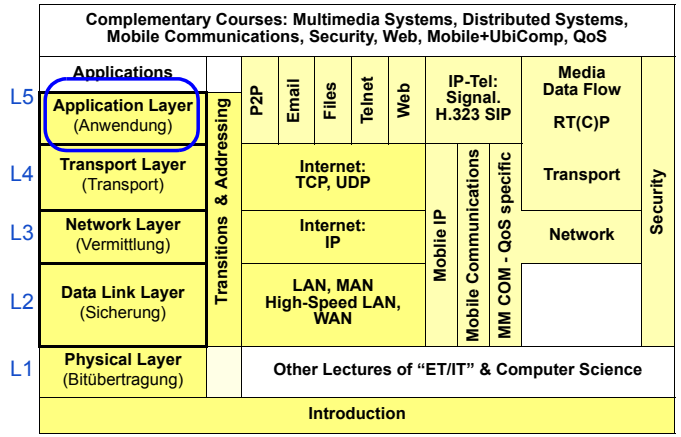
Communications Systems
Application Layer

Prof. Dr.-Ing. **Lars Wolf**

TU Braunschweig
Institut für Betriebssysteme und Rechnerverbund

Mühlenpfordtstraße 23, 38106 Braunschweig, Germany
Email: wolf@ibr.cs.tu-bs.de

Scope



Overview

1. **Addressing in General and DNS**
2. **Application-Oriented Communication Services**
3. **Data Presentation**
4. **Client / Server and Remote Procedure Call**
5. **Middleware - CORBA**

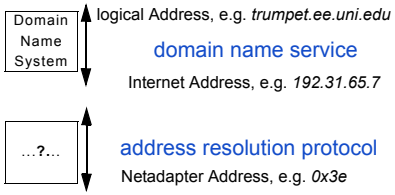
1. Addressing in General and DNS

Addressing means 3 types of identifiers: names, addresses and routes

"The **NAME** of a resource indicates WHAT we seek,
an **ADDRESS** indicates WHERE it is, and
a **ROUTE** tells HOW TO GET THERE [Shoch 78]

- **address identifies type of or specific**
  - application
  - service
  - network (e.g. subnet)
  - machine
  - interface
  - ...

⇒ **Addressing must occur at many levels of abstraction, e.g.**



Domain Name Service (DNS): Objective

Purpose:
- Internet Protocol address is a 32-bit integer
- People prefer to assign machines pronounceable names (host names)

192.168.128.73 ⟷ (DNS) ⟷ www.remember.tv

- hard-coded IP addresses within applications may become outdated
  - e.g., when moving mailserver / web server to other server with different address

⇒ **mapping from name to IP address needed**

Approaches:
- use file with mapping on every host (`hosts` file), updated regularly
  - doesn't scale nowadays (file too large, too many file update operations)
- use decentralized scheme

⇒ **DNS**

Domain Name Service (DNS): Basics

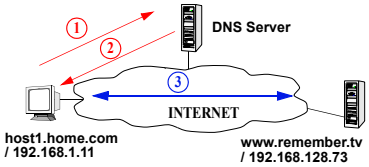
Standards:
- Basics: RFC 1034, RFC 1035
- many documents describe additional features

Idea:
- The Internet Protocol address is a 32-bit integer
- People prefer to assign machines pronounceable names (host names)

192.168.128.73 ⟷ (DNS) ⟷ www.remember.tv

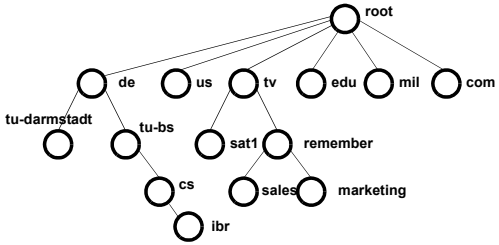
DNS characteristics:
- Distributed (responsibility and physical) database
- Provides mapping between host names and IP addresses
- Additional services: e.g. mail routing information

Operation - basic description:



1. Host1 sends a DNS request to its DNS Server and asks for the IP address of www.remember.tv
2. The DNS server sends the IP address (192.168.128.73)
3. Host1 is now able to communicate with www.remember.tv

DNS: Name Space



Top-level domains
- unnamed root
- 1 arpa domain
- generic domains: 7*3-char. domains (com, edu, gov, int, mil, net, org)
- country domains: based on (2-char.) country codes (ISO 3166: tv, de, us,...)

Registration
- geographical (e.g. remember.tv)
- organisational (e.g. remember.com)

Domains, subdomains, ...
- by local authorities (e.g. admin of remember)
- e.g. sales, marketing, ...

DNS: Name Space (2)

Tree leaves represent domains without further subdomains but with IP equipment (computers, printers, ...)

Distribution with regard to organizational issues but without regard to physical connections
- hierarchy can be distributed on the underlaying network

Allows multiple naming hierarchies to be embedded
- specified by object types: e.g. MX: Mail Exchanger, NS: Name Server

Several domains can be hosted by one server
- e.g. domains sales.remember.tv, marketing.remember.tv hosted by one server

'Popular' domains have been used up
- especially in .com

New top-level domains have been approved by ICANN recently

DNS: Name Server Types

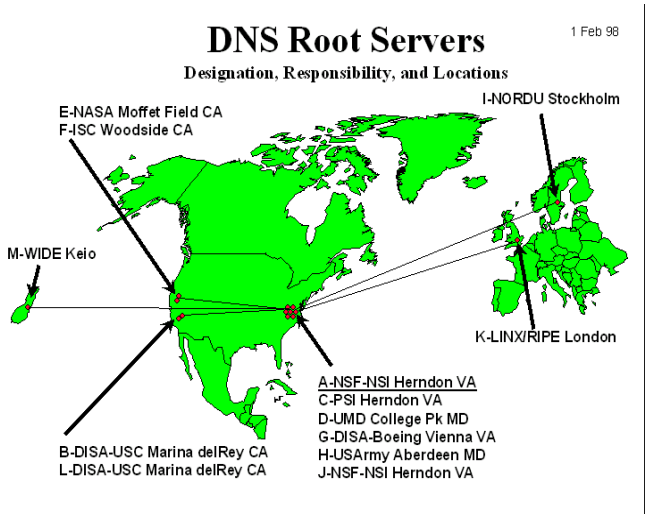
No server has all name-to-IP address mappings

Local name servers:
- **each ISP, company has *local (default) name server***
- **host DNS query first goes to local name server**

Authoritative name server:
- **for a host: stores that host's IP address, name**
- **can perform name/address translation for that host's name**

Root name server:
- **contacted by local name server that can not resolve name**
- **root name server:**
  - contacts authoritative name server if name mapping not known
  - gets mapping
- **returns mapping to local name server**

DNS: Root Name Servers



DNS: Resource Records

Each domain can have set of RESOURCE RECORDS (RR) associated with it
- **Different types: most common are IP address**

DNS maps domain names onto resource records

Resource record format are five tuples:

Domain_name	Time_to_live	Class	Type	Value
domain to which this record applies	'stability' of the record	IN for Internet information (others possible)	record type (see below)	number, domain, ASCII string depends on type

Some record types:

Type	Meaning	Value
A	IP address of named host	32 bit integer giving IP address
MX	Mail exchange associated with name	Priority, domain willing to accept email
NS	Name server	Name of server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
...		

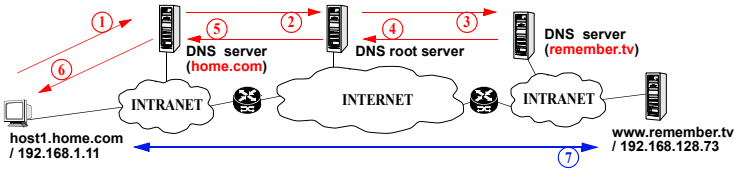
DNS Database

Example:

```
$TTL          86400
;
; Start of Authority:
;
;
; Nameserver:
;
@             IN            NS            agitator.ibr.cs.tu-bs.de.
@             IN            NS            infbssys.ips.cs.tu-bs.de.
@             IN            NS            oker.escape.de.
;
; Mail Exchanger fuer ibr.cs.tu-bs.de:
;
@             IN            MX            10 agitator.ibr.cs.tu-bs.de.
cip           IN            MX            10 pott.cip.ibr.cs.tu-bs.de.
mail.cip      IN            CNAME         pott.cip.ibr.cs.tu-bs.de.

;
; IPv6:
;
ipv6          IN            NS            agitator.ibr.cs.tu-bs.de.
              IN      NS      oker.escape.de.
              IN      NS      ns.ipv6.tm.uka.de.

asaft         IN            A             134.169.34.100
              IN            MX            10 agitator.ibr.cs.tu-bs.de.
osaft         IN            A             134.169.34.101
              IN            MX            10 agitator.ibr.cs.tu-bs.de.

salvator      IN            A             134.169.34.17
              IN            MX            10 agitator.ibr.cs.tu-bs.de.
nis           IN            CNAME         salvator
loghost       IN            CNAME         salvator
```

DNS: Protocol

typical operation - extended description



DNS recursive resolution:

- **in many steps**
  1. **Host1 sends a DNS request to its local DNS server and Host1 asks for the IP address of www.remember.tv**
  2. **...**
  3. **...**
  4. **...**
  5. **...**
  6. **...**
  7. **Host1 is now able to communicate with www.remember.tv**

DNS: Protocol (2)

1. **Application on Host1**
   - calls local 'resolver', asks for IP addr. of www.remember.tv (name as parameter)
2. **Host1**
   - sends a DNS request (using UDP) to its local DNS server and asks for IP addr.
3. **DNS server can not resolve the request**
   - forwards the request to one of the toplevel root server
   - request marked as "*recursive resolution*"
4. **toplevel DNS server**
   - knows the location of the DNS server(s) responsible for remember.tv
   - request is (also recursive) forwarded to this DNS server
5. **DNS server**
   - is capable to resolve the request
   - sends the IP address (192.168.128.73) back to the root server
6. **root server**
   - sends the answer to the home.com DNS server
7. **home.com DNS server**
   - sends the answer to host1
8. **Host1 is now able to communicate with www.remember.tv**

⇒ **Obviously optimizations are necessary**

⇒ **Efficient Translation, Caching name server**

DNS: Protocol (3)

Domain Server Message Format:

0	16	32
IDENTIFICATION	FLAGS	
NUMBER OF QUESTIONS	NUMBER OF ANSWER RRs	
NUMBER OF AUTHORITY RRs	NUMBER OF ADDITIONAL RRs	
QUESTION SECTION		
ANSWER SECTION		
AUTHORITY SECTION		
ADDITIONAL INFORMATION SECTION		

Fixed Header

Domain names are stored as sequence of labels, each beginning with an octet specifing its length.

=> repeatedly reading 1 octet (=n) and then reading the label with length n

Client fills in only the QUESTION SECTION, server returns QUESTIONS and ANSWERS in response

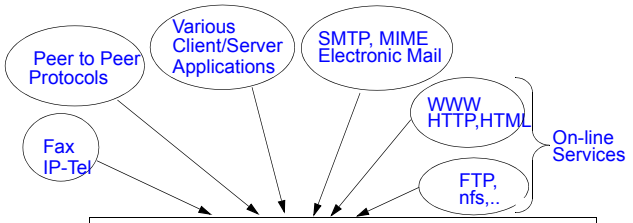
The QUESTION SECTION contains the queries consisting of:

- *QUERY DOMAIN NAME*: the name (stored as labels)
- *TYPE*: e.g. *MX* Mail Exchanger, *NS* Name Server
- *QUERY CLASS*: different classes, e.g. official Internet names

The ANSWER SECTION contains the answers consisting of:

- *TIME TO LIVE*: specifies how long the information can be cached
- *RESOURCE DATA*: each record describes one domain name and mapping
- Other fields: *TYPE, CLASS, RESOURCE DATA LENGTH, RESOURCE*

2. Application-Oriented Communication Services



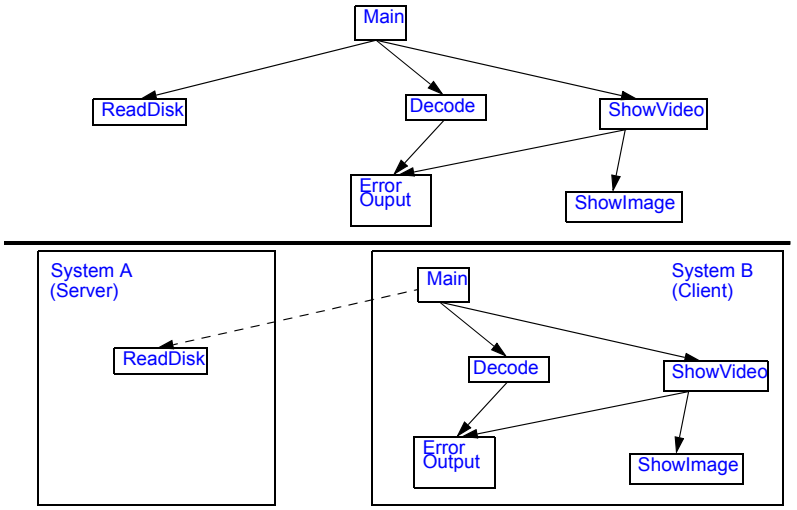
Approaches for Developing Distributed Programs

1. COMMUNICATION-ORIENTED approach
- **define messages and formats**
- **use, e.g., client-server design**
  - defined as reaction to incoming messages
- **use sockets**

⇒ **evaluation:**
- benefits
  - when all communication is executed on an equal basis
- disadvantages
  - program design depends on type of communication
  - an error in the protocol may lead to complete redesign of the program
  - development of communication protocols may be complex

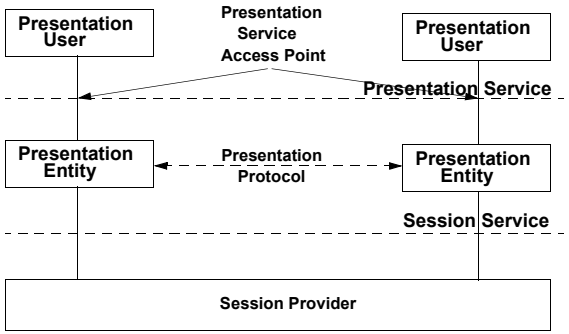
2. APPLICATION-ORIENTED approach
- **use conventional program development**
- **transfer modular approach to distributed programming**
- **functionality located in procedures/objects, not in communications**
- **communications between systems independent of programs, e.g., by using the Remote Procedure Call concept**

Example



Presentation & Session

To provide well understood data presentation for any communications between open systems



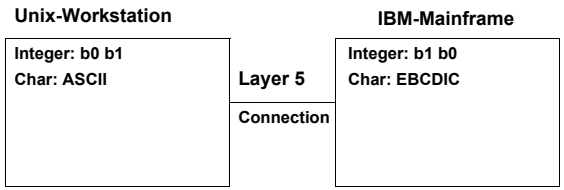
Presentation & Session: Task (2)

Functions:
- **to transfer communication control services**
- **to allow the specification of complex data structures**
- **negotiation of required data structures**
- **to convert the local representation into a global one**

Because
- connection does not mean communication
- communication implies a common understanding

Example:
- **understanding the words**
  - Igel (German) - eagle (English)

Presentation & Session: Presentation Task (3)



Situation:

- **even though correct connection at lower layers there is no further communication possible**

⇒ **Semantics are lost**

Coding regulations may depend on compiler

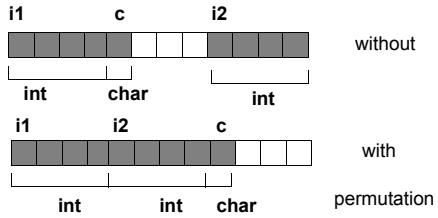
Presentation & Session: Example

Example:

```
struct {
  int   i1;
  char  c;
  int   i2;
}
```

- **char:** **one byte, no alignment conditions**
- **int:** **4 bytes, alignment according to an address divisible by 4**

compiling with and without permutation strategy



Coding Rules

Type (c)	Coding Rules
INT	Length Coding type Arrangement Justification (with word border)
FLOAT	Length of mantissa Length of the exponential Exponential basis Coding type Arrangement Justification
CHAR	Coding type

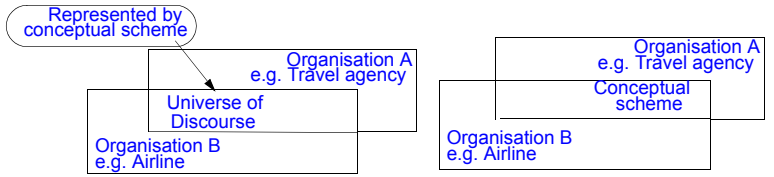
3. Data Presentation

Sender and receiver need common data presentation to allow understanding

- **'communication' of content not of bits**
- **needed for formats, data types, compression, coding, ...**

Generic view:

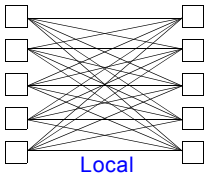
- **Universe of Discourse**
  - part of the real world which is to be processed in the system
- **Conceptual scheme**
  - formal description of the universe of discourse



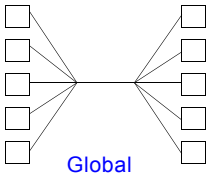
- **Requirements**
  - relation to the same universe of discourse
  - common conceptual scheme
  - comprehensible representation of the conceptual scheme's objects (i.e. data conversion) to both communication parties

Data Presentation: Methods

Local presentation of a communication partner
- **n x (n-1) conversion routines**
- **a maximum of one conversion per relationship**
  - local format f1 directly to local format f2



Global presentation
- **2 x n conversion routines**
- **2 conversions per relationship**
  - local f1 to global g,
  - global g to local f2
- **scheme:**



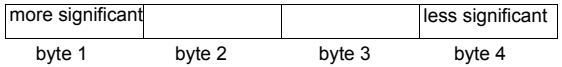
- **standards: XDR, ASN.1**

XDR, the Representation „Layer“ in the Internet

XDR: External Data Representation
- **presentation layer with very low functionality**

Example for a conversion issue: integers

1. BIG-ENDIAN **(byte 0 as the most significant (i.e. left)) versus** LITTLE-ENDIAN **(byte 0 as the least significant (i.e. right))**
   - comment: usually also relates to bits
   - Motorola 68x0, IBM 370 (Big Endian)

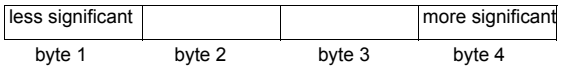


   - excerpt from a respective configuration header-file of an IBM RS6000:

```
/* Definitions for byte order, */
/* according to byte significance from low address to high. */
#define LITTLE_ENDIAN    1234    /* least-significant byte first
(vax)          */
#define BIG_ENDIAN      4321    /* most-significant byte first (IBM,
net)     */
#define PDP_ENDIAN       3412    /* LSB first in word, MSW first in
long (pdp) */
#define BYTE_ORDER       BIG_ENDIAN
```


XDR, the Representation „Layer“ in the Internet (2)

2. Intel 80x86 (LITTLE ENDIAN)



- **all data are mapped to a pre-defined transfer syntax (no negotiations)**
  - all integers as 4-byte big-endians
  - floating-point numbers in IEEE format:
    - mantissa 23 bits
    - exponential 8 bits
    - algebraic sign 1 bit
  - texts in ASCII code
  - all data elements aligned with 4-byte limit

Disadvantage:

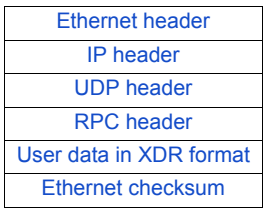
⇒ **Two systems which are completely identical have to convert twice**

XDR, the Representation „Layer“ in the Internet (3)

Essential component: XDR compiler
- **generates**
  - C data structures compatible with the XDR definition and
  - program pieces for coding and decoding

Summary/example of a typical data packet

start of the packet



end of the packet

Comment: XDR does NOT **need any own header**

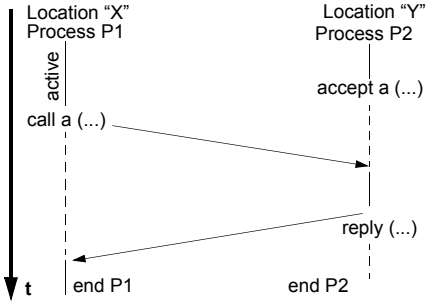
4. Client / Server and Remote Procedure Call

Server
- provides services
- waits for incoming service requests from clients
- processes requests and sends results as response
- may use other servers to process request (becomes client in that case)

Client
- uses services provided by server
- sends requests to server
- (typically) waits for response from server

For client conceptually similar to PROCEDURE CALL
- call procedure
- wait for result

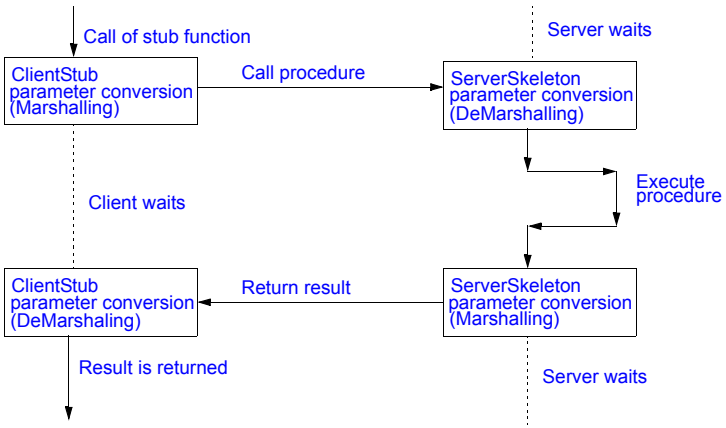
Remote Procedure Call - RPC



Concept
- synchronization between client and server
  - synchronous Remote Service Invocation (sRSI)
- characteristic: e.g. limited parallelism

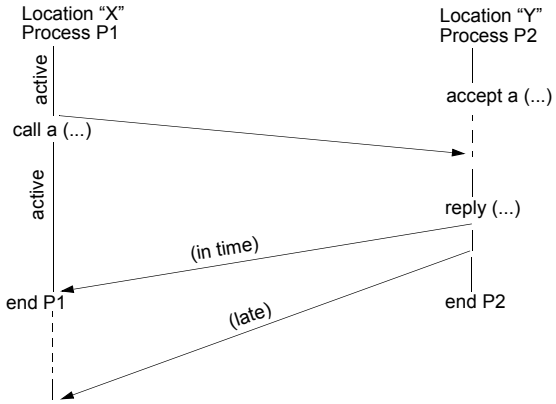
Basic idea:
- application cannot differentiate between
  - remote procedure call and
  - local procedure call

RPC Example



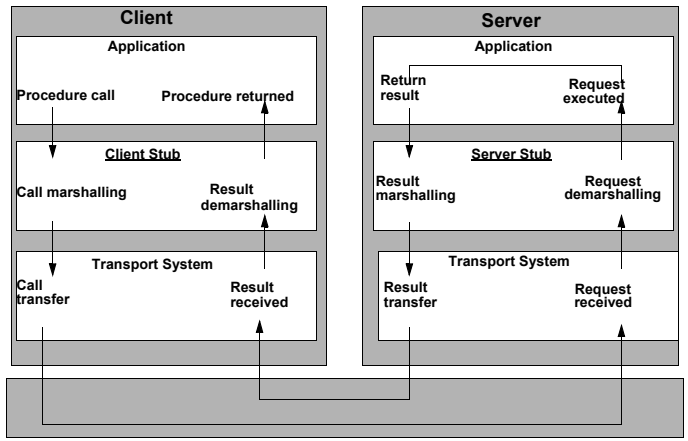
Asynchronous Remote Service Invocation (aRSI)

Alternatively to synchronous RPC



Characteristics (among others)
- parallelism between client and server possible
- associating requests and respective results more difficult

RPC: Cycle



RPC: Cycle (2)

Tasks of the "Stub" procedures:

- **to locate and bind server with client**
  - server registers its service at database (server) by providing its name (ASCII), network address and service number (any 32 bit number) (export)
  - client-stub sends request to database its name (which is also the name of the server) in ASCII
  - database service returns network address and unique server identification (binding)
- **marshalling/demarshalling (parameter arrangement) of parameters and results (guarantees transparency)**
  - client
    - collects all parameters of an RPC call and packs them into a message
  - server
    - unpacks the parameters, performs function(s) and packs results into a message
  - client unpacks results
- **error treatment, error semantics**
- **communications**
  - transport system interface
  - data representation
  - authentication/encryption

RPC: Error Semantics

Various errors may occur, e.g.

- **requests or replies get lost or are garbled during data transfer**
- **client or server crashes while RPC is ongoing**

Several error classes can be distinguished

Maybe-semantics

- **server process may have been executed once**

At-least-once-semantics

- **server process is executed error-free at least once (if not more)**

At-most-once-semantics

- **server process is executed error-free at most once**

Exactly-once-semantics

- **server process is executed error-free exactly once (guaranteed) including transmission**

RPC: Idea and Reality

Basic idea:

- **application cannot differentiate between**
  - remote procedure call and
  - local procedure call

Problems:

- **transparency:**
  - parameter treatment ("call by reference", "pointer", procedures ...)
  - side effects
- **efficiency**
  - additional effort for "marshalling/demarshalling"
  - error treatment, e.g., for recovery after a "server crash"
- **conception**
  - client and server roles may change
    - e.g. in streaming

Implementations

- **e.g., SUN RPC (RFC 1057)**
- **e.g., RPC at Open Software Foundation's (OSF) Distributed Computing Environment (DCE)**

5. Middleware - CORBA

Common Object Request Broker Architecture - CORBA

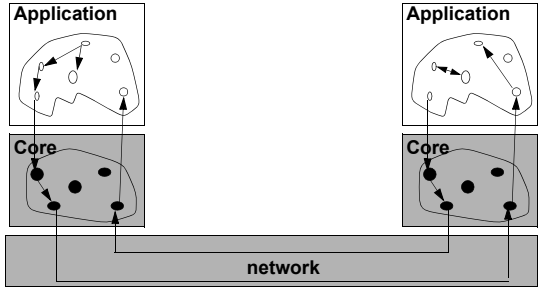
Middleware is

- **software/abstraction glue which allows separate applications to communicate TRANSPARENTLY**
- **i.e. ‚to inter-operate independently from**
  - hardware and system devices, operating systems capabilities
  - communications infrastructure

History

- **1987 - Sun RPC**
- **1988 - Distributed Computing Environment (DCE) of Open Software Foundation (OSF)**
  - coined term “middleware”
  - incl. naming service, fault semantics, Interface Definition Language (IDL)
  - but: no object-oriented model with inheritance, static binding of declared procedure, ...
- **today - Object-oriented approach of Object Management Group (OMG)**
  - Object Management Architecture (OMA)
  - Common Object Request Broker Architecture (CORBA)

Object-Oriented Software Development



Goals

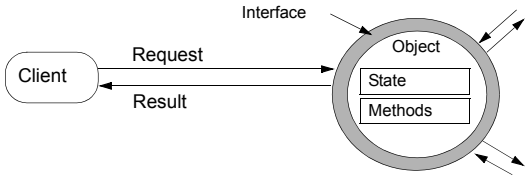
- **functionality, efficiency, robustness,**
- **reuse, future enhancements possible**

Alternative (unfortunately too often used) development methods

- **“to develop from scratch”**
- **“Copy, paste and adapt individual code”**
- **“Combine generic parts taken from libraries”**
- **“Use objects, inherit from and instantiate framework components”**

Objects in Distributed Systems

Abstraction



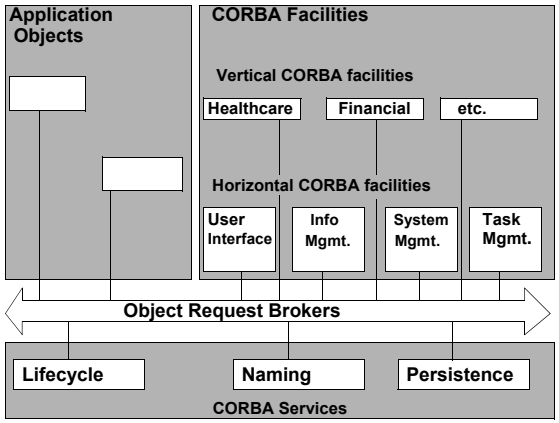
Request specifies

Target Object	Operation	Parameter	Context

Characteristics

- **object addressed via**
  - unique system wide (location independent) identifier
- **usage of Interface Definition Language IDL**
- **abstraction from local environment**
- **like the object-oriented paradigm**
  - inheritance, instantiation of object classes, polymorphism

Object Management Architecture OMA



Object Management Group (OMG)

- **1989 established**
- **as independent group (more than 400 companies involved)**

Object Management Architecture OMA (2)

OMA defines components:

- **Object Request Broker (ORB)**
- **Object Services**
- **Common Facilities**
- **Application Objects**
- **Notation via: Interface Definition Language (IDL)**

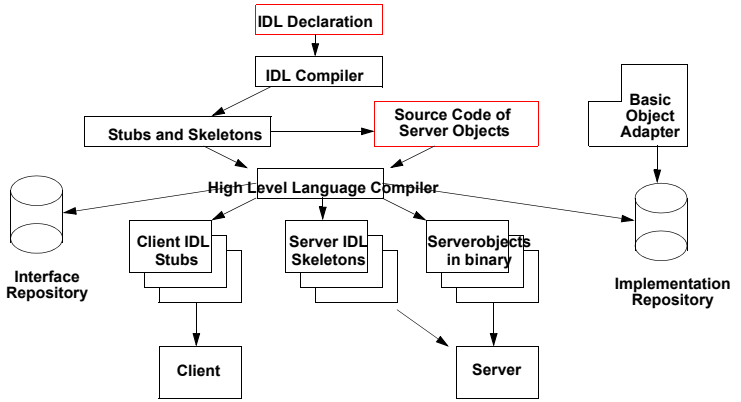
Interface Definition Language

- **Language to define the interfaces, syntax similar to C++**

Object Request Broker (ORB)

- **Client**
  - does NOT contact server directly
  - contacts object bus
- **Client does not care about**
  - transport services or protocols
  - object creation, management, storage at server side
- **Server and client similar**
  - i.e., should be the same

Interface Definition Language - Environment



Interface Repository:

- **stores interface information of objects used by clients at run-time**

Implementation Repository:

- **allows ORB to localize and activate object implementations**