Open-Source, Cross-Platform Java Tools Working Together on a Dialogue System

Oana NICOLAE

Faculty of Mathematics and Computer Science,
Department of Computer Science,
University of Craiova, Romania
oananicolae1981@yahoo.com

Abstract. The aim of this paper is to give an answer to the various raised questions about interaction between software technologies which allow adding speech and the capacity to reason to an application.We want to express our point of view concerning the technologies, their behavior and tools needed to build dialogue systems capable to return smart outputs based on reasoning, from an application development perspective.
Keywords: knowledge based system, rule-based system, rules, inference engine, Jess, Sphinx, FreeTTS, speech synthesis, speech recognition, speech understanding.

1 INTRODUCTION

Nowadays, a well designed dialogue system depends on a smart and friendly interface that can overcome the limitations of existing speech technologies: speech recognition, speech understanding and speech synthesis.

These interfaces must not only recognize the spoken words, but also understand the user's query and create a synthetic and accordingly answer.

This is possible because a speech understanding system could render in context the spoken commands, establishing this way, the execution of the actions for suitable inputs.

A knowledge base is also necessary to guide the system in the interpretation of the spoken inputs. The chosen technology to manipulate data is a rule-based one, because rules engines are widely recognized technology component for the applications of knowledge-based techniques.

2 CURRENT INFORMATION OF THE PROBLEM DOMAIN: KNOWLEDGE BASE SYSTEMS

Existing knowledge base systems are used to model complex behavior, through a large number of rules, over a set of data. The structure of a rule-based system contains a fact base called "*working memory of the system*", where the known

facts at different moments are stored, a base of rules which contains the rules used to infer new facts and an inference engine which selects some applicable rule to infer the new facts (*advantage of the Rete algorithm - 1982*).

The rules of the rule base are synthetic structures of the form:

IF
LHS (predicate or premises)
THEN
RHS (conclusions - > functions calling)

The conclusions represent the rule antecedent and they are patterns that are tested for the rule activation. If the conclusion match with the facts base, the rule can fire, and the actions representing the consequent (RHS) of the rule are performed.

The pattern matching process is performed during the inference process, and the variables appearing into the patterns are bounded to some constants from the facts of the fact base.

There are two important types of inference algorithms:

- forward chaining algorithms
- backward chaining algorithms

3 RULE-BASED SYSTEMS vs. RELATIONAL DATABASES SYSTEMS

Knowledge rule-based systems technology represent a combination of a "storehouse of expertise" (*the knowledge base*) and a reasoning mechanism, enhanced with deductive capabilities and inference.

A relational database stores all its data inside tables, and nothing more. All operations on data are done on the tables themselves or produce another tables as result. We never see anything except for tables. Standard relational databases have no deductive capabilities, and the development of a semantic foundation of such databases is practically null.

For our application, we could have designed a classic relational database, defining all the attributes, because we knew them at the time, and built the application on this design.

But we need a system which let us manage what gets stored in the knowledge base and how the application would react to the data going in and coming out of the system, without made changes all the time.

We choose **Jess** (*Java Expert System Shell*) to solve this problem, so we decided to use a rule-based engine.

4 WHY WE CHOOSE JESS?

Jess is an expert system shell and scripting language written entirely in Java language. Jess supports the development of rule-based expert systems which can be tied with code written in the powerful Java language.

Jess was for our application an attractive candidate. It's syntax is a simple one, and for those initiated in the old LISP language or in CLIPS (C Language Integrated Production System) language is relatively easy to integrate and understand Jess language. Jess can process a great number of rules quickly and is a software stable product. It is small, light and one of the fastest rule engines available.

Jess has come with all the portability and security advantages of Java along with the ability to write Java applications or applets. However, this also introduces Java's slower execution speed, but this is more than compensated by the fast Rete algorithm it employs.

Jess is a runtime engine which supports forward and limited backward-chaining. As the most rules engines, rules are specified as declarative patterns, not procedural code. As we know, the knowledge representation for an expert system means facts and rules.

For Jess, facts are primary means of passing data. We can pass a data (or a value) to Jess for some computation, but this thing it is not proper for Jess's type of action. Instead, simply assert a fact with that data and have a rule that fires when that data appears. The presence or modification of facts can cause rules to fire, which can then change other facts causing more rules to fire, until no further rules are activated (no patterns fully match)([1],[2],[3],[4]).

For the application we choose a mix between Jess language and Java language by embedding Jess library into Java, although Jess offers possibilities to use pure Jess language scripts or pure Java code.

Jess can also be used in a multithreaded environment. Each individual jess.Rete object represents an independent reasoning engine, so a single program can then include several independent engines.

5 THE APPLICATION ARCHITECTURE

Our application,VoiceMyth, is in fact a spoken dialogue system.Spoken dialogue systems are one of the main applications of speech and language processing. In spoken dialogue systems all the components of a speech and language processing systems are integrated. Traditionally, into a dialogue system, natural language understanding is integrated with a speech recognizer and a speech synthesizer.

On this context, the world purpose, is to develop spoken dialogue systems with a natural and flexible dialog flow based on interfaces by voice, in which the user has no restrictions in communicate his commands. The system should interpret each utterance and then find the intention of the user.

Our application, VoiceMyth is more a *system driven* type of dialogue system, because the dialogue flow is restricted to some fixed words or phrases.

The main modules of the application are: a speech recognizer, a speech understanding module, a text-to-speech tool, a dialog manager and an inference engine to manipulate the knowledge base.

The input data represents an acoustic sequence spoken by user at microphone. The recognizer module processes the acoustic data and passes a word graph to the language understanding module.

The role of the speech understanding module is to analyze the user query and to produce a representation of its semantic content that allows the dialogue manager to take a decision about the dialogue orientation taking into account the dialogue context.

The dialogue manager integrates the understood sentence or word into the system belief and decides on the necessary system actions. In the end, it passes the text string for text-to-speech module : the answer of the application. A knowledge base containing task-specific information may be accessed by the language understanding module and the dialogue manager.

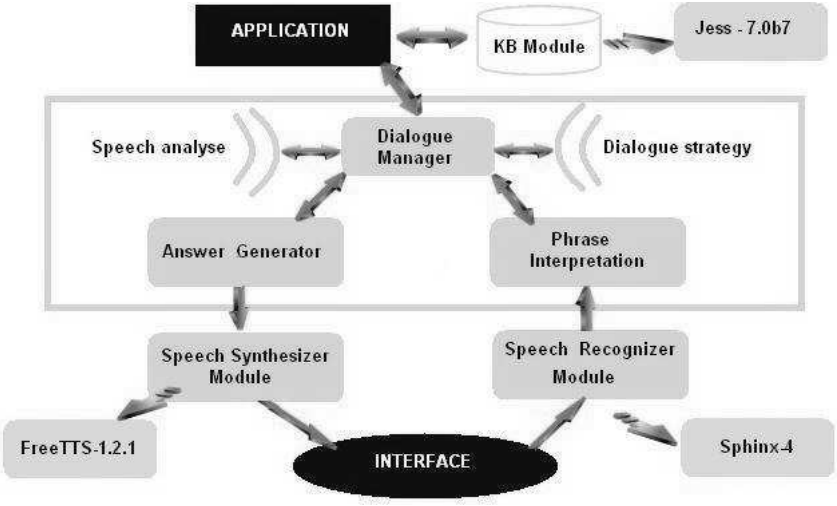

Fig. 1. The architecture of the VoiceMyth application

5.1 SPEECH RECOGNIZER MODULE

WHAT IS SPHINX? Sphinx is a flexible and modular system developed entirely on the Java platform, highly portable and easy to use with multithread. In our application we used the Sphinx-4 version of this open-source, Java tool.

USING SPHINX-4 ON VoiceMyth APPLICATION. The beauty of the Sphinx-4 system is its architectural design. Other open-source recognizers

are available, but Sphinx-4 has a nicely documented modular object-oriented framework that allows plug-in classes. ([1], [6])

In our application, the speech recognizer module has its own thread, that makes it an individual module and allow us to manipulate it easily, without any interfering. Its purpose is to acquire the acoustic input generated by speaker and to give it the specific parameters at the front-end module, which communicates the derived feature to the decoding block. The decoding block contains three components: the search manager, the linguist and the acoustic scorer, which work in tandem to perform the decoding.

It is well known that in the translation from sound to graph's words, some mechanism that affect the message may appear: the psychology of the speaker, semantics, rules of discourse, syntax, lexicon, prosodic system, phonemic system, ambient environmental noise and the microphone used.

The obtained sequence, supposed to be correct is then assumed by the speech understanding module.

When we run the application, we must specify to the JVM that the recognition module use an additional amount of memory for the recognizer's vocabulary and library loading, so we must add a parameter specifying the additional virtual memory used:(e.g.-mx256m).

The configuration of a particular Sphinx-4 system is determined by a configuration file in XML format. The configuration file for our application is available in the file *voicemyth.config.xml* and it determines which components are to be used in the system and establish detailed configuration of each of these components (e.g. the grammar configuration: where we must specify the grammar created by us in JSGF (*Java Speech Grammar Format*) format, where we define some words or phrase patterns for the recognized acoustic inputs).

5.2 SPEECH SYNTHESIZER MODULE

WHAT IS FreeTTS ? FreeTTS is a freely, available text-to-speech synthesizer written entirely in Java programming language and it is based on a small run-time speech engine developed at Carnegie Mellon University, named Flite1.1.

FreeTTS includes an engine for the vocal synthesis that supports a certain number of voices (male and female) at different frequencies. We used in our application the recent version of this engine: FreeTTS 1.2.1.

USING FreeTTS ON VoiceMyth APPLICATION. FreeTTS is a great toolkit for all Java programmers: to build their own applications and to contribute to the open-source community. FreeTTS provides partial support for JSAPI (*Java Speech API*), only a subset of JSAPI 1.0 javax.speech.synthesis specification. It is recommended to use JSAPI to interface with FreeTTS because JSAPI interface provides the best methods of controlling and using FreeTTS. ([1], [5])

The text-to-speech module creates the speech output for our application, being under the control of the dialogue manager. This one provides some templates for the answers of the application. These templates are filled in with values from the current system belief and the resulting text is passed to the text-to-speech module: the FreeTTS speech engine.

The speech synthesizer module has, in our application, its own thread. The resources for the speech engine are allocated and the engine moves into the resume state, each time we want an additionally speech output, besides the standard, written one. The application output represents the answer of the inference engine to the user spoken query. This answer is then mapped by the dialogue manager, into a proper form to display. This form represents the input for the text-to-speech engine.

The process of transforming text into speech contains two phases: first the text goes through analysis and then the resulting information is used to generate the synthetic speech signal. There are some steps in the complex process of post analysis. The text pre-processing step analyze the text for the special inputs of the English language: abbreviations, acronyms, dates, times, numbers, currency amounts, email addresses.Then next steps are: text-to-phoneme conversion, prosody analysis and the waveform production.

FreeTTS engine enable full control about the speech signal. The application VoiceMyth provides the possibility to choose a voice between three types of voices: a 8 khz, diphone male English voice named *kevin*, a 16 khz diphone male English voice named *kevin16* and a16khz limited domain, male US English voice named *alan*. The user could also set the properties of a chosen voice: the speaking rate, the volume and the pitch. FreeTTS engine also has it's limits as it ignores JSML Speech Markup. FreeTTS JSAPI will process JSML, but currently does not apply the markup to the generated speech. The speech signal is an artificial-sounding voice that is highly understandable.

5.3 HOW WORKS THE INFERENCE ENGINE MODULE?

As we said before, the inference engine of our application is the Jess's inference engine itself. The Jess's environment provides us a knowledge base where we could specify the templates we need, the facts, the rules and the queries.

The knowledge base is in fact a clp file which is loaded by the Jess's engine. The dialog manager and the speech understanding module model the user query so it could be passed to the inference engine, into a proper, understandable form. A fact containing the necessary data is passed into the knowledge base, and some rule responsible with this kind of facts fires, giving to the application the expecting answer. This answer is than take over by the dialog manager and the speech understanding module and mapped into a proper, particular form for the text-to-speech module.

5.4 THE DIALOGUE MANAGER MODULE

The dialogue manager is like a long-time goal of our application. Its role in developing spoken dialogue systems is to provide a natural and flexible dialogue flow which is adaptive to the user's spoken phrases or words. It is better known that for the recognizer module representing in our application by Sphinx-4, the long phrases are spoken by the user, the faster and correct recognition process is done.

The dialogue manager has to monitor the dialogue flow, collect the information given by the user, interact with the application background, and to decide whether and which further information is required or if an action has to be performed. From an utterance like *Show the winners from Japain* the system is able to assign the values country to Japain, and the value *Shizuka Arakawa* to the slot *namecompetitor* from knowledge base.

If there is only a single database entry with this name, no further information is required and the requested information is retourned to the user. Otherwise, the given information is not sufficient to meet the dialog goal, in which case the dialog manager tries to fill more slots with values in order to refine the knowledge base quest.

5.5 THE SPEECH UNDERSTANDING MODULE AND ITS FUNCTIONALITY

The speech recognizer module passes the word graph to the natural speech understanding module. The language understanding module has two task:

- to compute the meaning of the words or phrase
- to find and score the most probable path through the word graph

The user of our application always has in mind a particular purpose : he speaks some words or a phrase and he aspects the application should faster answer back a specific, correct information.Usually, the output of our application provides two formats of displaying the results: a persistent written one, and an additionally spoken one. In every spoken input, the user should insert some information items, that help the application to determine the dialogue goal and the current dialogue state. (e.g. the spoken utterance : "show the nagano podium skating figure" has the information items the list of this words:

- nagano, the host town of the Olympic Games
- podium, the information to display about this Olympic Games. In this case, all the four specific probes of the figure skating have a particular information to display: the three medals (name of the medal's winner, specific country, type of the medal) not necessarily in this order.

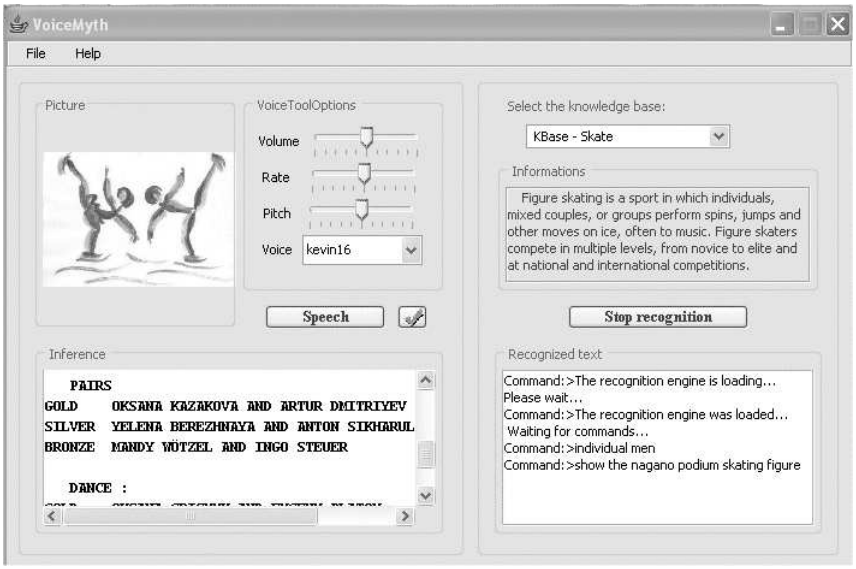

Fig. 2. The spoken command: "show the nagano podium skating figure"

Those information items are then transform in a particular query for the Jess's knowledge base. The dialogue state includes the system's current query and its current belief on what has already been stated by the user.

The core of our speech understanding module has a sub module for parsing the speech input and extract the meaningful words. One important point in our application is that the speech understanding module contain application specific knowledge information, that increase the speech understanding module performance. A different ideea to improve the quality is that our application reject an utterance if the word sequence of information items is incorrect.

6 FUTURE WORKS

In the light of a dialogue system implementation, the following steps concern the improvements that should be done on our application.

- One important goal of our application is to be able to extend an existing domain of the knowledge base (e.g. figure skating - Olympic Games results) or to add a new one, with as little manual work as possible. This involves new methods to populate the knowledge base (the *deffacts data* structure of the .clp file). Using spoken commands to populate the knowledge base is a challenging goal.
- Future improvements will also include some work on the dialogue manager, so this one will change the recognition lexicon and JSGF grammar depending on the dialogue flow.

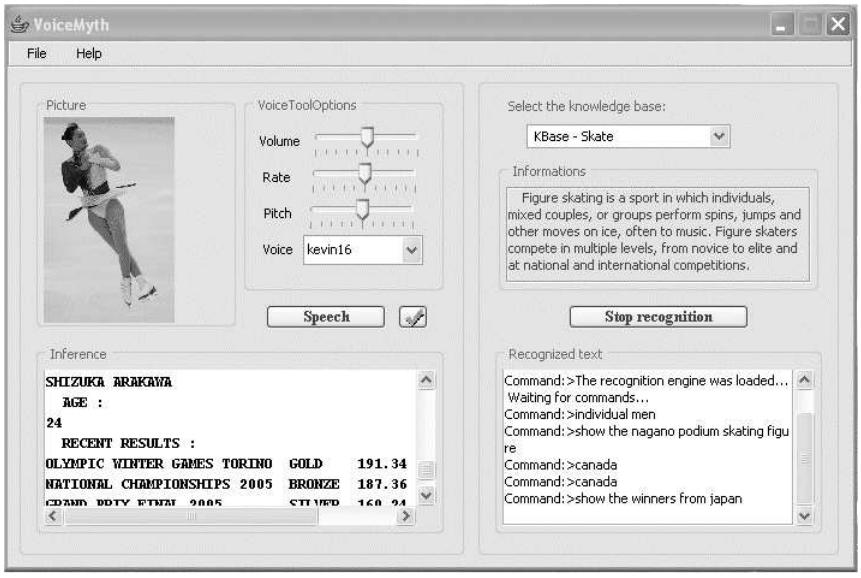

Fig. 3. The spoken command: "show the winners from Japain"

– Speech understanding technology module of our application requires some explicit semantic analysis and interpretation. Theoretically, speech understanding technology is substantially more complex and problematic issue, but practical procedures for improvements of speech understanding technologies, are not yet well established. Our future work on this feature will try to accomplish this task.

References

[1] **N. Ţăndăreanu**, Digital Signal Processing - Lecture Notes, 2005, University of Craiova
[2] **E. J. FriedmanHill. Jess**, The Rule Engine for Java Platform, http://herzberg.ca.sandia.gov/jess/docs/70/.Version 7.0b7
[3] **J. Morris**, Jess and the Art of Rule-Based Computing, http://www.jessrules.com/zen.shtml
[4] **E.J.Friedman-Hill**, Jess in Action, Manning Publications, 2003
[5] FreeTTS - A Speech synthesizer written entirely in Java language programming, http://freetts.sourceforge.net/docs
[6] The Recognition Engine : Sphinx-4, http://cmusphinx.sourceforge.net/sphinx-4