FileKommander – NLP Project Report

# Description

FileKommander is a GUI natural language application. It implements a simple file manager which analyses user input using a Conditional Pipeline to run simple file and folder tasks on the user operating system

It uses GATE to generate a set of annotations which are then used in Java to try to understand what the user wants and finally run the action asked for by a user .

# Extracting Information with GATE

We use a conditional pipeline to extract information from user input

The pipeline consists of

1. Document Reset PR
2. ANNIE Sentence Splitter
3. ANNIE English Tokenizer
4. POS tagger
5. ANNIE Gazetteer with a custom def file
6. NRTransducer which runs multiple phases of JAPE rules

## Gazetteer and JAPE Rules

All user input is split into two main categories

1. Actions – Create , Delete , Rename , Exists , Statistics , Insert, Replace, Remove . Every action has its own list file with multiple synonyms for each action
2. Objects – Files , Folders and Phrases
   1. Multiple JAPE rules are used to find file extentions and file names
   2. With POS tagging we try to find references to directories , files and phrases
   3. Objects in double quotes are considered to be Directories and Phrases depending on their use in a sentence , unless they have an extention which makes them filenames

## Annotations created

* All – One annotation for each sentence which is a collection of all annotations found in that sentence
  1. Actions – A set of all actions in that sentence with the MinorType and string stating which action it is
  2. File name – A set of all Filenames found in the sentence along with the file name its self
  3. Directories - A set of all Folder names found in the sentence along with the file name its self
  4. Phrases – A set of all phrases referred to in the sentence
  5. Position – Set of all annotations with the occurrence of words like before, after, start, end etc.
  6. Repetition - Set of all annotations with the occurrence of words like every, first, last.
  7. Content – Annotation containing the user sentence.

## Java System Implementation

* On running the main program, the system goes thru the following steps
  1. System initializes gate and creates the pipeline.
  2. ANNIE is initialized
  3. System then starts a new thread to listen to user input
  4. Swing GUI to take user input thru keyboard and microphone
  5. On User Input and submit , annotations are generated
  6. For each sentence in input , the sentence is analyzed to get the action and objects and an appropriate ActionModel is created
  7. The ActionModel is then processed to run the specific action which is then displayed on GUI
* For each action , there is a separate panel and action model created . Analysing the action sets information in the model which is then used when the action is run and the model is displayed

# Features Implemented

* Create – Single or multiple files and folders
* Delete – Delete single or multiple files and folders
* Rename – Rename a file/folder with new name
* Statics – Get information about files or folders
* Exists - Does a particular file or folder exist?
* Last Modified – Display the last modified date of a particular file or folder
* List files in a folder
* Count of words in a file
* Count of files in a folder
* Open – Open a file with system editor. OS independent (works on Win/Mac/\*nix) systems
* Insert/Remove/Replace phrases in a file at a particular location
* Errors during processing are displayed on screen or in tooltips
* Some actions display to the user what action will be performed so that the user can review the action before running it.

# Unimplemented Features

* Operations not implemented
  + Close file – We wanted the actions to be independent of each other . In Java a file(Process) can be closed only if it was opened in the same JVM . If this feature was implemented we would have to remember all files opened in FileKommander which would make the actions dependent on each other
* Regular Expression for insert , replace and delete operations
* Speech recognition is dependent on Google Chrome Speech API and is not usable for file names and special characters.
* Ability to modify options dynamically during runtime

## Speech Recognition

* We have used a Java-Speech-API library for speech recognition.
* It uses the speech engines created by Google to provide functionality for parts of the API.
* Speech captured by system is encoded to FLAC and sent as a post request to Google which replies with a JSON object giving possible text and confidence
* It uses a recognizer, synthesizer and microphone capture utility.
* It makes use of Google services for recognizer and synthesizer.

# Features to be Implemented

* In the future, this application can be more interactive with the user.
* Implement speech recognition with Sphinx
* To change the properties of the application, e.g. the working directory, GATE path, etc. at runtime.
* Show a progress bar while the operations are being done.
* Make code more modular so that users can add more actions and features
* Use regex where ever possible instead of strings
* Ability to use wild card characters for file and folder names
* Indexing all files with Apache Lucene to make search , insert and replace faster

# References

* <http://gate.ac.uk/>
* <https://github.com/The-Shadow/java-speech-api>
* <http://commons.apache.org/io/>
* <http://commons.apache.org/lang/>
* <http://logging.apache.org/log4j/1.2/>
* Source Code
  + - <http://goo.gl/JBYog>