Using Client-Server Technology to Query Knowledge Bases in Natural Language

Sorin Dincă
Faculty of Accounting and Financial Management Craiova
Spiru Haret University
sorin.dinca@gmail.com

Abstract
A major area of artificial intelligence deals with the design of knowledge representation and processing. From an intuitive point of view (abbreviated SRPC) is a collection of components that cooperate so that the system is able to reason, i.e. to answer a query. Such a system uses a knowledge base and a particular method of knowledge representation. A semantic schema has the form of a set of nodes and arcs, directed and labeled. Nodes are objects that can abstract concepts or attributes. Arcs are used to represent the links between these objects. Labels arcs have a sense arbitrary. In this paper we present an implementation of the knowledge base using the facilities of the Java language in implementing communication interfaces.
Keywords: *Token, Transition Network, ServerSocket, JTable*
ACM/AMS Classification: 68U99

1. Query in natural language knowledge bases

Besides formalism representation of syntactic structure in a text using context-free grammar, graphical representation can be used and applied successfully in parsing. This representation is called the Transition Network.

A transition diagram is a directed graph in which the arcs are labeled with the words of a natural language. The nodes of the graph are called states. An oriented arc designates transition from one state to another. A state that does not fall any arc is called the initial state. A state in which any arc out is called final state.

A sequence $(e_1, \ldots, e_n)$ of labels is called succession accepted if there is sequence of states $(s_0, \ldots, s_n)$, such that:

- s_0 is the initial state;
- s_n is the final state;
- for each $i \in \{0, \ldots, n-1\}$ there is an arc for the state s_i to state s_{i+1} in which the inscribed entity e_{i+1}.

There may be several initial states and several final states.

A sentence in a natural language is split into tokens, which separate an abstract concept of objects that are particular cases of this concept. For the sentence to be accepted, the sequence of tokens must match the chart a path that begins with an initial state and ends with a final state.

Usage is as follows. It starts from the initial state; being a node, it traverses an arc which leaves that node only if that arc is labeled with a syntactic category that represents the current word in the input sequence. Through an arc, the next word in the sequence becomes the current word. A sentence is syntactically correct if there is a path from the initial node to a final node that can be covered by the phrase.

To extend the generative power transition of diagrams we will replace entities of arcs with their corresponding syntactic categories. Such a Basic Transition Network (BTN) will be obtained. A certain phrase is accepted by BTN if the sequence of words in a sentence corresponds to an accepted sequence of syntactic categories.

We can enter commands to arcs in order to get a BTN:

- JUMP sk – unconditional jump from one state to another;
- WRD w – indicates that the transition that must consume the word w;
- CAT xzy – specific to the fact that it must consume an entity belonging to the syntactic category xyz.

Transition Network is the equivalent deterministic finite automata. For this reason, they can not cover the whole class of context independent languages. To do this, expand the definition of Recursive Transition Networks.

A recursive transition network RTN (Recursive Transition Network) is obtained from a network-based transition BTN if we allow that at least one of its arc to be labeled with the command: PUSH node.

So a Recursive Transition Network can be characterized by the equation:

RTN = BTN + PUSH

Order PUSH node performs the following operations:

- interrupts ongoing operations in;
- saves the current state of the network;
- moves to node **node**; this node is considered as the initial node to another network.

If the sequence of operations starting with node **node** leads to a final state then continues with execution sequence in place to make the jump to node **node**. Things are running as if launching a subprogram.

Recursive Transition Network allows implementation of context-free grammars.

Example generating mechanism phrases:

<proposition>→<elementary proposition>
<proposition>→<proposition><link><proposition>
<elementary proposition>→<verb><article><substantive>
<elementary proposition>→<verb><articulated substantive>

The following is an example of the Transition Diagram, as well as how to obtain a BTN:

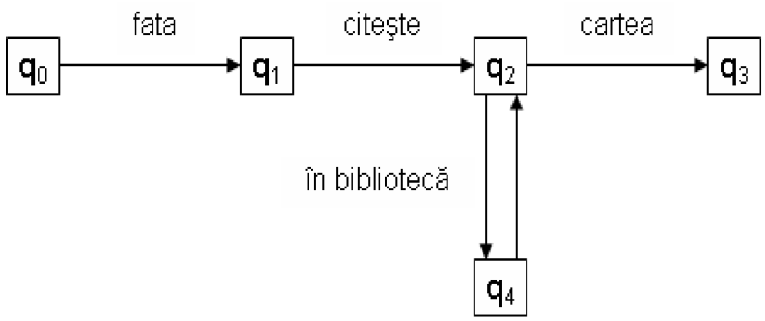

Figure no. 1. Transition Diagram

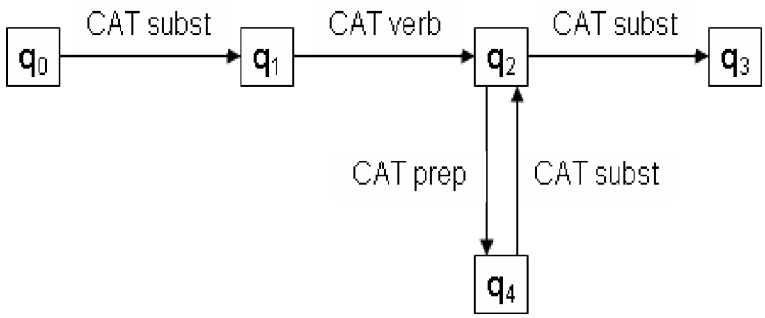

Figure no. 2. Basic Transition Network - BTN

2. **Client-server technology in querying knowledge bases**

A protocol is a convention used to represent data communication between two computers. Because any information sent over the network can be serialized to be sent sequentially, byte by byte, the destination requires establishment of conventions (protocols) to be used as computer sending data, and the one who receives.

Any computer connected to the Internet is uniquely identified by IP address. (Internet Protocol). This IP address is a number represented in 32 bits, abbreviated as 4 octets. Since in a computer there are concurrently several processes that have established network connections, waiting for various information, data sent to a destination must specify the IP address attached to the computer and the information bound. Identification of processes is done through ports. A port is a 16-bit number that uniquely identifies processes

running on a specific computer. Any application that provides a network connection will have to attach a port number to that connection.

A socket is a software abstract term used to represent each of the two ends of a connection between two processes running on a network. Each socket is attached to a port so that it can uniquely identify the program to which the data are intended. A socket is uniquely defined by a triplet: <domain,type,protocol>.

A network application that uses sockets fit in client/server model to design an application. In this model, application consists of two distinct categories of programs called servers respectively clients. Server programs are those that provide various services to potential customers, being able to hold as long as the client does not require any services. Client programs are initiating conversation with a server, requesting some service. A server must be able to handle multiple clients simultaneously, so each request to the server will be treated in a separate thread.

The advantage of the TCP/IP is that ensures a stable communication, permanent network, with the assurance that the information sent by a process will be received correctly and completely at the destination or an exception will be raised otherwise.

3. **Use Java client-server technology**

Java is an object oriented programming language. Java technology is on the one hand a programming language, on the other hand, a flatbed. Java is distributed with libraries implemented for networking. They provide TCP/IP, URL and charging network resources.

There are several types of Java applications:

1. stand-alone applications;
2. applications running on the client (appleturi);
3. applications running on the server (servleturi).

Programming with threads is an important aspect of Java. The ability of a program to execute multiple code sequences simultaneously called multithreading. Such a sequence of code called thread or thread. The possibility to create more threads allows Java program to execute multiple tasks simultaneously (e.g. an image animation, communication with a server).

Threads are executing instructions. Threads can run only within a process. Each thread (thread) has an execution priority. Java allocates to each thread one time quantum of execution, making apparent simultaneous execution of multiple threads.

Any class that describes threads will contain the method run ().

The easiest way to create a thread to do something is by extending the Thread class and supra-definition method run () of it. The general format of such a class is:

```
public class Fir extends Thread {
public Fir (String name) {
super (name);
//I call the superclass constructor Thread
}
public void run () {
//code executed by the thread
}
```

The general structure of a server-based connections is:

1. Creates a ServerSocket object to a specific port
   while (true)
2. Wait for a connection with a client using accepted methods;
   (it will be created a new object Socket type)
3. Treat requests from client:
   (a) Open an input and receive the request
   (b) Open an output stream and send the response
   (c) Close streams and socket newly created

It is recommended that the handling of requests to perform in separate threads to accept method can be recalled as soon as possible to establish a connection with another client.

Method accept () is the ServerSocket class which "listens" to the network. The parent process blocks until a request and returns a new Socket object that will ensure communication with the customer.

The general structure of a clinet-based connections is:

1. Read or declare server IP address and port on which it runs;
2. Create a Socket object with the specified address and port;
3. Communicate with the server:
   (a) Open an output stream and send the application;
   (b) Open an input and receive the answer;
   (c) Close streams and socket set.

For each of the two open sockets can be created two streams byte for reading or writing data. This is done through methods *getInputStream()* and *getOutputStream ()*. The flows obtained are used together with processing flows to provide easy communication between the two processes. Depending on the specific application, they can be pairs:

- *BufferedReader*, *BufferedWriter* and *PrintWriter* – for communication through strings;

- *DataInputStream*, *DataOutputStream* – for data communication primitive;
- *ObjectInputStream*, *ObjectOutputStream* – to communicate through objects.

Structure **JTable** is used to display and edit two-dimensional arrays. Each table uses a table model object's current data management a table. A table model object must implement the TableModel interface. If the programmer does not specify a TableModel, JTable automatically creates an instance of DefaultTableModel object.

JTable Class structure is as follows:

```
java.lang.Object
|__java.aw.Component
   |__java.awt.Container
      |__javax.swing.JComponent
         |__javax.swing.JTable
```

An object is a container part that can contain other components AWT (Abstract Window Toolkit).

Container *setLayout* method object is used to set the component manager for this container.

GridBagLayout class is a manager that aligns components vertically and horizontally, without requiring that all components have the same dimension.

An object *GridBagLayout* places its components as a rectangular array in which a component can occupy one or more cells of the matrix.

The area where all the components are placed is called the display area.

Each component managed by GridBagLayout is associated with an instance of GridBagConstraints object. This object will specify where the component appears in the display area and other useful information.

To use a custom GridBagLayout will have one or more GridBagConstraints objects associated to used components.

Customization is done by setting its variables:

- *GridBagConstraints.gridx*, *GridBagConstraints.gridy* – it specifies the position of the top left cell where the component starts;
- *GridBagConstraints.gridwidth*, *GridBagConstraints.gridheigh* – it specifies the number of cells in one row, a respective column of the area which covers the component;
- *GridBagConstraints.insets* – it specifies the minimum space between the component and the demarcation lines of the display area reserved for component;
- *GridBagConstraints.fill* – it contains information to resize the component described: *GridBagConstraints.NONE*, *GridBagConstraints.HORIZONTAL* (it realizes the extent of the component without affecting height), *GridBagConstraints.VERTICAL* (it

realizes the extent of vertical component without affecting its length), *GridBagConstraints.BOTH* (it fills the entire area reserved);

- *GridBagConstraints.anchor* – it refers to the positioning of the component.

4. **Aplication. Query in natural language knowledge base by client-server technology**

Entities based client-server technology are: client, server and network.

The basic concepts are: host, internet address, port and protocol.

A device connected to the Internet is called the host. Any host in the Internet is identified by IP address. Numeric IP addresses are mapped by name by the system called DNS (Domain Name System).

Ports are abstract logical entities that allow hosts to communicate with each other.

A *protocol* defines how two hosts communicate with each other.

In the client-server technology, the client knows the host running the server and the port on which the server listens for connections from a client.

Furthermore we present an application client addressed a question to the server.

1. Launch the application server (figure no. 3). The server awaits connection of a client.

2. Launch the client application that sends a message to the server, entered from the keyboard (figure no. 4).

The application server receives the message from the client, displays it on the screen and sends it back to the client, which in turn displays this message on the screen (figure no. 7).

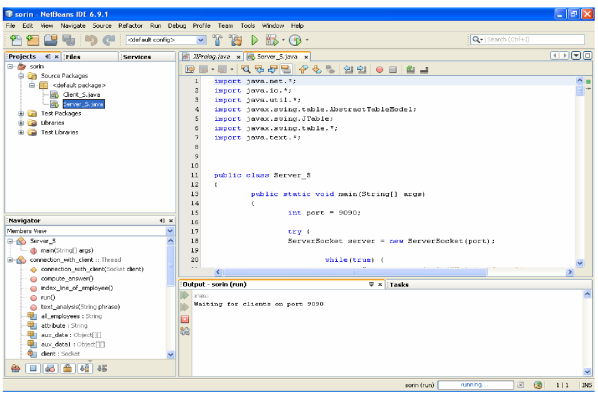

Figure no. 3. Execution of the application server

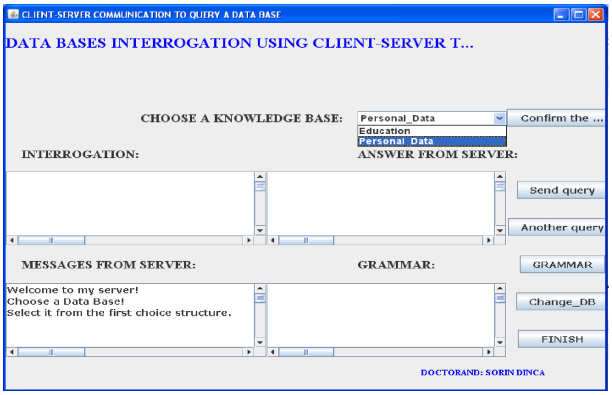

Figure no. 4. Execution of the application client

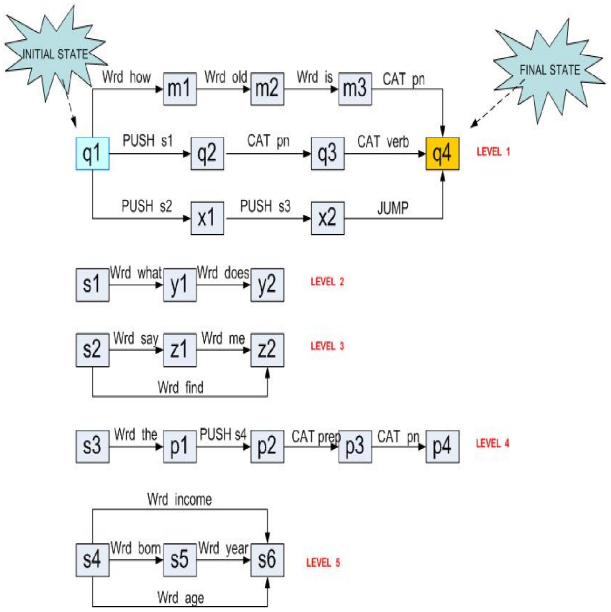

Figure no. 5. Grammar used

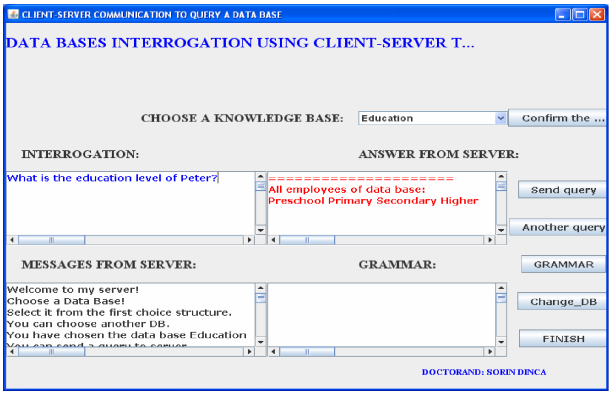

Figure no. 6. Choice knowledge base

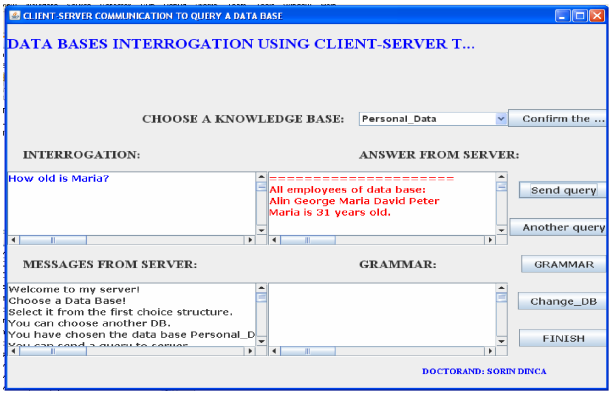

Figure no. 7. Question asked and the answer given by the server client

References

1. Nicolae Ţăndăreanu, *Tehnologii Java în inteligenţa artificială – Note de curs*, Universitatea din Craiova, 2011.
2. Gabriel Stoian, Claudiu Ionuţ Popîrlan, *Tehnologii Java pentru dezvoltarea aplicaţiilor*, Editura Universitaria, 2009.
3. Ştefan Tănasă, Ştefan Andrei, Cristian Olaru, *Java de la zero la expert*, Editura Polirom, 2007.
4. Nicolae Ţăndăreanu, *Baze de cunoştinţe*, Editura Sitech, Craiova, 2004
5. Eugen Petac, Tudor Udrescu, *Programarea calculatoarelor. Aplicaţii Java*, Editura MatrixROM, Bucureşti, 2003.
6. Daniel Danciu, George Mardale, *Arta programării în Java. Elemente suport fundamentale*, Editura Albastră, Cluj-Napoca, 2003.
7. Mircea Florin Vaida, Cosmin Porumb, Radu Vasile Fotea, Florin Ovidiu Hurducaş, Liviu Lazăr, *Java 2 Enterprise Edition (J2EE). Aplicaţii multimedia*, Editura Albastră, Cluj-Napoca, 2003.
8. $***http://en.wikipedia.org/wiki/Type\text{-}token\_distinction$.