TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineerin
Vol. 63, Issue I, March, 2020

USING NETWORKING SERVICES FOR REMOTE ACCESS TO A JAVA ROBOT SIMULATOR

Tiberiu Alexandru ANTAL

Abstract: *The paper presents the use of Java network services for the Internet connection of a Java client who remotely accesses a robot simulator implemented in Java, which integrates a server, implemented as a task, to perform the communication.*

Key words: *client, Java, networking, sever, simulation, socket, task.*

1. INTRODUCTION

Java is an object-oriented programming language that includes all the facilities needed to implement client-server simulation applications [1], [2] over the Internet with the help of the socket concept. The simulations are based on a mathematical model that describes a studied problem and can be purely numerical or have a graphical part [3] - [6].

1.1 Some words on the concept of socket

The concept of socket is connected to the 4.2BSD (Berkeley Software Distribution) Unix operating system, released in 1983. It was conceived as an Application Programming Interface (API) used for a virtual inter-process communication based, among others, on Internet sockets. Sockets can be of two types, stream (bi-directional) or datagram (fixed length destination-addressed messages). A socket is an abstract representation of a process called start-point (sometimes called endpoint if the communication is bi-directional) that connects over the network to a process called endpoint using input and output data streams attached to the socket. This simplifies socket communication as the network is viewed as a sequential file from which data is read or in which data is written. As long as the connection is on the data flows between the two remote processes (start-point and endpoint) in continuous streams, in both directions. A socket is created with the help of an API function and is assigned to a file descriptor which will be used further to access the socket. Socket communication takes place using a protocol. At the time of creation the socket is associated to a socket address, consisting of a port number (TCP port) and a local host network address (IP address). The port number is a logical channel number associated to an application. The inter-process networking is using the TCP/IP protocol to transfer data over the Internet as this adds reliable communication, flow-control, multiplexing and connection-oriented communication [7].

1.2 Java networking services

JDK (Java Developers Kit) 1.0, the first implementation of the cross-platform language called Java, was released by Sun Microsystems Inc. in 1996 providing the developers Internet and intranet application packages (libraries). Network related programming classes were implemented in the `java.net` package being divided into two categories of abstraction [8]:

- low level APIs:
  - network identifiers such as IP addresses;

- sockets: for bi-directional communication;
- interfaces: network interfaces;

- high level APIs:
  - Universal Resource Identifiers
  - Universal Resource Locators
  - Connections.

From JDK 11 enhanced support for HTTP clients was added in the `java.net.http` sub-package. The domain of data transfer over the network is a vast field for this reason the following work will only cover the concepts used to the implementation of the communication application between the client and the robot simulator.

2. THE JAVA CLIENT APPLICATION

The Java client application opens a socket communication point with the following code:

```
Socket s = new Socket(host, port);
```

The host is a `String` containing the IP address of the server. If the socket constructor can't find the host it throws an exception. Once the socket connects to the server two streams can be associated to it for reading (InputStream) and writing (OutputStream) purposes. As this application will only send data to the server an `OutputStream` will do for this purpose. This stream is using binary data organized as bytes. In order to communicate with the server using primitive types it must be turned into a `DataOutputStream`. This contains methods for writing standard data types including a `String` using the `writeUTF()` method. This is using a buffer to increase performance and must be flushed manually when a line of text is sent as follows:

```
OutputStream os=s.getOutputStream();
DataOutputStream out =
DataOutputStream(os);
out.print(line);
out.flush();
s.close();
```

The following class implements completely the computations for a set of data to be sent to the server in order to determine the workspace of the robot thru simulation.

```
class SimClient {
 public static void main(String args[])
throws Exception {
  double f, f1, f2;
  String L;
  Socket s = new Socket("localhost", 3333);
  DataOutputStream out = new
DataOutputStream(s.getOutputStream());

  for (f=0.;f<= 2.*Math.PI; f+=0.1)
   for (f1=0.;f1<=2.*Math.PI;f1+=0.1)
    for (f2=0.;f2<=2.*Math.PI;f2+=0.1) {
     L=String.format("%.6f,%.6f,%.6f",f,
f1,f2);
     out.writeUTF(L);
     out.flush();
    }
  out.writeUTF("stop");
  out.flush();
  out.close();
  s.close();
 }
}
```

3. THE JAVA SERVER APPLICATION

The Java server application is waiting for clients to connect on port 3333. This number is over 1024 which is the upper limit of the so called well-known ports or system ports. As the server will be only reading data from the clients a `DataInputStream` will be enough to get the data. To construct a server socket the port number must be give as follows (this also might throw and Exception):

```
ServerSocket ss = new ServerSocket(port);
Socket s = ss.accept();
```

The `accept()` method will wait for the first client to connect to the server. As long as the method is running and no client is initiating communication with the server the server application is blocked. When the connection is made it will obtain a normal socket to be used further to carry out communication with client. The following code implements the server that reads the `String` object lines from the client and using a `StringTokenizer` object breaks them into `double` type primitives (as `String` objects can't be used to perform mathematical operations).

```
class SimServer {
 public static void main(String args[])
throws Exception {
  long count = 1;
  String L = "";
  ServerSocket ss = new ServerSocket(3333);
  Socket s = ss.accept();
  DataInputStream      din      =      new
DataInputStream(s.getInputStream());
  while (!L.equals("stop")) {
   L = din.readUTF();
   System.out.println(c++ + ": " + str);
   StringTokenizer      stt      =      new
StringTokenizer(str,",");
   while (stt.hasMoreElements()) {

System.out.print(Double.parseDouble(stt.ne
xtElement().toString()));
   }
   System.out.println();
  }
  din.close();   s.close(); ss.close();
 }
}
```

In order to store the data received from the client the server is using a `Vector` class (as this can adapt its number of elements compared to a classical array where the maxim number of elements if fixed). The `JFrame` from Fig.1 contains a `JPanel` and three buttons. The `JPanel` is showing the φ_1, φ_2 and φ_3 angles (in radians) the counter of the current point as well as the total number of the received points.

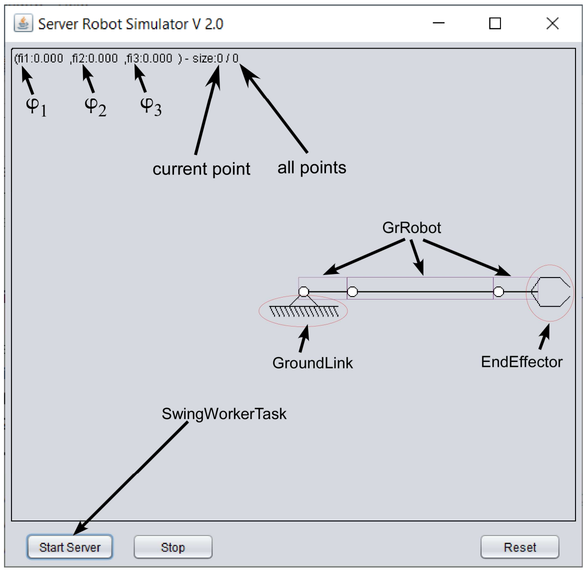

Figure 1. – The GUI of the server-side robot simulator.

The server simulator is built around the following classes: `GrFrame3V1` which extends `javax.swing.JFrame`, `GrPanel3` which extends `javax.swing.JPanel`, `SwingWorkerTask` for the data transfer and Swing object update during processing, `GrRobot` to represent a pivot-link pair, `GroundLink` to represent the ground link, `EndEffector` the represent the end effector of the robot. The Start Server button is starting the server side application to carry out the communication with the client. As long as the communication takes place the Swing GUI is not any more responsive. As given in [2] a separate thread can be used to manipulate the GUI objects if the thread is using `EventQueue.invokeLater()` method to update the GUI. The `SwingWorkerTask` abstract class was made to implement easy this task; the class must be extended and `init()`, `work()`, `update()` and `finish()` methods should be overridden as follows:

```
public void ServerMode() throws Exception,
InterruptedException {
 GrFrame3V1    topFrame    =    (GrFrame3V1)
SwingUtilities.getWindowAncestor(this);
 Runnable task = new SwingWorkerTask() {

  public void init() {
   count = 1;
   topFrame.setLabel("Started ...");
  }

  public void update() {
   topFrame.setLabel(String.format("%-
10d", count));
  }
  public void finish() {
   try {
    topFrame.setLabel("Start Server");
    din.close(); s.close(); ss.close();
    workerThread = null;
   } catch (IOException e) { }
   topFrame.setLabel("Start Server");
  }
  public void work() {
   try {
    ss = new ServerSocket(3333);
    s = ss.accept();
    din                =                new
DataInputStream(s.getInputStream());
    String str = "";
    while
(!Thread.currentThread().isInterrupted()&&
!str.equals("stop")) {
     str = din.readUTF();count++;
     vfi.add(str);doUpdate();
    }
   } catch (IOException e) {
    doFinish(); workerThread.stop();
    workerThread = null;
   }
   doFinish(); workerThread.stop();
   workerThread = null;
  }
 };
 workerThread = new Thread(task);
 workerThread.start();
}
```

To simplify running the code in the event dispatch tread `doUpdate()` and `doFinish()` convenience methods are provided to run `update()` and `finish()` methods when necessary. In Fig. 2 and Fig. 3 the GUI of the simulator is shown for two different cases. In Fig. 2 the communication task is running at the same time with the simulation. The **Strat Server** button is updated in real time to the current **count** of the read positions from the client. In Fig. 3 the communication task has been terminated and only the simulation task is running as long as all the read positions are processed. If the simulation is to fast the **Stop** button can stop the process which can be resumed later. In the simulation takes too long the **Reset** button can be used to clear all received data and the state of the simulator.

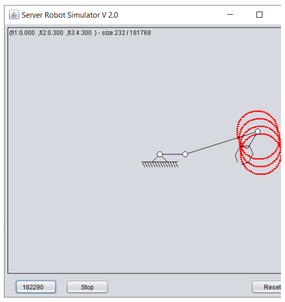
Server Robot Simulator V 2.0
(fi1:0.000 ,fi2:0.300 ,fi3:4.300) - size:232 / 181768
182290
Stop
Reset

Figure 2. – The GUI of the server-side robot simulator during client communication.

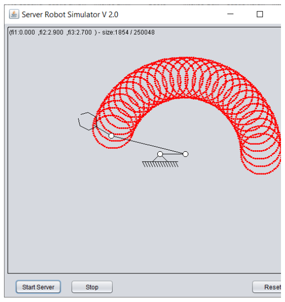
Server Robot Simulator V 2.0
(fi1:0.000 ,fi2:2.900 ,fi3:2.700) - size:1854 / 250048
Start Server
Stop
Reset

Figure 3. – The GUI of the server-side robot simulator after client communication while the simulation is running.

4. REFERENCES

[1] ANTAL, T. A., *Elemente de Java cu JDeveloper - îndrumător de laborator*, Editura UTPRES, 2013, p.150, ISBN: 978-973-662-827-6.

[2] ANTAL, T. A., *Java - Iniţiere - îndrumător de laborator*, Editura UTPRES, 2013, p. 246, ISBN: 978-973-662-832-0.

[3] DETESAN, Ovidiu-Aurelian et al. THE GRAPHICAL SIMULATION OF TRR SMALL-SIZED ROBOT. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 59, n. 3, sep. 2016. ISSN 1221-5872.

[4] DETESAN, Ovidiu-Aurelian. THE NUMERICAL SIMULATION OF TRR SMALL-SIZED ROBOT. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 58, n. 4, nov. 2015. ISSN 1221-5872.

[5] DETESAN, Ovidiu Aurelian. WORKSPACE DETERMINATION FOR THE RTTRR MODULAR SMALL-SIZED SERIAL ROBOT. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 60, n. 1, mar. 2017. ISSN 1221-5872.

[6] CRIŞAN, Adina - Veronica; ŞERDEAN, Florina - Maria; MORARIU - GLIGOR, Radu. THE ANALYSIS OF GEOMETRICAL ERRORS BASED ON POLYNOMIAL INTERPOLATION FUNCTIONS FOR A 5 D.O.F. SERIAL ROBOT. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, [S.l.], v. 60, n. 4, dec. 2017. ISSN 1221-5872.

[7] SCHILD, H. Java: The Complete Reference, Eleventh Edition, McGraw-Hill Education, 2019, p. 1208, 978-1-260-44023-2.

[8] HORSTMANN, C. S., CORNELL, Core Java 2: Volume II – Advances Features, Seventh Edition, Prentice Hall, 2005, ISBN13: 9780131118263.

Utilizarea serviciilor de rețea pentru accesul de la distanţă la un simulator de robot implementat în Java

Lucrarea prezintă modul de utilizare a serviciilor de rețea Java pentru conectarea prin Internet a unui client Java la un simulator de robot implementat în Java care integrează un server pentru derularea comunicaţiei sub forma unui task.

ANTAL Tiberiu Alexandru, Professor, Dr. Eng., Technical University of Cluj-Napoca, Department of Mechanical System Engineering, antaljr@bavaria.utcluj.ro, 0264-401667, B-dul Muncii, Nr. 103-105, Cluj-Napoca, ROMANIA.