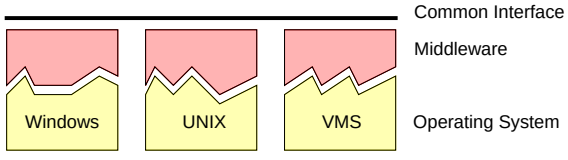
# E Distributed Objects in CORBA

- ★ Motivation
- Location
  - ◆ Transparency of location
- Heterogeneity
  - ◆ Different hardware,
  - ◆ different operating systems, and
  - ◆ different programming languages are used in distributed systems.
  - ◆ Transparency of heterogeneity
- Services
  - ◆ Name server
  - ◆ Time server
  - ◆ ...




# E.1 Middleware Approach

- Middleware: a piece of software between operating system and application




  - ◆ Middleware provides services for distributed programming
- **We need:** middleware for distributed object-based programming
- ★ CORBA – Common Object Request Broker Architecture, a standard of the OMG




# E.2 CORBA Overview

## 1 Design Goals

- CORBA is a standard
  - ◆ There are standard documents
  - ◆ Definition of CORBA compliance
  - ◆ Vendors build implementations of the standard (e.g., VisiBroker, OmniBroker, Orbix, etc.)
- CORBA wants to abstract from
  - ◆ hardware,
  - ◆ operating system, and
  - ◆ programming language.

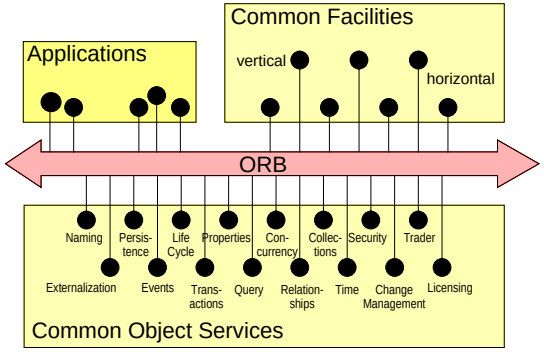



## 1 Design Goals (2)



- Interoperability
  - ◆ An application shall run on different CORBA implementations without major changes
  - ◆ Applications on different CORBA implementations shall be able to communicate
- Embedding of legacy applications
  - ◆ Legacy applications can be encapsulated and can act as CORBA objects
- Vision of business objects or components
  - ◆ CORBA objects represent all kind of data and proceedings of a company

## 2 Architectural Overview




- Object Management Architecture – OMA






## 3 CORBA Implementations

- A CORBA implementation must contain
  - the implementation of the core architecture
  - one language mapping (e.g., for C++)
- A CORBA implementation may contain
  - an arbitrary number of services
  - functionality for interoperability with other CORBA implementations (GIOP/IIOP)
- The implementation is free **how** it realizes the requirements of the standard
  - Many different implementations are possible
    - Daemon-based
    - Library-based
    - ...





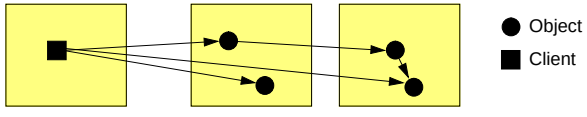

## 2 Architectural Overview (2)

- Application objects
  - Whole object approach for distributed objects
  - Client/server architecture
- Object Request Broker (ORB)
  - Communication backbone
  - Enables objects to communicate
- CORBA Services
  - Basic services for distributed programming
  - Extends the ORB's functionality
  - Services look like objects
- CORBA Facilities
  - Application-specific services




## E.2 Application Objects

- Distributed Objects
  - Identity
  - State
  - Operations (methods)
  - CORBA objects can be invoked (implement server side)
  - CORBA objects may act as clients




- Clients need not to be CORBA objects
  - CORBA clients can be processes, etc.

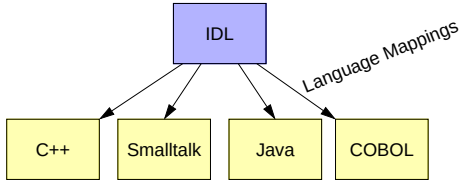
## E.2 Application Objects (2)

- Distributed objects form an application
  - Objects cooperate by communicating with each other
- ★ Example: printer management system
  - Client objects
  - Spooler objects
  - Printer objects
- "Whole object approach"
  - Client-side stub can contact the server object




## 1 Interface Definition Language (IDL)

- Language for describing object interfaces
  - Independent of implementation language of an object
  - Language mapping defines how IDL primitives are mapped to the concepts of a specific programming language
  - Language mapping is part of the CORBA standard
  - Language mappings exist for C, C++, Smalltalk, COBOL, and Java
  - IDL is similar to C++







## 1 Interface Definition Language (2)

- Example

```
module MyModule
{
    interface MyInterface
    {
        attribute int lines;
        void printLine( in string toPrint );
    };
};
```

IDL to C++

```
namespace MyModule {
    class MyInterface : ... {
        public:
         virtual CORBA:Long lines();
         virtual void lines( CORBA::Long _val );
         void printLine( const char *toPrint );
        ...
    };
};
```




## 1 Interface Definition Language (3)

- Example

```
module MyModule
{
    interface MyInterface
    {
        attribute int lines;
        void printLine( in string toPrint );
    };
};
```

IDL to Java

```
public interface MyModule.MyInterface extends ... {
    public int lines();
    public void lines( int lines );
    public void printLine(java.lang.string toPrint );
    ...
};
```

## 1 Interface Definition Language (4)

★ Advantages
- ◆ Transparency of implementation language
- ◆ Enables interoperability

▲ Disadvantages
- ◆ Object interface must be defined in the target language **and** in IDL
- ◆ IDL is quite expressive
  - Language that do not provide the right mechanisms will have a complex language mapping
- ◆ If a language has special features they cannot be used because they are not part of IDL

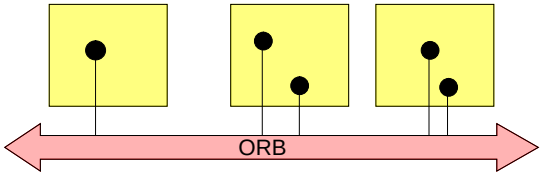



## 2 Process of Creation and Binding

- Creation of a server object
  - ◆ Description of the object interface in IDL
  - ◆ Programming the server object in an implementation language
  - ◆ Registration of the object at the ORB (the OA respectively)
    - The ORB creates an Interoperable Object Reference (IOR)
- Binding to the server object at client side
  - ◆ Retrieval of an object reference to the server
    - Result of a name-server query
    - Return parameter of a method call
    - Retrieval from outside of the system: user knows the reference as a string (IORs can be converted to strings and back)
  - ◆ Creation of the client stub
  - ◆ Method invocations using the stub object




## E.3 Object Request Broker – ORB

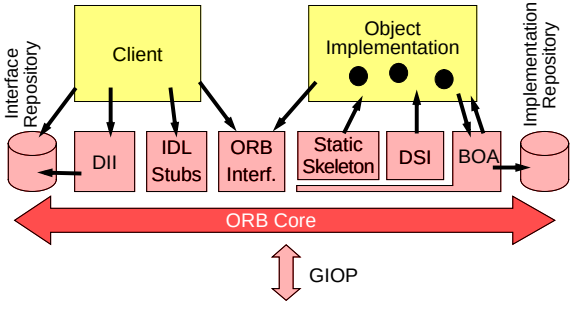



- The ORB is the communication backbone of a CORBA implementation
- All communication is handled by the ORB
  - ◆ ... within an address space / process
  - ◆ ... between address spaces / processes
  - ◆ ... between address spaces / processes of different ORBs
- The ORB implements location transparency




## E.3 Object Request Broker (2)

- Central components of an ORB



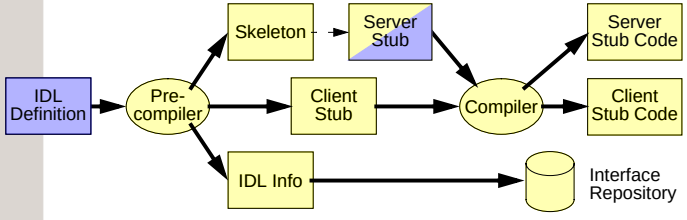
## 1 Static Stubs

- Stubs on client and server side
  - ◆ As soon as we exactly know the interface of an object, we can create static stubs from it.
  - ◆ Static stubs are automatically created from the IDL description
  - ◆ Server side stubs are called *skeletons*
    - CORBA skeletons have to be filled with the actual implementation
- Tasks of the stubs
  - ◆ Marshalling of parameters
  - ◆ Send and receive of request and reply messages using the ORB core




## 1 Static Stubs (2)

- Stub creation




- ◆ From IDL, a precompiler generates a number of output files.
- ◆ The skeleton contains a frame for filling in the actual methods of the object implementation. The skeleton already contains the server-stub functionality.
- ◆ The client-stub file contains the code for the client stub.
- ◆ An additional file may contain data for the interface repository.
- ◆ Usually the stubs have to be compiled by the target-language compiler.



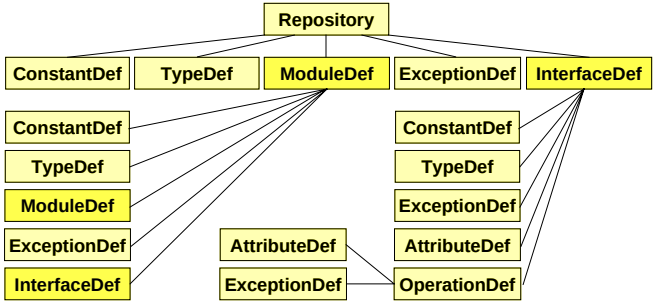

## 2 Interface Repository

- Data base for interface definitions in IDL
- Query the data base for
  - ◆ Type checking
    - Does the stub's type match the object's type?
    - Inter-ORB operations: insertion into multiple repositories
  - ◆ Retrieval of meta data by clients and tools
    - Dynamic invocations
    - Debugging
    - Class browser
  - ◆ Implementing method *get_interface* of each object
- Writing to the data base using
  - ◆ IDL compiler
  - ◆ Write methods




## 2 Interface Repository (2)

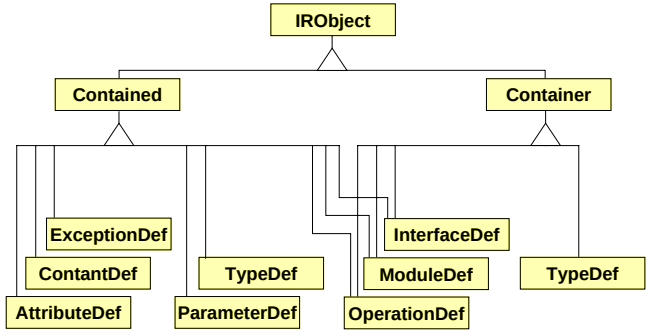
- Stored information / composition of an IDL file




- ◆ Hierarchy of the components of an IDL file, and of the interface repository respectively

## 2 Interface Repository (3)

- Inheritance hierarchy of the IDL interfaces of IR objects







## 2 Interface Repository (4)

- Standard types are represented by so-called *type codes*

**TypeCode**

  - Representation of basic types: **int, float, boolean**
  - Representation of composite standard types: **union, struct, enum**
  - Representation of template types and complex declarators: **sequence, string, array**

- Type codes represent type and structure as objects with an standard IDL interface
  - Operation for comparing type codes
  - Operation for obtaining the description of the type



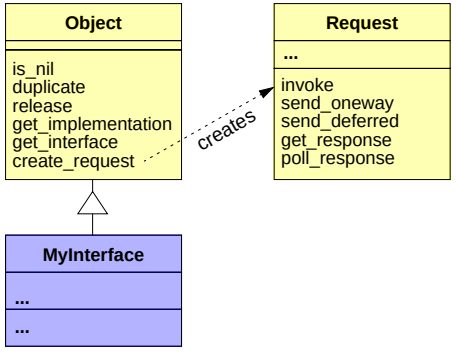

## 3 Dynamic Invocation Interface (DII)

- DII allows the invocation of methods whose object interface was not known at compile time
  - Interface description can be retrieved using the interface repository
- Single steps of an invocation (which have to be mapped by the language binding)
  - Retrieve signature of the method from the interface repository
  - Create list of parameters
  - Compose request
  - Invoke method
    - with RPC (synchronous call)
    - with asynchronous RPC (asynchronous call)
    - with datagramm message (no reply, no reliability)




## 3 Dynamic Invocation Interface (2)

- IDL definitions for the DII



## 4 Dynamic Skeleton Interface (DSI)

- DSI allows a CORBA server to accept method invocations for objects whose interfaces are only dynamically retrievable, e.g., in
  - Bridges to other ORBs
  - CORBA-encapsulated data bases
  - Dynamically created objects and interfaces
- The DSI identifies the object for which the invocation is made for
  - A call-back function is invoked that gets all the necessary information
  - The call-back function has to invoke the corresponding object
- ★ **Note:**
  DII and DSI are not distinguishable from static stubs, i.e., they are fully interoperable with static stubs.



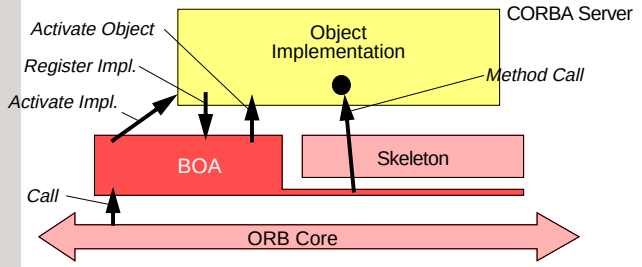

## 5 Object Adaptor

- The object adaptor is a local representative of ORB services
  - It generates object references (for new objects)
  - It maps object references to implementations
  - It forwards incoming method calls
  - It authenticates the caller (implementing CORBA's security functionality)
  - It activates and deactivates the object and its implementation
  - It registers server classes within the implementation repository
- CORBA defines the mandatory Basic Object Adaptor
  - Basic functionality
- Another example: an OODB Adaptor
  - Connecting an object-oriented data base to CORBA
  - The OODB Adaptor converts all objects in the data base to CORBA objects




## 6 Basic Object Adaptor (BOA)

- Activation and deactivation of CORBA servers
  - Implementation of an objects is not always active in the sense of the operating system




- **Note:** Do not mix up server objects with CORBA servers.

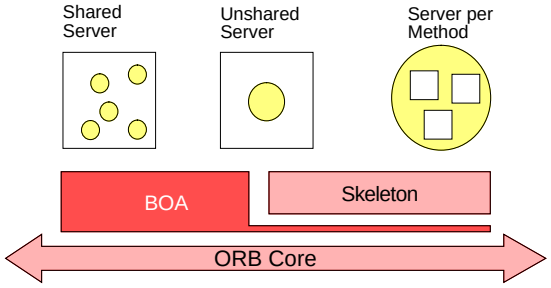



## 6 Basic Object Adaptor (2)

- Activation strategies
  - **Shared Server**
    A server process contains multiple objects of an implementation
  - **Persistent Server**
    Activation strategy is realized outside of the BOA; server is always running (persistent); server process hosts multiple objects of an implementation
  - **Unshared Server**
    One server process per object instance
  - **Server per Method**
    One server process per method invocation

## 6 Basic Object Adaptor (3)

- Activation strategies




  - Circles are objects
  - Squares are CORBA servers




## 7 Implementation Repository

- Data base for object implementations
  - Data to localize and activate object implementations
  - Information for debugging and administration
  - ...
- Implementation repository is highly implementation dependent.
- Query the data base
  - Implementation of the method *get_implementation* of each object
- Writing the data base by
  - External tools
  - The BOA at the start of a server




## 8 Inter-ORB Communication

- GIOP – General Inter-ORB Protocol
  - Basic protocol for the interaction of two ORBs
  - Common Data Representation (CDR) converts IDL-compliant parameters into a serial byte stream
  - IIOP – Internet Inter-ORB Protocol (GIOP over TCP/IP; implementation is mandatory)
  - Other implementations of GIOP are possible
- ESIOP – Environment Specific Inter-ORB Protocols, e.g., DCE/ESIOP
  - Inter-ORB protocol on the basis of DCE RPC
- IOR – Interoperable Object Reference
  - String representation
  - Several profiles (one of it is IIOP)




## E.4 Services

- Basic services of a distributed system as extensions of the ORB
  - Naming service
  - Transaction service
  - Persistent object service
  - Life cycle service
  - ...
- Services are accessible by IDl interfaces
  - No additional primitives necessary; method invocation is enough

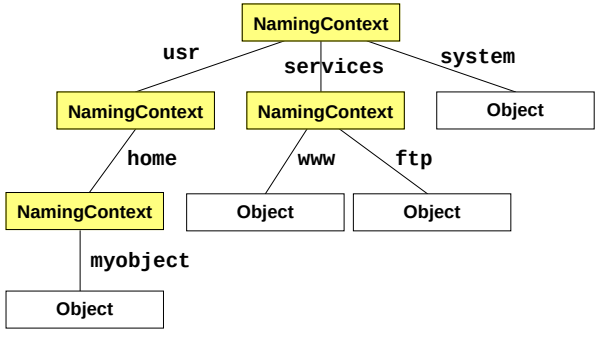
## 1 Naming Service

- CORBA defines an hierarchical naming service similar to the UNIX file system
  - Name space forms a tree
  - Names consist of multiple components (syllables)
    e.g., < "usr"; "home"; "myobject" >
    (UNIX: "/usr/home/myobject" )
  - Interface to the naming service is defined by IDL




## 1 Naming Service (2)

- Example of a name tree

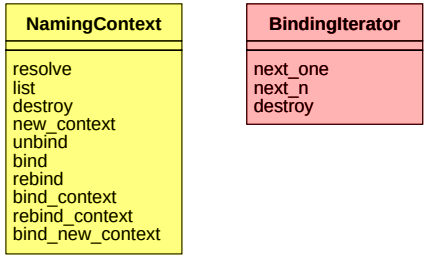






## 1 Naming Service (3)

- Interface of a Naming Context and the Binding Iterator

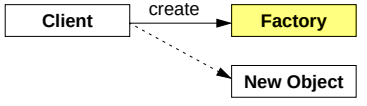



  - Naming Context acts like a directory
  - Objects or other contexts can be bound to a name
  - Listing the context returns an array and an iterator that allows you to look at the next entries




## 2 Life Cycle Service

- Controls the life cycle of an object
  - Creation
  - Copying, Migration
  - Deletion
- Life Cycle Service defines the common interface for controllable objects
- ★ Modell des Lebenszyklus
  - Creation of an Object:

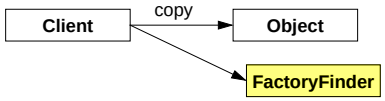



Call of a create method at a factory object

  - Protocol and interface of the factory is not defined and probably dependent on the kind of object to create

## 2 Life Cycle Service (2)

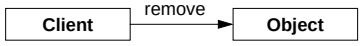
◆ Copying and migration:




FactoryFinder determines the location

- Locations can be a specific computer or a group of computers, etc.

◆ Deletion:

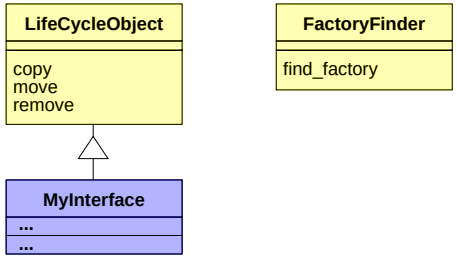



- With the call of remove the object is deleted.




## 2 Life Cycle Service (3)

■ Objects need to implement the LifeCycleObject interface




◆ copy and move need a FactoryFinder object to find a factory object which in turn will create the copy or migrated object




## 2 Life Cycle Service (4)

■ Example: implementation of the copy meethod

◆ Client invokes copy method and passes a reference to a FactoryFinder

◆ The copy method is provided by the programmer **or** by the CORBA implementation

◆ The copy methode calls the FactoryFinder and selects a suitable factory that is used to create a new object

◆ The new object will be initialized with the data of the existing object

▲ To the outside: clearly defined interface

▲ To the inside: open and implementation dependent, e.g., protocol between copy method and factory

■ In CORBA 2.0 relationship between objects is considered for copying and mobility: part-of relationship, etc.

◆ Parts of the object are also copied or migrated (deep copy vs. shallow copy)