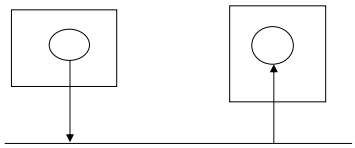
Distributed Programming

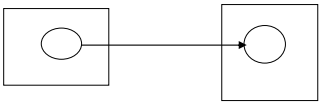
•low level: sending data among distributed computations



•network is visible (to the programmer)

•programmer must deal with many details

•higher level: supporting invocations among distributed computations

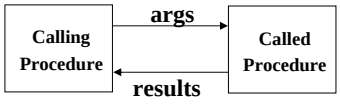


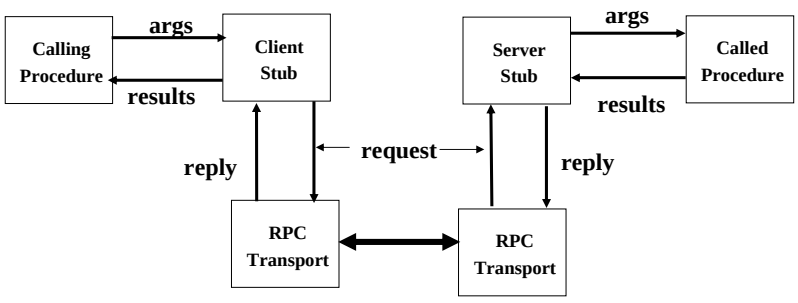
•network is invisible (to the programmer)

•programmer focuses on application

1

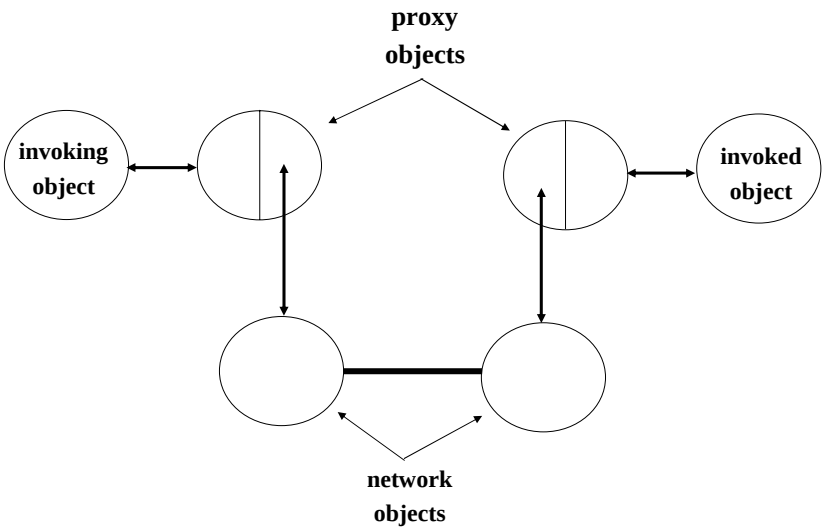
Remote Procedure Call





2

Remote Object Systems



3

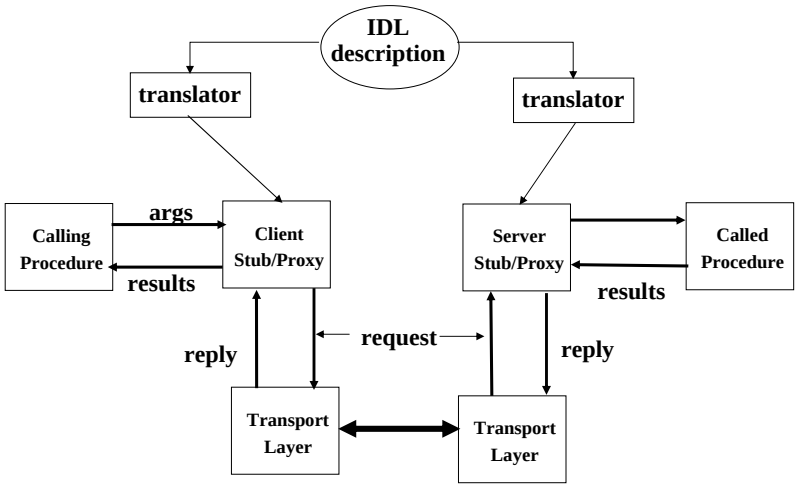
Remote Invocation Issues

- generating stubs/proxies
- serialization of arguments and return values
- heterogeneity of data representations
- locating servers in a distributed environment (*)
- authentication of called and calling procedures (*)
- semantics of invocation

(*) addressed in other sections of the course

4

Interface Definition Language



Language binding: how IDL is translated to a given programming language.

IDL Elements

```
module modulename {
  exception exceptionName { [type pname]* };
  typedef type newtype;

  interface newInterface {
   oneway type fname(in type pname1);
   attribute newtype;
  };

  interface newInterface2 : newInterface {
   type fname2 (out newInterface pname3) raises exceptionName;
  };
};
```

From: Ole Arthur Bernsen

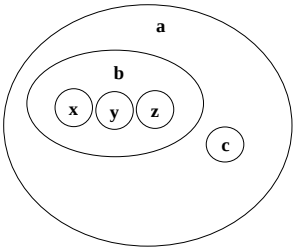
IDL Example

```
typedef unsigned long AccountNumber;
typedef unsigned long PersonalIdentificationNumber;

exception NoSuchAccount {};
exception InvalidPin{};
exception InsufficientFunds {};

interface Account {
  struct AccountRecord {
    string owner;
    float balance;
    string lastaccess; };
  void Credit (in float Amount);
  void Debit(in float Amount) raises (InsufficientFunds);
  void List (out AccountRecord List_R1);
};

interface Sbank {
  Account Access (in AccountNumber acct,
                  in PersonalIdentificationNumber pin)
                  raises (NoSuchAccount, InvalidPin);
};
```

From: Nigel Edwards

7

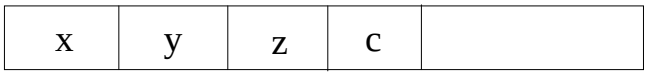
Serialization



Issues:

- how to represent base types (i.e. int)
- how to represent structured types (arrays)
- how to deal with references (pointers)
- how to treat duplicated objects

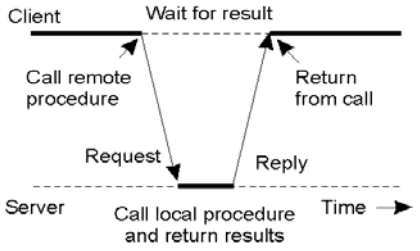
transforming a typed, highly structured object into a stream of bytes.



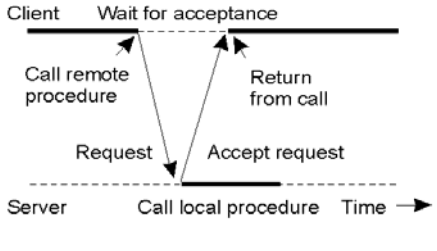
Transfer syntax: the description of the encoded data stream.

8

Invocation Semantics - Blocking



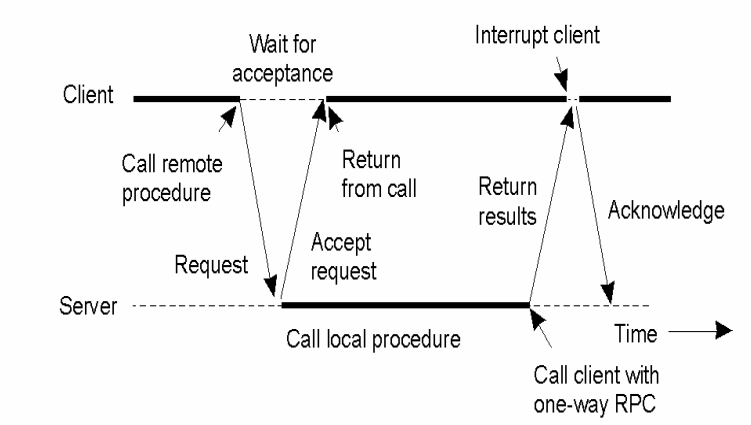
synchronous



asynchronous (one-way)

9

Invocation Semantics - Blocking



asynchronous (with returned result)

10

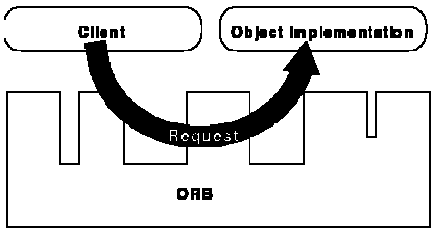
Invocation Semantics –Modes

- At-most once: it is guaranteed that the invocation will not occur or will occur exactly once.
- At-least-once: it is guaranteed that the invocation will occur though perhaps multiple times
- Best-effort: no guarantee

11

Corba

Goal: interoperability among application components
- written in different programming languages
- executing on heterogeneous architectures
- communicating over different networks.



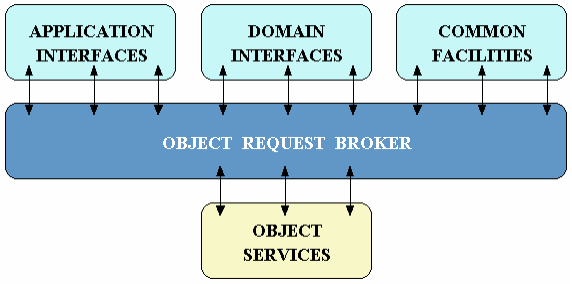
Corba: Common Object Request Broker Architecture

ORB: Object Request Broker

From: Object Mangagement Group

12

Role of the Object Request Broker

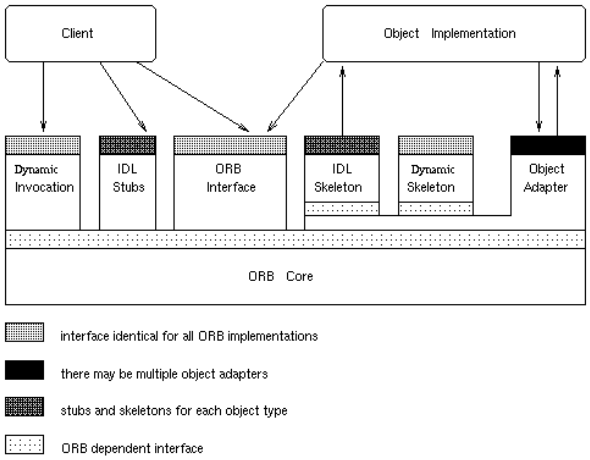


- **Application interfaces**: interfaces for a specific application
- **Domain interfaces**: interfaces shared across applications in a given application domain (publishing)
- **Common Facilities**: generic services that might be needed in several domains (document structure)
- **Object Services**: commonly needed across all applications (e.g., lifetime, naming, trading)

From: Doug Schmidt

13

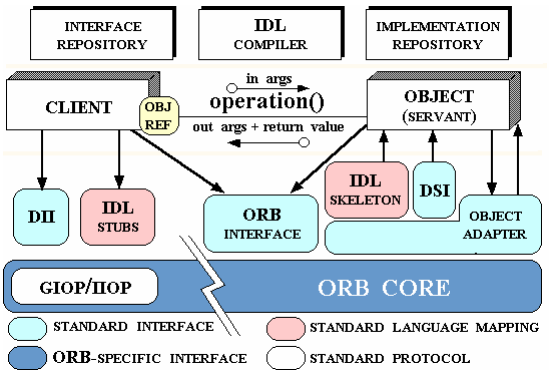
Elements of Corba



From: Kate Keahey

14

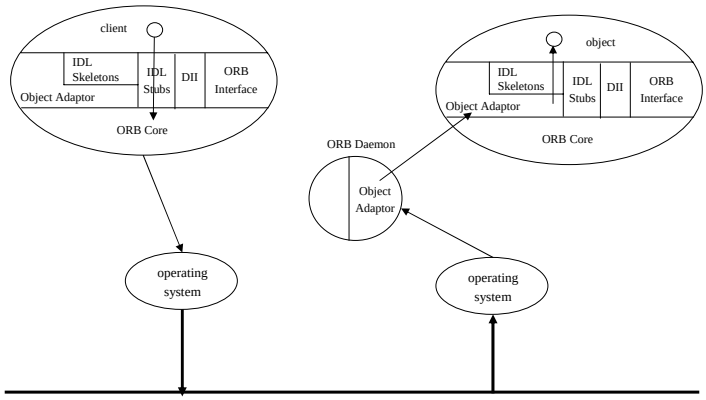
Elements of Corba



From: Doug Schmidt

15

Corba Process Structure



16

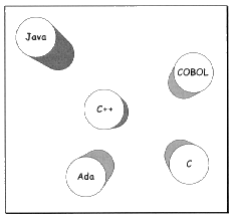
Corba Services

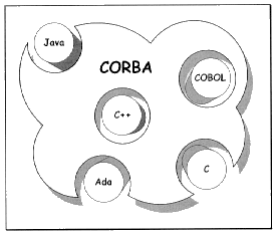
- Naming - bind of names to objects (*)
- Events - asynchronous notification (*)
- Lifecycle - object management
- Relationship - maintaining relationships among objects
- Transaction - structured, reliable, database operations (*)

(*) - see more about later in the course

17

Corba and Java





Corba is still needed to fill in the gaps between Java and system developed in other languages.

18