.ObjTimeServer_var.narrow(obj);
            …
            System.out.println(timeServer.getTime());
      }
      catch (CORBA.SystemException e){
            System.err.println(e);
      }
}
};
```

Execution


```
// starting server
C:\> java TimeServer_Server

IOR:000000000000122342435 …

// starting client
C:\> java TimeServer_Client IOR:000000000000122342435 …

Time: 14:35:44
```

Appendix
Corba Services

- (optional material)

Literature

- OMG. CORBA services: Common Object Service Specifications. http://www.omg.org.
- OMG: CORBAfacilities: Common Object Facilities Specifications.

Overview on Corba Services

- Services provide functionality a programming language might not provide (e.g, Cobol, Fortran)
- 16+ standardized service interfaces (i.e., a library)
  - Standardized, but status of implementation different depending on producer
- Object services
  - Deal with features and management of objects
- Collaboration services
  - Deal with collaboration, i.e., object contexts
- Business services
  - Deal with business applications
- The services serve for standardization. They are very important to increase reuse.
  - Remember, they are available for every language, and on distributed systems!

Object Services: Rather Simple

- Name service (directory service)
  - Records server objects in a simple tree-like name space
  - (Is a simple component system itself)
- Lifecycle service (allocation service)
  - Not automatic; semantics of deallocation undefined
- Property service (feature service for objects)
- Persistency service (storing objects in data bases)
- Relationship service to build interoperable relations and graphs
  - Support of standard relations reference, containment
  - Divided in standard roles contains, containedIn, references, referenced
- Container service (collection service)

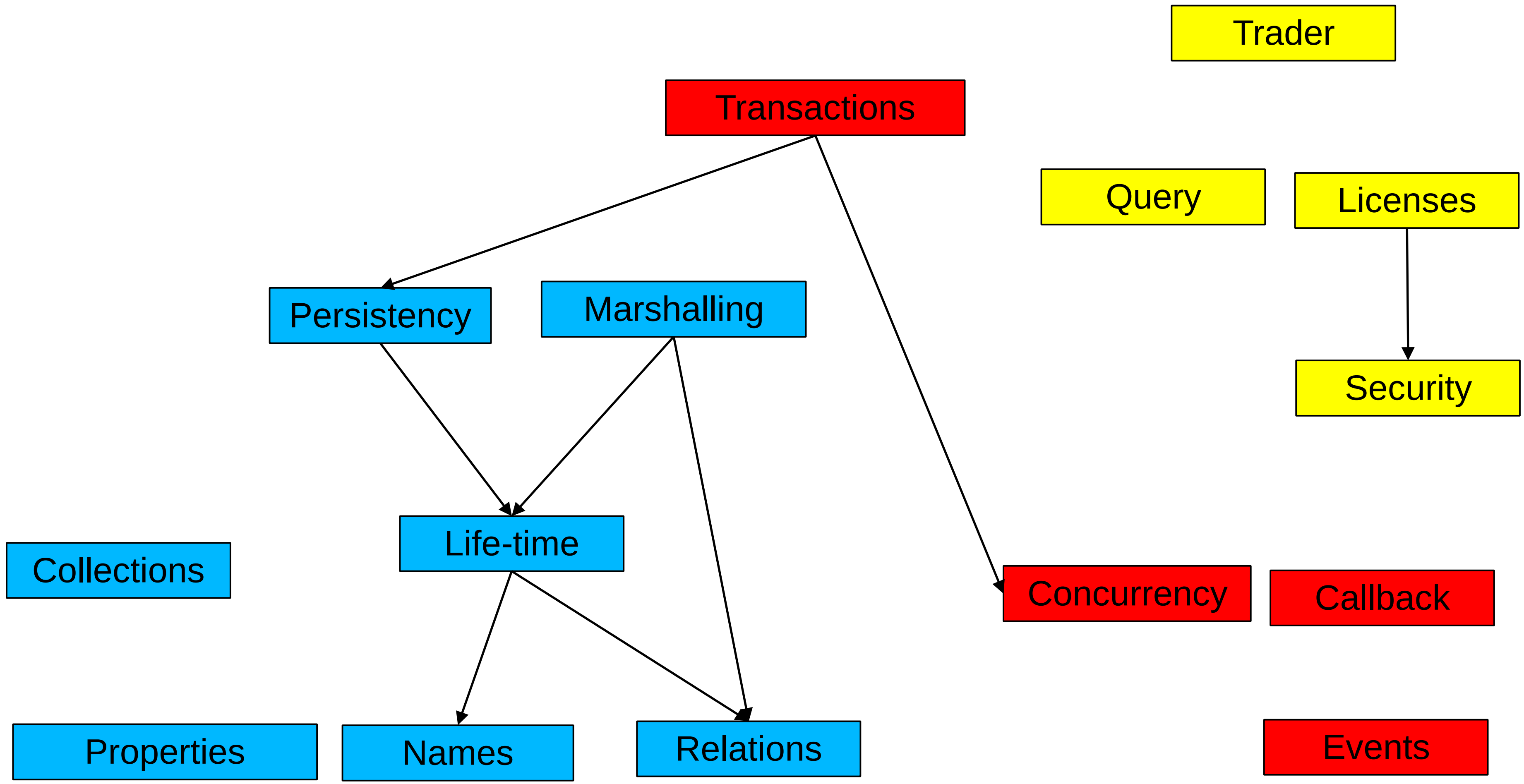
Collaboration Services

- Communication services
  - Resemble connectors in architecture systems, but cannot be exchanged to each other
  - Event service
    - push model: the components push events into the event channel
    - pull model: the components wait at the channel and empty it
  - Callback service
- Parallelism
  - Concurreny service: locks
  - Object transaction service, OTS: Flat transactions on object graphs
    - Nested transactions?

Business Services

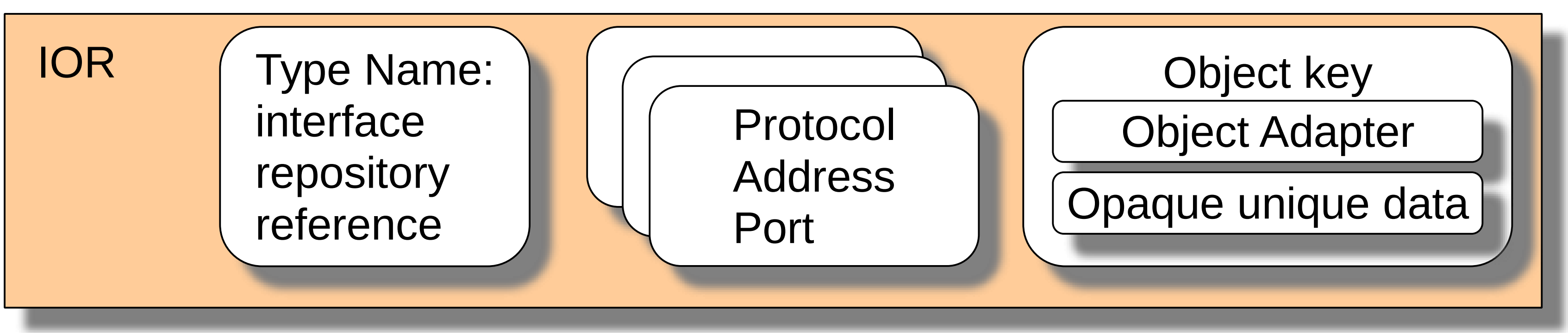
- Trader service
  - Yellow Pages, localization of services
- Query service
  - Search for objects with attributes and the OQL, SQL (ODMG-93)
- Licensing service
  - For application providers (application servers)
  - License managers
- Security service
  - Use of SSL and other basic services

Dependencies Between the Services

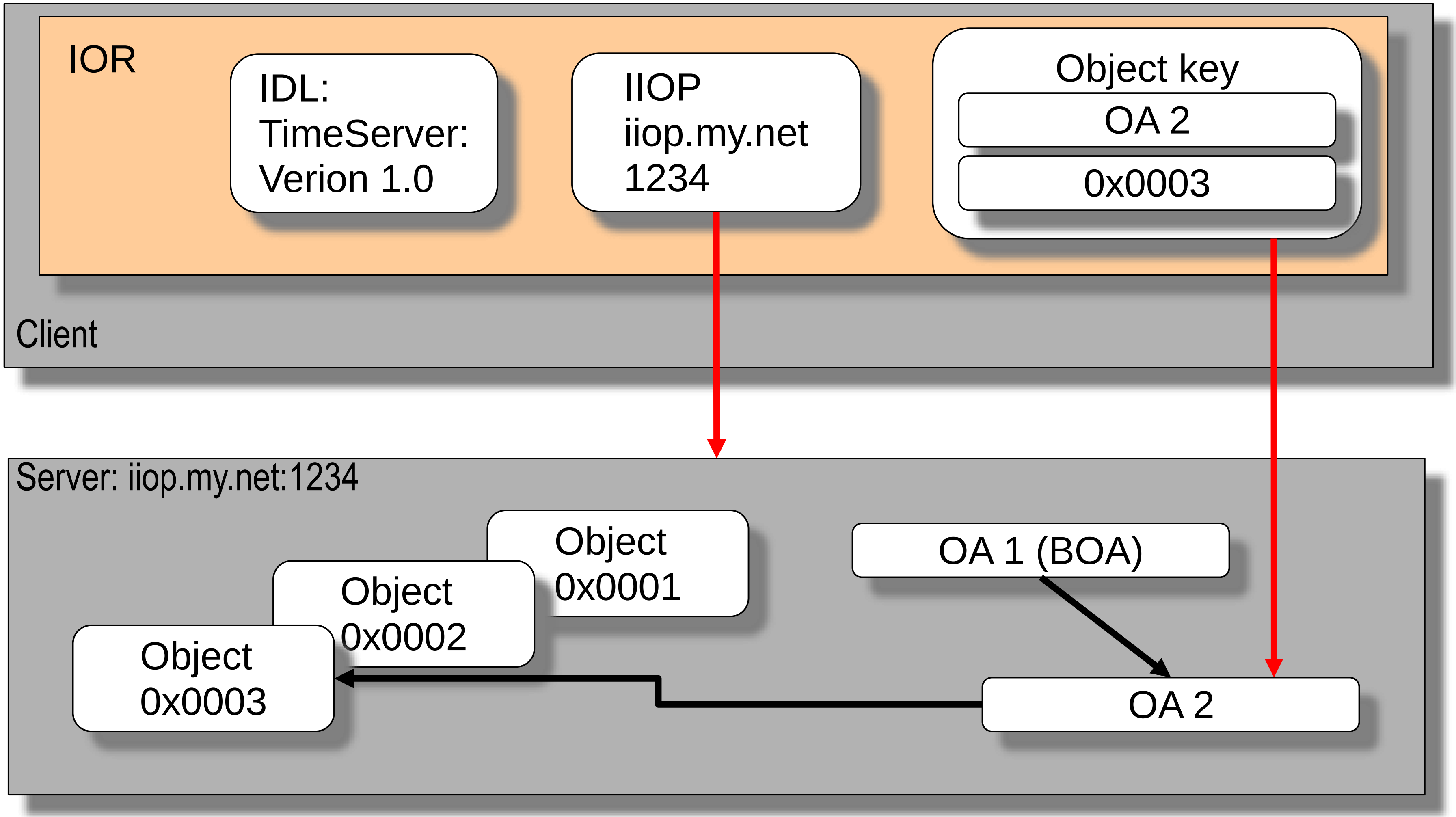


Example: CORBA Interoperable Object Reference – IOR

- A unique key for an object
  - Uniquely mapped per language (for all ORBs)
  - Hides object references of programming languages
- Consists of:
  - Type name (code), i.e., index into Interface Repository
  - Protocol and address information (e.g., TCP/IP, port #, host name), could support more than one protocol
  - Object key:
    - Opaque data only readable by generating ORB (pointer)
    - Object decorator (adapter) name (for BOA)



IOR Example


IOR
IDL:
TimeServer:
Verion 1.0
IIOP
iiop.my.net
1234
Object key
OA 2
0x0003
Client
Server: iiop.my.net:1234
Object
0x0001
Object
0x0002
Object
0x0003
OA 1 (BOA)
OA 2

Object Services: Names

- Binding of a name associates a name to an object in a name space (directory, scope, naming context)
  - A name space is an associative array with a set of bindings of names to values
  - Namespaces are recursive, i.e., they can reference each other and build name graphs
  - Others: Active Directory, LDAP
- The representation of a name is based on abstract syntax, not on the concrete syntax of a operating systemor URL.
  - A name consists of a tuple (Identifier, Kind).
  - The identifier is the real name, the Kind tells how the name is represented (e.g., c_source, object_code, executable, postscript,..).
  - For creation of names there is a library (design pattern Abstract Factory).

Name Service CosNaming


```
CosNaming::NamingContext
--------------------------------------------------
bind(in Name n, in Object obj) // associate a name
rebind(in Name n, in Object obj)
bind_context
rebind_context
mk_name(String s)
Object resolve
unbind(in Name n)  // disassociate a name
NamingContext new_context;
NamingContext bind_new_context(in Name n)
void destroy
void list(..)
_narrow()
```

Name Service


```
void bind(in Name n, in Object obj)
  raises(NotFound, Cannotproceed, InvalidName, AlreadyBoand);
void rebind(in Name n, in Object obj)
  raises(NotFound, Cannotproceed, InvalidName );
void bind_context(in Name n, in NamingContext nc)
  raises(NotFound, Cannotproceed, InvalidName, AlreadyBoand );
void rebind_context(in Name n, in NamingContext nc)
  raises( NotFound, Cannotproceed, InvalidName );
Name mk_name(String s);
Object resolve(in Name n)
  raises( NotFound, Cannotproceed, InvalidName );
void unbind(in Name n)
  raises( NotFound, Cannotproceed, InvalidName );
NamingContext new_context();
NamingContext bind_new_context(in Name n)
  raises( NotFound, AlreadyBoand, Cannotproceed, InvalidName );
void destroy()
  raises( NotEmpty );
void list(in unsigned long how_many,
       out BindingLis bl, out Bindingeserator bi );
```

Name Service in IDL


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

  enum BindingType { nobject, ncontext };
  struct Binding {
    Name binding_name;
    BindingType binding_type;
  };
  typedef sequence <Binding> BindingList;

  interface BindingIterator;
  interface NamingContext {
    enum NotFoundReason { missing_node,
not_context, not_object };
    exception NotFound {
      NotFoundReason why;
      Name rest_of_name;
    };
}
```

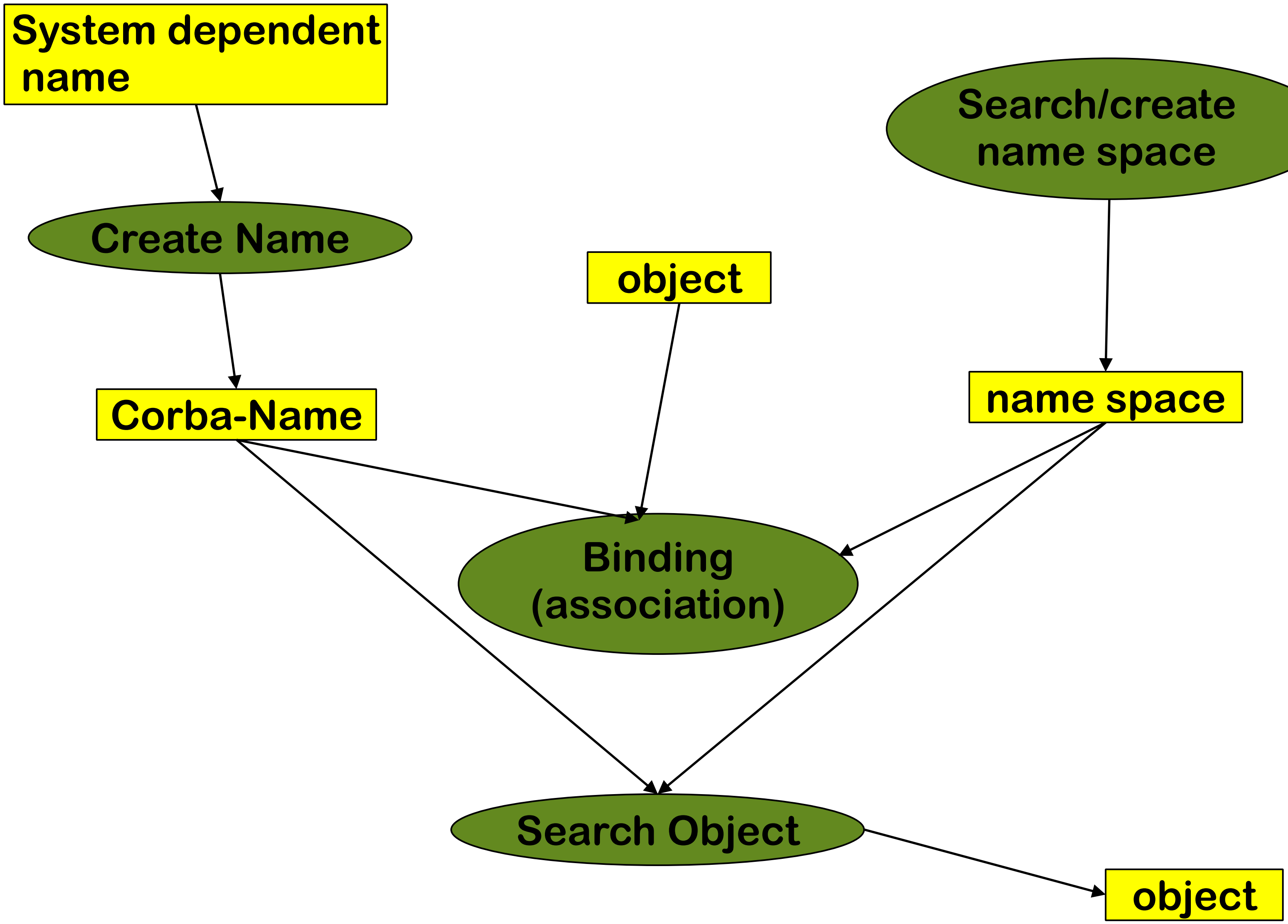
```
    exception Cannotproceed {
      NamingContext cxt;
      Name rest_of_name;
    };
    exception InvalidName {};
    exception AlreadyBoand {};
    exception NotEmpty {};

    // methods see previous slide
  };

  interface BindingIterator {
    boolean next_one(out Binding b);
    boolean next_n(in unsigned long
                      how_many,
                   out BindingLis bl);
    void destroy();
  };
```

...

Use of Names


System dependent name
Create Name
Corba-Name
object
Search/create name space
name space
Binding (association)
Search Object
object

Name Service: Example


```
// From: Redlich

import java.io.*;
import java.awt.*;
import IE.Iona.Orbix2.CORBA.SystemException;  // OrbixWeb
import CosNaming.NamingContext;    // name service/context
import CosNaming.NamingContext.*; // name service/Exceptions
import Calc5.calc.complex;         // Typ 'complex' from Calc5

class MyNaming extends CosNaming {
...
}
public class client  extends Frame  {
  private Calc5.calc.Ref calc;
  private TextField inR, inI;
  private Button setB, addB, multB,
divB, quitB, zeroB;

  public static void main(String argv[])
  {
     CosNaming.NamingContext.Ref  cxt;
     Calc5.calc_factory.Ref       cf;
     Frame f;
    try {
          cxt= NamingContext._narrow( MyNaming.
          resolve_initial_references(MyNaming.NameService));

          cf = Calc5.calc_factory._narrow(
                 cxt.resolve(MyNaming.mk_name("calcfac")));

          f = new client(cf.create_new_calc());
          f.pack();
          f.show();
      }
      catch (Exception ex)  {
          System.out.println("Calc-5/Init:" + ex.toString());
      }
  }
```

Object Services: Persistency

- Definition of a Persistent Object Identifier (PID)
  - references the *value* of CORBA-objects (in contrast to a CORBA-object)
- Interface
  - connect, disconnect, store, restore, delete
- Attachment to data bases possible (also ODMG compatible)

Object Services: Property Service

- Management of lists of features (properties) for objects
  - Properties are strings
  - Dynamically extensible
- Concept well-known as
  - LISP property lists, associative arrays, Java property classes
- Iterators for properties
- Interface:
  - define_property, define_properties, get_property_value, get_properties, delete_property,

Collaboration Services: Transactions

- What a dream: the Web as data base with nested transactions. Scenarios:
  - Accounts as Web-objects. Transfers as Transaction on the objects of several banks
  - Parallel working on web sites: how to make consistent?
- Standard 2-phase commit protocol:
  - begin_ta, rollback, commit
- Nested transactions
  - begin_subtransaction, rollback_subtransaction, commit_subtransaction

Appendix
CORBA Facilities
(Standards for Application Domains)

Horizontal Facilities

- User interfaces
  - Printing, Scripting
  - Compound documents: since 1996 OpenDoc is accepted as standard format. Source Code has been released of IBM
- Information management
  - Metadata(meta object facility, MOF)
  - Tool interchange: a text- and stream based exchangeformat for UML (XMI)
  - Common Warehouse Model (CWM): MOF-based metaschema for database applications

Vertical Facilities (Domain-Specific Facilities)

- The Domain technology committee (DTC) creates domain task forces DTF for a application domain
- Business objects
- Finance/insurance
  - Currency facility
- Electronic commerce
- Manufacturing
  - Product data management enablers PDM
- Medicine (healthcare CorbaMed)
  - Lexicon Query Service
  - Person Identifier Service PIDS
- Telecommunications
  - Audio/visual stream control object
  - Notification service
- Transportation

CORBA Facilities and UML Profiles

- Since 2000, the OMG describes domain-specific vocabularies with UML profiles
  - Probably, all CORBA facilities will end up in UML profiles
- A UML Profile is a UML dialect of a application specific domain
  - With new stereotypes and tagged values
  - Corresponds to an extension of the UML metamodel
  - Corresponds to a domain specific language with own vocabulary
  - Every entry in profile is a term
- Example UML Profiles:
  - EDOC Enterprise Distributed Objects Computing
  - Middleware profiles: Corba, .NET, EJB
  - Embedded and real time systems:
    - MARTE profile on schedulability, performance, time
    - Ravenscar Profile
    - HIDOORS Profile on real-time modelling www.hidoors.org

Appendix CORBA and the Web

Corba and the Web

- HTML solves many of the CORBA problems
- HTTP only for data transport
  - HTTP cannot call methods, except by CGI-Gateway-functionality (common gateway interface)
  - Behind the CGI-interface is a generals program, communicating with HTTP with untyped environment variables (HACK!)
  - http-Server are simple ORBs, pages are objects
  - The URI/URL-name schema can be integrated into CORBA
- IIOP becomes a standard internet protocol
  - Standard ports, URL-mappings and Standard-proxies for Firewalls are available
- CORBA is an extension of HTTP of data to code

CORBA and Java

- Java is an ideal partner for Corba :
  - Bytecode is mobile, i.e.,
    - Applets: move calculations to clients (thin/thick client problem)
    - can be used for migration of objects, ORBs and agents
  - Since 1999 direct Corba support in JDK 1.2
    - IDL2Java mapping, IDL compiler, Java2IDL compiler, name service, ORB
  - Corba supports for Java a distributed interoperable infrastructure
- Java imitates functionality of Corba
  - Basic services: Remote Method Invocation RMI, Java Native code Interface JNI
  - Services: serialization, events
  - Application specific services (facilities): reflection, properties of JavaBeans

Corba and the Web (Orblets)

- ORBs can be written as bytecode applets if they are written in Java (ORBlet)
- Coupling of HTTP and IIOP: Download of an ORBlets with HTTP: Talk to this ORB, to get contact to server
- Standard web services (see later) are slower than CORBA/ORBlets, because they incur interpretation overhead