A Java Template to Interrogate Knowledge Bases by Client-Server Technology

Nicolae Tăndăreanu

Abstract—The subject developed in this paper is related to some sort of remote connection with a knowledge base for its interrogation. More specifically, we propose to use client server technology to solve this problem. A methodological description is given based on Java technology. Both the server and client side of the application are presented step by step. The way of presentation is divided into stages, each stage is well defined according to the proposed tasks. Each step of the presentation can be easily modified and adapted by a person who wants to write his own application to query a knowledge base by client-server technology. The application is based on Java technology so that we can say that we obtained a Java template. The knowledge base used to exemplify the application running is based on inheritance property in an extended case ([8]). The choice of this kind of knowledge base can be explained by the fact that the inference engine in this case is easier to write than the inference engine for other methods of knowledge representation. The last section enumerates several developing directions.

Keywords—client, server, socket, graphical user interface, inheritance, query, Java technology

I. INTRODUCTION

There are two main entities for every client/server application: one entity is the client and another is the server. The client and server components of the application usually do not reside on the same computer. In other words, the client is installed on a computer that is different from the one hosting the server installation. But logically they are components of the same application.

The client/server programming supposes that the server is a provider of services. In a client-server application the client requests an action or service from the provider of service. The simplest example and the most used application is given by a Web browser and a Web server. When a person addresses a URL in the browser window, it becomes a client which requests a page from a Web server. The server returns an html page to the client, which displays it on its computer.

Client programs request service from a server by sending it a message. Server programs process client requests by performing the tasks requested by clients. Servers are generally passive as they wait for a client request. During these waiting periods servers can perform other tasks. Unlike the client, the server must continually run because clients can request service at any time. Clients on the other hand only need to run when they require service. Many server applications allow for multiple clients to request service.

A very common problem in the domain of databases is the connection to a remote database server. ODBC (Open Database Connectivity) provides a way for client programs to access a databases. This is a standardized API that enables connections to SQL database servers. It was developed according to the specifications of the SQL Access Group and defines a set of function calls, error codes, and data types that can be used to develop database-independent applications. Using MyODBC we can establish a connection to a remote database, export a database to the remote server, import a database from the remote server and establish a link from a local database to a remote database ([6]). Net8 is a fundamental networking technology to establish a network connection and facilitate the transfer of data between a client session and the database. Net8 allows Oracle services and clients to communicate with each other over a network. The most common application for Net8 is to allow clients to talk to database servers, but Net8 also enables server-to-server and other types of communication ([7]). Once the user established a connection to the database server and configured Net8 appropriately, the SQL*PLUS program allows the user to store and retrieve data in the relational database management system ORACLE ([1]).

WS-JDBC (Web Service - Java DataBase Connectivity) is a JDBC driver implemented using Web services, which allows clients to connect to remote databases using the standard JDBC interface. The WS-JDBC driver allows to use server-side classes that implement the necessary parts of the JDBC interface as Web services and client-side classes that are used by applications to invoke those Web service operations ([18]).

The modern network programming is based on a client/server model. A client/server application stores large quantities of data on a high-powered server, while most of the program logic and the user interface is handled by client software running on personal computers. In most cases, a server primarily sends data, while a client primarily receives it. In general the client initiates a conversation, while a server waits for clients to start conversations with it ([11], [9]).

As we can see, the remote interrogation of a data base is an usual action for practitioners in computer science. But the same problem for knowledge bases is less treated in literature. This is because there is a wide variety of knowledge bases.

In this paper we inaugurate a possible research line concerning the remote interrogation of a system of knowledge bases. Some inference engine to interrogate a knowledge base is used. The features of the method used for knowledge representation are fully integrated into this inference engine. To fulfill our purpose we used the client-server technology based on Java mechanisms. Both the server side and the client side of the application are presented. The system is exemplified for the extended inheritance knowledge bases.
The paper is organized as follows. Section II gives a short presentation of the communication by sockets in Java. In Section III we present a minimal set of concepts concerning the extended inheritance knowledge bases. Section IV contains the description of the application server side. In Section V we describe the application client side. The last section contains the conclusions and the possible extension of our study.

II. BASIC JAVA CONCEPTS

In the domain of client-server programming there are two main entities: the server and the client. These two entities communicate one by another by means of real network, that can be Internet or a private network. A computer is identified in a network by means of an IP- address. The processes running on a computer in a network are identified by means of a number on 16 bits named port.

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. The Java platform provides implementations of sockets in the java.net package. Java provides two classes for creating TCP sockets, namely, Socket and ServerSocket. The java.net.Socket class is used by clients to make a connection with a server. The constructors of this class have the following forms:

- Socket(String hostname, int port)
- Socket(InetAddress addr, int port)
- Socket(String hostname, int port, InetAddress localAddr, int localPort)
- Socket(InetAddress addr, int port, InetAddress localAddr, int localPort)

We emphasize that the Socket constructor attempts to connect to the remote server. Data is sent and received with output and input streams.

The java.net.ServerSocket class is used to by server to accept client connections. The constructors for this class have one of the following forms:

- ServerSocket(int port)
- ServerSocket(int port, int backlog)
- ServerSocket(int port, int backlog, InetAddress network-Interface)

The ServerSocket objects use their accept() method to connect to a client. This situation is represented in Figure 1.

![Communication and connections](image)

Fig. 1. Communication and connections

III. EXTENDED INHERITANCE KNOWLEDGE BASES

The best definition of a knowledge base is obtained if we define first the concept of knowledge representation and reasoning system, which is a tuple of components cooperating between them ([19]). Some component describes the representation language and other components define the inference relation and the update tasks. In this paper we use an extended inheritance knowledge base to exemplify a possible dialog server-client to interrogate such a base. Intuitively an extended inheritance knowledge base is a finite set of objects. An object has an unique description of the form (name, parents, attributes), where

- name is the name of the object and this entity identifies uniquely the object;
- parents is a list of object names; each element of this list is a parent of name;
- attributes is a list of attribute names for name and their descriptions; the simplest form of a description is attr(name_attr, val, p), where name_attr is an attribute name, val is the value of this attribute and p is a parameter.

The parameter of an attribute value can represent the certainty of the value, a risk coefficient or a cost for the use of this value.

An interrogation of such knowledge base is specified by a pair (name1, attr1), where name1 is an object name and attr1 is an attribute name. If this is an interrogation then the answer of the system gives the value of the attribute attr1 for the object name1. This value is computed by inheritance: if attr1 is specified by the object name1 and the description is of the form attr(attr1, v1, p1) then v1 is the value of the answer mapping. Otherwise this value is computed from the nearest predecessors of the object name1. If by inheritance we obtain a set \{attr(attr1, v1, p1), . . . , attr(attrk, vk, pk)\} of descriptions, where k ≥ 2, then we choose the value v1 which satisfies the ”choice strategy” given by parameters p1, . . . , pk.

IV. SERVER SIDE APPLICATION

In this section we describe the actions of the server. The actions are described in several steps.

Step 1: connection with a client.
- The server program begins by creating a new ServerSocket object to listen on a specific port 9090.
- If the server successfully binds to its port, then the ServerSocket object is successfully created and the server continues to the next step, accepting a connection from a client.
**Step 2:** A ConnectionHandler object is created to examine and process the requests sent to and the responses received from the client.

**Step 2.1:** Define data structures for knowledge bases and the communication channel between server and a client.

```java
import java.net.*;
import java.io.*;
import java.util.*;
...
public class MultiClientTcpServer{
    public static void main(String[] args)
    {int port = 9090;
        try (ServerSocket server =
                new ServerSocket(port))
        {while (true) {
                System.out.println("Waiting for clients on port ...");
                Socket client = server.accept();
                ConnectionHandler handler =
                        new ConnectionHandler(client);
                handler.start();}
        }catch(Exception ex) {...}
    }

    class ConnectionHandler extends Thread {
        private Socket client;
        BufferedReader reader;
        PrintWriter writer;
        static int count=0;
        String lista_KB="KB_1 KB_2 ...
...
        JTable kb = new JTable(model);
        ...}
    }
    
    Step 2.2: inference engines for knowledge bases.

    //....
    public String calculateAnswerKB1(...) { ...
    } // end method KB1
    ...
    public String calculateAnswerKB2(...) { ...
    } // end method KB2
    public String calculateAnswerKBm(...) { ...
    } // end method KBm
```
For each kind of knowledge base there is a proper inference engine. This is implemented by the method calculateAnswerKBj for \( j \in \{1, \ldots, m \} \).

**Step 2.3:** first communication.

```java
public void run() {
    String message = null;
    String name_obj = null;
    String name_attr = null;
    boolean rasp = true;
    boolean change_kb = true;
    try {
        // questions to and answers from client
        reader = new BufferedReader(new InputStreamReader(
            client.getInputStream()));
        writer = new PrintWriter(client.getOutputStream());
        writer.println("Welcome to my server");
        writer.flush();
        writer.println(lista_KB);
        writer.flush();
    } catch (Exception e) {
        // handle exception
    }

    // first communication
    writer.println("Choose a Knowledge Base!");
    writer.flush();
    writer.println("Select it from the first choice structure.");
    writer.flush();
    name_kb_client = reader.readLine();
    if (name_kb_client.equals("KB_1")) {
        // process KB_1
    } else if (name_kb_client.equals("KB_2")) {
        // process KB_2
    } // end if

    // second communication
    String Val21 = "";
    int jjj = 0;
    int ix = 0;
    String attr5 = "";
    boolean difera = true;
    boolean prima_data = true;
    for (int idx = 0; idx < nr_lines; idx++) {
        ix = 0; difera = true;
        Val21 = (String) kb.getValueAt(idx, 2);
        if (!Val21.equals("")) {
            String words_attr[] = new String[10];
            message = Val21;
            StringTokenizer st1 = new StringTokenizer(message);
            tokenCount = st1.countTokens();
            while (st1.hasMoreTokens()) {
                words_attr[ix] = st1.nextToken();
                ix++;
            } // end while
            for (int ix2 = 0; ix2 < tokenCount; ix2++) {
                String s3 = words_attr[ix2];
                String[] temp = null;
                temp = s3.split(",");
                attr5 = temp[0].substring(5);
                if (prima_data) {
                    all_attributes[jjj] = attr5;
                    prima_data = false;
                    jjj++;
                } else {
                    for (int j7 = 0; j7 < jjj; j7++) {
                        if (all_attributes[j7].equals(attr5)) {
                            difera = false;
                            break;
                        } else {
                            all_attributes[jjj] = attr5;
                            jjj++;
                        }
                    } // end else
                } // end if
            } // end for
            writer.println(jjj);
            writer.flush();
            for (int idx = 0; idx < jjj; idx++) {
                writer.println(all_attributes[idx]);
                writer.flush();
            } // end if
        } // end if
    } // end for

    // final communication
    String val = "";
    while (rasp) {
        writer.println("Select the object name!");
        writer.flush();
        name_obj = reader.readLine();
        writer.println("Select the attribute name!");
        writer.flush();
        name_attr = reader.readLine();
        writer.println("The value of attribute "+
            "name_attr" + " for " + name_obj + " is " + val + " //
            " and fc=" + vec_fcs[index_fcs]);
        writer.flush();
    } // end while
}
```

We observe the following steps:

- The client is invited to choose the name of the knowledge base which will be interrogated. This name is selected from a choice structure of the graphical user interface of the client and this value is sent to server. The server receives this value and assigns it to `name_kb_client`.
- As an effect of the sequence of code for(int i=0; i<nr_lines; i++) {
  writer.println(data2[i][0]);
  writer.flush();
  model.insertRow(i, new Object[] {data2[i][0], data2[i][1], data2[i][2]});
}

the object names of the selected knowledge base are sent to client and the structure `kb` wraps the knowledge base defined by `data2`.

**Step 2.5:** extract the attribute names of the inheritance knowledge base `kb` and send them to client.

```java
String Val21 = "";
int jjj = 0;
int ix = 0;
String attr5 = "";
boolean difera = true;
boolean prima_data = true;
for (int idx = 0; idx < nr_lines; idx++) {
    ix = 0; difera = true;
    Val21 = (String) kb.getValueAt(idx, 2);
    if (!Val21.equals("")) {
        int tokenCount;
        String words_attr[] = new String[10];
        message = Val21;
        String s3 = words_attr[ix2];
        String[] temp = null;
        temp = s3.split(",");
        attr5 = temp[0].substring(5);
        if (prima_data) {
            all_attributes[jjj] = attr5;
            prima_data = false;
            jjj++;
        } else {
            for (int j7 = 0; j7 < jjj; j7++) {
                if (all_attributes[j7].equals(attr5)) {
                    difera = false;
                    break;
                } else {
                    all_attributes[jjj] = attr5;
                    jjj++;
                }
            } // end else
        } // end if
    } // end if
} // end for
```
if(val.equals("")){
    writer.println("The value of attribute "+ //
    name_attr+" for "+name_obj+" is unknown.");
    writer.flush();
}
message=reader.readLine();
if(message.equals("yes")) rasp=true;
else rasp=false;
}
writer.println("Change KB? Change_KB/FINISH buttons");
writer.flush();
message4=reader.readLine();
if(message4.equals("yes")) change_kb=true;
}
client.close();
count--;
System.out.println("Active Connections = " + count);
}
}
This dialog consists of the following:

- The server sends to client the message Select the object name!. The answer of client is stored by name_obj.
- The client is invited by the message Select the attribute name! to indicate its choice. The answer is stored in name_attr.
- The variable var contains the value of the attribute name_attr for name_obj and this value is computed by the method calculateAnswerKB2. The value of val is sent to client.
- The program running on the client side asks whether or not the client wishes another interrogation of the same knowledge base. The client answer is sent to server and the variable message specifies this answer (yes or not).
- In the remainder of the dialog the client is asked if wishes to change the knowledge base or to finish its interrogation.

V. CLIENT SIDE APPLICATION

In this section we describe the actions of the client. For a friendly communication with the server we used a Graphical User Interface having two windows, three choice options and seven buttons. This interface is shown in Figure 3.

Step 1: the use of ApplicationFrame class, windows, choice structures and buttons.

public class C_S extends ApplicationFrame implements ActionListener, ItemListener{
static TerminalTextArea my_Area1;
static TerminalTextArea my_Area2;
// Choice structures
static Choice optiune1=new Choice();
static Choice optiune2=new Choice();
static Choice optiune3=new Choice();
//Buttons
Button ckb=new Button("Confirm KB");
Button cob=new Button("Confirm Ob");
Button cont=new Button("YES");
Button ncont=new Button("NO");
Button another_kb=new Button("Change KB");
Button finish=new Button("FINISH");
Button cat=new Button("Confirm Attr");
static PrintStream my_outs1 = null;
static PrintStream my_outs2 = null;
...

Fig. 3. Interface of the client

//restrictions.....

Step 2: establish a connection with server.

Step 3: read first message and initiate the structures of GUI.

Step 4: choose a knowledge base and introduce the object names and the corresponding attribute names into the Choice structures of GUI

...
sem_kb=false;
while(!semafor1) {
    if (sem_kb) {writer1.println(ales_kb);
        writer1.flush();semafor1=true;}
}

//the knowledge base contains nr_elem objects-----
int nr_elem=0;
nr_elem=Integer.valueOf(reader1.readLine());
//introduce the objects into optiune2---------
optiune2.removeAll();
String elem;
for (int j1=0;j1<nr_elem;j1++){
    elem=reader1.readLine();
    optiune2.addItem(elem);
//introduce the object attributes into optiune3---
optiune3.removeAll();
for (int j1=0;j1<nr_elem;j1++){
    elem=reader1.readLine();
    optiune3.addItem(elem);}

Step 5: the client chooses an object name and an attribute name and sends to server

while(raspuns){
    //read message "Select the object name!"------
    message=reader1.readLine();
    //display this message
    my_outs1.println(message);
    //select the object------------------------
    while(!semafor1) {
        if (sem_kb) {writer1.println(ales_kb);
            writer1.flush();semafor1=true;}
    }
    if (sem_kb) {writer1.println(ales_kb);
        writer1.flush();}
    semafor1=false; sem_kb=false;
    //read message "Select the attribute name!"--
    message=reader1.readLine();
    //display this message
    my_outs1.println(message);
    //select the attribute------------------------
    while(!semafor1) {
        if (sem_attr) {writer1.println(ales_attr);
            writer1.flush();}
    }
    semafor1=false; sem_attr=false;

Step 6: display the answer given by server

my_outs1.println("The answer from server : ");
my_outs1.println(reader1.readLine());

Step 7: the client can continue to interrogate the same or another knowledge base

my_outs1.println("Another interrogation? (yes/no)");
semafor1=false;
sem_cont=false;
sem_ncont=false;
while(!semafor1) {
    //end while 1
    //read message "Another interrogation? (yes/no)
    message=reader1.readLine();
    //display this message to client
    my_outs1.println(message);
    if (sem_cont){
        my_outs2.println("Yes.");
        writer1.println("Yes");writer1.flush();
        semafor1=true;raspuns=true;
    } else {
        my_outs2.println("You do not continue with the same knowledge base.");
        writer1.println("You do not continue with the same knowledge base.");
        raspuns=false;
    }
    semafor1=false; sem_cont=false;
}

 VI. CONCLUSIONS AND FUTURE WORK

In this paper we tried to model communication between the server and client to query a knowledge base. Both the server and client side of the application are presented step by step. With this features we can say that this work is a methodological one. The reader can easily adapt each step according to his vision.

We relieve the following features of the application presented in this paper:

1) In order to implement the application the Java mechanisms were used.
2) The client side uses a graphical user interface to visualize the communication with the server. This interface contains two windows: the first window is used to display the messages of the server and the second window is used to display messages which confirm the choice of the client. The client actions are guided by buttons.
3) There is a particular aspect of the client GUI. This is established by the GUI components. In order to exemplify the application execution we used an extended inheritance knowledge base. For this reason the choice components of GUI contains the object names and the attribute names.
As a future work we emphasize the following directions to improve the implementation presented in this paper:

- Describe the implementation of the inference engine for various kinds of knowledge bases.
- Adapt the GUI client components in the vision of the use of other kinds of knowledge bases. For example, the first Choice component can be preserved to specify the name of the knowledge base. If the server contains knowledge bases using a graph based knowledge representation (labeled stratified graphs, semantic schemas, conditional graphs, conceptual graphs etc) then the inference engine uses a path based reasoning. In this case the other two Choice components of the GUI can contain the initial and the final node respectively.
- Extend the implementation introducing the communication by text in natural languages. The design of the system GINLIDB (Generic Interactive Natural Language Interface to Databases) can be taken as a starting point in this study ([5]).
- Try to introduce the communication by voice in a natural language.

REFERENCES


