VoSyS: A System by Voice to Answer in Inheritance-Based Knowledge Systems

Nicolae ŢĂNDĂREANU

Faculty of Mathematics and Computer Science,
Department of Computer Science,
University of Craiova, Romania
ntand@oltenia.ro

Abstract. In a sequence of papers ([1], [2], [3]) some model to represent the knowledge by using the inheritance is presented and the computability of the answer mapping is studied. VoSyS is an implementation of this model and it is briefly described in this paper. To obtain this product we used a bidirectional connection Java-Prolog and a Java based speech technology. The capabilities of the graphical interfaces and interfaces by voice are combined to obtain a dialog user-system. VoSyS is able to receive an interrogation and to speak the corresponding answer obtained as a result of an inference in Prolog.
Math. Subject Classification 2000: 68T30, 68T10
Keywords: knowledge base, inheritance, speech technology

1 INTRODUCTION

The use of speech technology to accomplish interfaces by voice in the domain of knowledge bases is an attractive subject. In general this is an interesting problem for dialogue systems. A conversation user-system consists of parts of utterances: statements by the user, followed by replies from the system. More precisely, if we use knowledge bases then the statement of the user is an *interrogation* and the reply of the system is the *answer*. The answer is a result of some computations performed by using the entities of the knowledge base.

In this paper we consider the case of the inheritance-based knowledge systems. In such systems the knowledge is represented in a proper manner as objects and the processing uses an engine based on the inheritance mechanism. The following aspects are treated in this paper:

- Using several elements of lattice theory we studied in [3] several problems connected by the answer mapping in inheritance-based knowledge systems. The application presented in this paper uses an algorithm to compute the values of this mapping based on these results. This algorithm is concisely described in the last part of Section 2.
- We present the use of speech technology to implement the reply of the system to an interrogation in Section 3.

- The software presented in this paper can be applied to all knowledge pieces that can be modeled by the inheritance mechanism. In the modeling process it is possible to appear some particular constraints on the answer mapping. The adaptation of the computations to obtain the accurate answer is also discussed and a corresponding solution is given in Section 3.

2 INHERITANCE-BASED KNOWLEDGE REPRESENTATION

2.1 KNOWLEDGE BASES

The entities of the representation language L_{Repr} are of the form

$$\mathbf{frame}(name, name\_list, attribute\_list)$$

where *name* is the frame name; *name_list* is a list containing the name of the parents; *attribute_list* is a list whose elements are entities of the form $attr(attribute\_name, value)$, $attr(attribute\_name, proc(name))$ or $attr(attri–bute\_name, demon)$; the first case specifies the direct *value* of the attribute; in the second case, *name* is the name of some procedure which computes the corresponding value; the last form specifies that we have an abstract attribute and for some particular case its value is obtained by a procedure.

We consider the sets: V_{Attr} (values for attributes); L_{fr_name} (frame names); L_{Attr} (attribute names); L_{parent} (parent names); L_{proc} (procedure names); $Q_{Attr} = V_{Attr} \cup \{proc(x) \mid x \in L_{proc}\}$. A *knowledge base* is a finite subset of L_{Repr}.

2.2 THE ANSWER MAPPING

Let $\mathcal{K}$ be a knowledge base. We consider the graph $G_{\mathcal{K}} = (\mathcal{K}, \Gamma_{\mathcal{K}})$ where $(f_u, f_v) \in \Gamma_{\mathcal{K}}$ iff the name of f_u is a parent of f_v. The set of all the paths from x to y in $G_{\mathcal{K}}$ is denoted by $Path(x, y)$. An element $f \in \mathcal{K}$ is *a predecessor* of $g \in \mathcal{K}$ if there is a path from f to g. A predecessor f of g is *a nearest predecessor* having the property α if f satisfies α and does not exist a predecessor h of g such that h satisfies α and $dist(h, g) < dist(f, g)$, where $dist(x, y)$ is the length of the shortest path from x to y.

If $\mathbf{frame}(f, [p_1, \ldots, p_s], [\mathbf{attr}(a_1, v_1), \ldots, \mathbf{attr}(a_k, v_k)]) \in \mathcal{K}$ then we denote $Slot(f) = \{(a_1, v_1), \ldots, (a_k, v_k)\}$. $Pred(f)$ is the set of all the frame names which are predecessors of f. We say that the frame f *contains* the attribute name a if there is a slot $(a, u) \in Slot(f)$ for some $u \in Q_{Attr} \cup \{demon\}$. Equivalently we shall write $a \in f$.

For any attribute name $a \in L_{Attr}$ we define: $Near_a(f) = \{f\}$ if $a \in f$ and $Near_a(f) = \{g \in Pred(f) \mid \text{g is a nearest predecessor}, a \in g\}$ if $a \notin f$.

We denote by $Proc_{\mathcal{K}}$ the set of all elements $(f, a) \in L_{fr\_name} \times L_{Attr}$ for which the attribute a for f can be computed by means of a procedure. By $\Omega_{\mathcal{K}}(f, a)$ we denote the procedure name computing this attribute.

We denote by $M_{\mathcal{K}}$ the set of all the pairs (f,a) for which the attribute a can be computed for f but not by means of a procedure:

$$M_{\mathcal{K}} = \{(f,a) \mid (f,a) \notin Proc_{\mathcal{K}}, Near_a(f) \neq \emptyset\}$$

The mapping $Comp_{\mathcal{K}}$ which computes the value of an attribute for a given frame is recursively defined as follows:
• If $(f,a) \in M_{\mathcal{K}}$ then $Comp_{\mathcal{K}}(f,a) = u$, where $u \in V_{Attr} \cup \{demon\}$, $\{g\} = Near_a(f)$ and $(a,u) \in Slot(g)$
• Let be $(f,a) \in Proc_{\mathcal{K}}$ and $(b_1,\dots,b_t)$ the arguments of $(\Omega_{\mathcal{K}}(f,a))$. The value $Comp_{\mathcal{K}}$ is obtained as follows:
– If $Comp_{\mathcal{K}}(f,b_1),\dots,Comp_{\mathcal{K}}(f,b_t)$ can be computed and their values belong to $V_{Attr} \cup \{demon, unknown\}$ then

$$Comp_{\mathcal{K}}(f,a) = \Omega_{\mathcal{K}}(f,a)(Comp_{\mathcal{K}}(f,b_1),\dots,Comp_{\mathcal{K}}(f,b_t))$$

– Otherwise $Comp_{\mathcal{K}}(f,a) = undefined.$
• If $(f,a) \in L_{fr\_name} \times L_{Attr} \setminus (M_{\mathcal{K}} \cup Proc_{\mathcal{K}})$ then $Comp_{\mathcal{K}}(f,a) = unknown$

We consider the *query language* $L_{fr\_name} \times L_{Attr}$. The answer function is defined as follows: $Ans(\mathcal{K},(f,a)) = Comp_{\mathcal{K}}(f,a)$

2.3 COMPUTABILITY OF THE ANSWER MAPPING

We consider a finite set L and a decomposition $L = N_L \cup T_L$, where $N_L \cap T_L = \emptyset$. Let $\omega : N_L \longrightarrow \bigcup_{k\geq 1}\{k\} \times L^k$ be a pairwise mapping on L. We consider an element $b_0 \in N_L$ and we denote by $Tree_\omega(b_0)$ the set of all the ω-labelled trees $t = (A,R,h)$ such that $h(root(t)) = b_0$ ([3]).

We consider the equivalence relation ([3]) defined as follows: for $t_1, t_2 \in Tree_\omega(b_0)$ we write $t_1 \approx t_2$ if they are identical as structure.

Let be $t = (A,R,h) \in Tree_\omega(b_0)$. We denote by $S(t)$ the set defined by $((l_1,\dots,l_s),(b_1,\dots,b_s)) \in S(t)$ iff there is $(1,i_1,\dots,i_s) \in Path(t)$ such that $(1,i_1) \in R^{(l_1)},\dots,(i_{s-1},i_s) \in R^{(l_s)}$ and $h(i_1) = b_1,\dots,h(i_s) = b_s$.

We define the operations $\vee : Tree_\omega(b_0)/_\approx \times Tree_\omega(b_0)/_\approx \longrightarrow Tree_\omega(b_0)/_\approx$ and $\wedge : Tree_\omega(b_0)/_\approx \times Tree_\omega(b_0)/_\approx \longrightarrow Tree_\omega(b_0)/_\approx$ as follows:

$$[t_1] \vee [t_2] = [t], \text{ where } S(t) = S(t_1) \cup S(t_2)$$
$$[t_1] \wedge [t_2] = [t], \text{ where } S(t) = S(t_1) \cap S(t_2)$$

The structure $(Tree_\omega(b_0)/_\approx, \vee, \wedge)$ is a distributive lattice ([2]).

Let us consider $X \subseteq L$ and define:

$$U(X) = \{y \in L \mid \exists a \in X \cap N_L, \exists i \in \{1,\dots,\omega_1(a)\} : y = pr_i\omega_2(a)\}$$

For an arbitrary element $b \in L$ we define the sequence:

$$S_b^0 = U(\{b\});\ S_b^{n+1} = S_b^n \cup U(S_b^n) \text{ for } n \geq 0$$

There is $m(b) \geq 1$ such that $S_b^0 \subset \dots \subset S_b^{m(b)} = S_b^{m(b)+1} = \dots$. Thus we can define the mapping $T : L \longrightarrow 2^L$ as follows:

$$T(b) = \emptyset \text{ if } b \in T_L;\ T(b) = S_b^{m(b)} \text{ if } b \in N_L$$

The operator T is used to characterize the existence of the greatest element in $Tree_\omega(b_0)/_\approx$. The lattice $(Tree_\omega(b_0)/_\approx, \vee, \wedge)$ contains a greatest element if and only if $b \notin T(b)$ for every $b \in \{b_0\} \cup T(b_0)$ ([2]).

If we denote $T_{\mathcal{K}} = (L_{fr\_name}(\mathcal{K}) \times L_{Attr}(\mathcal{K})) \setminus (M_{\mathcal{K}} \cup Proc_{\mathcal{K}})$ then $T_{\mathcal{K}} = \{(f,a) \in L_{fr\_name}(\mathcal{K}) \times L_{Attr}(\mathcal{K}) \mid Near_a(f) = \emptyset\}$.
Finally we obtain the following decomposition into pairwise disjoint sets:

$$L_{fr\_name}(\mathcal{K}) \times L_{Attr}(\mathcal{K}) = M_{\mathcal{K}} \cup Proc_{\mathcal{K}} \cup T_{\mathcal{K}}$$

From the definition of $Comp_{\mathcal{K}}$ it follows that

1) $Comp_{\mathcal{K}}(f,a) \in V_{Attr} \cup \{demon\}$ for $(f,a) \in M_{\mathcal{K}}$

2) $Comp_{\mathcal{K}}(f,a) = unknown$ for $(f,a) \in T_{\mathcal{K}}$.

The computation of the value $Comp_{\mathcal{K}}(f,a)$ for $(f,a) \in Proc_{\mathcal{K}}$ can be characterized by means of the lattice theory, using the properties of the ω-labelled trees. In order to accomplish this aim we consider

$$L = M_{\mathcal{K}} \cup Proc_{\mathcal{K}} \cup T_{\mathcal{K}};\ N_L = Proc_{\mathcal{K}},\ T_L = M_{\mathcal{K}} \cup T_{\mathcal{K}}$$

We define the pairwise mapping $\omega : Proc_{\mathcal{K}} \longrightarrow \bigcup_{k\geq 1}\{k\} \times L^k$ as follows. For every $(f,a) \in Proc_{\mathcal{K}}$ we take the arguments $(b_1,\ldots,b_n)$ of the procedure $\Omega_{\mathcal{K}}(f,a)$ and take $\omega_1(f,a) = n$, $\omega_2(f,a) =< (f,b_1),\ldots,(f,b_n) >$.

Based on the results presented in this section we can relieve the following steps to compute the values of the mapping $Comp_{\mathcal{K}}$ ([3]):

Step 1: Compute $M_{\mathcal{K}}$ and $T_{\mathcal{K}}$.

Step 2: If $(f,a) \in M_{\mathcal{K}}$ then search $u \in V_{Attr} \cup \{demon\}$ such that $(a,u) \in Slot(g)$, where $\{g\} = Near_a(f)$; Let $Comp_{\mathcal{K}}(f,a) = u$; else go to Step 2.

Step 3: If $(f,a) \in T_{\mathcal{K}}$ then $Comp_{\mathcal{K}}(f,a) = unknown$; else go to Step 3.

Step 4: If the lattice $Tree_{\omega}(f,a)/\approx$ does not contain a greatest element then $Comp_{\mathcal{K}}(f,a) = undefined$; otherwise let $[t_0]$ be the greatest element; if the frontier of t_0 belong to $(M_{\mathcal{K}} \cup T_{\mathcal{K}})^* \setminus M_{\mathcal{K}}^*$ then $Comp_{\mathcal{K}}(f,a) = unknown$; else

$$Comp_{\mathcal{K}}(f,a) = \Omega_{\mathcal{K}}(f,a)(Comp_{\mathcal{K}}(f,b_1),\ldots,Comp_{\mathcal{K}}(f,b_t))$$

where $(b_1,\ldots,b_t)$ are the arguments of $(\Omega_{\mathcal{K}}(f,a))$.

3 VoSyS: AN ANSWERING SYSTEM BY VOICE

The main purpose of this section is to show an implementation of the model presented in the previous section. The implementation accomplished is a blend of the power offered by the graphical user interfaces and the flavor of the speech technology.

In order to implement the results described in the previous section we used Java technologies (*jdk*1.5.0_02, *Java Speech API*), the language *Prolog* and a bidirectional connection Java-Prolog (the product *JIProlog* $v3.0.2-8$ of Ugo Chirico, [4]). As auxiliary software we used *Apache Ant* and *XML* to allocate the resources for speech technology.

3.1 ARCHITECTURE OF VoSyS

The architecture of the system VoSyS is represented in Figure 1. There are two modules in Prolog:

- The *universal module UM* is represented by 5 files:

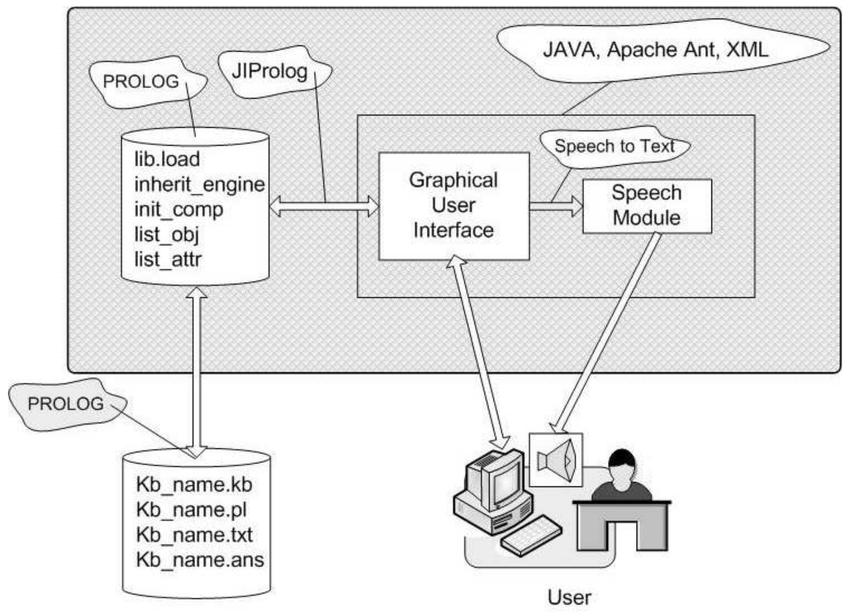


Fig. 1. VoSyS architecture

- **lib.load**: load the libraries *jipxterm.jar* and *jipxsystem* which are used to build the answer as a string;
- **inherit_engine.pl**: accomplishes the computation of the answer mapping $Comp_{\mathcal{K}}$;
- **init_comp.pl**: several steps necessary to initiate the computation of the mapping $Comp_{\mathcal{K}}$, especially to obtain the sets $M_{\mathcal{K}}$ and $T_{\mathcal{K}}$); the computability condition specified in the algorithm of the previous section is used;
- **list_obj.pl**: obtain the list of the objects from the file **Kb_name.pl**;
- **list_attr.pl**: obtain the list of the attributes from the file **Kb_name.pl**.

- The *specific module SM* is composed from 4 files:
  - **Kb_name.kb**: this file specifies the name of the knowledge base; the selection of this name is the starting point of the application;
  - **Kb_name.pl**: the main part of this file defines the content of the knowledge base;
  - **Kb_name.txt**: the content of this file is displayed if the user wishes to see the knowledge base content;
  - **Kb_name.ans**: this file builds the sentences associated to answer.

We observe that the module UM does not depend on the feature of the knowledge base. The second module, which is not a component of the system, depends on this feature (the content of the knowledge base, the output sentences of the answer mapping etc).

To implement the application we used: the Java language, the product JIProlog to obtain the connection Java-Prolog (Ugo Chirico, Shareware License,[4]) and software ”speech to text” (Java Speech API). To allocate and deallocate the resources required by the connection by voice Apache Ant and XML were used as auxiliary software.

3.2 FUNCTIONALITY OF VoSyS

Every time the system $VoSyS$ is launched, the user is informed about the tasks of the system and the following phrases are sent to user by voice:

```
Hello user! My name is VoSys.
I am a product of the Research Center for Artificial Intelligence,
University of Craiova from Romania. I can help you to find the
value of an object attribute in a knowledge system based on
inheritance.
```

In the next step the graphical user interface from Figure 2 is displayed. Two windows (TextArea in Java) for text communication from $VoSyS$ to user can be viewed: the first window contains the results of interrogation; the second windows is used to write in some auxiliary information (the actions performed by buttons, the content of the knowledge base and the values of the mapping $CompK$).

We observe the the graphical interface includes 10 buttons and they are divided into two parts:

- A group of 7 buttons is used to initiate the process of computation and to close the application.
- A group of 3 buttons are used to compute the value of the answer mapping, to generate the output sentence and to speak this sentence.

The main buttons are labeled from 1 to 8 to specify the order of action:

- *Start-processing*(1): select the name *Kb_name* of the knowledge base;
- $Load\_KB(2)$: load the selected knowledge base;
- *Load_objects*(3): compute the objects from *Kb_name.pl* and introduce them into the first structure of type *Choice* of the interface, labeled by *Choose an object name*;
- *Load_attributes*(4): compute the attributes from *Kb_name.pl* and introduce them into the second structure of type *Choice*, labeled by *Choose an attribute name*;
- $Init\_CompK(5)$: initiate the computation process of the answer mapping;
- *Load_library*(6): load the libraries *jipxterm.jar* and *jipxsystem.jar* used to perform some atom-string computations in Prolog and to generate the output sentences;

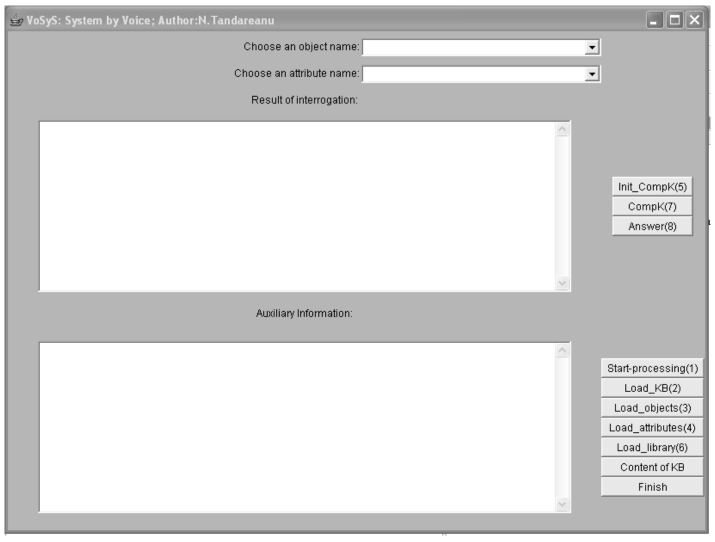


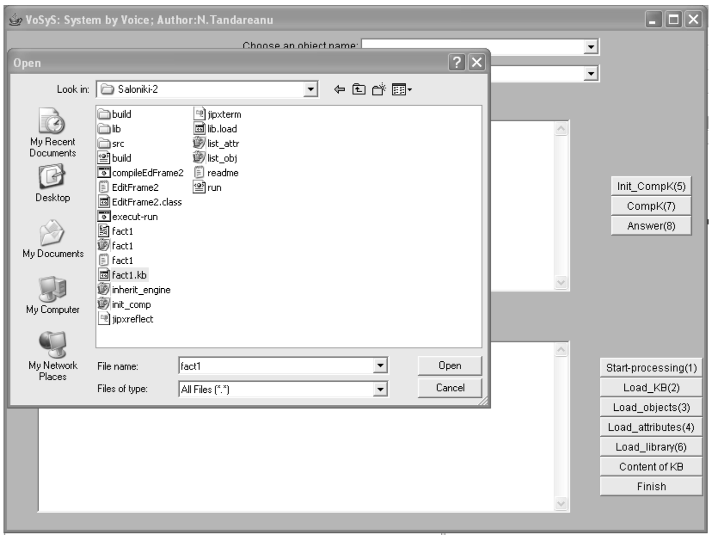
Fig. 2. The graphical interface of $VoSyS$

- $CompK(7)$: compute the values of the mapping $Comp_{\mathcal{K}}$ for the pair (f, a) where f and a are the frame name and the attribute name selected from the structures of type $Choice$;
- $Answer(8)$: introduce the value computed by $CompK(8)$ in a sentence, generate this sentence in a natural language, write the text on display and send it to user by voice;
- $Content\ of\ KB$: put the content of the knowledge base on display;
- $Finish$: close $VoSyS$.

3.3 CASE STUDY

In general we receive a knowledge piece given in a natural language. In order to use $VoSyS$ we have to transpose this piece into a finite subset of L_{Repr}. Equivalently this means that we have to obtain the file *Kb_name.pl*. The modeling process is not a simple process because various conditions can be encountered in the knowledge piece. We exemplify the modeling process for the following knowledge piece:

> *In a competition organized by some association, every candidate obtains the scores $s1$, $s2$ and $s3$. Consequently a general score $s1 + s2 + s3$ is obtained for each candidate. Peter, Elvis, Maria and Susan participate to this competition. Peter likes to play tennis. He has obtained the scores $s1 = 8$, $s2 = 9$ and $s3 = 7$. Elvis has obtained the scores $s1 = 9$ and $s2 = 5$. The score $s3$ is unknown for Elvis because he did not participate to the last test.*

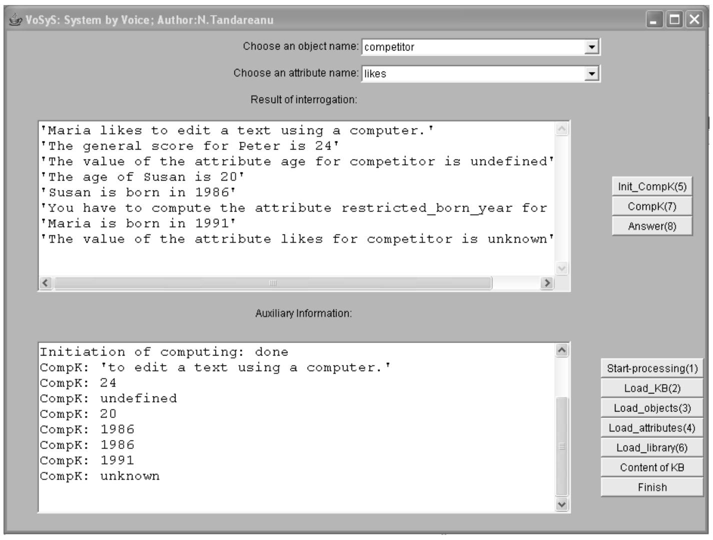


Look in: Saloniki-2

File name: fact1

Files of type: All Files (*.*)

Fig. 3. Step 1: choose the base name



Choose an object name: competitor

Choose an attribute name: likes

Result of interrogation:

```
'Maria likes to edit a text using a computer.'
'The general score for Peter is 24'
'The value of the attribute age for competitor is undefined'
'The age of Susan is 20'
'Susan is born in 1986'
'You have to compute the attribute restricted_born_year for
'Maria is born in 1991'
'The value of the attribute likes for competitor is unknown'
```

Auxiliary Information:

```
Initiation of computing: done
CompK: 'to edit a text using a computer.'
CompK: 24
CompK: undefined
CompK: 20
CompK: 1986
CompK: 1986
CompK: 1991
CompK: unknown
```

Fig. 4. The result of interrogation

> *Susan has obtained the scores* $s1 = 9$*,* $s2 = 8$ *and* $s3 = 5$*. She is a student. Every student likes to edit a text using a computer. Elvis was born in 1987. The age of Susan is 20. Maria obtained the same scores as Susan, she likes to edit a text using a computer also and she is 5 years younger than Susan.*

If we try to model this piece into a knowledge base we obtain the following description:

```
frame(competitor,[],[attr(general_score,proc(pr_score)),
  attr(age,proc(pr_age)),attr(born_year,proc(pr_born))]).
frame('Peter',[competitor],[attr(likes,'to play tennis'),
  attr(first_score,8),attr(second_score,9),attr(third_score,7)]).
frame('Elvis',[competitor],[attr(first_score,9),
  attr(second_score,5),attr(born_year,1987)]).
frame('Susan',[competitor,student],[attr(first_score,9),
  attr(second_score,8),attr(third_score,5),attr(age,20)]).
frame(student,[],[attr(likes,'to edit a text using a computer.')]).
frame('Maria',['Susan'],[attr(restricted_age,proc(pr_age_1)),
  attr(difference_of_age,5),attr(restricted_born_year,
  proc(pr_born1))]).
```

We remark that two distinct attributes to represent the *age* and also two distinct attributes for the *born year* were used. This is explained by the fact that the object $Maria$ is younger than $Susan$ and thus in order to find the age of $Maria$ we have to compute first the age of $Susan$ and then we obtain the age of $Maria$. For all other objects we use the attribute *age* and only for $Maria$ we use the attribute *restricted_age*. This is a restriction imposed by the model and not by the knowledge piece. A solution is given in the file *Kb_name.ans*: every time when the attribute *age* or *born_year* is required by the user for the object $Maria$, it is directed to *restricted_age* and *restricted_born_year* respectively. For example, the system gives the answer *Maria is born in 1991* as we can view in Figure 4.

The following information are also included in the file *Kb_name.pl*: the arguments of each procedure, the specific values of the answer mapping for the case when one of the arguments is $undefined$ or *unknown* and the information *male* or $female$ for a person.

```
procedura(pr_score,[first_score,second_score,third_score]).
procedura(pr_age,[born_year]).
procedura(pr_born,[age]).
procedura(pr_born1,[restricted_age]).
procedura(pr_age_1,[age,difference_of_age]).
calc_proc(_,Param,'undefined'):-exists_undef(Param).
calc_proc(_,Param,'unknown'):-exists_unkn(Param),!.
calc_proc(pr_score,[X,Y,Z],W):-W is X+Y+Z.
calc_proc(pr_age,[X],W):-date(Y,_,_),W is Y-X.
calc_proc(N,[X],W):-(N=pr_born;N=pr_born1),date(Y,_,_),W is Y-X.
calc_proc(pr_age_1,[X,Y],W):-W is X-Y.
exists_undef([X|_]):-X='undefined',!.
exists_undef([_|Q]):-exists_undef(Q).
exists_unkn([X|_]):-X='unknown',!.
exists_unkn([_|Q]):-exists_unkn(Q).
masc('Peter').
masc('Elvis').
fem('Maria').
fem('Susan').
```

Let us compare the content of the communication windows from the graphical interface. If we send the sequence of interrogations

$(Maria, likes)$, $(Peter, general\_score)$, $(competitor, age)$,
$(Susan, age)$, $(Susan, born\_year)$, $(Maria, born\_year)$,
$(Maria, restricted\_born\_year)$, $(competitor, likes)$

then we obtain the following results:

In the window for "Result of interrogation":

```
'Maria likes to edit a text using a computer.'
'The general score for Peter is 24'
'The value of the attribute age for competitor is undefined'
'The age of Susan is 20'
'Susan is born in 1986'
'You have to compute the attribute restricted_born_year for Maria'
'Maria is born in 1991'
'The value of the attribute likes for competitor is unknown'
```

In the window "Auxiliary information":

```
CompK: 'to edit a text using a computer.'
CompK: 24
CompK: undefined
CompK: 20
CompK: 1986
CompK: 1986
CompK: 1991
CompK: unknown
```

We can observe that the value $Comp_{\mathcal{K}}(Maria, born\_year)$ is 1986 but the answer is the sentence 'You have to compute the attribute restricted_born_year for Maria'. If we ask for $Comp_{\mathcal{K}}(Maria, restricted\_born\_year)$ then the value is 1991 and the answer is the sentence 'Maria is born in 1991'.

4 FUTURE WORK

The system can be extended to use the recognition voice. Thus the query can be introduced as a text or can be spoken at a microphone.

References

[1] **N. Ţăndăreanu**, Lattices of labelled ordered trees (I), *Annals of the University of Craiova*, Mathematics and Computer Science Series, Vol. XXVIII, 2001, p.29-39

[2] **N. Ţăndăreanu**, Lattices of labelled ordered trees (II), *Annals of the University of Craiova*, Mathematics and Computer Science Series, Vol. XXIX, 2002 (to appear)

[3] **N. Ţăndăreanu**, Inheritance-based Knowledge Systems and Their Answer Functions Computation Using Lattice Theory, *Romanian Journal of Information Science and Technology*, Volume 6, Numbers 1-2, 2003, 227-248

[4] **Ugo Chirico**, JIPrologRefManual.pdf, http://www.ugosweb.com/jiprolog