Annals of the University of Craiova, Mathematics and Computer Science Series
Volume 37(3), 2010, Pages 1–11
ISSN: 1223-6934

Extended Inheritance Knowledge Bases and their Interrogations by Client-Server Technology

Nicolae Ţăndăreanu

Abstract. In this paper we describe an implementation of the interrogation process for an extended inheritance knowledge base. The implementation uses the client-server technology offered by Java platform. Both the server side and the client side of the application are described. The application allows to use one server and several clients and was tested in a private wireless network. The processing steps and the results given by the server are presented.

2010 Mathematics Subject Classification. Primary 68T30; Secondary 68N19.
Key words and phrases. directed ordered graph, tree, ordered tree, partial order, embedding mapping.

1. Introduction

Various theoretical and practical aspects of the concept of inheritance were studied. Several results concerning the mixed inheritance, relationship between inheritance and knowledge representation, inheritance networks, inheritance in object oriented programming languages, partial equivalence relations, inheritance mechanisms in the OBJLOG language are treated in [1]. In [4] an inheritance system for knowledge bases in which IS-A relation and IS-NOT-A relation is proposed. A formal model of knowledge representation for uncertain and vague domains is described in [3]. The model is based on a fuzzy inheritance procedure that uses the dynamical properties of a Fuzzy Petri Net. The lattice theory was applied to characterize the properties of the answer mapping for an inheritance knowledge based systems ([6]). The use of the voice user interface to interrogate an inheritance knowledge based systems is described in [5].

In this paper we consider the interrogation process for an extended inheritance knowledge base and its implementation using the client-server technology based on Java platform. The paper is organized as follows: in Section 2 we present the main concepts connected by the concept of extended inheritance knowledge base; in Section 3 we explain the structure of the application, both the server side and the client side; in Section 4 we present several details concerning the results given by our application.

2. Extended inheritance knowledge bases

The concept of extended inheritance knowledge base was introduced in [2] and the mathematical properties of this structure were studied in [7] and [8]. A brief description of the main results is given in this section. The model proposed in these papers has the following features:

Received June 17, 2010. Revision received August 09, 2010.

(1) The model allows to use multiple inheritance for the computation of an attribute value. More precisely, the value of a given attribute can be inherited from more that one object. In this case a choice function is introduced to define some strategy and this function selects only one attribute from the set of all inherited attributes. Moreover, this function can specify a new attribute value which is obtained by means of the selected values. In this manner the value of an attribute can be the inherited value, but can be a modified value also.
(2) The value of an attribute can specify some parameter that can represent the uncertain knowledge (fuzzy and probabilistic) or a risk factor.
(3) An object can specify several values for a given attribute. In this case an object can inherit some of them or can obtain a new value by means of these values.

An object is characterized by:

- The *name* of the object; it identifies uniquely the object.
- A finite set of symbolic names. Each of them designates another object that is named a *parent* of the object. A given object may contain zero or more parents.
- A finite set of *slots*; a slot is an ordered pair of the form $(attribute, value)$, where *attribute* is the symbolic name of some feature and the *value* is its corresponding value.

The objects are virtually linked by the relation parent-child. A child has proper features and can *inherit* other features from its parents. Each feature of an object is given by an *attribute*. An attribute can specify some immediate value (for example the temperature value, the height, the length, the color, the weight etc) or can identify the name of a method that computes the value of the attribute. Moreover, we suppose that some parameter is assigned to each value of an attribute. This parameter establishes the uncertainty of the value, a risk coefficient or a cost value.

The value V of an attribute A of the object F is computed as follows:

(1) Suppose that the description of F contains at least one slot of the form (A, V, p) or (A, P, p), where V is an immediate value, P is the name of a procedure and p is a parameter associated to (A, V). We denote by $Slot(F)$ the set of all these slots. Intuitively we suppose that we have a choice strategy given by a mapping $Choice$ such that $Choice(Slot(F)) = (A, T, p)$ is the selected slot. If $T = V$ then the value of A is V and p is the parameter associated to (A, V). If $T = P$ then the value V is returned by the procedure P for some values of its arguments and p is the parameter associated to this value.
(2) Suppose that the description of F does not contain any slot such that its first component is A. In this case the attribute value is obtained by means of some parent of F. In other words the corresponding attribute is *inherited* from its parents. This situation is iterated and this means that if each of these parents does not contain the corresponding attribute then we search it for the parents of the parents and so on.

In order to exemplify the concepts we represent an object as a system of three entities as follows:

$$(n_1, \{q_1, \ldots, q_s\}, \{(a_1, v_1, p_1), \ldots, (a_k, v_k, p_k)\})$$

where

- n_1 gives the object name;
- $\{q_1, \ldots, q_s\}$ defines the set of all parents of n_1;
- $\{(a_1, v_1, p_1), \ldots, (a_k, v_k, p_k)\}$ represents the set of attributes for n_1, their values and the values of the corresponding parameters.

Let us consider the following objects:

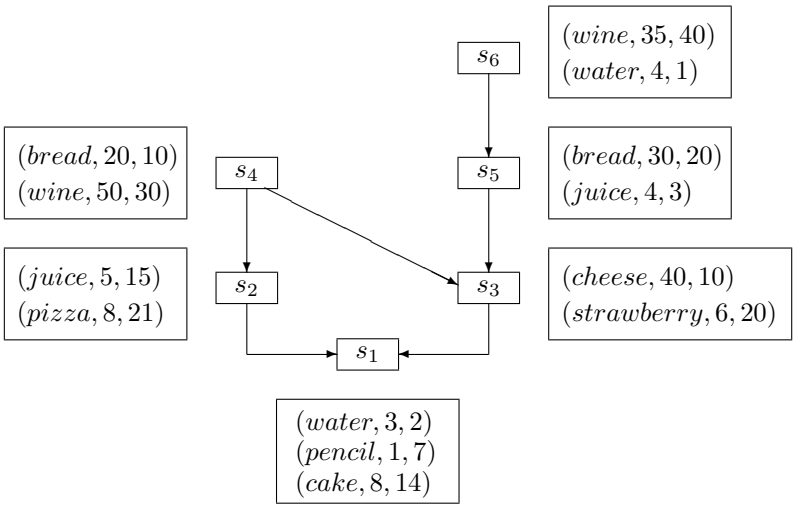

FIGURE 1. An example of knowledge base

$(s_1, \{s_2, s_3\}, \{(water, 3, 2), (pencil, 1, 7), (cake, 8, 14)\})$
$(s_2, \{s_4\}, \{(pineapple, 5, 15), (pizza, 8, 21)\})$
$(s_3, \{s_5\}, \{(cheese, 40, 10), (strawberry, 6, 20)\})$
$(s_4, \{\}, \{(bread, 20, 10), (wine, 50, 30)\})$
$(s_5, \{s_6\}, \{(bread, 30, 20), (juice, 4, 3)\})$
$(s_6, \{\}, \{(wine, 35, 40), (water, 4, 1)\})$

The set $\{s_1, s_2, s_3, s_4, s_5, s_6\}$ represents an extended inheritance knowledge base. The relation parent-child allows to represent this knowledge base as in Figure 1. Let us choose the maximal value strategy for parameters and we denote by $V(s, attr)$ the value of the attribute *attr* for the object s then we obtain the following examples of values:

- $V(s_1, bread) = 30$ because the set of the nearest predecessors of s_1 containing the attribute *bread* is the set $\{s_4, s_5\}$ and the maximal parameter is 20;
- $V(s_1, wine) = 50$ because s_4 is the nearest predecessor of s_1 which contains the attribute *wine*.

Remark 2.1. *The example considered in Figure 1 does not contain the case of values computed by a procedure. The reader can find details for this case in* [2], [7] *and* [8]. *The application presented in this paper includes an implementation of a Java method to compute the general value of an attribute and we emphasize only the main aspects of this method. The focus of this paper is to present an implementation in client-server technology.*

3. Interrogation by client-server technology

3.1. Interrogation. An inheritance knowledge base is a finite sequence L of entities of the form

$$(n_1, \{q_1, \ldots, q_s\}, \{(a_1, v_1, p_1), \ldots, (a_k, v_k, p_k)\})$$

as we specified in the previous section. An interrogation is defined by a pair (n_r, a_r), where n_r is an object name of L and a_r is an attribute name.

In what follows we consider that the value p_i is the certainty factor of the value v_i of the attribute a_i. The answer of the interrogation $(obj, attr)$ is a sentence of the form *The value of the attribute attr for the object obj is* v_s *with fc* p_r or *The value of the attribute attr for the object obj is unknown* (fc denotes certainty factor). We denote by $Ans(obj, attr)$ such a sentence. We exemplify the answer for the example presented in the previous section:

- $Ans(shop1, bread)$=*The value of the attribute bread for the object shop*1 *is* 30 *with certainty factor* 20.
- $Ans(shop3, wine)$=*The value of the attribute wine for the object shop*3 *is* 50 *with certainty factor* 30.
- $Ans(shop2, water)$=*The value of the attribute water for the object shop*2 *is unknown.*

We implemented our application in Java. A knowledge base is defined as a Java structure of the following form:

```
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)"}
                              };
```

Each line defines an object. For example, the first line represents the object *shop*1 that contains:

- the parents *shop*2 and *shop*3;
- the attributes *water*, *pencil* and *cake* and the corresponding values 3, 1 and 8 of these attributes; the third argument defines the parameter of the attribute value.

3.2. Client-server programming. A client-server application uses a server and one or more clients. In Figure 2 we represented such a case based on a network communication. The server provides some service, such as storing databases and processing database queries or can offer information to user by means of a dialogue system. The client uses the service provided by the server. It can display the database query results performed by the server or display the information obtained by server processing. The communication that occurs between the client and the server must ensure an ordered transfer. That is, the data must arrive on the client side in the same order in which the server sent it.

The mechanism for communication between two computers is provided by means of *sockets*. Particularly the sockets provide the communication between a client and a server. A socket can be viewed as a software unit used to represent the terminal of a connection between two computers. For a given connection, there's a socket on each computer. A socket on one computer can talk to a socket on another computer and thus a communication channel is obtained. A programmer can use that channel to send data between the two computers.

Our application is implemented in Java. The Java platform provides implementations of sockets in the *java.net* package. The *java.io* package gives the tools to

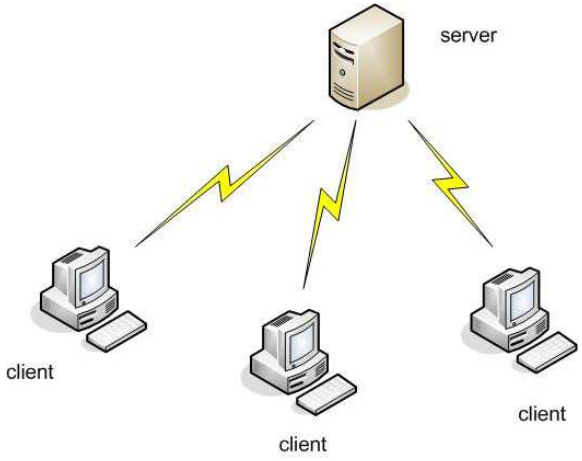

FIGURE 2. Connections client-server

read and write streams, which is the only way the user can communicate with TCP sockets. A client-server application includes two parts: the server part and the client part. The application is the pair of these two parts. In what follows we present each component of this pair.

3.3. The server side. The general structure of this part is described as follows:

```
import java.net.*;
import java.io.*;
import java.util.*;
import javax.swing.table.AbstractTableModel;
import javax.swing.JTable;
import javax.swing.table.*;

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 " + port);
Socket client = server.accept();
ConnectionHandler handler = new ConnectionHandler(client);
handler.start();}
} catch(Exception ex) {//
    System.out.println("Connection error: "+ex);}
      }
                                    }
//=========
class ConnectionHandler extends Thread {
private Socket client;
BufferedReader reader;
PrintWriter writer;
static int count;
//----------------------------
```

```
String date[][] = new String[0][0];
  String col[] = {"Object Name", "Parents", "Attributes"};
  DefaultTableModel model = new DefaultTableModel(date,col);
  JTable kb = new JTable(model);
//----------------------------
private Object[][] data1 = {
            {"Mary", "","attr(a,5,6) attr(b,7,2)"},
            {"Alison", "Mary","attr(a,8,3)"},
            {"Peter","Alison Mary Helen","attr(c,99,7)"},
            {"Helen","","attr(d,8,1) attr(b,3,10)"}
                                  };
 ----------------------------
String lista_KB="KB_1 KB_2 KB_3";
//----------------------------
public int choice_max(int[] t) {
-----------------------------
}   //end method max
//----------------------------
public String calculateAnswer(String n_obj, String n_attr) {
-----------------------------
     }  //end calculateAnswer
//----------------------------
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
public void run() {
String message=null;
        String name_obj=null;
        String name_attr=null;
        boolean rasp=true;
 boolean change_kb=true;
try {
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();
----------------------------
                  }  //end run
                                       }
```

We emphasize now the following details concerning the above implementation:

- The ServerSocket class is used to by server to accept client connections. ServerSocket objects use their *accept()* method to connect to a client.

- ConnectionHandler class uses its *getInputStream()* and *getOutputStream()* methods to provide streams for reading and writing to the client.
- The method *choice_max* compute the greatest value of the parameters if there are several predecessors that contribute to the computation of a attribute value.
- The entity *writer* can be viewed as a channel by means of which the server sends a message or data to client. In an opposite manner the entity *reader* is a channel by means of which the server receives a message from the client.

3.4. The client side. The skeleton of the client part can be described as follows:

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.Locale;
import java.io.IOException;
import java.security.Security;
import java.util.Properties;
import java.net.URL;
import javax.swing.*;

public class C_S extends ApplicationFrame //
     implements ActionListener, ItemListener
{
--------------------
public static  void main(String args[]) //
    throws UnknownHostException,IOException
    {
int port = 9090;

        C_S console = new C_S();
        console.setVisible(true);
//--------------------------------------
try {
String host = "192.XXX.X.XXX";
       Socket client = new Socket(host, port);
PrintWriter writer1 = //
  new PrintWriter(client.getOutputStream());
BufferedReader reader1 = new BufferedReader( //
    new InputStreamReader(client.getInputStream()));

BufferedReader stdin = new //
    BufferedReader(new InputStreamReader(System.in));
----------------------------------------
     }  //end try
 public C_S()
    {
 super("CLIENT-SERVER Communication to Interrogate //
    a Knowledge Base");
        init();
```

```
    }

    public void init()
    {
    --------------
    } //end init
public void actionPerformed(ActionEvent evt)
    {
    --------------
    } //end actionPerformed
}
```

In order to simplify the processing and for a better visualization of the processing steps the client application contains a graphical user interface. This interface includes:

(1) an window to specify the messages from the server; an welcome message from server is displayed after the first connection to server; the result of the interrogation is also displayed into this window;
(2) an window to specify auxiliary information concerning the processing steps on the client: the name of the knowledge base chosed by the user; the object name and the attribute name selected by the user; confirmation concerning the changing of the knowledge base;
(3) buttons to confirm:
    - the name of the knowledge base selected by the user;
    - the name of the object selected;
    - the name of the attribute selected.
(4) buttons to perform the following actions:
    - to specify another interrogation of the same knowledge base;
    - to specify the fact that the user intends to change the knowledge base;
    - to specify the changing of the knowledge base;
    - to finish the processing steps of the client.

4. Running the application

The server side is contained by the file $MultiClientTcpServer.java$ and the client side is included into the file $TcpClient.java$. In order to exemplify our application we used a server and two computers as clients (named client1 and client2), connected in a wireless network. We performed the following steps:

(1) We execute the file $exec - server.bat$ that contains the following command:

$$java\ MultiClientTcpServer$$

As a consequence the server is launched.
(2) On client1 and client2 we execute the file $exec - client.bat$ that contains the following command:

$$java\ TcpClient$$

On the sever side we obtain the image presented in Figure 3.

On the client side we obtained the images from Figure 4 and Figure 5 for client1 and client2 respectively.

We selected two distinct knowledge bases: for client1 we selected the knowledge base KB_3 (described in Section 3.1) and for client2 we selected the knowledge base KB_1 (described in Section 3.3 by the structure $data1$).

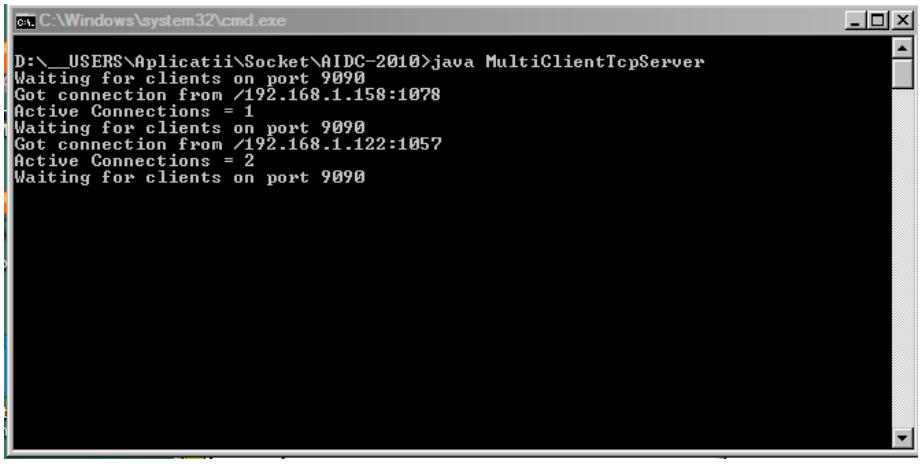

FIGURE 3. Two connections client-server

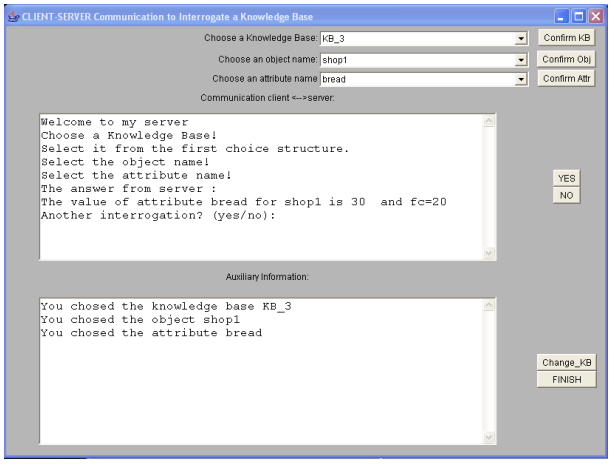

FIGURE 4. First step for client1

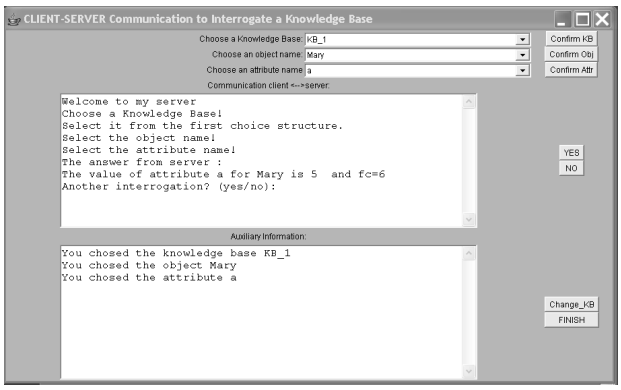

FIGURE 5. First step for client2

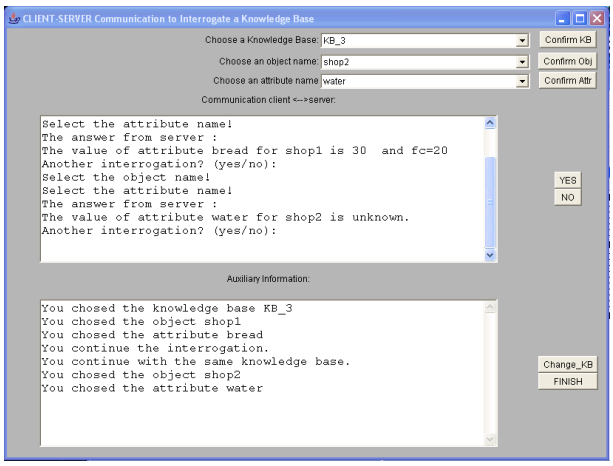

FIGURE 6. Second step for client1

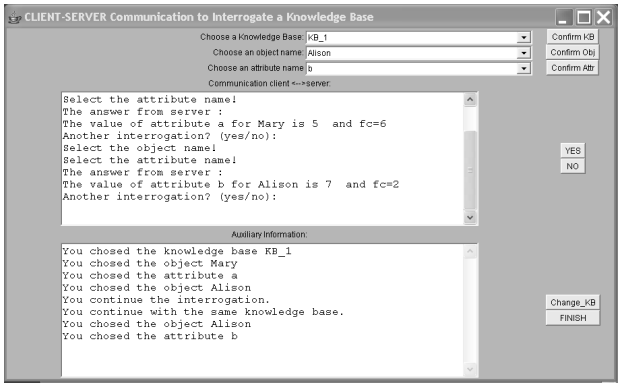

FIGURE 7. Second step for client2

If we continue the processing we obtain the images from Figure 6 and Figure 7.

We remark the following answers for KB_3:

- The value of attribute bread for shop1 is 30 and fc=20.
- The value of attribute water for shop2 is unknown.

and for KB_1 we obtained:

- The value of attribute a for Mary is 5 and fc=6.
- The value of attribute b for Alison is 7 and cf=2.

We observe that the value of the attribute b for Alison is inherited from the object Mary.

5. Conclusions

In this paper we considered the concept of extended inheritance knowledge base and and the process of interrogation of such a structure is implemented. The implementation uses the client-server technology offered by Java platform. The main steps of the application are described and exemplified by capture images. The application

was tested in a private wireless network using one server and two clients. The server contains several knowledge bases and each client can interrogate a distinct knowledge base. The same technology can be used to interrogate other kinds of knowledge bases, with a corresponding implementation of the answer mapping.

References

[1] M. Lenzerini, D. Nardi and M. Simi, *Inheritance Hierarchies in Knowledge Representation and Programming Languages*, John Wiley and Sons, 1991.

[2] C.-I. Popîrlan and N. Ţăndăreanu, An Extension of Inheritance Knowledge Bases and Computational Properties of their Answer Functions, *Annals of the University of Craiova, Mathematics and Computer Science Series* **35** (2008), 149–170.

[3] S. Ribaric and N. Pavesic, An Inheritance Procedure for a Knowledge Representation Scheme Based on Fuzzy Petri Nets, *Proceedings of the Third International Conference on Natural Computation* **1** (2007), 3–9.

[4] M. Tsukamoto and S. Nishio, Inheritance reasoning by regular sets in knowledge-bases with dot notation, Deductive and Object-Oriented Databases, *Lecture Notes in Computer Science* **1013** (1995), 247–264.

[5] N. Ţăndăreanu, Communication by Voice to Interrogate an Inheritance Based Knowledge System, *Research Notes in Artificial Intelligence and Digital Communications, 7th International Conference on Artificial Intelligence and Digital Communications* **107** (2007), 1–15.

[6] N. Ţăndăreanu, Inheritance-based knowledge systems and their answer functions computation using lattice theory, *Romanian Journal of Information Science and Technology* **6** (2003), no. 1-2, 227–248.

[7] N. Ţăndăreanu and C.-I. Popîrlan, Factorization of an inheritance knowledge base (I), *Annals of the University of Craiova, Mathematics and Computer Science Series* **37** (2010), no. 2, 62–74.

[8] N. Ţăndăreanu and C.-I. Popîrlan, Factorization of an Inheritance Knowledge Base (II), *Annals of the University of Craiova, Mathematics and Computer Science Series* **37** (2010), no. 4, (to appear).

[9] G. Yang and M. Kifer, Well-Founded Optimism: Inheritance in Frame-Based Knowledge Bases, *Lecture Notes in Computer Science, in On the Move to Meaningful Internet Systems* **2519** (2008), 1013–1032.

(Nicolae Ţăndăreanu) Faculty of Mathematics and Computer Science, University of Craiova, A.I. Cuza Street, No. 13, Craiova RO-200585, Romania
E-mail address: ntand@rdslink.ro