Script **generated by TTT**

Title: Distributed_Applications (15.05.2012)

Date: Tue May 15 14:30:31 CEST 2012

Duration: 91:59 min

Pages: 27

LDAP - Lightweight Directory Access Protocol

LDAP is a ***protocol*** supporting the access to and update of directory information. It is an open industry standard.

LDAP is used by the **IntegraTUM** project to provide a university-wide directory service at TUM.

Basics

LDAP architecture

Information model

Naming model

Functional model

ldif - exchange format

Generated by Targeteam

The functional model defines operations for accessing and modifying directory entries. Among others LDAP supports the following directory operations:

- create a LDAP entry
- delete a LDAP entry
- update a LDAP entry, e.g. modification of the distinguished name (= move in DIT)
- compare LDAP entries
- search for LDAP entries which meet certain criteria

Search

Generated by Targeteam

The search operation allows a client to request that an LDAP server search through some portion of the DIT for information meeting user-specified criteria in order to read and list the result(s).

Examples

- find the postal address for cn=John Smith,o=IBM,c=DE.
- find all entries which are children of ou=Informatik,o=TUM,c=DE.

Search constraints.

- base object: defines the starting point of the search. The base object is a node within the DIT.
- scope: specifies how deep within the DIT to search from the base object, e.g.
  - baseObject: only the base object is examined.
  - singleLevel: only the immediate children of the base object are examined; the base object itself is not examined.
  - wholeSubtree: the base object and all of its descendants are examined.
- filter: search filter on entry attributes; Boolean combination of attribute value assertions
  - example: (&(cn=schmi*)(!(c=de)))

Code example

Generated by Targeteam

```
#define SEARCHBASE "o=TUM,c=DE"
LDAP *ld;
char *User = NULL;
char *Passwd = NULL;
char searchfilter[] = "cn=Mayr";
/* open a connection */
if ((ld = ldap_open("ldapserver.in.tum.de", LDAP_PORT)) == NULL) exit(1);
/* authenticate as nobody */
if (ldap_simple_bind_s(ld, User, Passwd) != LDAP_SUCCESS) {
    ldap_perror(ld, "ldap_simple_bind_s");
    exit(1);
}
/* search the database */
if (ldap_search_s(ld, SEARCHBASE, LDAP_SCOPE_SUBTREE, searchfilter, NULL,
0) != LDAP_SUCCESS) {
    ldap_perror(ld, "ldap_search_s");
    exit(1);
}
.....
/* close and free connection resources */
ldap_unbind(ld);
```

ldif - exchange format

ldif = LDAP Data Interchange Format; it is used to import and export directory information.

```
dn: cn=Informatik
cn: Informatik
objectclass: top
objectclass: groupOfNames
member: cn=Baumgarten,Uwe, mail=baumgaru@in.tum.de
member: cn=Schlichter,Johann, mail=schlicht@in.tum.de
....
dn: cn=Baumgarten,Uwe, mail=baumgaru@in.tum.de
cn: Baumgarten,Uwe
modifytimestamp: 20001213084405Z
mail: baumgaru@informatik.tu-muenchen.de
givenname: Uwe
sn: Baumgarten
objectclass: top
objectclass: person
....
dn: cn=Schlichter,Johann, mail=schlicht@in.tum.de
cn: Schlichter, Johann
modifytimestamp: 20001213084406Z
mail: schlicht@in.tum.de
```

Client-server model

The client-server model implements a sort of ***handshaking principle*** , i.e., a client invokes a server operation, suspends operation (in most of the implementations), and resumes work once the server has fulfilled the requested service.

Terms and definitions

Concepts for client-server applications

Processing of service requests

File service

Time service

Definition: A **time service** provides a synchronized system-wide time for all nodes in the network.

Name service

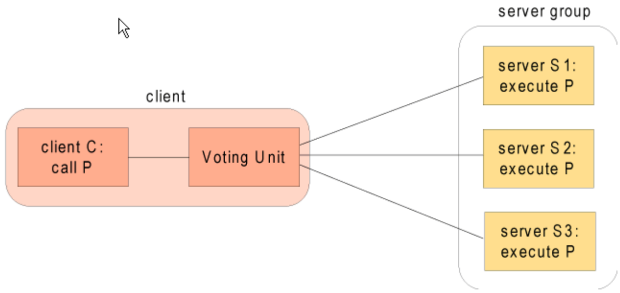
LDAP - Lightweight Directory Access Protocol

Failure tolerant services

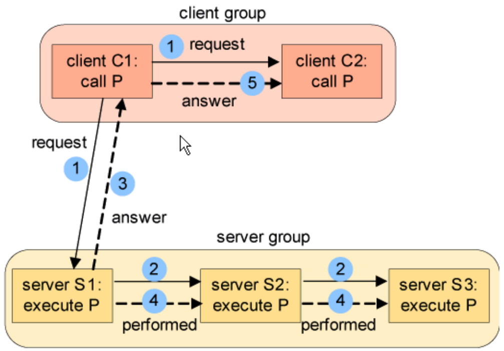
Generated by Targeteam

Modular redundancy

Client requests are sent to and processed by all server replicas (active replication). Each server replica sends its result to the voting unit of the client. The voting unit decides on the received results (e.g. majority voting).



Generated by Targeteam

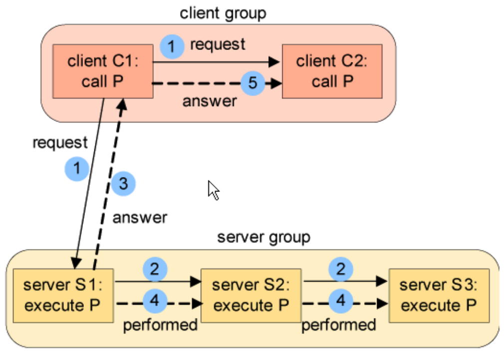


At any specific time, there is only one replica acting as master (primary replica); RPC requests are always propagated to the primary replica; at checkpoints the current state is propagated to the secondary replicas.

in case of an error the master is replaced by a backup replica.

distinction between hot and cold standby.

Generated by Targeteam



At any specific time, there is only one replica acting as master (primary replica); RPC requests are always propagated to the primary replica; at checkpoints the current state is propagated to the secondary replicas.

in case of an error the master is replaced by a backup replica.

distinction between hot and cold standby.

Generated by Targeteam

Distributed Applications - Verteilte Anwendungen

- Prof. J. Schlichter
  - Lehrstuhl für Angewandte Informatik / Kooperative Systeme, Fakultät für Informatik, TU München
  - Boltzmannstr. 3, 85748 Garching

    Email: schlichter@in.tum.de

    Tel.: 089-289 18654

    URL: http://www11.in.tum.de/

Generated by Targeteam

Definition: Birrell and Nelson (1982) define an **RPC** as a synchronous flow of control and data passing scheme achieved through procedure calls between processes running in separate address spaces where the needed communication is via small channels (with respect to bandwidth and duration time).

synchronous : The calling process (client) is blocked until it receives the answer of the called procedure (server); the answer contains the results of the processed request.

procedure calls : the format of an RPC call is defined by the signature of the called procedure.

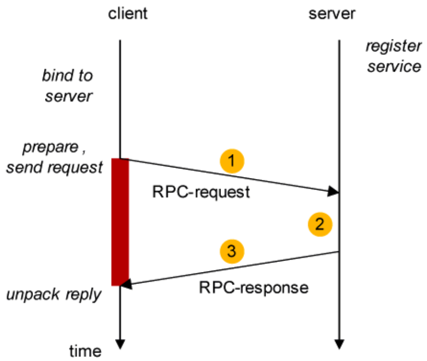
different address spaces : it is necessary to handle pointers during parameter passing different from local procedure calls.

small channel : reduced bandwidth for communication between involved computers.

Generated by Targeteam

Neither the client nor the server assume that the procedure call is performed over a network.

Control flow for RPC calls



[Differences between RPC and local procedure call]

[Basic RPC characteristics]

[RPC and OSI]

[RPC vs message exchange]

For an RPC, the caller and the callee run in different processes.

both processes (caller and callee) have

- no shared address space.
- no common runtime environment.
- different life span of client and server .

Handle errors occurring during a RPC call, e.g. caused by machine crashes or communication failures

- RPC-based applications must take communication failures into consideration.

Basic RPC characteristics

An RPC can be characterized as follows

1. uniform call semantics.
2. "type-checking" of parameters and results.
3. parameter functionality.
4. Optimize response times rather than throughput.
5. new error cases

   bind operation failed; request timed out; arguments are too large

   goal is some transparency concerning exception handling and communication failures (relevant for the programmer).

Integration of the RPC into ISO/OSI protocol stack

layer 7 application layer	client-server model	
layer 6 presentation layer	RPC	hides communication details
layer 5 session layer	message exchange , e.g. request-response protocol	Operating system interface to underlying communication protocols
layer 4 transport layer	transport protocols e.g. TCP/UDP or OSI TP4	transfer of data packets

transport protocols: UDP (User Datagram Protocol) transports data packets without guarantees; TCP (Transmission Control Protocol) verifies correct delivery of data streams.

message exchange: socket interface to the underlying communication protocols.

RPC: hides communication details behind a procedure call and helps bridge heterogeneous platforms.

RPC vs message exchange

RPC	message exchange
synchronous (generally)	asynchronous
1 primitive operation (RPC call)	2 primitive operation (send, receive)
messages are configured by RPC system	message specification by programmer
one open RPC	several parallel messages possible

The RPC protocol defines only the structure of the request/answer messages; it does not supply a mechanism for secure data transfer.

RPC exchange protocols

RPC exchange protocols

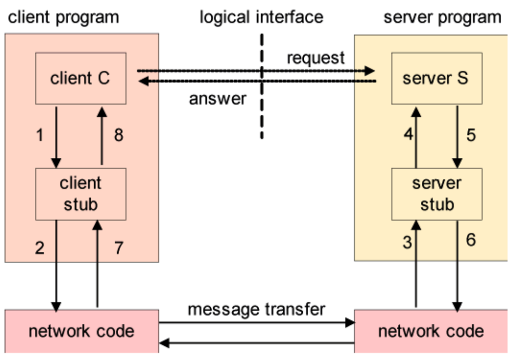
There are different types of RPC exchange protocols

- the request (R) protocol
- the request-reply (RR) protocol
- the request-reply-acknowledge (RRA) protocol.

Remote Invocation (RPC/RMI)

Integration of software handling the communication between components of a distributed application.

- Stubs encapsulate the distribution specific aspects.
- Stubs represent interfaces.
- ***Client Stub*** : contains the proxy definition of the remote procedure P.
- ***Server Stub*** : contains the proxy call for the procedure P.



Client and server stubs have the following tasks during client - server interaction.

1. **Client stub**

   specification of the remote service operation; assigning the call to the correct server; representation of the parameters in the transmission format.

   decoding the results and propagating them to the client application.

   unblocking of the client application.

2. **Server stub**

   decoding the parameter values; determining the address of the service operation (e.g. a table lookup).

   invoking the service operation.

   prepare the result values in the transmission format and propagate them to the client.

How to implement distributed applications based on remote procedure calls?

Distributed application

In order to isolate the communication idiosyncrasy of RPCs and to make the network interfaces transparent to the application programmer, so-called ***stubs*** are introduced.

Stubs

Stub functionality

Implementing a distributed application

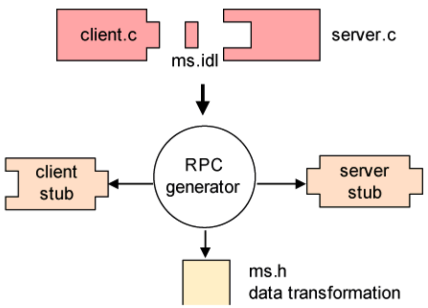
RPC language

Phases of RPC based distributed applications

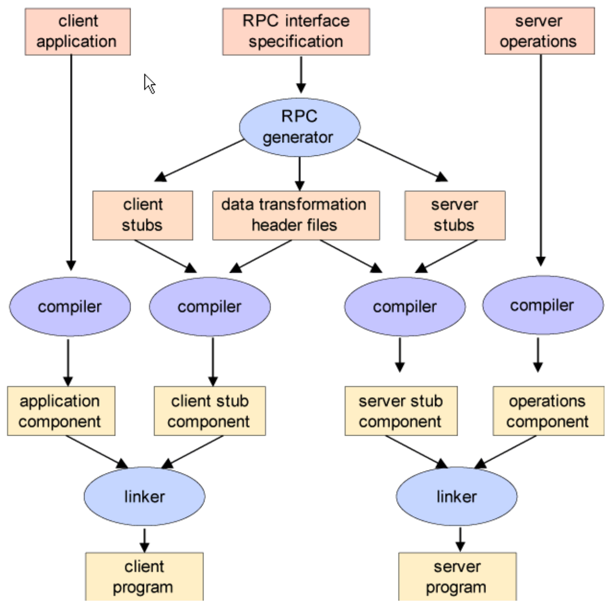
An RPC generator

reduces the time necessary for implementation and management of the components of a distributed application.

a declarative interface description is easier to modify and therefore less error-prone.

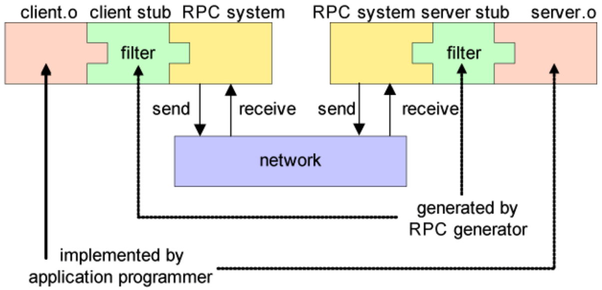


The individual steps for generating a distributed application are illustrated in the following figure.



Structure of a distributed application

The internal structure of a distributed application created using an RPC generator is as follows:



Implementing a distributed application

Manual implementation of stubs is error-prone ⇒ use of a ***RPC generator*** to generate stubs from a declarative specification.

RPC generator

Applying the RPC generator

Structure of a distributed application

Distributed applications based on RPC

How to implement distributed applications based on remote procedure calls?

Distributed application

In order to isolate the communication idiosyncrasy of RPCs and to make the network interfaces transparent to the application programmer, so-called ***stubs*** are introduced.

Stubs

Stub functionality

Implementing a distributed application

RPC language

Phases of RPC based distributed applications

Component binding

The components of a distributed application (client and server) may be started independently; linking of components to enable RPC calls.

Static binding

Static binding takes place when the client program is generated. In this case, the server address is hard-coded within the client program.

Semistatic binding

Dynamic binding

binding sometimes integrates a solution to the factory problem, i.e. the startup of a non-operational server.