Java Interrogation of an Homogeneous System of Inheritance Knowledge Bases by Client-Server Technology

Nicolae Ţăndăreanu

Abstract—The subject developed in this paper is connected by the remote interrogation of a knowledge base. We suppose we have a collection of the same kind of knowledge bases, namely, extended inheritance knowledge bases. We use the client-server technology to query each element of such a system of knowledge bases. To implement the application we used Java technology. The reasoning process is based on an inference engine. The mechanism of this engine is based on the extended inheritance presented in [18], [22] and [23]. 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/her own application to query a knowledge base by client-server technology. The use of the extended inheritance knowledge bases 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 ([8]). 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 ([9]). 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 ([25]).

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 ([13], [10]).

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.

N. Ţăndăreanu, Faculty of Mathematics and Computer Science, University of Craiova, http://inf.ucv.ro/˜ntand/en/, email: ntand@rdslink.ro

The features of the method used for knowledge representation are fully integrated into this inference engine. Java can create applications that are distributed between clients and servers in a network. For this reason in order 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 server side application. In Section V we describe the client side application. The last section contains the conclusions of our study and several possible extensions of this application.

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 form:

- ServerSocket(int port)
- ServerSocket(int port, int backlog)
- ServerSocket(int port, int backlog, InetAddress networkInterface)

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

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

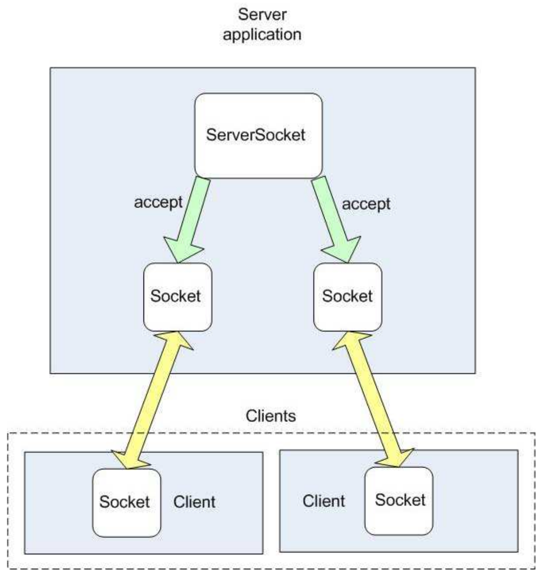


Fig. 1. Comunication and connections

between them ([26]). 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 $(name_1, attr_1)$, where $name_1$ is an object name and $attr_1$ is an attribute name. If this is an interrogation then the answer of the system gives the value of the attribute $attr_1$ for the object $name_1$. This value is computed by inheritance: if $attr_1$ is specified by the object $name_1$ and the description is of the form $attr(attr_1, v_1, p_1)$ then v_1 is the value of the answer mapping. Otherwise this value is computed from the nearest predecessors of the object $name_1$. If by inheritance we obtain a set $\{attr(attr_1, v_1, p_1), \ldots, attr(attr_k, v_k, p_k)\}$ of descriptions, where $k \geq 2$, then we choose the value v_i which satisfies the ”choice strategy” given by the parameters $p_1, \ldots, p_k$.

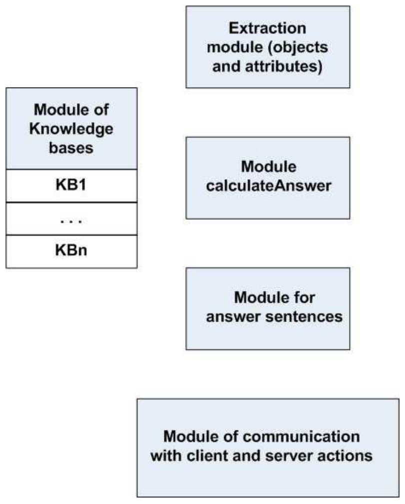

Extraction module (objects and attributes)
Module of Knowledge bases
KB1
. . .
KBn
Module calculateAnswer
Module for answer sentences
Module of communication with client and server actions

Fig. 2. Modules used by server actions

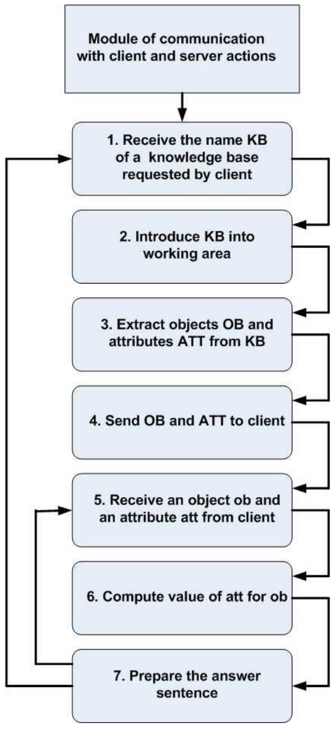

Module of communication with client and server actions
1. Receive the name KB of a knowledge base requested by client
2. Introduce KB into working area
3. Extract objects OB and attributes ATT from KB
4. Send OB and ATT to client
5. Receive an object ob and an attribute att from client
6. Compute value of att for ob
7. Prepare the answer sentence

Fig. 3. The server actions

IV. Server side application

The server side application contains *modules* and *actions*. The modules are presented in Figure 2 and they are used by the server actions. The actions are presented in Figure 3.

In the remainder of this section we describe step by step the actions of the server.

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.
- When a connection is successfully established, the accept method returns a new Socket object which is bound to the same local port and has it's remote address and remote port set to that of the client.
- The constructor for ServerSocket throws an exception if it can't listen on the specified port (for example, the port is already being used).
- The server can communicate with the client over this new Socket and continue to listen for client connection requests on the original ServerSocket.
- The server can service simultaneously several clients through the use of threads - one thread per each client connection. The basic flow of logic can be described as follows:

```
while (true) {
  accept a connection ;
  create a thread to deal with the client ;
           }
```

  The thread reads from and writes to the client connection as necessary.

All these tasks are performed by the following code:

```
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) {System.out.println("Connection
      error: "+ex);}
   }
}
```

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.

```
class ConnectionHandler extends Thread {
private Socket client;
BufferedReader reader;
PrintWriter writer;
static int count=0;
...
String date[][] = new String[0][0];
  String col[] =
   {"Object Name", "Parents", "Attributes"};
  DefaultTableModel model =
   new DefaultTableModel(date,col);
   JTable kb = new JTable(model);
...
data structures for knowledge bases
...
String lista_KB="Hobbies Shopping";
public ConnectionHandler(Socket client) {
this.client = client;
System.out.println("Got connection from "+
    client.getInetAddress()+":"+client.getPort());
count++;
System.out.println("Active Connections = "+ count);
       } //end constructor
```

We relieve the following remarks:

- The variable *count* contains at every time the number of the active connections to the server.
- The server contains a number m of knowledge bases. The data structure $lista\_KB$ is a string of the form `"KB_1 KB_2 ....KB_m"` including the names of these knowledge bases such that two consecutive names are separated by a blank character.
- Each knowledge base is represented by means of a specific data structure. For example, the extended inheritance knowledge base Hobbies used in our application is defined as follows:

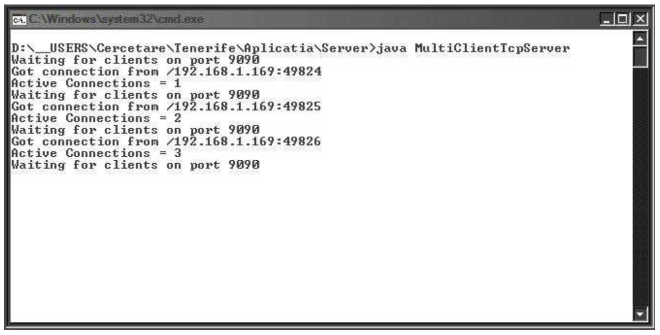
```
 private Object[][] data2 = {
   {"Alin", "","attr(likes,to_drink_coffee,60)
                attr(speaks,English,30)
                attr(main_hobby,fishing,30)"},
   {"George", "Alin",
               "attr(likes,to_drink_tea,10)"},
   {"Susan","George David",
               "attr(speaks,English,20)
                attr(main_hobby,cooking,40)"},
   {"David","","attr(likes,to_drink_juice,15)
                attr(plays,tennis,30)
                attr(main_hobby,hunting,75)"},
   {"Melvin","Susan","attr(drives,Logan,25)
                attr(speaks,French,10)
                attr(main_hobby,gardening,80)"}
```

  We used also the knowledge base Shopping defined as follows:

```
 private Object[][] data3 = {
   {"shop1", "shop2 shop3","attr(water,3,2)
               attr(pencil,1,7)
               attr(cake,8,14)"},
   {"shop2", "shop4","attr(juice,5,15)
               attr(pizza,8,21)"},
   {"shop3","shop4 shop5","attr(cheese,40,10)
               attr(strawberry,6,20)"},
   {"shop4","","attr(bread,20,10)
                attr(wine,50,30)"},
   {"shop5","shop6","attr(bread,30,20)
               attr(juice,4,3)"},
   {"shop6","","attr(wine,35,40)
                attr(water,4,1)"}
                                    };
```

- In order to manipulate the extended inheritance knowledge bases we used in Java the *TableModel* interface which specifies the methods the JTable will use to interrogate a tabular data model. The class DefaultTableModel is an implementation of TableModel that uses a Vector of Vectors to store the cell value objects. As well as copying the data from an application into the DefaultTableModel, it is also possible to wrap the data in the methods of the TableModel interface so that the data can be passed to the JTable directly.
- The BufferedReader object *reader* is used to receive data from client and the PrintWriter object *writer* is used to send data to client.
- As we mentioned above the server accepts several clients. In Figure 4 we exemplified a connection of three clients.

Step 2.2: inference engine for extended inheritance knowledge bases; this part implements the module CalculateAnswer presented in Figure 2.



```
C:\Windows\system32\cmd.exe
D:\__USERS\Cercetare\Tenerife\Aplicatia\Server>java MultiClientTcpServer
Waiting for clients on port 9090
Got connection from /192.168.1.169:49824
Active Connections = 1
Waiting for clients on port 9090
Got connection from /192.168.1.169:49825
Active Connections = 2
Waiting for clients on port 9090
Got connection from /192.168.1.169:49826
Active Connections = 3
Waiting for clients on port 9090
```

Fig. 4. Connection of three clients

```
//....
public String calculateAnswer(...) {
...
} // end method
```

Step 2.3: first communication.

```
public void run() {   //begin 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();
```

The first communication includes the message ”Welcome to my server!”; a list of knowledge base names is sent to client.

Step 2.4: invitation for client to choose a name of knowledge base and data initiation of this structure.

```
while(change_kb){   //while 5---------------
rasp=true;
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("Hobbies"))
{nr_lines=data2.length;writer.println(nr_lines);
writer.flush();
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]}); };
                      }; //end if
if(name_kb_client.equals("Shopping"))
{nr_lines=data2.length;writer.println(nr_lines);
writer.flush();
for(int i=0; i<nr_lines; i++){
    writer.println(data3[i][0]);
    writer.flush();
    model.insertRow(i, new Object[]
    {data3[i][0], data3[i][1], data3[i][2]}); };
```

```
                              };  //end if
```

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]}); };
  };  //end if
  ```
  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.

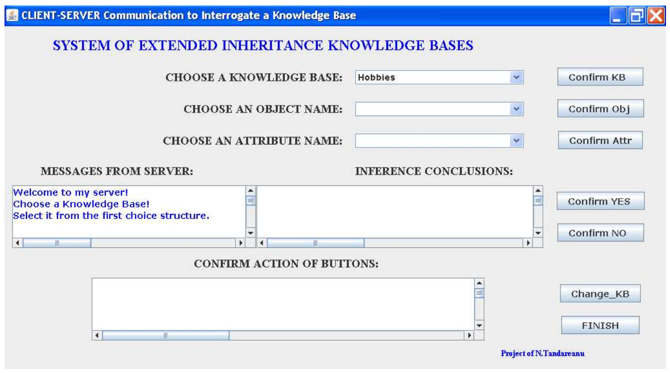
```
public void compute_all_attributes_of_kb(){
//begin "Extraction module"
String Val21="";
int jjj=0;
int ix=0;
String attr5="";
boolean difera=true;
boolean prima_data=true;
for (int idx=0;idx<nr_lines; idx++) { //for200
  ix=0; difera=true;
  Val21=(String)kb.getValueAt(idx,2);
if (!Val21.equals("")) {   //if100
 int tokenCount;
 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++) {  //for300
   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 {  //else400
 for(int j7=0;j7<jjj;j7++){
  if(all_attributes[j7].equals(attr5))
      difera=false;}; //end for
 if(difera) {all_attributes[jjj]=attr5;
           jjj++;}; //end if
           }  //end else400
           }  //end for300
           }  //end if100
           }  //end for200
writer.println(jjj);
writer.flush();
for (int idx=0;idx<jjj; idx++) {
   writer.println(all_attributes[idx]);
   writer.flush();}
   } // end  "Extraction module"
```

Step 2.6: Final dialog with client.

```
while(rasp){      //while 7
val="";
writer.println("Select the object name!");
writer.flush();
name_obj = reader.readLine();
writer.println("Select the attribute name!");
writer.flush();
name_attr = reader.readLine();
calculateAnswer(name_obj,name_attr);
Communication(name_obj,name_attr,val);
message=reader.readLine();
if(message.equals("yes")) rasp=true;
else rasp=false;
                   }  //end while 7
writer.println("Change KB? Change_KB/FINISH buttons");
writer.flush();
String message4;
message4=reader.readLine();
if(message4.equals("yes")) change_kb=true;
                   }      //end while 5
client.close();
count--;
System.out.println("Active Connections = " + count);
} catch (Exception ex) {
count--;
System.out.println("Active Connections = " + count);}
    }  //end run Step 2.3
 }  // end ConnectionHandler
```

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.
- The method *Communication* implements ”Module for answer sentences”:
  ```
  public void Communication(String name_obj1,
             String name_attr1,String value1){
    if(!(val.equals(""))){  //first if
    if(name_kb_client.equals("Shopping"))
      {nr_answers++;writer.println(nr_answers+
      ") "+"You can buy "+name_attr1+
      " from "+name_obj1+" with "+value1+" Euro.");
          writer.flush();
          writer.println(vec_fcs[index_fcs]);};
         // send the certainty factor
    if(name_kb_client.equals("Hobbies")){
    if(name_attr1.equals("main_hobby"))
      {nr_answers++;writer.println(nr_answers+") "
      +"The main hobby of "+name_obj1+
      " is "+value1+".");
      writer.flush();
      writer.println(vec_fcs[index_fcs]); }
    if(name_attr1.equals("likes"))
      {nr_answers++;writer.println(nr_answers+") "
      +name_obj1+" likes "+value1+".");
      writer.flush();
      writer.println(vec_fcs[index_fcs]); }
    if(name_attr1.equals("speaks"))
      {nr_answers++;writer.println(nr_answers+") "
  ```



CHOOSE A KNOWLEDGE BASE: Hobbies

```
Welcome to my server!
Choose a Knowledge Base!
Select it from the first choice structure.
```

Fig. 5.   Interface of the client

```
        +name_obj1+" speaks "+value1+".");
        writer.flush();
        writer.println(vec_fcs[index_fcs]); }
      if(name_attr1.equals("drives"))
        {nr_answers++;writer.println(nr_answers+") "
        +name_obj1+" drives a "+value1+".");
        writer.flush();
        writer.println(vec_fcs[index_fcs]); }
                          };
        writer.flush();
                          }  //end first if
      if(val.equals("")){nr_answers++;  //second if
        writer.println(nr_answers+") "+
        "The value of attribute "+name_attr1+
        " for "+name_obj1+" is unknown.");
        writer.flush();
        writer.println("unknown");
        writer.flush();
                          }  //end second if
    }
```

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 three windows, three choice option structures and seven buttons. This is shown in Figure 5.

Step 1: use JFrame class, windows, choice structures and buttons to build the graphical user interface of the user.

```
public class C_S extends JFrame implements
    ActionListener, ItemListener{
 boolean sem_cont=false;
 boolean sem_ncont=false;
 boolean sem_kb;
 boolean sem_obj=false;
 boolean sem_attr=false;
 boolean sem_finish=false;
 boolean sem_another=false;
 JTextArea _resultArea;
 Font font = new Font("Verdana", Font.BOLD, 14);
 JTextArea _resultArea1;
 JTextArea _resultArea2;
 Insets insets = new Insets(0, 0, 0, 0);
 Choice optiune1=new Choice();
 Choice optiune2=new Choice();
 Choice optiune3=new Choice();
 JButton ckb=new JButton("Confirm KB  ");
 JButton cob=new JButton("Confirm Obj ");
 JButton cont=new JButton("Confirm YES ");
 JButton ncont=new JButton("Confirm NO  ");
 JButton another_kb=new JButton("Change_KB");
 JButton finish=new JButton("   FINISH   ");
 JButton cat=new JButton("Confirm Attr");
 ...
//restrictions.....
 ......
```

Step 2: establish a connection with server and define the output and input channel (writer1/reader1) to communicate with the server.

```
public static  void main(String args[])
  throws UnknownHostException,IOException{
  C_S console = new C_S();
  console.setVisible(true);
            } //end main
public C_S(){
  super("CLIENT-SERVER Communication to
    Interrogate a Knowledge Base");
  init();
  comm_Client_Server();
    }
public void comm_Client_Server(){
  int port = 9090;
  try {
    String host = "192.168.1.169";
    Socket client = new Socket(host,port);
    PrintWriter writer1 = new PrintWriter
               (client.getOutputStream());
    BufferedReader reader1 = new BufferedReader
    (new InputStreamReader(client.getInputStream()));
```

Step 3: read the first message from server and the list of knowledge bases.

```
//receive the message "Welcome to my Server!"
//display this message
write10(reader1.readLine());
  boolean sem_change_kb=true;
  String message;
//obtain the list of knowledge bases
  message=reader1.readLine();
//introduce them into first choice structure
  StringTokenizer st=new StringTokenizer(message);
  int tokenCount=st.countTokens();
  while(st.hasMoreTokens()) {
           optiune1.addItem(st.nextToken());}
```

Step 4: choose a knowledge base, receive the objects and attributes from server and introduce them into the corresponding Choice structures of GUI

```
while(sem_change_kb){// while 5-----------
// read the message "Choose a Knowledge Base! "
//display this message
write10(reader1.readLine());
// read the message "Select it from the first
//choice structure."
//display this message
write10(reader1.readLine());
//choose kb by button--------
boolean raspuns=true;
boolean done;
boolean semafor1= false;
sem_kb=false;
this.setVisible(true);
while(!semafor1) {
    if (sem_kb) {writer1.println(ales_kb);
    writer1.flush();semafor1=true;}
               }
//---------------------------
semafor1=false;
//the knowledge base contains nr_elem obiects-----
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();
nr_elem=Integer.valueOf(reader1.readLine());
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 them to server

```
while(raspuns){        //while 1
//read message "Select the object name!"------
  message=reader1.readLine();
//display this message
  write10(message);
//select the object--------------------------
while(!semafor1) {
    if (sem_obj) {writer1.println(ales_obj);
        writer1.flush();semafor1=true;}
               }
semafor1=false; sem_obj=false;
//read message "Select the attribute name!"--
  message=reader1.readLine();
  //display this message
  write10(message);
//select the attribute------------------------
while(!semafor1) {              //while
    if (sem_attr) {writer1.println(ales_attr);
        writer1.flush();semafor1=true;}
               }
semafor1=false; sem_attr=false;
```

Step 6: display the answer given by server

```
//receive the server answer and display
//this message at INFERENCE CONCLUSIONS
write20(reader1.readLine());
//receive the certainty factor and display
//this value
write20("(certainty factor="+
         reader1.readLine()+")");
```

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

```
write30("Another interrogation? (yes/no):
    click on Confirm YES or Confirm NO");");
semafor1=false;
sem_cont=false;
sem_ncont=false;
while(!semafor1) {              //while
  if (sem_cont) {
      write30("You continue with
              the same knowledge base.");
      writer1.println("yes");
      writer1.flush();
      semafor1=true;raspuns=true;}
  if (sem_ncont) {
      write30("You does not continue with
              the same knowledge base.");
      writer1.println("no");writer1.flush();
      semafor1=true;raspuns=false;}             }
semafor1=false; sem_cont=false;
              }        //end while 1
//read message "Change KB? Change_KB/FINISH"
//display this message to client
write30(reader1.readLine());
while(!semafor1) {
if(sem_another){
          write30("Send yes to server");
          writer1.println("yes");writer1.flush();
          sem_change_kb=true;semafor1=true;}
                 }
semafor1=false;sem_another=false;
```

```
     }  // end while 5
client.close();
}catch (UnknownHostException e) {
System.err.println("Server is not ready "+e);}
    catch (IOException e) {System.err.println(e);}
}  //end comm_Client_Server
```

Remark 5.1: We used above the following methods:

```
public  void write10(String s){
    _resultArea.append(s+"\n");
}
public  void write20(String s){
    _resultArea1.append(s+"\n");
}
public void write30(String s){
    _resultArea2.append(s+"\n");
}
```

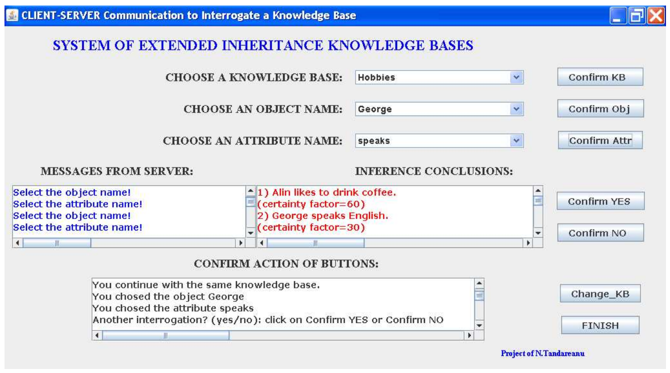
Remark 5.2: The *init* method includes the definition of the components for the graphical user interface.

```
public void init(){
// set dimension
Dimension ds =
  Toolkit.getDefaultToolkit().getScreenSize();
int a1= (int) ((double)ds.width * (8.0/10.0));
int b1=(int) ((double)ds.height * (8.0/10.0));
setSize(a1,b1);
setBackground(Color.gray.brighter());
this.setLayout(new GridBagLayout());
_resultArea = new JTextArea(50, 80);
_resultArea1 = new JTextArea(50, 80);
_resultArea2 = new JTextArea(50, 120);
_resultArea.setFont(font);
_resultArea1.setFont(font);
_resultArea2.setFont(font);
_resultArea.setForeground(Color.BLUE);
_resultArea1.setForeground(Color.RED);
JScrollPane scrollingArea =
  new JScrollPane(_resultArea);
JScrollPane scrollingArea1 =
  new JScrollPane(_resultArea1);
JScrollPane scrollingArea2 =
  new JScrollPane(_resultArea2);
Panel my_Panel10=new Panel();
my_Panel10.setLayout(new GridLayout(1,1));
my_Panel10.add(scrollingArea, BorderLayout.CENTER);
Panel my_Panel20=new Panel();
my_Panel20.setLayout(new GridLayout(1,1));
my_Panel20.add(scrollingArea1, BorderLayout.CENTER);
Panel my_Panel30=new Panel();
my_Panel30.setLayout(new GridLayout(1,1));
my_Panel30.add(scrollingArea2, BorderLayout.CENTER);
optiune1.setFont(font);
optiune2.setFont(font);
optiune3.setFont(font);
...
           }   //end init
```

Remark 5.3: The action of the buttons are described by the method *actionPerformed*.

```
public void actionPerformed(ActionEvent evt){
 if(evt.getSource()==ckb){
 ales_kb=optiune1.getSelectedItem();
 write30("You chosed the knowledge base "+ales_kb);
 sem_kb=true;                               }
if(evt.getSource()==cob){
 ales_obj=optiune2.getSelectedItem();
 write30("You chosed the object "+ales_obj);
 sem_obj=true;
...                                  }
```

In figures 6 and 7 we illustrate two steps of the same session of the client execution: two queries for the knowledge base Hobbies and one query for the knowledge base Shopping.



CHOOSE A KNOWLEDGE BASE: Hobbies
CHOOSE AN OBJECT NAME: George
CHOOSE AN ATTRIBUTE NAME: speaks

MESSAGES FROM SERVER:

```
Select the object name!
Select the attribute name!
Select the object name!
Select the attribute name!
```

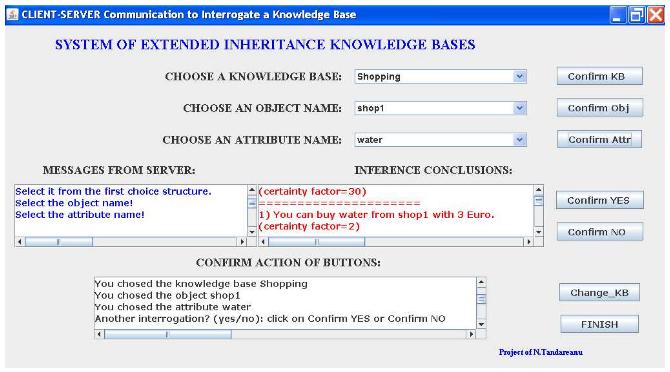
INFERENCE CONCLUSIONS:

```
1) Alin likes to drink coffee.
(certainty factor=60)
2) George speaks English.
(certainty factor=30)
```

CONFIRM ACTION OF BUTTONS:

```
You continue with the same knowledge base.
You chosed the object George
You chosed the attribute speaks
Another interrogation? (yes/no): click on Confirm YES or Confirm NO
```

Fig. 6. Example of interrogation



CHOOSE A KNOWLEDGE BASE: Shopping
CHOOSE AN OBJECT NAME: shop1
CHOOSE AN ATTRIBUTE NAME: water

MESSAGES FROM SERVER:

```
Select it from the first choice structure.
Select the object name!
Select the attribute name!
```

INFERENCE CONCLUSIONS:

```
(certainty factor=30)
=====================
1) You can buy water from shop1 with 3 Euro.
(certainty factor=2)
```

CONFIRM ACTION OF BUTTONS:

```
You chosed the knowledge base Shopping
You chosed the object shop1
You chosed the attribute water
Another interrogation? (yes/no): click on Confirm YES or Confirm NO
```

Fig. 7. Changing the knowledge base

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 three windows: the first window is used to display the messages of the server, the second shows the conclusions of the inference engine and the third 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. Various structures of knowledge bases can be considered:
  - object oriented knowledge bases (the inference mechanism is based on inheritance, [15], [18]);
  - knowledge bases with output, represented by means of labeled stratified graphs ([14], [16]);
  - knowledge bases represented by means of semantic schemas ([17], [20]);
  - conditional knowledge bases ([3], [21], [24]);
  - knowledge bases represented by conceptual graphs ([2], [5], [12])

  and other kind 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 ([6]).
- Imply the use of mobile agents to satisfy a query ([19]).
- Try to introduce the communication by voice in a natural language ([4], [11]).
- Our approach presented in this paper considers a very simple query, a pair (object,attribute) of two entities. A subject of general interest is connected by the use of complex queries. An interesting approach of this problem can be obtained from [7]. The query is processed and translated in one or more queries that are submitted to one or more sources. The results are then merged together and reinterpreted to obtain an unique answer.

References

[1] **N.R.Adam, R.H.Holowczak, J-H.Maeng**:- ORACLE SQL*Plus and SQL*Forms: An Introduction and Tutorial, Technical Working Paper 89, CRAMTD Project, Rutgers University, September, 1994
[2] **Ganter Bernhard, Guy W. Mineau, eds.**:- Conceptual Structures: Logical, Linguistic, and Computational Issues, Lecture Notes in AI 1867, Springer-Verlag, Berlin, 2000
[3] **M.Colhon, N.Ţăndăreanu**:- The Inference Mechanism in Conditional Schemas, Annals of the University of Craiova, Mathematics and Computer Science Series, Vol.37, No.1, p.55-57, 2010
[4] **Daniel Jurafsky, James H. Martin**:- Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2nd Edition, Pearson Education, 2006.
[5] **Ellis Gerard, Robert A. Levinson, William Rich, John F. Sowa, eds.**:- Conceptual Structures: Applications, Implementation, and Theory, Lecture Notes in AI 954, Springer-Verlag, Berlin, 1995
[6] **Faraj A. El-Mouadib, Zakaria S. Zubi, Ahmed A. Almagrous, Irdess S. El-Feghi**:- Generic Interactive Natural Language Interface to Databases (GINLIDB), International Journal of Computers, Issue 3, Volume 3, 301-309, 2009

[7] **Mario Arrigoni Neri**:- Knowledge integration through semantic query rewriting, Proceedings of the 9th WSEAS International Conference on APPLIED COMPUTER SCIENCE, ISBN: 978-960-474-127-4, 229-234, 2009

[8] **Steve Knoblock**:- Using MyODBC to Connect to a Remote Database, http://www.devarticles.com/cp/bio/Steve-Knoblock/

[9] **Hugo Toledo, Jonathan Gennick**:- Oracle Net8 Configuration and Troubleshooting, O'Reilly Media, 2000

[10] **Elliotte Rusty Harold**:- Java Network Programming, Third Edition, O'Reilly Media, 2004

[11] **M. Selvam, A.M. Natarajan**:- Improvement of Rule Based Morphological Analysis and POS Tagging in Tamil Language via Projection and Induction Techniques, International Journal of Computers, Issue 4, Volume 3, 357-367, 2009

[12] **J.F.Sowa**:- Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, 2000.

[13] **Dick Steflik, Prashant Sridharan, Richard Steflik**:- Advanced Java Networking, Prentice Hall, 2000

[14] **N.Ţăndăreanu**:- Knowledge Bases with Output, Knowledge and Information Systems, 2(4), 438-460, 2000

[15] **N.Ţăndăreanu**:- Inheritance-based knowledge systems and their answer functions computation using lattice theory, Romanian Journal of Information Science and Technology, Vol.6, Numbers 1-2, 227-248, 2003

[16] **N.Ţăndăreanu**:- Knowledge Representation by Labeled Stratified Graphs, The 8th World Multi-Conference on Systemics, Cybernetics and Informatics, July 18-21, 2004, Orlando, Florida, USA, Vol. V: Computer Science and Engineering, p.345-350, 2004

[17] **N.Ţăndăreanu**:- Semantic Schemas and Applications in Logical representation of Knowledge, Proceedings of the 10th International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA2004), July 21-25, Orlando, Florida, USA, Vol.III, 82-87, 2004

[18] **Claudiu Popirlan, N.Ţăndăreanu**:- An Extension of Inheritance Knowledge Bases and Computational Properties of their Answer Functions, Annals of the University of Craiova, Vol. 35, 149-170, 2008

[19] **Claudiu Ionut Popirlan**:- A Multi-Agent Approach for Distributed Knowledge Processing in Contact Centers, 14th WSEAS International Conference on Computers, Corfu Island, Greece, July 23-25, ISBN 978-960-474-201-1, 214-219, 2010

[20] **N.Ţăndăreanu**:- Transfer of knowledge by semantic schemas, The 11th IASTED International Conference on Intelligent Systems and Control (ISC 2008), Ed.K.Grigoriadis, ISBN 978-0-88986-777-2, 2008

[21] **N.Ţăndăreanu, M.Colhon**:- Conditional graphs generated by conditional schemas, Annals of the University of Craiova, Mathematics and Computer Science Series, Vol.36, No.1, p.1-11, 2009

[22] **N.Ţăndăreanu, Claudiu-Ionut Popirlan**:- Factorization of an inheritance knowledge base (I), Annals of the University of Craiova - Mathematics and Computer Science Series, Vol 37, No 2, 62-74, 2010

[23] **N.Ţăndăreanu, Claudiu-Ionut Popirlan**:- Factorization of an Inheritance Knowledge Base (II), Annals of the University of Craiova, Mathematics and Computer Science Series, Vol.37, No.4, 1-8 (2010).

[24] **N.Ţăndăreanu, Mihaela Colhon, Cristina Zamfir**:- Embedding Conditional Knowledge Bases into Question Answering Systems and Java Implementation, International Journal of Computers, Issue 4, Volume 4, 169-176, 2010

[25] *** :- Web Service - Java DataBase Connectivity, http://ws-jdbc.sourceforge.net/WS-JDBC_wp.pdf

[26] **G. Wagner**:- Vivid Logic: Knowledge-Based Reasoning with Two Kinds of Negation, Knowledge-Based Reasoning with Two Kinds of Negation, Lecture Notes in Artificial Intelligence 764, Springer-Verlag, 1994