Script **generated by TTT**

Title: Distributed_Applications (30.04.2013)

Date: Tue Apr 30 14:30:50 CEST 2013

Duration: 88:56 min

Pages: 27

Bidirectional communication

Usage of the request-answer scheme for message exchange.

Sockets

Call semantics

Communication between sender and receiver is influenced by the following situations

loss of request messages.

loss of answer messages.

sender crashes and is restarted.

receiver crashes and is restarted.

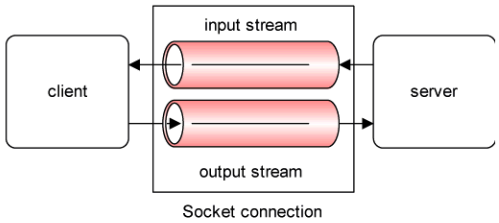
Different types of call semantics

Sockets

Sockets provide a low level abstraction for programming bidirectional communication.

A socket is an application created, OS-controlled interface into which application can both send and receive messages to/from another application.

unique identification: IP-address and port number.



Socket connection

Sockets in Java

Java package java.net

Socket constructors - methods

Socket constructors - methods

constructors of java.net.socket

Socket(): Creates an unconnected socket, with the system-default type of SocketImpl.

Socket(InetAddress address, int port): Creates a stream socket and connects it to the specified port number at the specified IP address.

Socket(Proxy proxy) Creates an unconnected socket, specifying the type of proxy, if any, that should be used regardless of any other settings.

Socket(String host, int port) Creates a stream socket and connects it to the specified port number on the named host.

methods of java.net.socket

void bind(SocketAddress bindpoint): Binds the socket to a local address.

void close(): Closes this socket.

void connect(SocketAddress endpoint): Connects this socket to the server.

void connect(SocketAddress endpoint, int timeout): Connects this socket to the server with a specified timeout value.

Example

Example

Example from the client perspective

```
import java.io.*
import java.net.*

public class EchoClient {
public static void main(String[] args) throws IOException {
Socket echoSocket = null;
Printwriter out = null;
BufferedReader in = null;

try {
echoSocket = new Socket("www.in.tum.de", 7); //create Socket
// create Writer, Reader
out = new PrintWriter(echoSocket.getOutputStream(), true);
in = new BufferedReader(
    new InputStreamReader(echoSocket.getInputStream()) );
}
catch (UnknownHostException e) {
    System.err.println("unkown host in.tum.de");
    System.exit(1);
}
catch (IOException e) {
```

Example

```
    System.err.println("unkown host in.tum.de");
    System.exit(1);
}
catch (IOException e) {
    System.err.println("No I/O from in.tum.de");
    System.exit(1);
}

//read streams
BufferedReader stdIn = new BufferedReader (
    new InputStreamReader(System.in));
String userInput;
while ((userInput = stdIn.readLine()) != null) {
    out.println(userInput);
    System.out.println("echo: " + in.readLine());
}

// close streams and sockets
out.close();
in.close();
stdIn.close();
echoSocket.close();
} }
```

Bidirectional communication

Usage of the request-answer scheme for message exchange.

Sockets

Call semantics

Communication between sender and receiver is influenced by the following situations

- loss of request messages.
- loss of answer messages.
- sender crashes and is restarted.
- receiver crashes and is restarted.

Different types of call semantics

Different types of call semantics

Any communication between a sender and a receiver is subject to communication failures. Therefore, we distinguish between different call semantics.

at-least-once semantics

exactly-once semantics

last semantics

Under a last semantics, the requested service operation is processed once or several times, however, only the last processing produces a result and, potentially, some side-effects.

at-most-once semantics

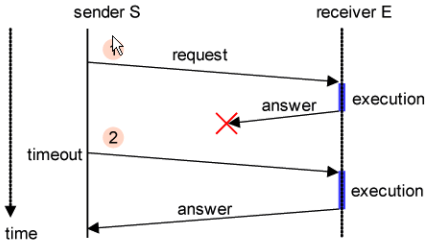
Under an at-most-once semantics, the requested service operation is processed once or not at all.

Example for providing at-most-once semantics

- After timeout at the sending site the request is not retransmitted.
- The request is transmitted in the context of a transaction.

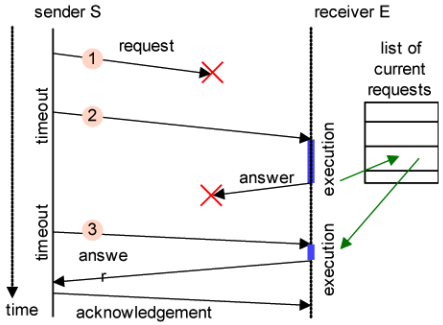
at-least-once semantics

Under an at-least-once semantics, the requested service operation is processed once or several times.



exactly-once semantics

requested service operation is processed exactly once: detection of repeatedly sent requests.



Different types of call semantics

Any communication between a sender and a receiver is subject to communication failures. Therefore, we distinguish between different call semantics.

at-least-once semantics

exactly-once semantics

last semantics

Under a last semantics, the requested service operation is processed once or several times, however, only the last processing produces a result and, potentially, some side-effects.

at-most-once semantics

Under an at-most-once semantics, the requested service operation is processed once or not at all.

Example for providing at-most-once semantics

After timeout at the sending site the request is not retransmitted.

The request is transmitted in the context of a transaction.

Bidirectional communication

Usage of the request-answer scheme for message exchange.

Sockets

Call semantics

Communication between sender and receiver is influenced by the following situations

loss of request messages.

loss of answer messages.

sender crashes and is restarted.

receiver crashes and is restarted.

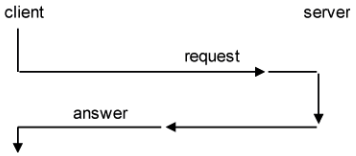
Different types of call semantics

Paradigms for distributed applications

Information Sharing
Message exchange
Naming entities
Bidirectional communication
Producer-consumer interaction
Client-server model
Peer-to-peer model
Group model
Taxonomy of communication
    Message serialization
Levels of Abstraction

Request-Answer Interaction

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.

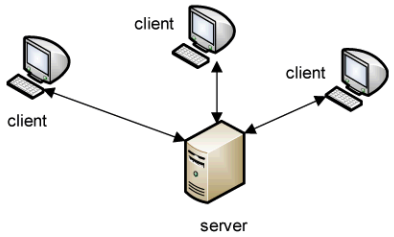


SOA

Service-oriented architecture (SOA): abstract architectural approach
    loose coupling and dynamic binding between services
    based on principles of modularized software and interface/component-based design
Collection of services
    services communicate with each other, e.g. data passing or remote invocation
    each service must manage its own data
SOA contains 3 roles: service requestor, service provider and service registry.
Web services represent an implementation of SOA concept (currently the most important one)

Client-server model

A central component (the server) provides a service to requesting clients.



Request-Answer Interaction
SOA
Examples for servers
    In a distributed environment, a server manages access to shared resources (e.g. a file server).
    Problems:
        server crash ⇒ resource is no longer available in the network.
        server becomes a bottleneck for accessing the resource.
    Internet Explorer, Netscape/Opera browser are examples for clients and Apache Web-Server is an example for a server.
Web Server - HTTP

Web Server - HTTP

Communication between Web Browser and Web Server is based on the HTTP protocol

- stateless protocol.
- based on TCP sockets using typically port 80.
- session information is handled by the application layer (cookies).

HTTP protocol supports

- the methods get, put, post, ..
- return values / status code, such as
  - 404: not found
  - 401: unauthorized
  - 400: bad request

Paradigms for distributed applications

Information Sharing
Message exchange
Naming entities
Bidirectional communication
Producer-consumer interaction
Client-server model
Peer-to-peer model
Group model
Taxonomy of communication
- Message serialization

Levels of Abstraction

Client-Server vs. Peer-to-Peer

Client-Server

- Servers are centrally maintained and administered
- Client has fewer resources than a server

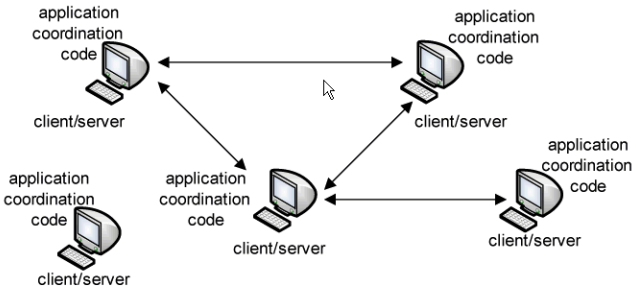
Peer-to-Peer (P2P)

- A peer's resources are similar to the resources of the other participants.
- peers communicate directly with other peers and share resources.
- Issues of P2P
  - Peer discovery and group management
  - Data location and placement
  - Reliable and efficient file exchange
  - Security/privacy/anonymity/trust

Peer-to-peer model

All processes play a similar role

- interacting cooperatively as peers to perform a distributed computation.
- there is no distinction between clients and servers.
- clients talk directly to one-another.



Client-Server vs. Peer-to-Peer
Napster
Gnutella
Other System Examples

Legal issues

When service was launched, Napster designers hoped they had a way around the legal limits of sharing music:

- clients advertise stuff
- if some of that stuff happens to be music. That is the responsibility of the person who does it.
- the directory server "helps clients to advertise stuff" but it does not endorse the sharing of protected intellectual property.
- Napster is making money by integrating Ads.
- In the court case the judges saw it differently: Napster's clear purpose is to facilitate theft of intellectual property.

Gnutella

Gnutella was one of the first examples of a pure P2P system

- system is not run by a single company
- no nodes which act only as servers; Gnutella eliminates the directory server.

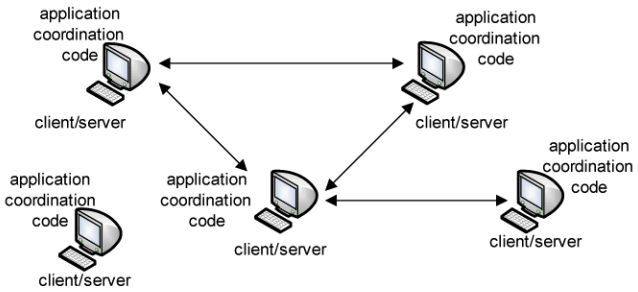
for sharing files the user must connect to the Gnutella network, a loose federation of computers running Gnutella

- for connection the computer only has to know the address of one other Gnutella machine, e.g. machines published at well known web sites.
- at first connection the computer receives hundreds of addresses of machines which may be used at subsequent occasions.
- a Gnutella program tries to maintain 3 or 4 connections to other Gnutella machines at any one time.
- find a file: send request with file name and current hop count to its neighbors.
  - neighbor has matching file: respond with the location of the file
  - increment hop count;
  - if hop count < maximum hop count, then propagate request to its neighbors.

Peer-to-peer model

All processes play a similar role

- interacting cooperatively as peers to perform a distributed computation.
- there is no distinction between clients and servers.
- clients talk directly to one-another.



Client-Server vs. Peer-to-Peer

Napster

Gnutella

Other System Examples

Gossip-based Approach

Propagate information in the same way as epidemic diseases spread.

- approach explained informally
  - time t_0 : suppose I know something new
  - time t_1 : I pick a friend and tell him; now 2 people know.
  - time t_2 : we each pick a friend and tell them; now 4 people know
  - time t_3 : .....
- information spreads at exponential rate.
  - due to re-infection information spreads at approx. 1.8^k after k rounds.
- combination of push and pull works best.
- algorithm is quite robust and scalable
  - information travels on exponentially many paths.
  - difficult to slow down.
  - the load of the participating nodes is independent of the system size.
  - information spreads in log(system size) time.
  - network load is linear in system size.

Other System Examples

BitTorrent is a P2P communications protocol for file sharing

the recipients of data also supply data to newer recipients,

reducing the cost and burden on any given individual source.

reducing dependence upon the original distributor.

eDonkey is a P2P file sharing network

used primarily to exchange audio and video files and computer software.

Files identified using compound MD4 hash checksums, which are a function of the bit content of the file.

Overnet as successor of eDonkey protocol.

Gossip-based Approach

Paradigms for distributed applications

Information Sharing
Message exchange
Naming entities
Bidirectional communication
Producer-consumer interaction
Client-server model
Peer-to-peer model
Group model
Taxonomy of communication
Message serialization
Levels of Abstraction

Message serialization

certain order for message delivery

messages to a group of recipients: messages arrive in different order, due to different transmission times.

One sender

There are the following ordering schemes:

according to the message arrival on the recipient's side; different receivers can have different message arrival sequences.

according to message sequence number generated by the sender; this approach is sender-dominated.

receiver creates a serialization according its own criteria.

Several senders

Several senders

If several senders are involved, the following message ordering schemes may be applied:

1. no serialization.
2. loosely-synchronous.

   There is a loosely synchronized global time which provides a consistent time ordering.
3. virtually-synchronous.

   The message order is determined by causal interdependencies among the messages. For example, a message N has been sent after another message M has been received, i.e. N is potentially dependent on M.
4. totally ordered.

   by token: before a sender can send a message, it must request the send token.

   a selected component (the coordinator) determines the order of message delivery for all recipients.