Annals of the University of Craiova, Mathematics and Computer Science Series
Volume 37(3), 2010, Pages 78–91
ISSN: 1223-6934

DiaSys - A Dialogue System based on Conditional Knowledge

Cristina Zamfir

Abstract. In this paper the DiaSys application is presented. The DiaSys is a dialog system based on conditional schemas that allows end users to interrogate the knowledge base through questions that are formed in natural language. The system interprets the question and produces the correct answer. After generating the answer this will be transformed in a natural language text that will be presented to the user.

2010 Mathematics Subject Classification. Primary 68T30; Secondary 68T35.
Key words and phrases. natural language, dialogue system, question answering, conditional schemas, knowledge base.

1. Introduction

A dialogue system is a computer system intended to converse with a human, with a coherent structure. Dialogue systems have employed text, speech, graphics, gestures and other modes for communication on both the input and output channel.

In general by a *dialogue system* we understand a software system designed to provide answers to questions that are formulated by an user in natural language. Usually a dialog system maintains the conversation by asking the user different questions in order to produce a more accurate answer. This will be further discussed in the "The system architecture" and "Using DiaSys" sections.

To find the answer to a question, a system may use a database, a collection of natural language documents or a knowledge base. Various such systems were developed.

In this paper we describe an architecture and a Java implementation of a dialogue question answering system based on conditional knowledge. The main features of this system are the following ones:

- The communication user-system is based on a graphic interface.
- The information is represented into a knowledge base that uses conditional knowledge.
- The implementation is platform independent because Java technology is used.
- The product can be used successfully in automatic training, to build health systems consultancy and systems for diagnosing the malfunctions of a device.

2. Conditional knowledge

The concept of conditional knowledge was introduced in [12]. The inference mechanism of this structure is treated in [5].

A conditional schema is a tuple $\mathcal{S} = (Ob, C_s, E_r, A, V, B_{cr}, h, f)$ such that:

Received June 30, 2010. Revision received August 27, 2010.
This article was produced under the project "Supporting young Ph.D students with frequency by providing doctoral fellowships", co-financed from the EUROPEAN SOCIAL FUND through the Sectoral Operational Program Development of Human Resources.

- $Ob = Ob_{ind} \cup Ob_{abstr}$, where Ob_{ind} is the set of the individual objects and Ob_{abstr} is the set of the abstract objects; we suppose that $Ob_{ind} \cap Ob_{abstr} = \emptyset$;
- C_s is the set of conditional mappings;
- E_r is the set of the symbols for conditional binary relations;
- A is the set of attributes for the elements of Ob_{ind} while V is the set of their values;
- $B_{cr} \subseteq 2^{((Ob \times I) \times (Ob \times I)) \times C_s}$ is the set of conditional binary relations and $I = \{i, a\}$, where i is used to designate individual objects and a is used to specify abstract objects. Thus an individual object is specified as (x, i) and an abstract object has the form (x, a).
- $h : E_r \longrightarrow B_{cr}$ maps a conditional binary relation for every symbol of E_r;
- $f : Ob_{ind} \longrightarrow 2^{A \times V}$ assigns declarative knowledge to the individual objects of Ob_{ind}.

  The conditional graph ([12]) generated by $\mathcal{S}$ is the system $G_{\mathcal{S}} = (X \cup Z, \Gamma_X \cup \Gamma_Z)$, where:
- $X \subseteq Ob \times I$ is the set of nodes such that $x \in X$ if and only if there are $r \in E_r$, $y \in X$ such that $(x, y) \in_c h(r)$ or $(y, x) \in_c h(r)$;
- $\Gamma_X \subseteq X \times E_r \times X$ and $((n, \omega_1), r, (m, \omega_2)) \in \Gamma_X$ if and only if $((n, \omega_1), (m, \omega_2)) \in h(r)$; the elements of Γ_X are named *arcs of first category*;
- $Z = \{f(x) \mid x \in Ob_{ind}\}$ and $\Gamma_Z = \{(f(x), x) \mid x \in Ob_{ind}\}$; the elements of Γ_Z are named *arcs of the second category.*

3. The system architecture

A possible architecture is shown in Figure 1. In this section the components of the system and the connection between them are described.

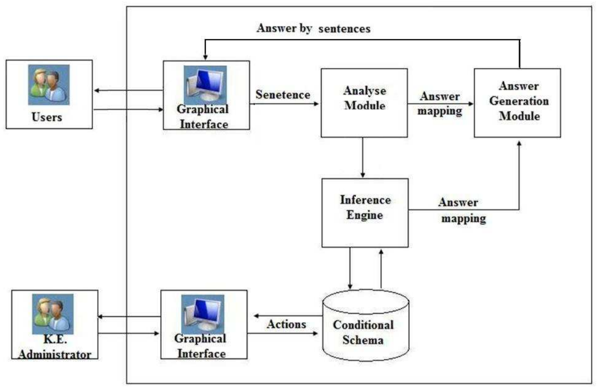


FIGURE 1. An architecture for conditional schema implementation

The communication between user and system is based on a *graphical interface* which collaborates with the Analysis Module; a context free grammar is used to guide the communication in natural language. A knowledge piece that can be represented by our model can contain two kinds of information:

- *declarative knowledge* - propositions or facts describing the current state of the problem (facts that are known to be true); the user can submit questions which refer only to declarative knowledge.

- *procedural knowledge* - the logical rules: IF *condition* THEN *action* ELSE *action*; these rules are used to compute the values of the conditional symbols.

The module ***Conditional Schema*** contains the component of a conditional schema.

The ***Inference Engine*** performs the inference (receives two nodes and gives the result of the computation).

This inference mechanism is a path-driven one. The conditional graph of a conditional schema contains two kinds of nodes: individual nodes and abstract nodes. An individual node contains pairs of the form *(attribute,value)*, where *attribute* represents an attribute name and *value* gives the value of the corresponding attribute.

A path from n_1 to n_{k+1} is a pair $d = ([(n_1, \omega_1), \ldots, (n_{k+1}, \omega_{k+1})], [a_1, \ldots, a_k])$, where $n_1, \ldots, n_{k+1} \in X$, $\omega_1, \ldots, \omega_{k+1} \in \{a, i\}$ and $a_1, \ldots, a_k \in E_r$.

The knowledge engineer must define a partial mapping Φ such that $\Phi([a_1, \ldots, a_k])$ is a new label representing a "compounded" label of $a_1, \ldots, a_k$. We denote by E_r^* the union set of E_r and the set of the compounded labels. In order to apply the inference mechanism we suppose that:

- there is $j \in \{1, \ldots, k+1\}$ such that $w_j = i$
- $[a_1, \ldots, a_k] \in dom(\Phi)$

In order to interrogate such a system, the user specifies two nodes α_1 and α_2. Each arc (k_i, k_j) of a given path contains two kinds of labels:

- a label to specify a binary relation between k_i and k_j; this is the set E_r from the definition of a conditional schema;
- a label that identifies a certain condition which must be verified/satisfied by the nearest individual object of k_i; the set of these labels is C_s and the connection between an element of E_r and the conditional binary relations is given by the mapping h from the definition of a conditional schema;

The node α_1 must be an individual node. The condition imposed on the arc (k_i, k_j) can be viewed as a semaphore. The value of a semaphore is computed by means of some rule of the form IF-THEN-ELSE, where the attribute values of the individual objects are used. If the condition is true then the semaphore is "on", otherwise is "off".

If all semaphores of the path d are "on" then some conclusion is obtained by the inference mechanism. If some semaphore is "off" then either no conclusion is obtained or a "negative" conclusion is specified.

The answer mapping is defined using a semantic function Sem. If G is the grammar of user-system communication and $L(G)$ is the language generated by G then $Sem : Ob \times E_r^* \times Ob \times \{\text{on, off}\} \longrightarrow L(G)$ and this mapping is defined by knowledge engineer. We denote by $dom(\Phi)$ the domain of the mapping Φ.

Let us consider the path $d = ([(\alpha_1, \omega_1), \ldots, (\alpha_{k+1}, \omega_{k+1})], [a_1, \ldots, a_k])$.

We suppose that:

- there is $j \in \{1, \ldots, k+1\}$ such that $w_j = i$
- $[a_1, \ldots, a_k] \in dom(\Phi)$
- $[t_1, \ldots, t_k]$ is the list of all conditional symbols of d, where the order of the arcs of d is preserved by this list.

We define $ans(d)$ as follows:

- If $t_1[d] = \ldots = t_k[d] = \text{on}$ and $(\alpha_1, \Phi([a_1, \ldots, a_k]), \alpha_{k+1}, on) \in dom(Sem)$ then $ans(d) = Sem(n_1, \Phi([a_1, \ldots, a_k]), n_{k+1}, \text{on})$.
- If there is $u \in \{1, \ldots, k\}$ such that $t_u[d] = $off and $(\alpha_1, \Phi([a_1, \ldots, a_k]), \alpha_{k+1}, \text{off}) \in dom(Sem)$ then $ans(d) = Sem(\alpha_1, \Phi([a_1, \ldots, a_k]), \alpha_{k+1}, \text{off})$
- $ans(d) = unknown$ otherwise.

The Inference Engine is also responsible for the maintaining of the dialogue between the users and the system. This is done by assessing the information needed in order to select a proper knowledge base which is critical if the system is to generate a correct and precise answer to the user. The Inference Engine transferees a question in the form of an answer mapping to the Answer Generation Module. The Answer Generation Module will formulate an natural language question that will be displayed to the user. By repeating this process the Inference Engine will obtain the necessary information to select a knowledge base and construct an answer.

The main task of the ***Analyse Module*** is to extract the semantics from the received sentences.

In this module, the text input by the user is analyzed and the semantics of the text are extracted. This is done using a Recursive Transition Network. We consider that, in time, due to the different use of the application, the grammar should be able to be changed according to the needs of the end users. This is why the grammar will be used from a file, so it can be easily configured at a later point in time.

If the text obtained by the automatic speech module is grammatically correct then, using an algorithm, we will obtain objects and relations. If the text is not grammatically correct then the message received from the user will be considered as not valid. This information will be passed to the Answer Generation Module. The Analyse Module sends the Inference Engine module the objects on which to perform the inference.

The ***Answer Generation*** module receives the value of the answer mapping and produces the answer sentences. We relieve the following facts:

- $Sem(x, a, y, \mathrm{on})$ specifies the semantics of the relation between the objects x and y, which is identified by the symbol a;
- $Sem(x, a, y, \mathrm{off})$ specifies the converse property.

For example, if $Sem(Peter, is\_a, student, \mathrm{on})$=*Peter is a student* then $Sem(Peter, is\_a, student, \mathit{off})$ = *Peter is not a student.*

Frequently the answer given by this module is not the same as the sentence received from inference engine. This can be viewed from the following example. Suppose that the user ask the system "Does Maria like to eat pizza?" If the value of the mapping *ans* is the sentence "A nephew of Maria likes to eat pizza" then the answer given by this module is the sentence "No, a nephew of Maria likes to eat pizza".

The ***Graphical Interface*** module is an interface for administrator. By means of this interface the knowledge engineer or the administrator performs the following tasks:

- create a conditional schema; add new components to the conditional schema; modify the existing components of the conditional schema; delete the components of the conditional schema; delete the conditional schema; visualize a conditional schema;
- define, modify and visualize the mapping Φ;
- define, modify and visualize the mapping Sem.

4. Java implementation

The implementation was performed by means of Java platform. The Java API for XML Processing (JAXP) is used for processing XML data using applications written in the Java programming language. JAXP leverages the parser standards Simple API for XML Parsing (SAX) and Document Object Model (DOM).

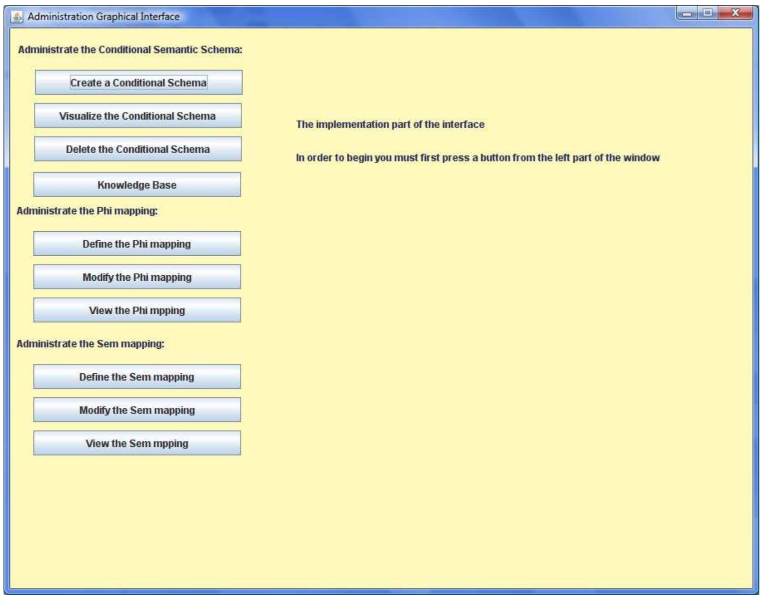
To process all the xml files that this application uses we have worked with DOM and SAX. DOM is the acronym for Document Object Model which is a API component of the Java API for XML Processing. A DOM is a garden-variety tree structure, where each node contains one of the components from an XML structure. The two most common types of nodes are element nodes and text nodes. Using DOM functions we can you create nodes, remove nodes, change their contents, and traverse the node hierarchy

The Simple API for XML (SAX) is the event-driven, serial-access mechanism that does element-by-element processing. The API for this level reads and writes XML to a data repository or the web. The interfaces provided in the SAX package is an important part of our toolkit for handling XML.

After the application is installed the first step is to configure the application. This must be done by an user with administrative capabilities.

An example of the grammar xml file is presented below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<grammar>
<Prop>
    <andgroup>
        <S/>
        <P/>
    </andgroup>
    <P/>
</Prop>
<S>
    <andgroup>
        <art/>
        <adj/>
        <subst/>
    </andgroup>
    <andgroup>
        <art/>
        <subst/>
    </andgroup>
    <andgroup>
        <adj/>
        <subst/>
    </andgroup>
    <subst/>
</S>
<P>
    <andgroup>
        <verb/>
        <GV/>
    </andgroup>
    <verb/>
</P>
<GV>
    <andgroup>
        <art/>
```


Administration Graphical Interface
Administrate the Conditional Semantic Schema:
Create a Conditional Schema
Visualize the Conditional Schema
Delete the Conditional Schema
Knowledge Base
Administrate the Phi mapping:
Define the Phi mapping
Modify the Phi mapping
View the Phi mpping
Administrate the Sem mapping:
Define the Sem mapping
Modify the Sem mapping
View the Sem mpping

The implementation part of the interface

In order to begin you must first press a button from the left part of the window

FIGURE 2. The Administration Graphical Interface

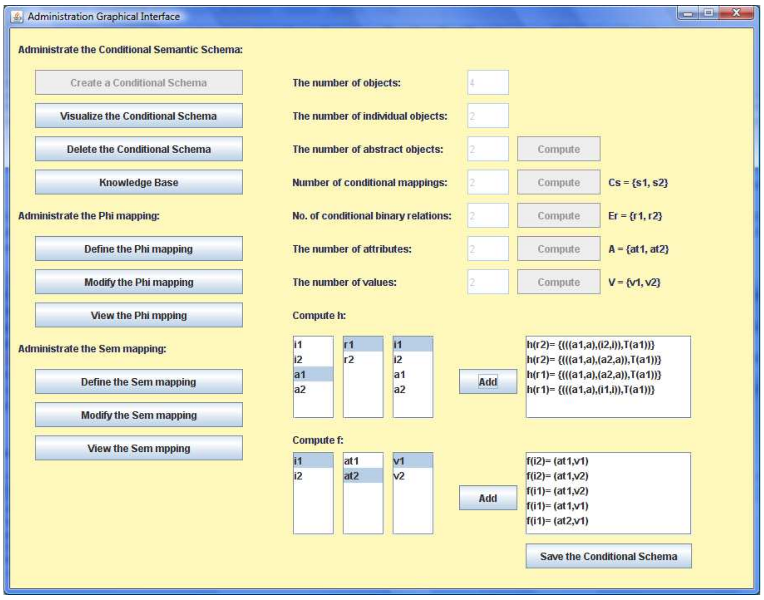
```
            <adj/>
            <subst/>
        </andgroup>
        <andgroup>
            <art/>
            <adj/>
        </andgroup>
        <andgroup>
            <art/>
            <subst/>
        </andgroup>
        <adj/>
        <subst/>
</GV>
</grammar>
```

The rest of the configuration can pe performed by the administrator through a graphical interface or directly in the xml files.

As shown in Figure 2, the administrator can:

- Create a Conditional Schema
- View the Conditional Schema
- Delete the Conditional Schema
- Create a Knowledge Base
- Define, Edit and Delete the Φ mapping;
- Define, Edit and Delete the Sem mapping;

When creating a Conditional Schema the administrator will input the number of individual and abstract objects. The application will create unique symbols for the objects. The next step is to input the number of Conditional Mappings so that the


Administration Graphical Interface
Administrate the Conditional Semantic Schema:
Create a Conditional Schema
Visualize the Conditional Schema
Delete the Conditional Schema
Knowledge Base
Administrate the Phi mapping:
Define the Phi mapping
Modify the Phi mapping
View the Phi mpping
Administrate the Sem mapping:
Define the Sem mapping
Modify the Sem mapping
View the Sem mpping
Compute
Add
Save the Conditional Schema

The number of objects: 4
The number of individual objects: 2
The number of abstract objects: 2
Number of conditional mappings: 2 Cs = {s1, s2}
No. of conditional binary relations: 2 Er = {r1, r2}
The number of attributes: 2 A = {at1, at2}
The number of values: 2 V = {v1, v2}

Compute h:

i1	r1	i1
i2	r2	i2
a1		a1
a2		a2

```
h(r2)= {(((a1,a),(i2,i)),T(a1))}
h(r2)= {(((a1,a),(a2,a)),T(a1))}
h(r1)= {(((a1,a),(a2,a)),T(a1))}
h(r1)= {(((a1,a),(i1,i)),T(a1))}
```

Compute f:

i1	at1	v1
i2	at2	v2

```
f(i2)= (at1,v1)
f(i2)= (at1,v2)
f(i1)= (at1,v2)
f(i1)= (at1,v1)
f(i1)= (at2,v1)
```

FIGURE 3. The Administration Graphical Interface

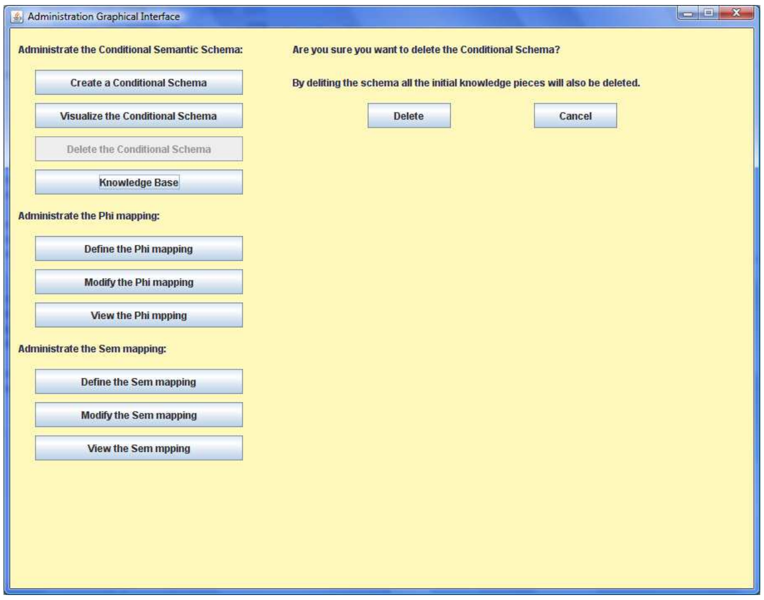
application can generate unique symbols for the mappings. Using the interface, the administrator will also create the E_r and A sets and the h mapping. After completing this operations the user will save the conditional schema. By pressing the button ”Visualize the Conditional Schema” the administrator can later view and modify the prior created schema. Modifying the schema is done by adding or removing symbols of objects or of conditional mappings or by adding or removing elements of the C_s, E_r or A sets.

In the Figure 3 we can see how the user constructs the Conditional Schema. When the process is finished the user must press the ”Save the Conditional Schema” button. This will save the schema in an xml file format that will be used later on by the application when interacting with normal users.

As mentioned earlier the administrator can create, modify and delete the conditional schema directly from the xml file. An example of a conditional schema is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<conditional_schema>
<iobjs>
    i1
    i2
    i3
    i4
</iobjs>
<aobjs>
    a1
    a2
    a3
</aobjs>
```

```
<CS>
    s1
    s2
</CS>
<ER>
    r1
    r2
    r3
    r4
</ER>
<Attributes>
    at1
    at2
    at4
</Attributes>
<Values>
    v1
    v2
    v3
    v4
</Values>
<h_mapping>
    h(r1)={(((i1,i),(i2,i)),T(i1))}
    h(r2)={(((i1,i),(i3,i)),T(i1))}
    h(r2)={(((i2,i),(i4,i)),T(i2))}
    h(r3)={(((i3,i),(a2,a)),s1)}
    h(r3)={(((i4,i),(a3,a)),s2)}
    h(r4)={(((i1,i),(a1,a)),T(i1))}
</h_mapping>
<f_mapping>
    f(i1)=(at1,v1)
    f(i1)=(at1,v2)
    f(i1)=(at2,v1)
    f(i1)=(at2,v2)
    f(i2)=(at1,v1)
    f(i2)=(at2,v2)
    f(i2)=(at2,v1)
    f(i2)=(at1,v2)
</f_mapping>
<u>
    u(r1)=(i1,i2)
    u(r2)=(i1,i3)
    u(r2)=(i2,i4)
    u(r3)=(i3,a2)
    u(r3)=(i4,a3)
    u(r4)=(i1,a1)
</u>
</conditional_schema>
```


Administration Graphical Interface
Administrate the Conditional Semantic Schema:
Create a Conditional Schema
Visualize the Conditional Schema
Delete the Conditional Schema
Knowledge Base
Administrate the Phi mapping:
Define the Phi mapping
Modify the Phi mapping
View the Phi mpping
Administrate the Sem mapping:
Define the Sem mapping
Modify the Sem mapping
View the Sem mpping
Delete
Cancel

Are you sure you want to delete the Conditional Schema?

By deliting the schema all the initial knowledge pieces will also be deleted.

FIGURE 4. The Graphical Interface

The administrator can view or modify the schema later on by pressing the "Visualize the Conditional Schema" button. The schema can be viewed and modified in directly from the xml file or trough the application graphical interface.

When the engineer wants to delete the conditional schema it is very important to remember that all the initial knowledge pieces created on this schema will also be deleted. It is recommended to save copies of the xml files in order to have a back-up for latter. In the delete interface the administrator is asked if he wants to delete the conditional schema and he is reminded that the initial knowledge pieces will be deleted. This is shown in Figure 4

When the "Knowledge Base" button is pressed the interface used to create a knowledge base is shown. The administrator is allowed to create more than one knowledge base that is constructed on the initial schema. Trough this interface the administrator establishes connections between the symbols of the objects and the objects, he constructs the Φ and *Sem*.

The construction of Φ and *Sem* is done in a similar matter as to the conditional schema. The administrator inputs the number of the necessary relations, the application creates the symbols for the relations and the administrator, using lists of values, creates the desired mappings. The Φ and *Sem* mappings are also saved in an xml file format.

As mentioned earlier the administrator can create, modify and delete the knowledge base directly from the xml file. An example of a knowledge base is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<knowledge_base>
<natural_language_text>
Helen is the mother of Peter. She is 40. Helen made cheese cake.
Peter likesto eat cheese cake if this is made by his mother.
Susan is Helen’s sister and George is her son.
```

```
George likes to eat fruits if they are bought by his mother.
Susan is 30. She bought some bread. Helen likes to eat pizza.
</natural_language_text>
<iobjs>
    i1=Helen
    i2=Susan
    i3=Peter
    i4=George
</iobjs>
<aobjs>
    a1=pizza
    a2=cheese cake
    a3=fruits
</aobjs>
<CS>
    s1
    s2
</CS>
<ER>
    r1=is mother
    r2=is sister
    r3=likes
    r4=eats
</ER>
<Attributes>
    at1=age
    at2=made
    at4=bought
</Attributes>
<Values>
    v1=40
    v2=cheese cake
    v3=30
    v4=fruits
</Values>
<Reg>
    s1=R1(i3)=IF(y,i3) in u(r2) and f(y)=(at2,v2) then s1(i3)=on
  else s1(i3)=off
    s2=R2(i4)=IF(y,i4) in u(r2) and f(y)=(at4,v4) then s2(i4)=on
  else s2(i4)=off
</Reg>
<phi>
    phi(r1,r2)=b1
    phi(b1,r3)=b2
    phi(r2,r3)=b3
</phi>
<b>
    b1
    b2
```

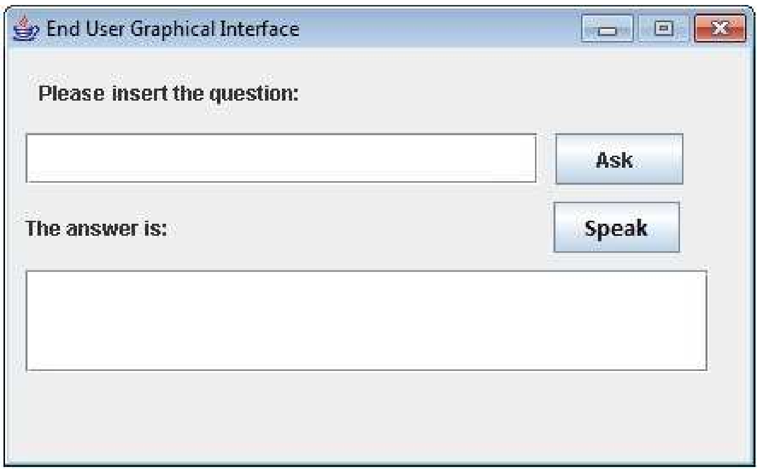


FIGURE 5. The End User Graphical Interface

```
    b3
</b>
<Big_phi>
    PHI([r1,r2])=b1
    PHI([r1,r2,r3])=b2
    PHI([r2,r3])=b3
</Big_phi>
<sem>
    Sem(x,r1,y,on) = x is siter of y
    Sem(x,r1,y,on) = x is not sister of y
    Sem(x,r2,y,on) = x is mother of y
    Sem(x,r2,y,off) = x is not mother of y
    Sem(x,r3,y,on) = x eats of y
    Sem(x,r3,y,off) = x does not eat of y
    Sem(x,r4,y,on) = x likes to eat of y
    Sem(x,r4,y,off) = x does not like to eat of y
    Sem(x,b1,y,on) = x is aunt of y
    Sem(x,b1,y,off) = x is not aunt of y
    Sem(x,b2,y,on) = the nephew of x eats y
    Sem(x,b2,y,off) = the nephew of x does not eat y
    Sem(x,b3,y,on) = The sun of x eats y
    Sem(x,b3,y,off) = The sun of x does not eat y
</sem>
</knowledge_base>
```

After the administration part of the application is properly configured the end users can start using the application. An end user will connect to the application trough the graphical interface shown in Figure 5.

The obtained text will be submitted to the following algorithm:

- Parsing - Dividing the proposition into words

- Semantical and grammatical evaluation - In order to analyze and work with a text we must first determine if the text is grammatically correct (from our defined context free grammar point of view);
- Extracting objects - in this step we extract the objects from the sentence that are also found in the initial knowledge piece
- Extracting relations - in this step the algorithm identifies the relations;
- Applying Simb - in order for the inference to be performed the symbols for the found objects must be used. In this step the objects are assigned the symbols from the conditional schema;
- Applying Label - in order for the inference to be performed the labels for the found relations must be used. In this step the relations are assigned the labels from the conditional schema;

Once that the message input by the user is transformed and can be interpreted the inference will be performed.

Using the initial knowledge pieces xml file, the objects and the relations that where introduced by the user trough the verbal signal will be identified in the conditional schema.

If an object or an relation can not be identified in the conditional schema, the Inference Engine will send an error message to the Answer Generation Module that will be transformed in a verbal message. The user will be asked to contact the administrator by e-mail or telephone. This approach will ensure that the user will receive the answer for his question and that the administrator will be informed of the issue that needs to be corrected.

If all the objects and relations can be identified in the conditional schema then using the inference mechanism and the answer mapping generation the application will calculate all the answers to the question of the user.

The answer generated by the Inference Engine will be sent to the Answer Generation Mapping where the natural language answer will be formed. The text will be transmitted to the Text to Speech module that will transform the text into a verbal signal if the user actions the "Speak" button.

For the construction of this application we have used java packages like

- org.w3c.dom.*, org.xml.sax.SAXException, javax.xml.parsers.*, java.io.*, javax.xml.transform.*, javax.xml.transform.dom.DOMSource for working with the xml files;
- java.util.StringTokenizer and java.util.Vector for the parsing of the text;
- java.awt.*, java.awt.event.*, javax.swing.* for constructing and working with the graphical interface;

5. Using DiaSys

As mentioned before, the DiaSys is a system that is used to perform a conversation between a user and the system. The conversation uses natural language.

An user must connect to the system trough the graphical end user interface that is presented in Figure 5. The user can address the system with a direct question or it can start a so called person-to-person dialogue.

For example, the phrase "What car should I buy?" is considered to be a direct question. In this case the system needs to ask some additional questions in order to select the appropriate knowledge base. This will start a person-to-person dialogue. An example of the possible dialogue is presented below:

```
User: What car should I buy?
System: Hello. I am DiaSys. In order to answer you're question I will
need some information about you.
How old are you?
User: 25.
System: Are you a male or a female?
User: Male.
System: Do you own a driving license?
User: Yes.
System: What should be the price range for the car?
[... Some other questions that will help the system narrow down the
list of cars will be asked. The system will send the response by
mail to the user...]
```

All the questions that the system has asked the user are meant to narrow the list of possibilities that the system has. Further more by answering the questions the user will obtain a personalized answer that will satisfy the needs of the user. The questions are asked based on the attributes described in the knowledge base.

An example of a person-to-person dialogue is shown below.

```
User: Hello
System: Hello. I am DiaSys. Can you please tell me you're name?
User: I am Peter.
System: How can I help you, Peter?
User: I would like to receive a copy of my phone invoice.
System: I need to verify some information first.
[... The system will ask the user for personal information ...]
System: A copy of the invoice has been sent to the e-mail that you
provided. Is there some other information that I can help you with?
User: Yes.I wold like to know the address of the University of Pitesti.
System: The address is Str. Targu din Vale nr. 1, Pitesti, Romania.
Is there some other information that I can help you with?
User: No, thank you.
System: Thank you for using DiaSys. Have a nice day.
```

Based on the information gathered from the user, the dialogue the system is capable of selecting the appropriate knowledge base. If the information that the user offers dose not suffice additional questions will be asked by de system.

6. Conclusions and future work

In this paper an architecture of a dialogue question answering system is proposed. The inference engine and the knowledge base are built taking into account a knowledge representation method named conditional knowledge. The user can interrogate the system only by sentences concerning the declarative facts. An interesting extension of this study refers to the case when the user asks the system in the area of procedural knowledge. In this way we are led to the idea to add a way to explain how the system obtained the conclusion. This can be named *Module of Explanation.* In this way our system can be used more successfully in automatic training. We intend to extend the dialogue system based on conditional knowledge. In a future work we exemplify the use of such a system in automatic training, to build health systems consultancy and systems for diagnosing the malfunctions of a device.

References

[1] E. Agichtei and L. Gravano, Snowball: Extracting Relations from Large Plain-Text Collections, *Proceeding of the 5th ACM International Conference on Digital Libraries (DL'00)*, San Antonio, TX. June 2-7, (2000).

[2] E. Breck, J. Burger, L. Ferro, D. House, M. Light and I. Mani, A sys called Qanda, *Eighth Text REtrieval Conference (TREC-8)*, Gaithersburg, MD. November 17-19, (1999).

[3] E. Breck, J. Burger, L. Ferro, W. Greiff, M. Light, I. Mani and J. Rennie, Another sys called Qanda, *Ninth Text REtrieval Conference(TREC-9)*, Gaithersburg, MD. November 13-16, (2000).

[4] E. Brill, S. Dumais and M. Banko, An analysis of the AskMSR question-answering system, *Proceedings of the ACL-02 conference on Empirical methods in natural language processing* (10) (2002), 257–264.

[5] M. Colhon and N. Ţăndăreanu, The Inference Mechanism in Conditional Schemas, *Annals of the University of Craiova, Mathematics and Computer Science Series* **37** (2010), no.1, 55–70.

[6] A. Fujii and T. Ishikawa, Organizing Encyclopedic Knowledge based on the Web and its Application to Question Answering, *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL-EACL 2001)*, Toulouse, France, July 6-11, (2001).

[7] S. Harabagiu, D. Moldovan, C. Clark, M. Bowden, A. Hickl and P. Wang, Employing Two Question Answering Systems in TREC-2005, *In: Text Retrieval Conference (TREC-14)*,Gaithersburg, Maryland (2005).

[8] B. Katz, Using English for Indexing and Retrieving, In *Artificial Intelligence at MIT: Expanding Frontiers* **1** (1990), Cambridge, Massachusetts.

[9] H. Kim, K. Kim, G.G. Lee and J. Seo, MAYA: a fast Question-answering system based on a predictive answer indexer, *Proceedings of the workshop on Open-domain question answering* **12** (2001), Toulouse, France, 1–8.

[10] D. Moldovan, M. Bowden and M. Tatu, A Temporally-Enhanced PowerAnswer in TREC 2006, In: Text Retrieval Conference (TREC-15), Gaithersburg, Maryland (2006).

[11] D.I. Moldovan, C. Clark, S.M. Harabagiu, D. Hodges, COGEX: A semantically and contextually enriched logic prover for question answering, *J. Applied Logic* **5** (2007), no. 1, 49–69.

[12] N. Ţăndăreanu and M. Colhon, Conditional graphs generated by conditional schemas, *Annals of the University of Craiova, Mathematics and Computer Science Series* **36** (2009), no.1, 1–11.

[13] Z. Zheng, AnswerBus Question Answering System, *Proceeding of HLT Human Language Technology Conference (HLT 2002)*. San Diego, CA. March 24 – 27, (2002).

[14] M.A. Yarmohammadi, M. Shamsfard, M.A. Yarmohammadi and M. Rouhizadeh, SBUQA Question Answering System, Advances in Computer Science and Engineering, *13th International CSI Computer Conference, CSICC 2008* (2009), Kish Island, Iran, March 9-11, 2008 Revised Selected Papers, Springer Berlin Heidelberg, 316–323.

(Cristina Zamfir) Department of Computer Science, University of Pitesti, Faculty of Mathematics and Computer Science, Str. Targu din Vale nr. 1, cod 110040, Pitesti, Romania
E-mail address: cristina.tudorache@star-storage.ro