**BatterBot Version 2.0 Documentation**

Bryce Paterson

13212071

**1.0 Goal**

The purpose of this code is to provide a basic chatter bot engine. The chat agent provides a conversation with Batman. In addition to asking him questions about his crime fighting career, you can also ask him some more general, and personal questions.

**2.0 Features**

The following are the features of this project, how they interact, and what purpose each serves.

**2.1 Simple System Overview:**

We begin by examining the flow of data by defining five main components in our system. We have the Language Processor, the Response Selector, the Response builder, and the IO controller. Another critical component of the system is the Response Template class, which we discuss more in the next section. Using these components, we can construct the basic cycle observed when the user enters an input string.

The program starts with the GUI reading the user input. This input String is the given to the Language Processor, which extracts from the input all of the recognized keywords. These matched keywords are then passed to the Response Selector. The Response Selector ranks all of the relevant response templates, selecting the one with the most matched keywords. This Response Template is then passed to the Response Builder. Each Response Template contains a String with some blanks and possible variable substitutions. The Response Builder then fills in these blanks with the variable substitutions, and hands the complete string to the GUI to be printed. The cycle then repeats.

**2.2 BatterBotDriver:**

We use a driver class BatterBotDriver which contains all of the components needed for the Chatter Bot. This class also stores the main function cycle() which iterates through the user input and chatter bot response cycle. Additionally, there is a method setup(), which can be used to load our sample Batter Bot Response Template files.

**2.3 IO Console:**

The current IO class is very simple, and is simply used to read user input from the eclipse console, and then print the chatter bot responses.

**2.4 Language Processor:**

Our implementation of the Language Processor is relatively simple. The Language Processor takes in the user input as a String. The Language Processor then loops through a list of all of the known keywords, and searches through the user input looking for each one. Every keyword that is found is then stored in a list, which is passed to the Response Selector.

Two external language toolkits were added to the language processor. Both libraries are from the Stanford Natural Language Processor. The first library is the Parts Of Speech (POS) tagger. The second library added is the Named Entity Recognition (NER) classifier.

The Language Processor automatically pulls out any proper nouns, base verbs, locations, or organizations in the input, and commits them to memory. This allows for more interesting responses to be constructed.

**2.5 POS Tagger:**

The POS tagging library for the Stanford Toolkit is a library that allows tagging each word in a given string. The classifications include various verb classifications, nouns, proper nouns, prepositions....etc. The output from the tagger is a string with "\_" followed by the words type appended to each word. EX: "Bryce"-->"Bryce\_NNP", "\_NNP" being the code for a proper noun.

**2.6 NER Classifier:**

The NER classifying library is a library that allows certain words, particularly nouns, to be classified from a given string. Some classifications include locations, and organizations. The output from the classifier is the input string with "/" followed by the location appended to words of interest. EX: "Kelowna"-->"Kelowna/LOCATION", "/LOCATION" being the code for a location.

**2.7 Response Selector:**

The Response Selector contains a reference to a Hash Table storing pointers to all of the Response Templates. The keys for the hash table are the keywords, and the values are then the pointers to the Response Templates. This allows for quickly finding all of the relevant Response Tables based on a set of found keywords.

**2.8 Response Builder:**

Given a Response Template as input, the Response Builder fills in the canned sentence with its variable substitutions and returns the response as a string. A variable substitution may optionally come from the memTable, which contains previous user input.

**2.9 MemTable:**

This is a dictionary data structure used to store previous user input. For example, the key may be “Name” and the value “John Doe”. When the question “What is your name?” Is asked, the response would then be parsed and stored in the memTable.

**2.10 Response Template:**

The Response Template is a class used to store the ‘canned’ sentences and their variable substitutions. Each Response Template will contain:

- An output sentence with possible blanks to be filled in

- The Array of buckets, which contain the variable substitutions

- A list of key words that will trigger that response to be selected

Additionally, a Response Template may contain a scaleRules function definition, which allows for more complicated selection rules to be defined. This function should return a value that will multiply the normal ranking determined by the number of relevant keywords. For example, supposing that the given response should only be said once, the function scaleRules could return 1 if the response has not yet been used and a value of 0 if the response already has been used. Alternatively, this function could be used to increase the weight of the responses ranking by using a number larger than 1. This should be used with caution.

If a Response Template is asking a question of the user, for example, “What is your phone number?” then the response Template must also define and additional step for interpreting the following user input. In our example, this would mean extracting the phone number from the response. The Response Template then also needs to have a memTable key, called memEntry, which is the key that will be used to store that user response in the memTable.

**2.11 KeyWordList:**

We have created our own class, the KeyWordList, which is used to store a list of keywords. This class is very simple, and was designed to facilitate the integration of new features in the future. Currently it simply wraps an array of Strings.

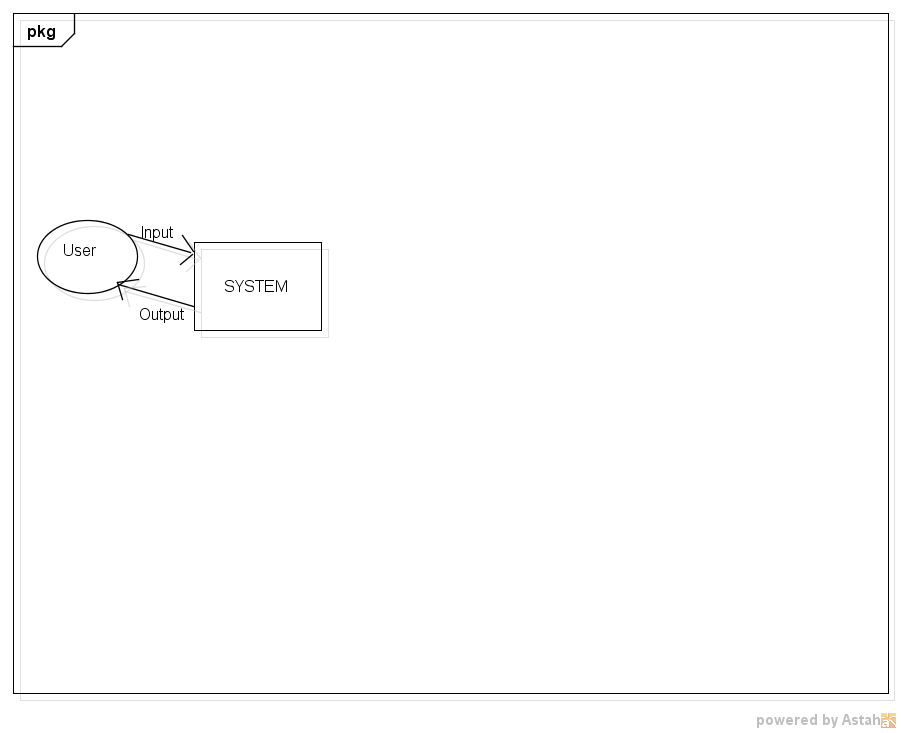
**2.12 Batman Convo:**

This is the GUI class. Running this class results in the GUI version of the program. The GUI was constructed using the Eclipse plugin Window Builder. The theme of the GUI is the traditional Batman colours: black, grey, and yellow.

**3.0 Data Flow Diagrams**

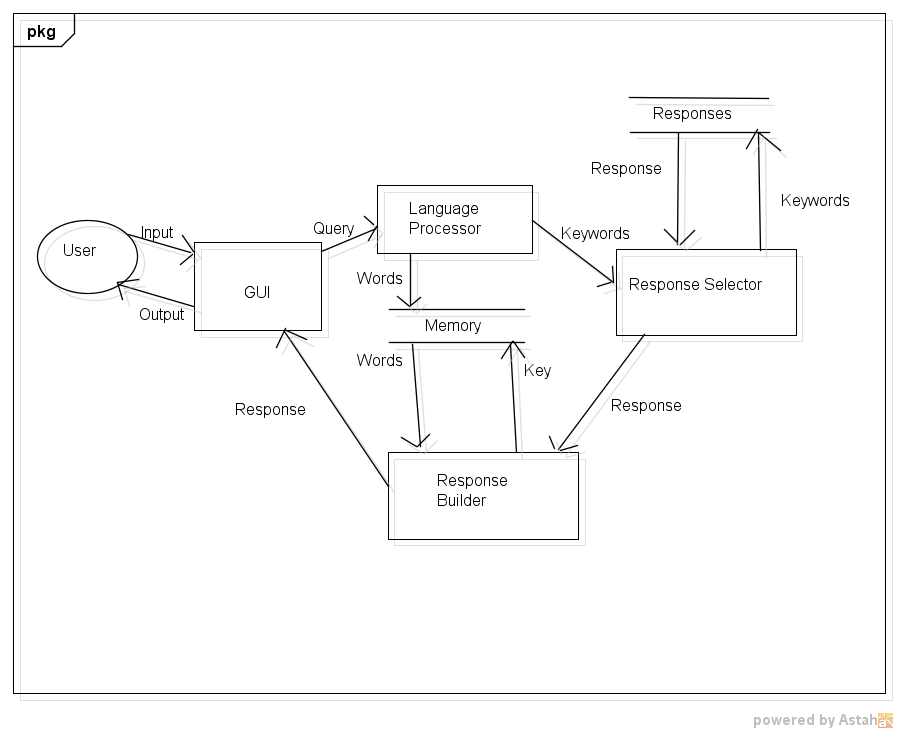
**3.1 Level 0 Data Flow Diagram**

The level-0 DFD is very simple. The user only needs to type an input string and the system returns an output string as Batman’s response.

****

**3.2 Level 1 Data Flow Diagram**

An in depth description of each process and their interactions can be seen in Section 2.

****

**4.0 Sample Output**

**5.0 API**

Here are some features of the BatterBot that could be extracted and used as an API:

1. The language processor could be ported and used as an API. It makes use of the POS and NER toolkits.

2. The Batterbot GUI could easily be ported as an API.

3. The memory system/response builder of the Batterbot could be used as an API. It allows construction of a variable response, taking either hardcoded words, or words from memory, and inserting them into a base string.