D50: Advances in Software Engineering
Distributed Objects

Wolfgang Emmerich

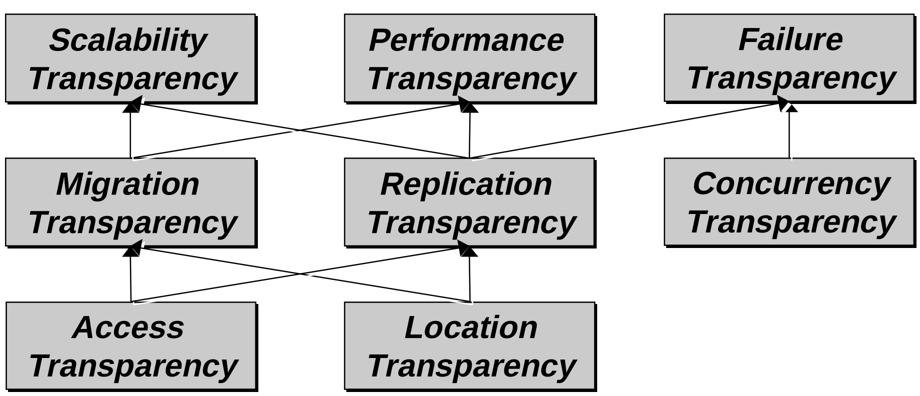
Lecture Overview

- *Transparency*
- *OO MIddleware*
- *Resolving Language Heterogeneity*
- *Resolving Data Heterogeneity*
- *OMG/CORBA*
- *Genericity*

Transparency

- Users and application programmers perceive distributed system as a whole rather than a collection of components
- Transparency has multiple dimensions that were identified by ANSA [ANSA89] and in the ISO ODP Standard [ISO92]

Transparency Dimensions



Access Transparency

- *Enables local and remote information objects to be accessed using identical operations*
- *Examples*
  - *File system operations in NFS*
  - *Navigation in the Web*
  - *SQL queries*

Location Transparency

- *Enables information objects to be accessed without knowledge of their location*
- *Examples*
  - *Files in NFS*
  - *Pages in the Web*
  - *Tables in distributed databases*

Concurrency Transparency

- ***Enables several processes to operate concurrently using shared information objects without interference between them***
- ***Examples***
  - ***Automatic teller machine network***
  - ***Database management system***

Replication Transparency

- ***Facilitates use of multiple instances of information objects to increase reliability and performance without knowledge of the replicas by users or application programs***
- ***Examples***
  - ***Distributed DBMS***
  - ***Mirroring Web pages.***

Failure Transparency

- **_Enables concealment of faults_**
- **_Allows users and applications to complete tasks despite failure of other components._**
- **_Example_**
  - **_Database Management System_**

Migration Transparency

- **_Allows movement of information objects within system without affecting operations of users or application programs_**
- **_Examples_**
  - **_NFS_**
  - **_Web Pages_**

Scaling Transparency

- ***Allows the system and applications to expand in scale without changing system structure or application algorithms.***
- ***Examples***
  - ***World-Wide-Web***
  - ***Distributed Databases***

Distribution Middleware Needed

Requirements for middleware:

- ***Component type definition***
  - ***Services offered by components***
  - ***Component state***
  - ***Relationships between components***
- ***Resolution of heterogeneity***
  - ***Platforms***
  - ***Programming languages***
  - ***Networks***
- ***Support in achieving transparency***

What is Middleware?

- ***Layered between Application and OS/Network***
- ***Makes distribution transparent***
- ***Resolves heterogeneity of***
  - ***Hardware***
  - ***Operating Systems***
  - ***Networks***
  - ***Programming Languages***
- ***Provides development and run-time environment for distributed systems.***

Categories of Middleware

- ***Message-Oriented Middleware***
  - ***IBM MQSeries***
  - ***DEC Message Queue***
  - ***NCR TopEnd***
- ***Transaction-Processing Middleware***
  - ***IBM CICS***
  - ***BEA Tuxedo***
  - ***Encina***
- ***Object-Oriented Middleware***
  - ***OMG/CORBA***
  - ***DCOM***
  - ***Java/RMI***
- ***These are converging! We focus on OO.***

ISO/OSI Reference Model

Application
Presentation
Session
Transport
Network
Data link
Physical

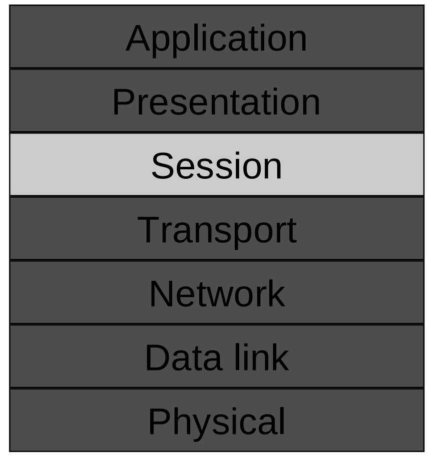
Transport Layer

- ***How are we going to transmit object requests between hosts?***
- ***Two facets in UNIX networks:***
  - ***TCP and***
  - ***UDP.***

Application
Presentation
Session
Transport
Network
Data link
Physical

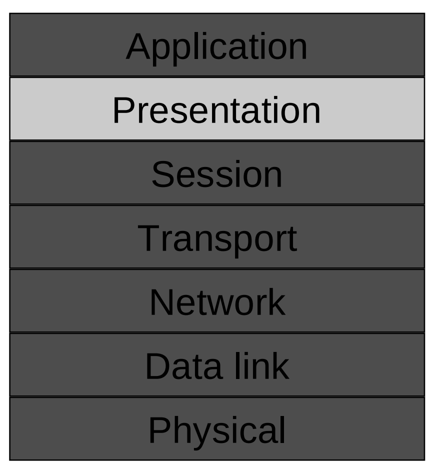
ISO/OSI Session Layer

- ***Which object runs on which machine?***
- ***Layering object request on top of transport***
- ***Activating objects***
- ***Object Adapters and Registries***



ISO/OSI Presentation Layer

- ***At application layer: complex data types & Object references***
- ***How to transmit complex values through transport layer?***
- ***Presentation layer issues:***
  - ***Complex data structures and***
  - ***Heterogeneity.***



Complex Data Structures

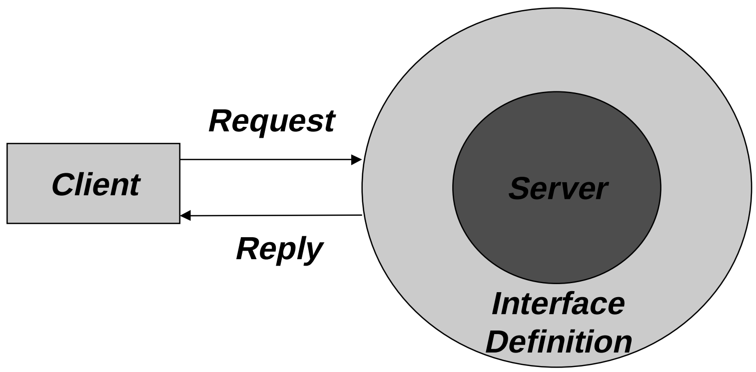
- **Marshalling: Disassemble data structures into transmittable form**
- **Unmarshalling: Reassemble the complex data structure.**

```
class Person {
 private:
  int dob;
  char * name;
 public:
  char * marshal() {
    char * msg;
    msg=new char[strlen(name)+10];
    sprintf(msg,"%d,%d,%s", dob,
            strlen(name),name);
    return(msg);
  };
};
```


Type Safety

- **How can we make sure that**
  - **servers are able to perform operations requested by clients?**
  - **actual parameters provided by clients match the expected parameters of the server?**
  - **results provided by the server match the expectations of client?**
- **Middleware acts as mediator between client and server to ensure type safety.**
- **Achieved by interface definition in an agreed language.**

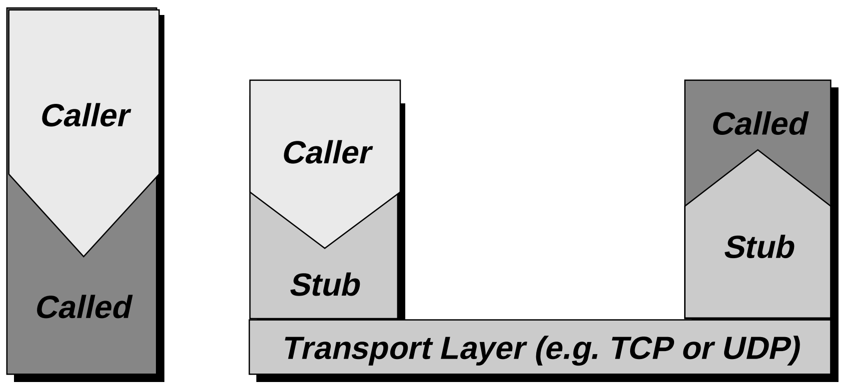
Facilitating Type Safety



Stubs

- *Creating code for marshalling and unmarshalling is tedious and error-prone.*
- *Code can be generated fully automatically from interface definition.*
- *Code is embedded in stubs for client and server.*
- *Client stub represents server for client, Server stub represents client for server.*
- *Stubs achieve type safety.*
- *Stubs also perform synchronization.*

Local Call vs. Remote Request



Synchronization

- ***Goal: achieve similar synchronization to local method invocation***
- ***Achieved by stubs:***
  - ***Client stub sends request and waits until server finishes***
  - ***Server stub waits for requests and calls server when request arrives***

Facilitating Access Transparency

- ***Client stubs have the same operations as server objects***
- ***Hence, clients can***
  - ***make local call to client stub***
  - ***or local call to server object***

  without changing the call.
- ***Middleware can accelerate communication if objects are local by not using the stub.***

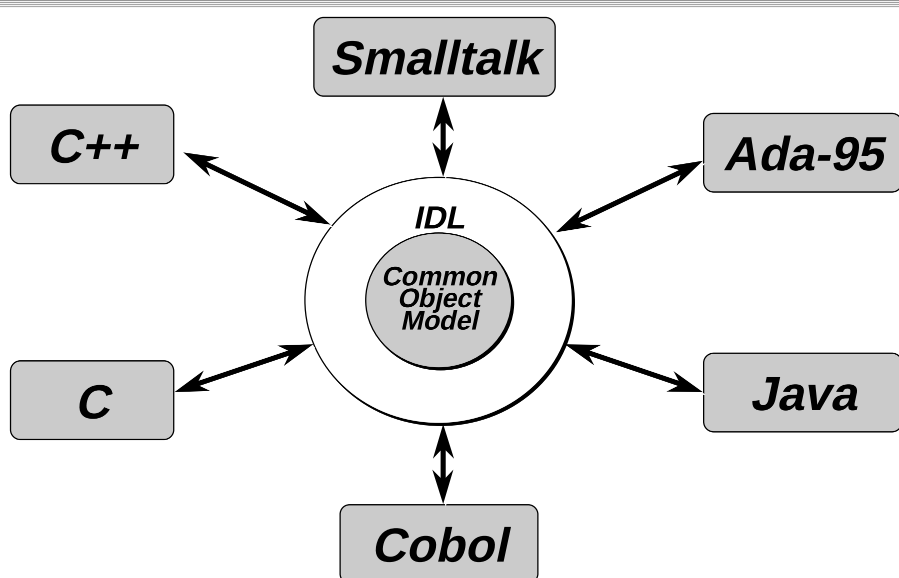
Facilitating Location Transparency

- ***Object identity***
- ***Object references***
- ***Client requests operation from server object identified by object reference***
- ***No information about physical location of server necessary***
- ***How to obtain object references?***

Motivation

- Components of distributed systems are written in different programming languages
- Programming languages may or may not have their own object model
- Object models largely vary
- Differences need to be overcome in order to facilitate integration

Resolving Language Heterogeneity



Purpose of Common Object Model

- ***Meta-model for middleware's type system***
- ***Defines meaning of e.g.***
  - ***object type***
  - ***operation***
  - ***attribute***
  - ***request***
  - ***exception***
  - ***subtyping***
- ***Defined general enough for mappings to most programming languages***

Interface Definition Language

- ***Language for expressing all concepts of the middleware's object model***
- ***Should be***
  - ***programming-language independent***
  - ***not computationally complete***
- ***Bindings to different programming languages are needed***
- ***As an example: OMG object model and OMG/IDL***

Programming Language Bindings

- *Atomic data types / type constructors*
- *Constants*
- *Interfaces and multiple inheritance*
- *Object references*
- *Attribute accesses*
- *Operation execution requests*
- *Exception declaration / handling*
- *Modules*
- *Middleware interface invocations*

Standardisation of Bindings

- *Facilitate portability with respect to:*
  - *Object requests*
  - *Object implementations*
  - *ORB interface invocations*
- *Decrease learning curve of developers*

Data Heterogeneity

- **Hosts of client and server might use different data representation formats. E.g.:**
  - **Mainframes are big-endian**
  - **Unix servers & NT workstations are little-endians**

		n+3	n+2	n+1	n
little-endians	memory	●			
		sign			

		n	n+1	n+2	n+3
big-endians	memory	●			
		sign			

Data Heterogeneity (cont'd)

- **Different programming languages use different data representations, e.g. Character string "abc" in Pascal or C++:**

Pascal	memory	3	a	b	c
C++	memory	a	b	c	\0

Motivation

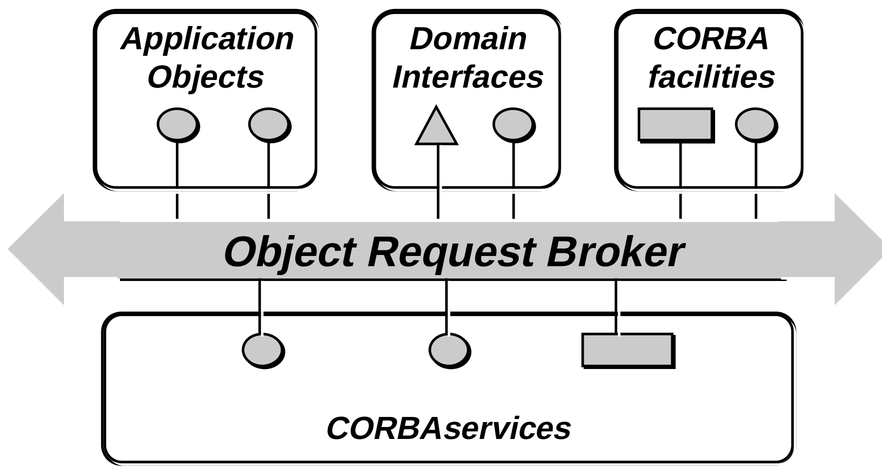
- *Data representations have to be converted between client and server*
- *Conversion should be transparent to application developer*
- *Generally achieved by middleware within presentation layer implementation*

Approaches

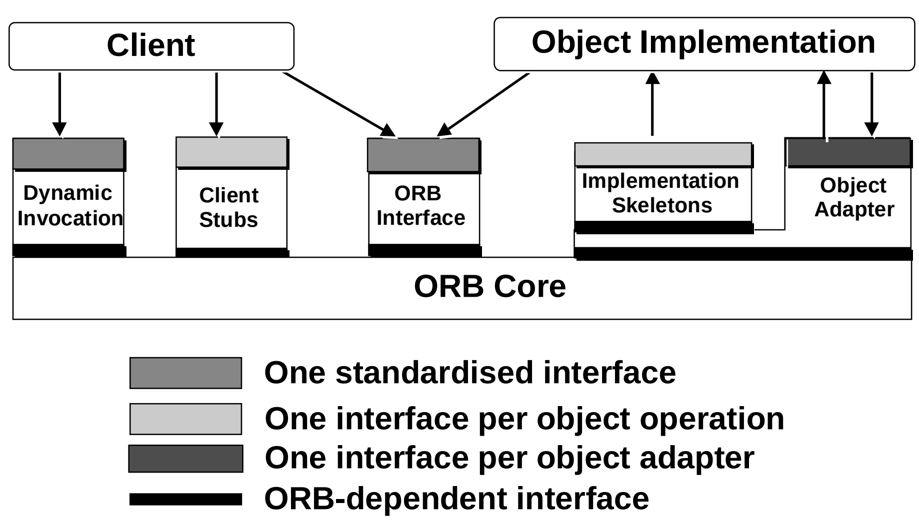
- *Mappings between native representations*
  - *Standardized data representation, e.g.*
    - *Sun's external data representation (XDR)*
    - *OMG's common data representation (CDR)*

    No transmission of the type definition
  - *Transmission of values and their types using an abstract syntax notation e.g*
    - *ASN.1*

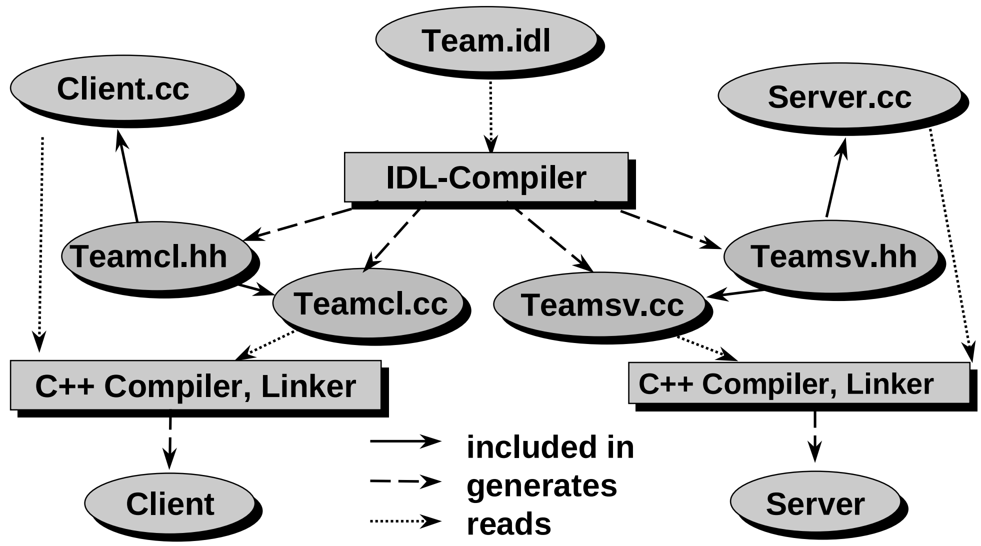
Object Management Architecture



Components involved at run-time



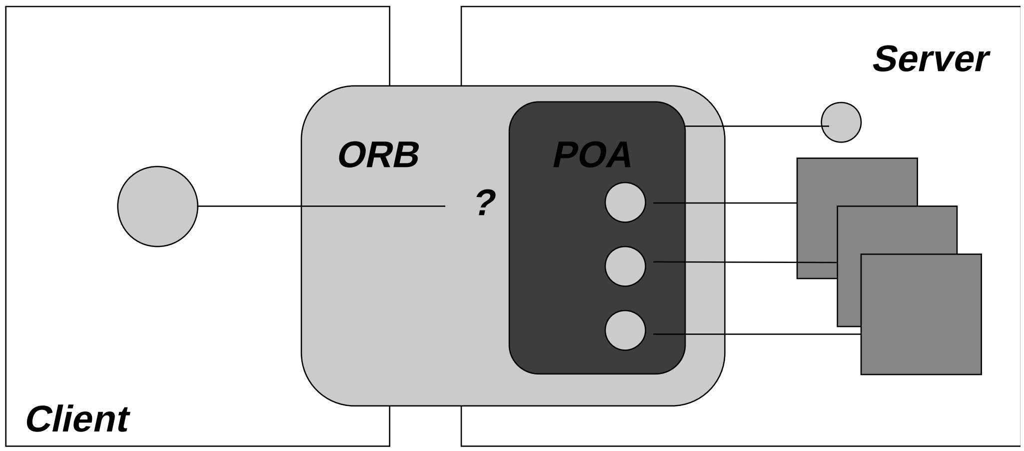
Generation of Stubs/Skeletons



Portable Object Adapter (POA)

- ***Facilitate object implementation portability between different ORBs***
- ***Support light-weight transient objects***
- ***Support persistent object identities (e.g. in ODBMSs)***
- ***Allow servants to implement multiple objects***
- ***Support transparent object activation***
- ***Extensible mechanism for activation policies***
- ***Multiple POAs in one server***

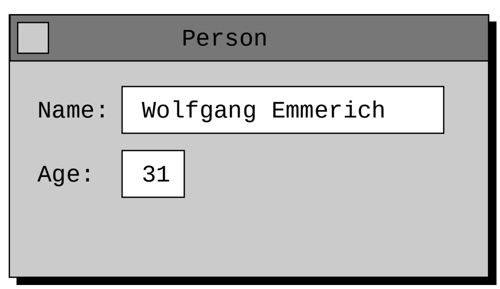
Abstract POA Model



Generic Applications

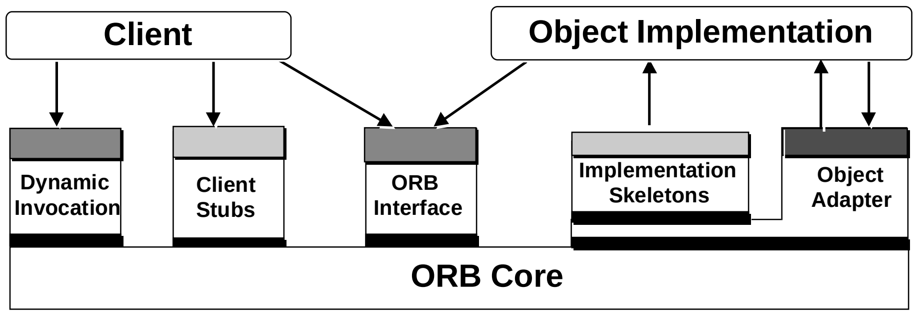
Generic applications use components whose types are not (yet) known.

Example: Object Browser



Static vs. Dynamic Invocation

■ Example: OMG/CORBA



Static Invocation

■ Advantages:

- Requests are simple to define.
- Availability of operations checked by programming language compiler.
- Requests can be implemented fairly efficiently.

■ Disadvantages:

- Generic applications cannot be build.
- Recompilation required after operation interface modification.

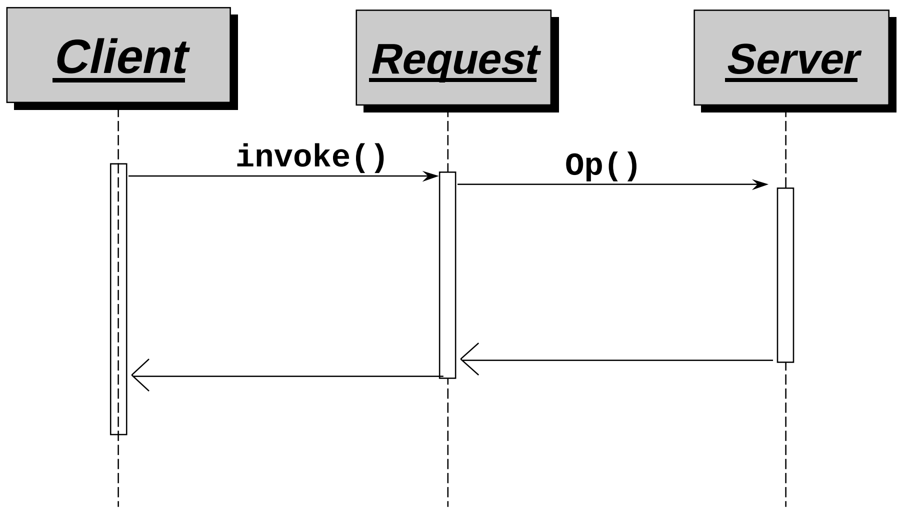
Dynamic Invocation

- *Interface to create operation execution requests dynamically.*
- *Requests are objects.*
- *Attributes for operation name, parameters and results.*
- *Operations to*
  - *change operation parameters,*
  - *issue the request and*
  - *obtain the request results.*

Creation of Requests

```
interface Object {
 ORBstatus create_request (
  in Context ctx,          // operation context
  in Identifier operation,// operation to exec
  in NVList arg_list,      // args of operation
  inout NamedValue result,// operation result
  out Request request      // new request object
  in Flags req_flags       // request flags
  );
  ...
};
```

Synchronous Request



Dynamic Invocation

- **Advantages:**
  - ***Components can be built without having the interfaces they use,***
  - ***Higher degree of concurrency through deferred synchronous execution.***
  - ***Components can react to changes of interfaces.***
- **Disadvantages:**
  - ***Less efficient,***
  - ***More complicated to use and***
  - ***Not type safe!***

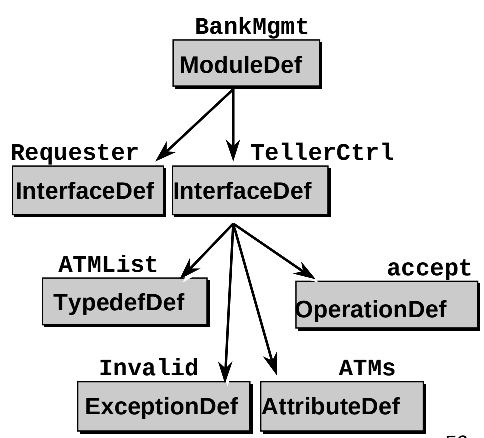
Interface Repository

- Makes type information of interfaces available at run-time.
- Enables development of generic applications.
- Achieves type-safe dynamic invocations.
- Supports construction of interface browser.
- Used by Middleware itself.

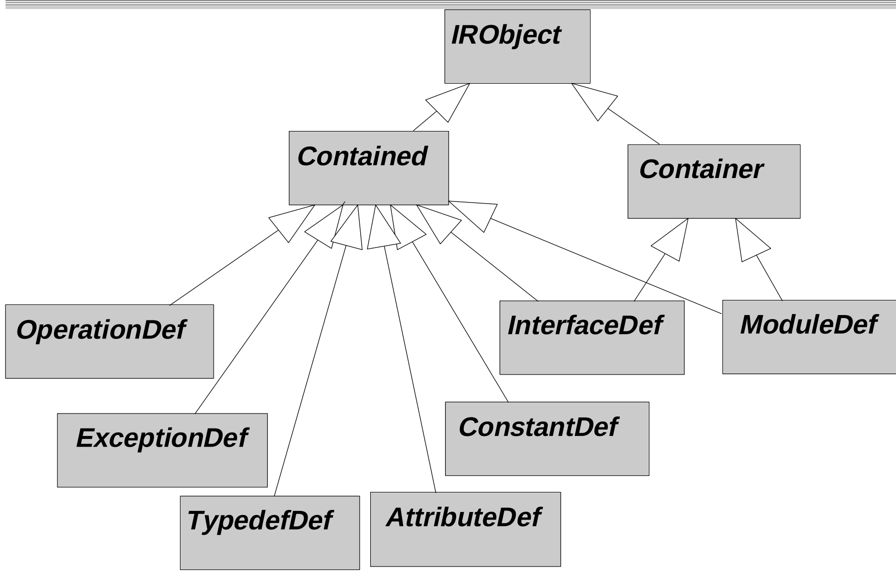
Abstract Syntax Trees (ASTs)

- Interface repository persistently stores ASTs of IDL modules, interfaces, types, operations etc.

```
module BankMgmt {
 interface Requester;
 interface TellerCtrl {
  typedef sequence<ATM>
          ATMList;
  exception Invalid {};
  attribute ATMList ATMs;
  void accept(
    in Requester req,
    in short amount);
 };
};
```



AST Node Types



Container (node with children)

```
interface Container : IRObject {
  Contained lookup(in ScopedName search_name);
  sequence<Contained> contents(
      in DefinitionKind limit_type,
      in boolean        exclude_inherited);

  sequence<Contained> lookup_name(
      in Identifier     search_name,
      in long           levels_to_search,
      in DefinitionKind limit_type,
      in boolean        exclude_inherited);
  ...
};
```

Contained (child)

```
interface Contained : IRObject {
  attribute Identifier         name;
  attribute RepositoryId       id;
  attribute VersionSpec        version;
  readonly attribute Container defined_in;
  struct Description {
    DefinitionKind  kind;
    any             value;
  };
  Description describe();
  ...
};
```


Interface Definition

```
interface InterfaceDef : Container,Contained {
 attribute sequence<InterfaceDef> base_interfaces;
 boolean is_a(in RepositoryId interface_id);
 struct FullInterfaceDescription {
    Identifier                     name;
    RepositoryId                   id;
    RepositoryId                   defined_in;
    RepositoryIdSequence           base_interfaces;
    sequence<OperationDescription> operations;
    sequence<AttributeDescription> attributes;
    ...
 };
 FullInterfaceDescription describe_interface();
};
```

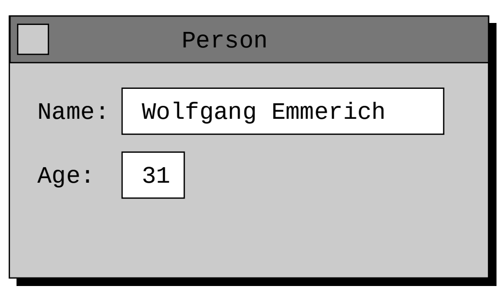
Locating Interface Definitions

Alternatives:

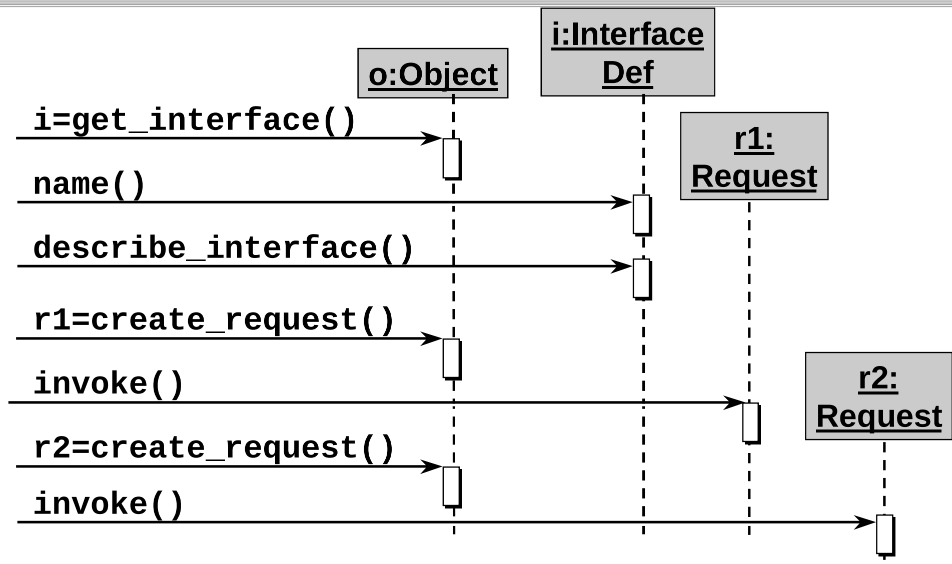
- ***Any interface inherits the operation InterfaceDef `get_interface()` from `Object`.***
- ***Associative search using `lookup_name`.***
- ***Navigation through the interface repository using `contents` and `defined_in` attributes.***

Example: Object Browser

- ***Use interface repository to find out about object types at run-time***
- ***Use dynamic invocation interface to obtain attribute values***



Sequence Diagram



Summary

- *Transparency*
- *OO MIddleware*
- *Resolving Language Heterogeneity*
- *Resolving Data Heterogeneity*
- *OMG/CORBA*
- *Genericity*