**Ontology Module**

After extracting the relevant information from the tweets based on their respective categories, they are now stored to an ontology that contains object relations between the different extracted information. The actual structure of the ontology was made using an external tool called Protegé that makes use of the OWL API. This module takes an instance or a list of instance of categorized tweet classes. The categorized tweet classes include the following: the CallForHelpTweet class for containing the information that were gathered under the Call For Help category; CasualtiesAndDamageTweet class for containing the information that were gathered under the Casualties and Damage category; CautionAndAdviceTweet class for containing the information that were gathered under the Caution and Advice category; lastly, DonationTweet class for containing the information that were gathered under the Casualties and Damage category. This module has two sub-parts that are both responsible for storing and accessing information in the ontology. The OntologyModule class is responsible for storing the extracted information to the ontology and the OntologyRetriever class is responsible for retrieving the information that was stored in the ontology.

**OntologyModule: Implementation**

The OntologyModule class is the main class responsible for working around the storage and verification of the extracted information in the ontology that was designed for the use of the system. It has respective functions for the different pre-requisite steps that shall be taken before actually accessing and modifying the contents of the ontology. There are respective functions for loading, saving and removing the ontology from its manager. Also, general-purpose functions were included to streamline the process of verifying the information to the ontology. These functions include a categorized tweet information viewer and a data property value viewer.

To store information into the ontology, certain classes have to be initialized so that they can be manipulated within the module. A sample code listing below shows how necessary classes should be initialized.

// Classes for containing the extracted information per category

CallForHelpTweet oCH = new CallForHelpTweet();

CasualtiesAndDamageTweet oCD = new CasualtiesAndDamageTweet();

CautionAndAdviceTweet oCA = new CautionAndAdviceTweet();

DonationTweet oD = new DonationTweet();

// Class for actually initializing the module

OntologyModule oModule = new OntologyModule();

After initializing the necessary classes, the extracted information will now be stored into the new initialized classes based on its given category. For instance, let’s take the information that was extracted from a tweet that was categorized to be a Caution and Advice tweet. A sample code listing below shows how to temporary store the extracted information into its respective categorized tweet class.

// SAMPLE INITIALIZATION FOR CAUTION AND ADVICE REPORTS

oCA.setTweetHandle("theonlykyleeeee");

oCA.setTweetContent(":(( RT WARNING! Baha sa Guadalupe!");

oCA.setTweetGeoLocation("10.00000121, 145.345300023");

oCA.setLocationInTweet("Guadalupe");

oCA.setTweetTimestamp("12/27/2014:00:13:67:40");

oCA.setTweetDate("December 27, 2014");

oCA.setTweetAdvice("WARNING! Baha sa Guadalupe!");

After temporarily storing the extracted information, actual storage of the information to the ontology now follows. Storing the extracted information is a fairly simple process; as it would only require calling one function and requires only one input. The catch, though, is that there are different methods for certain categories of tweet. Also, if you would want to verify if the storage process has been successful, you could do a rough view of the contents of the ontology just by calling a simple view method. A sample code listing below shows how to permanently store the extracted information to the ontology (the one written in bold characters) and how to do a rough view of the ontology contents (the one written in italicized characters).

try {

oModule.loadOntology();

**oModule.addCautionAndAdviceReport(oCA);**

**oModule.addCasualtiesAndDamageReport(oCD);**

**oModule.addDonationReport(oD);**

**oModule.addCallForHelpReport(oCH);**

*oModule.displayStoredTweets();*

oModule.removeOntologyFromManager();

} catch (OWLOntologyCreationException e) {

e.printStackTrace();

}

**OntologyModule: Issues**

In developing the OntologyModule class, we had some issues with its actual implementation because of a number of reasons. First, the documentation that came with the API was not that comprehensive and streamlined in a way that there were actually no complete descriptions about the different methods that can be utilized; though, there were code samples that were a bit confusing. With this type of documentation, we had difficulty customizing the implementation of the different methods within the API to suit the needs and requirements of our system. Also, we had difficulty in debugging/testing the module because we have no complete reference for the actual functions of each method including its parameters and its outputs. Second, we have difficulty in manipulating and modifying the actual ontology because we have to work with two different tools to assure that changes are properly reflected. If there changes to the ontology, we have to work first with the Protegé to fix the structure of the ontology and the different dependencies that might be affected upon performing the changes and then, we then have to work with our preferred code editor to reflect the changes in the ontology to the Java code that we’ve made for it so that there will be seamless interaction between our system and the ontology behind it.

**OntologyRetriever: Implementation**

The OntologyRetriever class is the main class responsible for working around the retrieval of the extracted information in the ontology that was designed for the use of the system. It has respective functions for the different pre-requisite steps that shall be taken before actually accessing and modifying the contents of the ontology. There are respective functions for loading and removing the ontology from its manager. Also, general-purpose functions were included to streamline the process of retrieving the information to the ontology. These functions include a categorized tweet information retriever and a data property value retriever. Also, category-specific functions were included to properly organize the information that was retrieved. These functions include a constructor method for retrieved Caution and Advice, Casualties and Damage, Call For Help, and Donation tweets, which just make each of the set of extracted information an instance of each of the classes representing the mentioned categories.

To retrieve the information from the ontology, one class has to be initialized so that other modules can manipulate it. A sample code listing below shows how the necessary class should be initialized.

OntologyRetriever or = new OntologyRetriever();

After initializing the needed class, actual retrieval of the information from the ontology now follows. Retrieving the information stored in the ontology is a fairly simple process, as it would only require calling one function and requires no input. All you need to secure is a variable of the type *RetrievedTweet*to contain the information that will be returned by the retrieval function of the module. The variable should be of type *RetrievedTweet* because the retrieval function, logically, returns four ArrayLists for the different categories of the information. A sample code listing below shows a snippet of the structure of the class *RetrievedTweet* and following this code listing is another code listing showing how to retrieve information from the ontology.

public class RetrievedTweet {

ArrayList<CallForHelpTweet> retrievedCFHTweets;

ArrayList<CasualtiesAndDamageTweet> retrievedCADTweets;

ArrayList<CautionAndAdviceTweet> retrievedCATweets;

ArrayList<DonationTweet> retrievedDTweets;

......

}

try {

or.loadOntology();

RetrievedTweet rt = or.getStoredTweets();

or.removeOntologyFromManager();

} catch (Exception e) {

e.printStackTrace();

}

**OntologyRetriever: Issues**

In developing the OntologyRetriever class, we had almost the same as the ones mentioned in the issues encountered in the OntologyModule. First, the documentation that came with the API was not that comprehensive and streamlined. With this type of documentation, we had difficulty customizing the implementation of the different methods that are concerned with ontology retrieval within the API to suit the needs and requirements of our system’s retrieval method. Also, we had difficulty in debugging/testing the module because we have no complete reference for the actual outputs of each method including the effects of combining different methods to achieve a certain output from the ontology. Second, we had difficulty in structuring and implementing the actual retrieval method code because we have to look at the initial contents of the ontology and its actual relations and structures to be able translate the logical connections and relations between the instances stored in the ontology to a physical working code that will facilitate the exchange of information between the ontology and system. We have to be able to remember the specific the relational connections between the instances to and “simulate” them in order for us to be able to reverse the process of adding/storing to the ontology; thus, essentially, enabling us to do a retrieval process. Also, with this, we essentially had the same difficulty of working with two separate tools.