FRANKFURT UNIVERSITY OF APPLIED SCIENCES

How can we achieve access transparency?

Distributed Systems
Remote Invocation

Prof. Dr. Oliver Hahm
Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
oliver.hahm@fb2.fra-uas.de
https://teaching.dahahm.de

Agenda

Agenda

Motivation

- Message oriented communication
  - asynchronous exchange of messages
  - explicitly via `send()` and `receive()` operations
  - Summary
    - \+ very flexible, all communication patterns possible
    - \- explicit, I/O paradigm

Motivation

- Message oriented communication
  - asynchronous exchange of messages
  - explicitly via `send()` and `receive()` operations
  - Summary
    - \+ very flexible, all communication patterns possible
    - \- explicit, I/O paradigm
- Goal of **remote invocation**
  - Communication transparency
  - Appears like an usual local procedure call
  - → Remote Procedure Call
- Supports ...
  - Service orientation → Service = Set of functions
  - RPC for calling the functions
  - Object orientation → Remove Method Invocation (RMI)

History

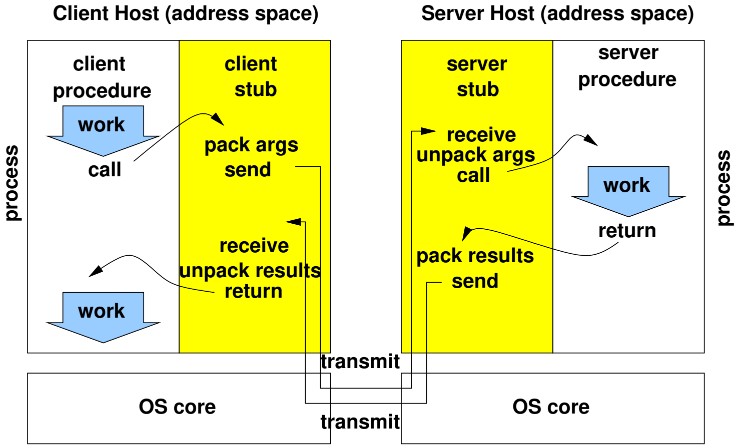
- First comprehensive presentation:
  - Dissertation Nelson (1981, XPARC)
  - Derived Paper Birrel/Nelson (1984, ACM ToCS)
- Definition:
  - RPC as a synchronous mechanism '"which transfers control flow and data as a procedure call between two address spaces over a narrowband network."'
- Nelson's Thesis:
  - RPC is an efficient concept for implementing distributed applications
  - RPC facilitates the development of distributed systems

History

- First comprehensive presentation:
  - Dissertation Nelson (1981, XPARC)
  - Derived Paper Birrel/Nelson (1984, ACM ToCS)
- Definition:
  - RPC as a synchronous mechanism '"which transfers control flow and data as a procedure call between two address spaces over a narrowband network."'
- Nelson's Thesis:
  - RPC is an efficient concept for implementing distributed applications
  - RPC facilitates the development of distributed systems
- Today:
  - Nelson's vision has been widely accepted
  - Many produces work on RPC systems
  - Typical examples: SunRPC and NFS, OSF DCE RPC, Apache Thrift, D-Bus

Agenda

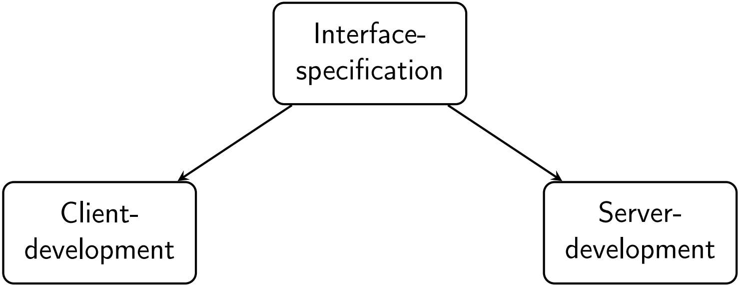
Main Principle



pack/unpack = marshalling/unmarshalling
Proxy components: stub, proxy, skeleton

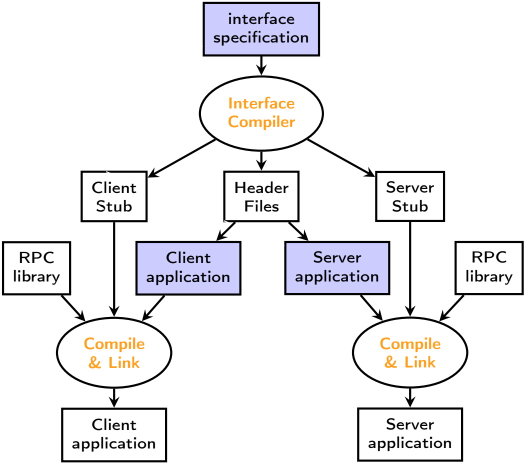
Application Development (high level)

Coarse structure:



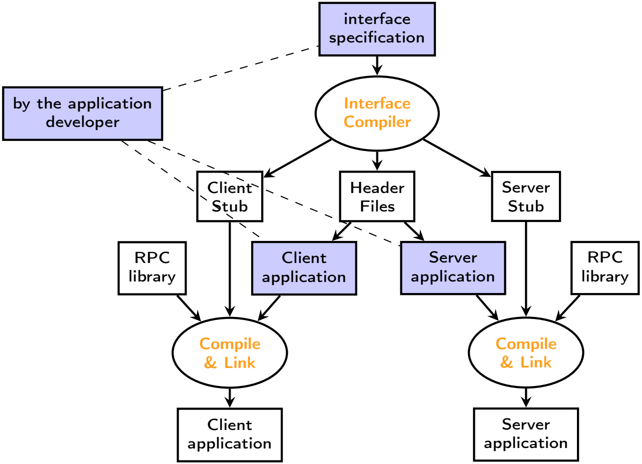
Application Development (Zoom in)

more detailed, but still independent of the particular RPC system:

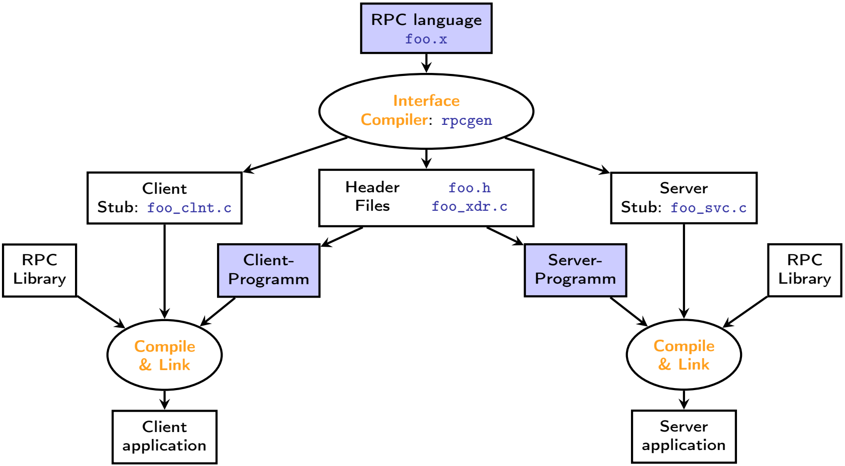


Application Development (Zoom in)

more detailed, but still independent of the particular RPC system:



Example: SunRPC



Example: Interface Description SunRPC (1)

```
    const MAX_FILENAME_LEN = 255;
    typedef string t_filename<MAX_FILENAME_LEN>;
    const MAX_CONTENT_LEN = 255;
    typedef string t_content<MAX_CONTENT_LEN>;
```

```
struct s_filewrite {
    t_filename filename;
    t_content content;
};
struct s_chmod {
    t_filename filename;
    long mods;
};
```

```
struct s_fstat {
    long dev;
    long ino;
    long mode;
    long nlink;
    long uid;
    long gid;
    long rdev;
    long size;
    long blksize;
    long blocks;
    long atime;
    long mtime;
    long ctime;
};
```

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Example: Interface Description SunRPC (2)

```
program fileservice {
    version fsrv {
        int fsrv_mkdir(string) = 1;
        int fsrv_rmdir(string) = 2;
        int fsrv_chdir(string) = 3;
        int fsrv_writefile(s_filewrite) = 4;
        string fsrv_readfile(string) = 5;
        s_fstat fsrv_fileattr(string) = 6;
        int fsrv_chmod(s_chmod) = 7;
    } = 1;
} = 0x30000001;
```

Example: Interface Description DCE

```
[ uuid(5ab2e9b4-3d48-11d2-9ea4-80c5140aaa77),
version(1.0), pointer_default(ptr)
]
interface echo {
    typedef [ptr, string] char * string_t;
    typedef struct {
        unsigned32 argc;
        [size_is(argc)] string_t argv[];
    } args;
    boolean ReverseIt(
        [in] handle_t h,
        [in] args* in_text,
        [out] args** out_text,
        [out,ref] error_status_t* status
        );
}
```

Example: Interface Description Thrift

```
typedef i32 MyInteger
enum Operation { ADD = 1,
                 SUBTRACT = 2,
                 MULTIPLY = 3,
                 DIVIDE = 4
}
struct Work {
    1: MyInteger num1 = 0,
    2: MyInteger num2,
    3: Operation op,
    4: optional string comment,
}
exception InvalidOperation { 1: i32 what, 2: string why }
service Calculator {
    void ping(),
    i32 add(1:i32 num1, 2:i32 num2),
    i32 calculate(1:i32 logid, 2:Work w)
    throws (1:InvalidOperation ouch),
    oneway void quit()
}
```

Security

- Problems
  - Mutual authentication
  - Authorization wrt executable functions on the server
  - Encryption of transmitted data

→ Detailed consideration in a separate chapter of this lecture

Agenda

Binding

- Binding/Trading:
  - Problem: Binding of a client to a server is mandatory
  - Problem exists for other paradigms as well
  - Aspects: Naming & Locating

Binding

- Binding/Trading:
  - Problem: Binding of a client to a server is mandatory
  - Problem exists for other paradigms as well
  - Aspects: Naming & Locating

⇒ **Naming**

  - How does the client specify what it wants to be bound to (service)
  - Interface names are structured in a system wide **namespace**
  - Extending this concept by interface attributes → **Trading**

  → Directory and name services

Binding

- Binding/Trading:
  - Problem: Binding of a client to a server is mandatory
  - Problem exists for other paradigms as well
  - Aspects: Naming & Locating

⇒ **Naming**

- How does the client specify what it wants to be bound to (service)
- Interface names are structured in a system wide **namespace**
- Extending this concept by interface attributes → **Trading**

→ Directory and name services

⇒ **Locating**

- Determine the (location dependent) **address** of a server which exports the desired interface and can be used for the service
- often: IP address of the host and port number

Locating Types

- Static address as part of the application
  - Benefit: requires no search process
  - Drawback: often not flexible enough

  ⇒ binding too early

Locating Types

- Static address as part of the application
  - Benefit: requires no search process
  - Drawback: often not flexible enough

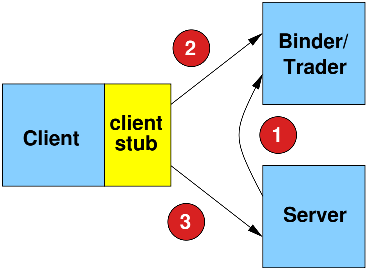
  ⇒ binding too early
- Search for exporting servers at runtime, e.g., via broadcast
  - Benefit: very flexible
  - Drawback: increased runtime
  - Drawback: Broadcasting across subnet boundaries is not desirable

  ⇒ binding too late

Locating Types

- **Static address** as part of the application
  - Benefit: requires no search process
  - Drawback: often not flexible enough
  - ⇒ binding too early
- Search for exporting servers at runtime, e.g., via broadcast
  - Benefit: very flexible
  - Drawback: increased runtime
  - Drawback: Broadcasting across subnet boundaries is not desirable
  - ⇒ binding too late
- Manage binding information via intermediary instance
  - Mediating instance is called **binder**, trader, or **broker**
  - Exporting server **registers** interface (along with all attributes)
  - Binding request of an importing client causes assignment by the binder

Basic Procedure



1. Exporting the interface
   - Register the interface at binder
   - Binder has known address
2. Importing
   - At first use of the service from stub
   - Provides handle with address
3. Remote invocation
   - Client stub uses the address for the call to server

Binder/Trader

Typical interface

`Register(` service name, version, address[, attributes])
`Deregister(` services name, version, address)
`Lookup(` name, version[, attributes]) ⇒ address

- Advantages:
  - Very flexible
  - Works with multiple servers of the same type
  - Basis for **load balancing** between equivalent servers
- Drawbacks:
  - Additional effort for exporting and importing of a services is required
  - Can be problematic with short-lived servers and clients

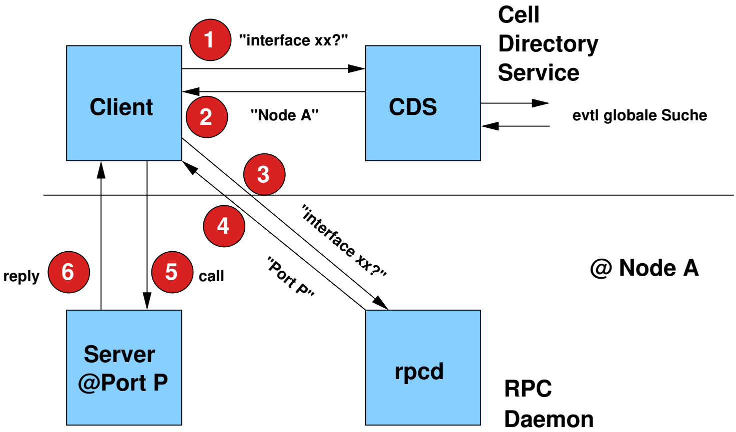
Example: SunRPC

- Names
  - Pairs (Program number, version number)
- Addresses
  - Pairs (IP address of host, port number)
- Binder: **Portmapper**
  - Mapping from names to port numbers
  - IP address of host must be known → the portmapper located there will be used
  - The portmapper itself is a SunRPC service (port 111)

Example: DCE RPC

- Names
  - **UUID (Universal Unique Identifier)**
  - Worldwide unique string
  - Generated by the tool `uuidgen`
- Addresses
  - Pairs (IP address of host, port number)
- Binding
  - Two-tiered within a DCE cell
  - No additional knowledge required
  - Binder is called RPC daemon

Example: DCE RPC (2)



Agenda

Error Problem

- Local function call:

→ Caller and callee are aborted simultaneously

- RPC:

→ Failure of single components in a distributed environment is possible

- Additional error cases caused by the messaging system itself need to be considered
  - Message loss
  - Unknown transmission times
  - Out of order delivery of messages

RPC Error Semantics: at-least-once

- at-least-once semantics
  - successful execution of the RPC
    $\Rightarrow$ called procedure is executed at least once,
    i.e., multiple executions may happen
  - Can cause arbitrary effects in an error case
  - In general, only suited for idempotent operations, i.e., multiple executions do not change state and result
- Implementation
  - Most simple form
  - If the client does not receive a result in time, the call is repeated by the stub
  - No precautions on the server are are necessary

RPC Error Semantics: at-most-once

- **at-most-once** semantics
  - Successful execution of the RPC
    $\Rightarrow$ Called procedure gets executed exactly once
  - Unsuccessful execution of the RPC
    $\Rightarrow$ Called procedure gets never executed
  - No partial error effects can be left behind
- Implementation
  - More complex
  - Requires duplicate detection

RPC Error Semantics: exatly-once

- exactly-once semantics
  - Successful execution of the RPC
    $\Rightarrow$ Called procedure is executed exactly once
- Implementation
  - Very complex (almost impossible)

Orphan Problem

- Problem: The client dies after calling an RPC
- Generated call may cause further activities even though no one is waiting for it any more
- After restart responses from a "'former life"' may be received
- Solutions:
  - **Extermination: Targeted abort of orphaned RPCs based on stable memory (practically unusable)**
  - (Gentle) Reincarnation: Introduce epochs on client side
  - **Expiration: RPCs are extended by timeouts**

Agenda

RPC Protocol

- RPC protocol: rules for processing of RPCs
- Depends on the underlying transport system
  - Datagram service (e.g., UDP)
    - \+ resource-efficient, low latency
    - \- Duplicates (via timeouts), permutations and loss are possible
  - Reliable transport service (e.g., TCP)
    - \+ Less error causes on the upper layers
    - \- Potentially possible performance reducing

⇒ The selection happens dependent on the service requirement

Example: SunRPC

- Also: Open Network Computing (ONC) RPC
- Embedding in the C language
- Underlying transport service:
  - TCP or UDP
  - Does not add any reliability enhancing measures
    - ⇒ UDP plus timeouts on the application layer can be used for a **at-least-once** semantics
    - ⇒ TCP and message transaction IDs on the application layer can be used for a **at-most-once** semantics
- Binding via portmapper
  - Portmapper protocol itself is based on RPC
- Parameters
  - only call-by-value
- Security
  - Authentication: Null, UNIX, DES

Example: SunRPC

- Also: Open Network Computing (ONC) RPC
- Embedding in the C language
- Underlying transport service:
  - TCP or UDP
  - Does not add any reliability enhancing measures
    - ⇒ UDP plus timeouts on the application layer can be used for a **at-least-once** semantics
    - ⇒ TCP and message transaction IDs on the application layer can be used for a **at-most-once** semantics
- Binding via portmapper
  - Portmapper protocol itself is based on RPC
- Parameters
  - only call-by-value
- Security
  - Authentication: ~~Null~~, ~~UNIX~~, ~~DES~~, RPCSEC_GSS

OSF DCE/RPC

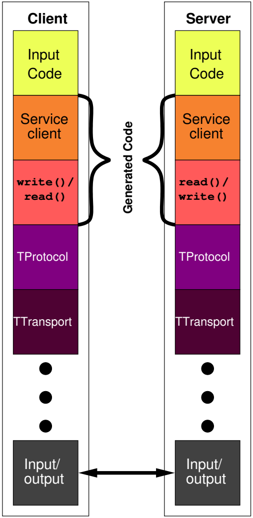
- Part of the OSF Distributed Computing Environments
- Foundation of Microsoft's DCOM and ActiveX
- Embedding for C/C++
- Multiple semantics possible (emphat-most-once as default)
- Arbitrary parameter types

→ *long* parameters via *pipe* mechanism

- Security is based on the Kerberos framework
- Relevancy has decreased

Modern RPC system: Apache Thrift

- Apache Thrift project (http://thrift.apache.org/)
  - Origins at Facebook, published in 2007
  - Supports all common programming languages
  - Siple Thrift IDL
  - IDL Compiler generates client and server stubs
  - Multiple server architectures available:
    - TNonBlockingServer
    - TThreadedServer
    - TThreadPoolServer
    - TForkingServer
    - ...
  - Multiple protocols and transports can be configured
  - Protocols: binary and text based (like JSON)
    $\Rightarrow$ low overhead
  - Transports: Tsocket, TMemoryTransport, ...
- Well-known users
  - Facebook, last.fm, Pinterest, Uber, NSA



Transparency of RPC Systems

- Access transparency
- Location transparency
- Migration transparency
- Failure transparency
- Concurrency transparency
- Replication transparency
- Performance transparency
- Scaling transparency

Transparency of RPC Systems

- Access transparency
  Yes, the same operation gets executed
- Location transparency
- Migration transparency
- Failure transparency
- Concurrency transparency
- Replication transparency
- Performance transparency
- Scaling transparency

Transparency of RPC Systems

- Access transparency
  Yes, the same operation gets executed
- Location transparency
  Yes, via the locating
- Migration transparency
- Failure transparency
- Concurrency transparency
- Replication transparency
- Performance transparency
- Scaling transparency

Transparency of RPC Systems

- Access transparency
  Yes, the same operation gets executed
- Location transparency
  Yes, via the locating
- Migration transparency
  Yes, via the naming service
- Failure transparency
- Concurrency transparency
- Replication transparency
- Performance transparency
- Scaling transparency

Transparency of RPC Systems

- Access transparency
  Yes, the same operation gets executed
- Location transparency
  Yes, via the locating
- Migration transparency
  Yes, via the naming service
- Failure transparency
  Maybe, depends on the used error semantics
- Concurrency transparency
- Replication transparency
- Performance transparency
- Scaling transparency

Transparency of RPC Systems

- Access transparency
  Yes, the same operation gets executed
- Location transparency
  Yes, via the locating
- Migration transparency
  Yes, via the naming service
- Failure transparency
  Maybe, depends on the used error semantics
- Concurrency transparency
  No
- Replication transparency
- Performance transparency
- Scaling transparency

Transparency of RPC Systems

- Access transparency
  Yes, the same operation gets executed
- Location transparency
  Yes, via the locating
- Migration transparency
  Yes, via the naming service
- Failure transparency
  Maybe, depends on the used error semantics
- Concurrency transparency
  No
- Replication transparency
  Sometimes
- Performance transparency
- Scaling transparency

Transparency of RPC Systems

- Access transparency
  Yes, the same operation gets executed
- Location transparency
  Yes, via the locating
- Migration transparency
  Yes, via the naming service
- Failure transparency
  Maybe, depends on the used error semantics
- Concurrency transparency
  No
- Replication transparency
  Sometimes
- Performance transparency
  No
- Scaling transparency

Transparency of RPC Systems

- Access transparency
  Yes, the same operation gets executed
- Location transparency
  Yes, via the locating
- Migration transparency
  Yes, via the naming service
- Failure transparency
  Maybe, depends on the used error semantics
- Concurrency transparency
  No
- Replication transparency
  Sometimes
- Performance transparency
  No
- Scaling transparency
  For RMI yes, by the object orientation, for other RPCs sometimes

Important takeaway messages of this chapter

- RPCs provide a possibility to call functions on a remote host as if this would happen locally
- Important elements of an RPC system are the IDL, its compiler, and the binder
- Multiple error semantics exist which can be handled below or on top of the RPC system