Slide 1

Centre International de Mathématiques Pures et Appliquées
International Center for Pure and Applied Mathematics

A CIMPA-UNSA-INRIA-UNESCO-VIETNAM SCHOOL ON

Objects and Network

Part : Sockets and Client/server Courses

Emmanuel Chailloux

Emmanuel.Chailloux@lip6.fr

http://www-spi.lip6.fr/~emmanuel

Slide 2

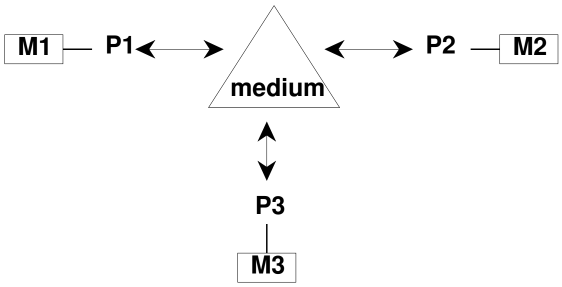
SUMMARY

- Distributed memory model
- Sockets : Generic client/server
- Internet services : HTTP reader
- Dynamic pages for the Web
- Databases : SQL server and JDBC, SQLJ
- Architectures for client/server

Slide 3

Distributed memory

Each processus has its **own** memory space and communicates with the others *through* a medium (protocols).



⇒ explicit communication and implicit synchronization!!!

Slide 4

Communication on sockets : socket types

- UDP (User Datagram Protocol)/IP (Datagram sockets)
  no order - may lose packets - fast
- TCP (Transmission Control Protocol)/IP (Stream sockets)
  delivery in the order - no lost packets - slow

Sockets communication have :

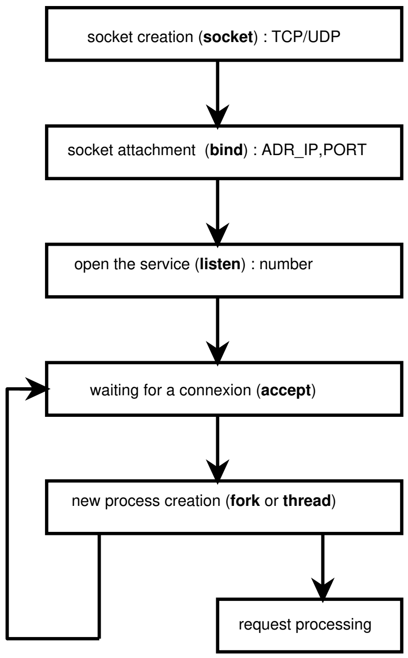
- a server : waiting for incoming requests
- a client : initiates a connection, and then passes or requests informations to the server.

Communications are **asymmetric**!!!

Slide 5

Communication on sockets

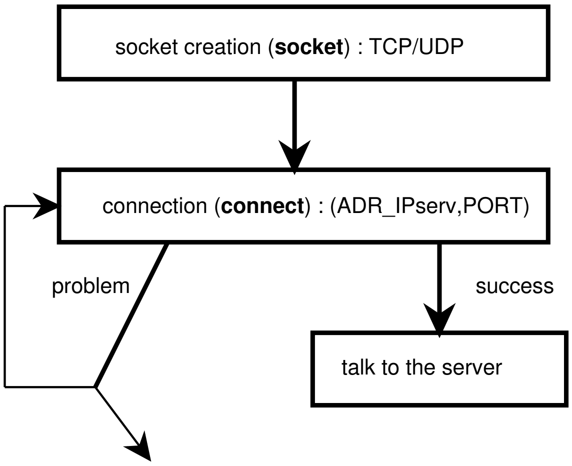
Server scheme :



Slide 6

Communication on sockets

Client scheme :



Slide 7

Main classes for networking

in package `java.net` :

Low level:

- InetAddress : IP address class
- DatagramPacket : datagram packet class
- DatagramSocket : using UDP

High level:

- ServerSocket : server Socket class
- Socket : client Socket class
- URL : Uniform Resource Locator (an object on the Web)

Slide 8

ServerSocket and Socket classes

Main methods:

ServerSocket:	**Socket:**
ServerSocket(int port)	**Socket**(String host,int port)
Socket **accept**()	
void **close**()	
	InputStream **getInputStream**()
	OutputStream **getOutputStream**()

All these methods can throw an `IOException`,
and need `throws IOException`.

Slide 9

Example : CAPITALIZER server

Service: the server receives a line, converts it in CAPITALIZE and sends it to its client.

Organization:
Server side : two classes :

- *Server* class : creates a socket and waits for a connection
- *Connection* class : treats the request

Client side : one trivial class :

- *Client* class : waits for a inpt line from the keyboard, sends it to the server, waits for the answer and displays it.

Protocol:

A line which contains only the word *end* closes the connection with the client!!!

Slide 10

Server part

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

public class Server extends Thread {
  protected static final int PORT =45678;
  protected ServerSocket hear;


Server ()
{ try
     { hear = new ServerSocket(PORT);}
  catch (IOException e)
     {System.err.println(e.getMessage());
      System.exit(1);
     }
  System.out.println("Server hears on port : "+PORT);
  this.start();
}

public void run ()
{ try
     {while (true)
         {Socket client=hear.accept();
          Connection c = new Connection (client);}}
  catch (IOException e)
     {System.err.println(e.getMessage());
      System.exit(1);}
}

public static void main (String[] args)
  {new Server();}

}
```

Slide 11

Connection part

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

class Connection extends Thread {
  protected Socket client;
  protected BufferedReader in;
  protected PrintWriter out;

  public Connection(Socket client_soc)
  { client=client_soc;
    try
     { in=new BufferedReader(
          new InputStreamReader(client.getInputStream()));
       out=new PrintWriter(client.getOutputStream());}
    catch (IOException e)
     {try {client.close();}
      catch (IOException e1){}
      System.err.println(e.getMessage());
      return;}
    this.start();
  }
  public void run()
  { try
      { while (true)
         {String line=in.readLine();
          if (line.toUpperCase().compareTo("END")==0) break;
          System.out.println(line.toUpperCase());
          out.println(line.toUpperCase());
          out.flush();
         }}
     catch (IOException e)
      {System.out.println("Connection : "+e.toString());}
     finally
      {try {client.close();} catch (IOException e) {}}

  }

}
```

Slide 12

Client part

```
import java.io.*;
import java.net.*;
public class Client {
  protected static final int PORT=45678;
  public static void main(String[] args)
  {
    Socket s=null;
    if (args.length ≠ 1)
      {System.err.println("Usage: java Client <hote>");
       System.exit(1);}
    try
      { s=new Socket (args[0],PORT);
        BufferedReader inChannel = new BufferedReader(
          new InputStreamReader(s.getInputStream()));
        DataInputStream console = new DataInputStream
(s.getInputStream());
        PrintWriter outChannel = new
PrintWriter(s.getOutputStream());
        System.out.println("Connection : "+ s.getInetAddress()+"
port : "+s.getPort());
       String line = new String();
       char c;
       while (true)
         { System.out.print("?"); System.out.flush();
           line = "";
           while ((c=(char)System.in.read()) ≠ '\n')
{line=line+c;}
           outChannel.println(line);
           outChannel.flush();
           line=inChannel.readLine();
           if (line == null)
             {System.out.println("Connection disapears"); break;}
           System.out.println("!"+line);
         }
      }
    catch (IOException e) {System.err.println(e);}
    finally { try {if (s ≠ null) s.close();} catch (IOException e2)
{} }
  }
}
```

Slide 13

Execution

Server side:

```
$ java Server
Server hears on  port : 45678
THIS IS THE END
MY ONLY FRIEND THE END
```

Client side:

```
$ java Client asimov
Connection : asimov/132.227.69.15 port : 45678
?this is the end
!THIS IS THE END
?my only friend the end
!MY ONLY FRIEND THE END
?end
Connection disappears
```

Slide 14

Internet services

name	port	socket type	description
ftp	21	tcp	file transfer
telnet	23	tcp	connection to remote systems
smtp	25	tcp	mail protocol
name	42	udp	nameserver
tftp	69	udp	trivial ftp
http	80	tcp	for the Web
pop2	109	tcp	Post office

A same service can exist on different ports as http. The Internet protocols are described in documents *Request For Comments* (RFCs),available online at : `http://www.internic.net/std`

Slide 15

Others services

From Unix world:

- NFS (Network File System), LPD (printer daemon)
- X-window
  - server manages screen, keyboard and mouse
  - clients are graphical applications, which talk with the server ; a special client, the window manager, gives the *look and feel.*

Others examples:

- databases servers : SQL servers
- networking games (xpilot, xgis, Chess Server, Go Server (IGS : http://igs.nuri.net/), MUD, . . .)

Slide 16

World Wide Web

- WWW : World Wide Web
- HTTP : Hyper Text Transfert Protocol
- HTML : Hyper Text Markup Language
  Hyper Text : multimedia text + links
- URL : Uniform Resource Locater
  service://machine-name/dir1/dir2/resource:port
- Browser : a HTTP Client which can visualize HTML source
  Communicator (Netscape), Explorer (Microsoft), ...
- MIME : Multipurpose Internet Message Extensions
  define Content-type, access-type, ...

Slide 17

HTTP small description

HTTP protocol
(http://www.w3.org/pub/WWW/Protocols/) :

HTTP messages consist of requests from client to server and responses from server to client.

```
HTTP-message   = Request | Response

Request        = Request-Line
                     ...   CRLF

Request-Line = Method SP Request-URI
                SP HTTP-Version CRLF

Request-URI  = absoluteURI | abs_path

Method         = "GET"
               | "HEAD"
               | "POST"
```

Example :
GET /pub/WWW/TheProject.html HTTP/1.0

Slide 18

Example : an http client

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

public class http {
  public static void main(String []args){
    String webserver ="ufr-info-p6.jussieu.fr";
    int http_port = 80;
    Socket sock;
    DataInputStream dis;
    PrintStream ps;

    switch (args.length) {
      case 0: break;
      case 1: {webserver=args[0]; break;}
      case 2: {http_port = (int)Integer.parseInt(args[1]);break;}
      default : { System.err.println("Usage: java http <hote>
<port>");
                  System.exit(1);}
    }

   try {
     sock = new Socket(webserver, http_port);
     dis = new DataInputStream(sock.getInputStream());
     ps = new PrintStream(sock.getOutputStream());

     ps.println("GET /index.html");
     String s= null;
     while ( (s=dis.readLine()) ≠ null) {
       System.out.println(s );
     }
     sock.close();
   }
   catch (Exception e) {System.err.println("IO:"+e.getMessage());
                        System.exit(2);}


  }
}
```

Slide 19

Running http client

```
$ java http www.ufr-info-p6.jussieu.fr
<!-- index.html -->
<HTML>
<HEAD>
<TITLE>Serveur de l'UFR d'Informatique
de Pierre et Marie Curie</TITLE>
</HEAD>
<BODY>
....
<A HREF="mailto:webdev@ufr-info-p6.jussieu.fr">webdev</A>
derni&egrave;re modification le 30/04/98 &agrave; 14h43
</BODY>
</HTML>
<!-- index.html -->
```

Slide 20

using URL class

in `java.net` package

- creating a URL : *URL(URL baseURL, String relativeURL)*
- parsing a URL : getProtocol(), getHost(), getFile(), getPort(), getRef()
- connecting to a URL : openConnection()
- reading from and writing to a URL : getInputStream(), getOutputStream()

Slide 21

http client : new version

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

public class wurl {

  public static void main (String a[]) {
    BufferedReader dis;
    try {
      URL u = new URL("http://cadillac.lip6.fr/index.html");
      dis = new BufferedReader(
         new InputStreamReader(u.openConnection().getInputStream() ));
      String s;
      while ( (s=dis.readLine()) ≠ null) {
        System.out.println( s );
      }
    } catch (Exception e) {System.out.println( e ); }
  }

}
```

Slide 22

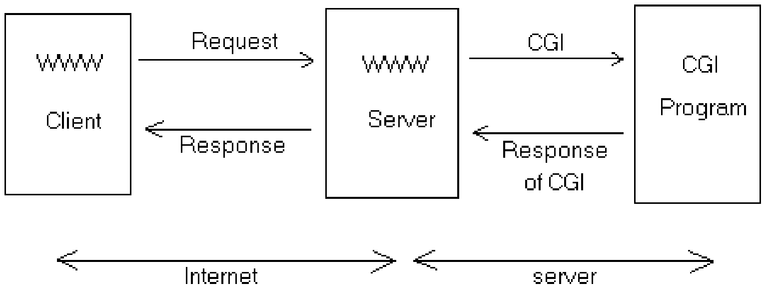
Http servers and dynamic pages

- static pages
  HTML document (an existing file)
- dynamic pages with CGI (*Common Gateway Interface*)
  to run programs in the server
- applets
  to load and to run small graphic application in the client

Dynamic pages don't exist before in the server, but are **generated** by programs!!!

Slide 23

CGI : Common Gateway Interface



- Part of HTTP server to communicate with others programs running in the server
- a standard interface for browsers to run a program in the server.

Slide 24

Input/Output to CGI

- Input :
  - information about the client
  - From data supplied by the user :
    2 methods : GET and POST
    Data encoding
- Output :
  HTTP header through stdout + data (HTML)

Slide 25

Remarks

- use HTML forms (button, list,...)
- can be used with any languages (with stdin and stdout) :
  Java (Servelt API), C, Perl, Lisp, ML, ...
- without state (needing to conserve state inside HIDDEN fields)
- all the work is delegated to the server.
- connection to DBMS.

Slide 26

Applet

- better distribution between client and server : 2 or 3 tiers
- real GUI (Graphic User Interface)
- direct connection to DBMS
- RMI or CORBA calls (remote method invocation)
- better security

Slide 27

Database servers

API JDBC: *Java DataBase Connectivity*

- standard SQL database access interface
- providing uniform access to a wide range of relational databases

To use JDBC:

- you need a JDBC Driver to mediate between JDBC and the database.
- ODBC bridge, implements JDBC API in terms of the ODBC standard C API.

Slide 28

JDBC interfaces

in `java.sql` package

DriverManager:

- to manage drivers loaded list
- to create TCP connections (bfConnection)
- to map URL to connections

Connection:

- to create a stream to a DB
- *in* URL format : `jdbc:mysql://"+asimov+":3306/+table`
- with connection properties : username, passwd
- to create **Statements** (SQL requests)
- to manage transactions

Slide 29

Statement:

- to manage basic SQL requests
- 2 sub-types
  - **PreparedStatement** : pametrized requests (IN)
  - **CallableStatement** : saved procedures (OUT)
- adding parameters : methods : set, ..., indexed access
- to create **ResultSet**

ResultSet:

- to access to a *tuple* from result (SELECT)
- to access via columns : by name, by index
- to convert types between SQL and Java
- to use a stream for the raw data
- to manage basic cursors

Slide 30

basic example : SELECT

```
Connection con = DriverManager.getConnection (
            "jdbc:mysql:"+machine+":"+port+"/"+dbname,
"login", "password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT name, age FROM Table1");
while (rs.next()) {
  int x = rs.getInt("age");
  String s = rs.getString("name");
  System.out.prinln("name: "+ name+" age: "+age);
  }
```

Slide 31

basic example : UPDATE

```
java.sql.PreparedStatement ps =
  conn.createStatement(UPDATE T1 SET cat=? WHERE age=?);

ps.setString(1, Young);
for (int i = 14; i < 19; i++) {
  ps.setInt (2, i);
  int nbTuples = ps.executeUpdate();
  System.out.println ("age: " + i + " nb: " + nbTuples);
  }
```

Slide 32

Interest

Advantages:

- integrated inside core Java
- complete API for dynamic SQL
- a lot of converting functions
- multiple drivers : configuration 2-tier, 3-tier

Disadvantages:

- no typechecking from database
- hard to write a program (verbose,...)
- results are not java objects

Slide 33

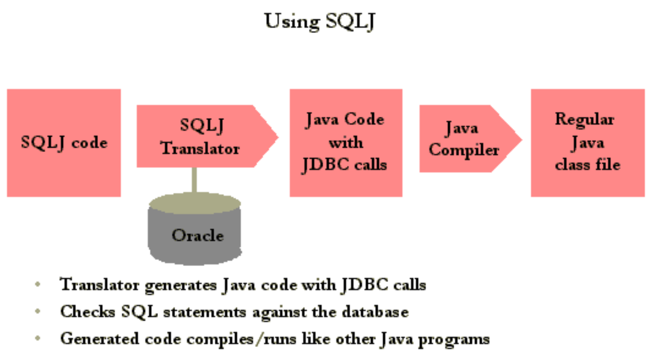
Bridge with ODBC

ODBC : (Open Database Connectivity) :
standard interface between application program and database
using for MS Access on Windows, Informix on Linux, ...

Bridge ODBC:JDBC : URL format :
jdbc:odbc:machine:port/myDB

needs JDBC-ODBC bridge + driver ODBC (given with DBMS)

SQLJ : Embedded SQL



Slide 34

- proposed by IBM, Oracle, Sybase
- `#sql` statements inside Java programs
- precompilation :
  - to check and to optimize SQL source
  - to generate Java classes (Connection, ResultSet)
  - to generate JDBC code

Slide 35

Simplified code

// SQLJ

float w; java.sql.Date x; **int** y; String z;

...

#sql { SELECT C1, C2 INTO :w, :x FROM TAB WHERE C3 = :y AND C4 = :z };

// JDBC

float w; java.sql.Date x; **int** y; String z;

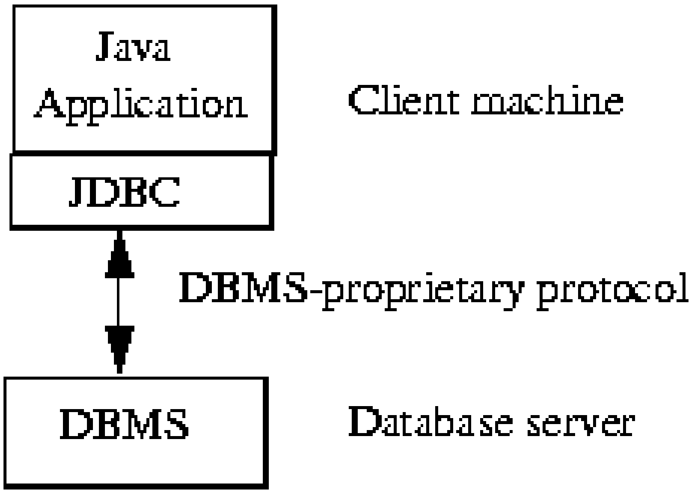
...

PreparedStatement s = connection.prepareStatement(
`"SELECT C1, C2 FROM TAB WHERE C3 = ? AND C4 = ?"`);
s.setInt(1, y);
s.setString(2, z);
ResultSet r = s.executeQuery();
r.next();
w = r.getFloat(1);
x = r.getDate(2);
r.close();
s.close();

Slide 36

Architecture for clients/servers

two tiers:



Slide 37

Architecture for clients/servers

three or more tiers: