The actual storing of information involves more sub modules or methods which helps streamlined the process. The process starts with the call to the respective method for storing the information extracted from its tagged category. Inside this method, there are certain variables that should be initialized so that they can be interfaced with the actual ontology. A sample code listing below shows the different variables to be initialized – take the ***addCautionAndAdviceReport*** for example.

// PREPARE THE TWEET INDIVIDUAL

OWLIndividual tweetIndividual = addTweetInformation(ca);

// PREPARE THE REASONER AND PREREQUISITE STUFF

OWLDataFactory dataFactory = manager.getOWLDataFactory();

// INSTANTIATE CLASSES NEEDED FOR THIS OPERATION

OWLClass cautionAdviceClass = dataFactory.getOWLClass(IRI.create(CAUTION\_ADVICE\_CLASS\_IRI));

OWLClass adviceClass = dataFactory.getOWLClass(IRI.create(ADVICE\_CLASS\_IRI));

// CREATE THE NEEDED INDIVIDUALS FOR THIS OPERATION

OWLIndividual caIndividual = dataFactory.getOWLNamedIndividual(IRI.create(BASE\_IRI + "\_CA-I\_" + ca.getTweetAdvice().replace(" ", "\_")));

OWLIndividual adviceIndividual = dataFactory.getOWLNamedIndividual(IRI.create(BASE\_IRI + "\_A-I\_" + ca.getTweetAdvice().replace(" ", "\_")));

After initializing the different needed variables, the next step is to prepare the Tweet individual that will serve as the root individual for storing all the extracted information into the ontology. To do so, call the ***addTweetInformation*** method with an instance of the ***CautionAndAdviceTweet*** class as a parameter for this method. This method will return an ***OWLIndividual*** that will be used as a prerequisite for completing the storage process. Furthermore, this method prepares the Tweet individual to include other information that was extracted including the Tweet Location and the Tweet Timestamp. A sample code listing below shows how the ***addTweetInformation*** method works.

// PREPARE THE REASONER AND PREREQUISITE STUFF

OWLDataFactory dataFactory = manager.getOWLDataFactory();

// INSTANTIATE CLASSES NEEDED FOR THIS OPERATION

OWLClass tweetClass = dataFactory.getOWLClass(IRI.create(TWEET\_CLASS\_IRI));

OWLClass locationClass = dataFactory.getOWLClass(IRI.create(LOCATION\_CLASS\_IRI));

OWLClass timestampClass = dataFactory.getOWLClass(IRI.create(TIMESTAMP\_CLASS\_IRI));

Code Listing: Prerequisite Class Instantiations

// CREATE THE NEEDED INDIVIDUALS FOR THIS OPERATION

OWLIndividual tweetIndividual = dataFactory.getOWLNamedIndividual(IRI.create(BASE\_IRI + "\_TI\_" + t.getTweetContent().replace(" ", "\_")));

OWLIndividual locationIndividual = dataFactory.getOWLNamedIndividual(IRI.create(BASE\_IRI + "\_LI\_" + t.getTweetContent().replace(" ", "\_")));

OWLIndividual timestampIndividual = dataFactory.getOWLNamedIndividual(IRI.create(BASE\_IRI + "\_TSI\_" + t.getTweetContent().replace(" ", "\_")));

Code Listing: Ontology Individual Creations

Set<OWLClassAssertionAxiom> classAssertion = new HashSet<OWLClassAssertionAxiom>();

....

OWLClassAssertionAxiom lAssertion = dataFactory.getOWLClassAssertionAxiom(locationClass, locationIndividual);

classAssertion.add(lAssertion);

....

manager.addAxioms(filietOntology, classAssertion);

Code Listing: Class-Individual Assertions

Set<OWLDataPropertyAssertionAxiom> dataAssertion = new HashSet<OWLDataPropertyAssertionAxiom>();

// These are the data properties for the Tweet Individual

OWLDataProperty tweetHandle = dataFactory.getOWLDataProperty(IRI.create(BASE\_IRI + "tweetHandle"));

OWLDataPropertyAssertionAxiom tweetHandleDataPropertyAssertion = dataFactory.getOWLDataPropertyAssertionAxiom(tweetHandle, tweetIndividual, t.getTweetHandle());

dataAssertion.add(tweetHandleDataPropertyAssertion);

OWLDataProperty tweetContent = dataFactory.getOWLDataProperty(IRI.create(BASE\_IRI + "tweetContent"));

OWLDataPropertyAssertionAxiom tweetContentDataPropertyAssertion = dataFactory.getOWLDataPropertyAssertionAxiom(tweetContent, tweetIndividual, t.getTweetContent());

dataAssertion.add(tweetContentDataPropertyAssertion);

....

manager.addAxioms(filietOntology, dataAssertion);

Code Listing: Data Property Assertion

Set<OWLObjectPropertyAssertionAxiom> objectAssertion = new HashSet<OWLObjectPropertyAssertionAxiom>();

OWLObjectProperty hasThisInfo = dataFactory.getOWLObjectProperty(IRI.create(BASE\_IRI + "has\_this\_information"));

OWLObjectPropertyAssertionAxiom hasLocationAssertion = dataFactory.getOWLObjectPropertyAssertionAxiom(hasThisInfo, tweetIndividual, locationIndividual);

objectAssertion.add(hasLocationAssertion);

OWLObjectPropertyAssertionAxiom hasTimestampAssertion = dataFactory.getOWLObjectPropertyAssertionAxiom(hasThisInfo, tweetIndividual, timestampIndividual);

objectAssertion.add(hasTimestampAssertion);

manager.addAxioms(filietOntology, objectAssertion);

Code Listing: Object Property Assertion

After preparing the Tweet individual, the next step is to assert or tell the ontology that the newly instantiated classes are instances of the instantiated Ontological classes. A sample code listing below shows how to assert individual instances as class instances.

Set<OWLClassAssertionAxiom> classAssertion = new HashSet<OWLClassAssertionAxiom>();

OWLClassAssertionAxiom caAssertion = dataFactory.getOWLClassAssertionAxiom(cautionAdviceClass, caIndividual);

classAssertion.add(caAssertion);

OWLClassAssertionAxiom adAssertion = dataFactory.getOWLClassAssertionAxiom(adviceClass, adviceIndividual);

classAssertion.add(adAssertion);

manager.addAxioms(filietOntology, classAssertion);

After asserting that the newly instantiated classes are instances of the instantiated Ontological classes, the next step is to add the extracted information to the different individuals as their respective data properties. A sample code listing below shows how to add data properties to the individuals that were instantiated.

Set<OWLDataPropertyAssertionAxiom> dataAssertion = new HashSet<OWLDataPropertyAssertionAxiom>();

OWLDataProperty tweetAdvice = dataFactory.getOWLDataProperty(IRI.create(BASE\_IRI + "tweetAdvice"));

OWLDataPropertyAssertionAxiom adviceDataPropertyAssertion = dataFactory.getOWLDataPropertyAssertionAxiom(tweetAdvice, adviceIndividual, ca.getTweetAdvice());

dataAssertion.add(adviceDataPropertyAssertion);

manager.addAxioms(filietOntology, dataAssertion);

After adding data properties to the individuals that were instantiated, the next step is to assert or tell the ontology the different relations that connect the different individuals that were instantiated. In ontological terms, this is called Object Properties. This is the part where connecting the different individuals are connected so that they would make sense when stored into the ontology. Examples of these connections include the “has\_this\_information” connection, the “gives\_out\_an” connection and the like. A sample code listing below shows how to add object properties between the individuals that were instantiated.

Set<OWLObjectPropertyAssertionAxiom> objectAssertion = new HashSet<OWLObjectPropertyAssertionAxiom>();

// Say that the Tweet Individual is of the category CA Individual

OWLObjectProperty canBeOfTheCategory = dataFactory.getOWLObjectProperty(IRI.create(BASE\_IRI + "can\_be\_of\_the\_category"));

OWLObjectPropertyAssertionAxiom canBeAssertion = dataFactory.getOWLObjectPropertyAssertionAxiom(canBeOfTheCategory, tweetIndividual, caIndividual);

objectAssertion.add(canBeAssertion);

OWLObjectProperty givesOutAn = dataFactory.getOWLObjectProperty(IRI.create(BASE\_IRI + "gives\_out\_an"));

OWLObjectPropertyAssertionAxiom givesAssertion = dataFactory.getOWLObjectPropertyAssertionAxiom(givesOutAn, caIndividual, adviceIndividual);

objectAssertion.add(givesAssertion);

manager.addAxioms(filietOntology, objectAssertion);

After adding the object properties, the complete extracted information can now be finally stored to the ontology. Storing information to the ontology is simple done by calling the function ***saveFilietOntology***.