COLLECTING KNOWLEDGE FROM SOCIAL MEDIA

CM3203 One Semester Individual Project- 40 credits – FINAL REPORT

Abstract

This project asks the question: Can we extract actionable information (knowledge) from tweets relating to a particular event (music festival) and use CENode to allow users to query that knowledge base for useful information. To answer this question, I developed a client-server based system that identifies useful information from tweets and stores them in a knowledge base. I also implemented a web application that allows users to interface with the knowledge base. The final system was able to take in tweets, extract information from it, and then store it in a knowledge base. The web application was able to take in user input and query the knowledge base. It was unable to display the response from the server back to the user.

AUTHOR: M TAHIR (1308094)
SUPERVISOR: PROF. A PREECE
MODERATOR: DR. M CHORLEY

Acknowledgements

First, I would like to take this opportunity to thank my supervisor Prof. Alun Preece for his continual support and guidance throughout the dissertation. I would also like to thank Will Webberley for giving me guidance with CENode, without his assistance I would have spent a lot more time debugging!

I would also like to thank my family for their continuing support throughout my time at university and indeed the whole of my education. Without them, I would not be where I am today.

A mention must also go out to my friends, both at university and at home, without whom there would be a lot less laughter in my life.

Contents

Table of Figures

1 Introduction

1.1 Social Media and Knowledge

Social media is the term used to describe a variety of web-based platforms, applications and technologies that enable people to interact with each other online. Examples of social media sites include Facebook,YouTube,Del.icio.us and Twitter. The content of these websites is based upon user participation and user generated content. The invention of the smartphone has allowed people to share rich content, with precise location (coordinates) and visual in the form of pictures or video, which enriches the relevance of the content.[4] Knowledge is understanding of someone/something, such as facts, information or descriptions, which is acquired through perceiving, discovering, or learning.

1.2 Gathering knowledge from social media

Since social media is a huge part of everyday life, with people using it to log events and use it to express their opinions about a particular subject. This means that there is a wealth of information in the content produced by users. Social media intelligence refers to the collectives tools and solutions that allow organizations to monitor social channels and conversations, respond to social signals and synthesize social data points into meaningful trends and analysis based on the user's needs. This can be particular helpful when trying to detect events as they happen.

The aim of this project is to investigate gathering knowledge from social media. That is by parsing publicly made tweets from users, can we gather actionable information about a major event such as an music event and then allow attendees/staff of that event to query the knowledge base for up-to-date information.

2 Background

2.1 Twitter

Twitter is a microblogging service that allows registered members to broadcast short posts called tweets. Unlike other social networking sites such as Facebook, which require you add people to access content produced by that particular user, twitter is by default public. Therefore tweets are permanent, they are searchable and they are public. Anyone can search tweets on Twitter, whether they are a member or not. A key feature of twitter is the hashtag(#). A hashtag is a type of label or metadata tag which makes it easier for users to find messages with a specific theme or content. Users create and use hashtags by placing the hash character (#) in front of a word or unspaced phrase, either in the main text of a message or at the end. Searching for that hashtag will then present each message that has been tagged with it.

2.2 Three V's of Big Data

3Vs (volume, variety and velocity) are three defining properties or dimensions of big data. Volume refers to the amount of data, variety refers to the number of types of data and velocity refers to the speed of data processing. According to the 3Vs model, the challenges of big data management result from the expansion of all three properties, rather than just the volume alone -- the sheer amount of data to be managed.[19]

2.3 Structured knowledge base

A knowledge base is a database used for knowledge sharing and management. It promotes thecollection, organization and retrieval of knowledge. Many knowledge bases are structured, which means that they store the data in a distinct way.

2.4 What is controlled English?

A Controlled Natural Language is a restricted form of a language that is readable by humans and unambiguous to computers. Using a controlled natural language can help to make complex computing tasks accessible to non-technical users in their own language. Therefore these languages are useful and can be used as knowledge-representation languages. As part of the International Technology Alliance (ITA), a Controlled Natural Language called "ITA Controlled English" (CE) has been developed.[20] CE is an unambiguous subset of full English and can be used to model a domain and reason about it.

The domain model is created (conceptualised) using CE sentences starting with the word conceptualise:

conceptualise a ~ man ~ M that is a person.

conceptualise a ~ woman ~ W that is a person.

The user states facts using CE sentences:

there is a woman named Jean.

there is a man named James.

2.5 Why use a controlled english KB?

By using controlled English to create a KB means that the data goes into a structured KB. This will allow machine agents to help humans make sense of lots of information, this approach can help with all 3 Vs of big data, especially velocity. [19]

Another reason for using a controlled English KB is that users with less training in KBS to extend and query the knowledge model as well as the instances. Therefore it will help in building and extending a KB relatively quickly, which is necessary to adapt to fast-moving and fast-changing situations such as unexpected things happening at a large scale event.

2.6 Previous work

The Traditional method using embedded sensor networks is plagued with problems such as difficulty adapting to complex spaces and various others such as aesthetics etc. To combat this there has been a rise in the use of human involvement, which is particularly useful in sensing various processes in complex social and urban spaces. By using people who frequent/visit these places and by using their knowledge of the area, human-centric sensing makes it feasible to get information that otherwise is not possible. [21]

Social media is the medium that has seen a surge of interest as a source of actionable information that can be used for situational awareness. Twitter, the popular microblogging service has been a particular favourite of researchers.[22] This is due to its characteristics of twitter:

- Real time characteristics - when an event occurs, people make many Twitter posts (tweets) related to that particular event. By analysing the tweet you can detection of occurrence of the event
- Follower Model - It takes only a fraction of a second to hit the retweet button on Twitter…..for example a person at the scene tweeting can be retweeted by a news agency or user with a large follower base and so content can reach people very quickly.
- 140 character limit - due to this people are forced to tweet in a precise manner getting to the heart of the matter.

An example of events that first broke on Twitter includes the Boston marathon bombings. The tragic news originated from people on the ground at the race's finish line. They posted images of the explosions moments after the tragedy occurred.[5]
This combined with natural language question-answering systems and contextually aware mobile applications allows for improved Sensemaking.[2] Sensemaking is the process by which people give meaning to experience. Users equipped with mobile devices act as sensors (able to acquire information).[24] There is growing recognition that users need more flexible styles of conversational interaction, where they are able to freely ask or tell, be asked or told, seek explanations and clarifications. Ideally, such conversations should involve a mix of human and machine agents, able to collaborate in collective sensemaking activities with as few barriers as possible.
A recent study, tasking the tweeters, the researchers were attempting to gather situational awareness by identifying individual tweeters as “sensing assets”. These assets had already tweeted about the event , by engaging with these particular tweeters and asking for clarifications or amplifications can lead to the value of the information provided being even greater due to verification of facts. [1]

2.7 The Problem & proposed solution

Both of these studies make use of human involvement in order to enrich the data before adding to a KB but can we remove the human element from the system if we already know information about an event before it takes place such as a music festival?

Music festivals are commonly held outdoors, and are often inclusive of other attractions such as food and merchandise vending, performance art, and social activities. With so many various pieces of information, an attendee requires can we crowd source up to date information and make this available to attendees of that event.

Can we use twitter as a source of information for large-scale events such as a music festival and then use a “conversational agent” that can provide facility for attendees to query this information?

Figure 1 An example of the tweets of the festival I am interested in

2.8 Approach to solution

Data collection - I will be using Twitter to collect information to enrich. This is mainly due to the 140 character imposed which causes users to condense their ideas, shorten your words, and really get at

the essential point they are trying to make. It forces users of twitter to think about what you're going to say and get at the heart of the matter, rather than writing two or three paragraphs before you get to the point with other forms of social media such as Facebook for example.

- The process of gathering information about a particular event is simple and can be broken down into four steps following the DCPD model of collecting and managing sensing assets (in this case, the publicly available tweets) [6]:
    - **Direction -** involves establishing what data to collect from the available streams. In principle with Twitter it is feasible to collect all available data, though the cost of this is substantial, as are the computational resources required to handle that volume of data. Use a query to get tweets that are relevant to the domain that is being investigated.
    - **Collection -** Twitter is particularly convenient in this respect, offering a number of APIs for streaming, searching and sampling.
    - **Processing -** The objective of this step is generally to provide semantic enrichment of the data, to make it useful in situation understanding
    - **Dissemination** of the results of the processing step may involve visualisation. In my case I will be making use of CENode. CENode is a JavaScript based "conversational agent designed to mediate interactions between human users and machine agents". It allows users to query the knowledge base by asking it questions in English.

Stakeholders for this project will be first of all the attendees of the event. They can query the system to get up to date information about the status of certain things such as which band is playing, the status of the facilities (i.e. toilets) and the current weather for example. Another stakeholder would be the organisers of the event. They can have up to date situational awareness of events happening at the festival and they can respond accordingly if a situation arises.
The one constraint of my approach is the context of the tweets. Many users on twitter usually tweet their own opinion of the event rather than what is actually going on. For example, the majority of the tweets about football matches tends to be their opinion of certain footballers rather than the actual action happening on the pitch. Therefore, the no. of tweets actually providing information about what's going on might be low

3 Selection of Approach

3.1 Twitter

The Application Programming Interface, API, is a large part of Twitter's success. Twitter has two main group of APIs called REST and Streaming API. As of version 1.1, the Twitter API now requires OAuth authentication, either application-only authentication or application-user authentication. The latter requires your Twitter user to click through to the Twitter website, sign in with their credentials, and then return to your site. Application-user authentication is required for many user-specific API calls.[3]

3.1.1 REST

The REST APIs provide programmatic access to read and write Twitter data.[7] It allows you to author a new Tweet, read author profile and follower data, and more. The REST API identifies Twitter applications and users using OAuth; responses are available in JSON. The REST API is best used then to fill in the past 7 days for your subject matter.

3.1.2 Streaming

The Streaming APIs give developers low latency access to Twitter's global stream of Tweet data.[8] A proper implementation of a streaming client will be pushed messages indicating Tweets and other events have occurred, without any of the overhead associated with polling a REST endpoint.

3.1.3 Search Queries

The REST API has a fairly rich set of operators that can filter results based on attributes like location of sender, language, and various popularity measurements. The streaming API has a more limited approach of only collecting tweets containing words, sent by specific accounts, or within a geographic area.

3.2 CENode

The CE Store is a Java program which provides a processing environment for ITA Controlled English (CE), including domain modelling, inference, and natural language (NL) to CE interpretation.[9] It is based on client server architecture where the CE is processed on the server side and the clients can gain access to the functionality of the programme by the means of application programming interfaces (API).

CENode is a pure JavaScript implementation of CEStore, which is lightweight and can be deployed in a variety of different ways (mobile applications, web browsers, servers etc). It's lightweight due to the fact it's not designed to be a fully-fledged CE engine, it is capable of limited inference and NL processing. Due to this it doesn't require high amount of network bandwidth for it to be downloaded and for it to operate. A CENode instance, when loaded, can function independently without a network connection by maintaining a local knowledge base (KB). This property makes it well suited to deployments at the network edge.

CENode can be run independently or as a part of a multi node system, where one of the nodes must be run as a server (through Node.js). Each node in the systems has the same functionality and behaviour. Getting/providing information from a node is done via CE.

CENode has a few features are really useful in operating at the network's edge:

• The first advantage is that it's a lot more decentralised than CEStore, which relies on a single CEStore server with a centralised KB. CENode on the other hand supports a network of peers who could all potentially have different local KB with different variations.

• Users can use their device to directly access a CENode agent and can interact with it. All CE is parsed on device (locally) and local knowledge that has been stored can be relayed to other agents when a connection to the network is established

• Any NL is also parsed locally. This means that any input by the user is processed locally and only validate CE is transmitted between node. This saves not only bandwidth because only valid cards are transmitted within the network but also time as they don't have to processed.

3.2.1 Supported CE & Modelling

To make any modifications to CENode's conceptual model, have to be made through a **conceptualise** statement. An example could be creating a new concept called 'teacher', which is going to be a subclass of a 'person' (for this example we are going to assume that 'person has already been conceptualised). The following sentence achieves this:

Conceptualise a ~teacher~ T that is a person.

CENode will now allow instances of teacher to made up with properties assoicated with the person concept. The following adds further properties to the teacher concept (assume that 'subject' and 'age' have been conceptualised already):

Conceptualise the teacher T ~teaches~ the class C and has the subject S as ~subject~ and has the value A as ~age~.

CENode will now allow instances of 'teacher' to have a teaches relationship and have **subject** and **age** values.

Now let's create a new instance of these existing concepts, which are declared via regular CE:

There is a teacher named 'Mrs Smith'.

This creates an instance of 'Mrs Smith'. We can add more information to this instance:

The teacher 'Mrs Smith' teaches the class 'Year 11' and has the subject 'Biology' as a subject and has '50' age.

For this example, we are going to assume that the class 'Year11' had not been declared as an instance of **class.** CENode attempts to do some of the work for the agent which provides this information to it. Therefore, it will create a new instance of type **class** named 'Year 11. Therefore, if we assume that we did not also declare 'Biology' as an instance of **subject,** it will create it for us. As for age, as 50 is value and therefore has no type, it's just embedded in the 'Mrs Smith' object type. If a property in the input CE that hasn't been declared in the 'teacher' conceptual model or in any of its ancestors, then this property will be ignored. CENode does not support the deletion either of an instance or concept same as CEStore.

CENode is also able to understand some additional sentence structures to make interaction a little easier and to support information extraction.

3.2.2 Questions

CENode has the ability to answer questions. Users can get information using 'what/where/who' , which extracted from the node and relayed back in an easy-to-understand gist format. CENode treats 'who' and 'what' questions in the same way. Using the Mrs Smith example, the following questions have equal meaning:

- *Who is mrs smith?*
- *What is mrs smith?*

These question will both get the same response back from the node:

Mrs Smith is a teacher. Mrs Smith teaches the class 'Year 11' and has the subject 'Biology' as subject and has '45' as age.

You can also query the node for ‘Where’ based questions. They require the core model to be loaded which includes the **location** concept. This can then be used the parent to declare other types of locations such a room, building or stage etc.

Using the Mrs Smith example, where the **person** concept (parent of **teacher** and therefore inherits from it) supports a relationship called ‘lives in’ that targets an instance of type **house**, which is a child of **location**:

The teacher Mrs Smith lives in the house ‘No.21’.

Now that it's been declared, we can now ask CENode a ‘where’ question regarding Mrs Smith:

Where is Mrs Smith?

And this gets the following response from the node:

Mrs Smith lives in the house ‘No.21’.

You can also query for the location directly using CENode:

Who is in No.21?

This would get the following response:

The teacher Mrs Smith lives in the house No.21.

3.2.3 Node Models

A CENode instance comprises of a KB and an agent. The KB is used to store the concepts and instances that CENode instance knows about. To update the KB, the node is given CE and similarly the KB can be queried by asking questions.

Models allow for the creation of an empty KB to which knowledge can be added via CE that is added to the node. A model is a collection of CE sentences that are sent to the node to develop its conceptual model and create instances. CENode as it's written in JavaScript. The model is therefore a simple array of CE sentences. For example:

```
Var myModel = [
        “Conceptualise a ~teacher~ T.”,
        “There is a teacher named ‘Mrs Smith’.”
];
```

Models can be loaded into an instance of CENode when the node is instantiated. The core model contains concepts such as the location concept and others that serve as useful parent class when adding to the model. These CORE models are included inside cenode.js MODEL object and is accessed by MODELS.CORE.

3.2.4 CENode Agents

Each CENode has its own agent. In a multi node system, agents handle the node to node interactions through the use of policies (see Agents and Policies). A node's agent is distinct from the nodes KB; it has equal access to the node's conceptual model as someone accessing it programmatically. For agents to function correctly the CORE model has to be loaded, furthermore each node in a multi-node system should have a unique name to avoid confusion. By default, each node is given the name 'Moira' which can easily be changed. The KB can be made aware of agents (local and other agents) using CE:

There is an agent named 'agent1'.

3.2.5 Cards

To make use of agents, 'cards' should be used to deliver CE, which is the basis of the blackboard architecture implemented by the CEStore and is the recommended means of communication, whether that its between human to node or node to node. There are different types of cards, all of which inherit from **card** concept and they are included in the CORE model. Cards work in the following way, they wrap CE in a value property and this allows the information that is contained inside to be sent to different agents. The agent will only read cards that have it as the intended recipient.

- Tell Card - the card is used to tell the intended agent some information. I

An example of a **tell card,** using the Mrs Smith example:

there is a tell card named 'msg1' that is to the agent 'agent1' and is from the agent 'Moira' and has the timestamp '123456' as timestamp and has 'there is a teacher named \'Mrs Smith\'" as content.

- Ask Card - used to query information from the intended node's KB. The nodes local agent will respond by returning a suitable response or an error if the question is invalid as it does not conform to the supported question structure as mentioned previously.
- NL Card - These are used when the type of information is unknown. There is an order step for dealing with these (failure leads to the next step being processed):
    - **Test for valid CE:** if the content has valid CE, the agent writes a **tell card**, which has exactly the same content and addresses it to itself. It has the same outcome as adding a **tell card** that has valid CE. This process is known as auto confirming.
    - **Test for question:** if the content contains a valid question, the agent will create an **ask card**, which has the same and addressed it to itself. This has the same outcome as an **ask card** with a valid question. This process is called auto tasking.
    - **Node:** The final stage is that content is passed onto the node. The node will attempt to parse the NL and if this is successful, it will return a **confirm card** containing its guess at the CE that best represents the content that was inputted. The content of the **confirm card** can then be used in a **tell card**, which can be used to update the node's KB.

3.2.6 CENode Architecture

Agents continuously check their node's KB for any cards that are addressed to themselves. If a card is found that is addressed to and hasn't yet been seen by the agent, then the agent will act upon it. If a card instance exists in a node's KB and the node's local agent is not a recipient, then no further action will occur for this card on this node. This forms the basis of blackboard architecture which in which agents and users can read and write cards from and to a node. Policies (3.2.8 Agents & Policies) which allow agents to communicate with one another automatically.

Any CE that is submitted to the node will be parsed immediately and if it's valid the node's conceptual model will be updated according to the CE. The node will also return a response immediately (programmatically or in a response to a HTTP request) that contains relevant content. This happens when question are asked of the node in the form of what/where/who questions as previously described (3.2.2 Question Asking)

3.2.7 Manipulating the KB

Each CENode instance is made up of the following:

- A KB
- A local CE agent - responsible for maintaining the KB

When any CE is received, the CENode instances will attempt to process it and if necessary update it KB. As previously mentioned (3.2.6 CENode Architecture) CENode supports blackboard architecture. The blackboard architecture means that users and agents can use CE cards (3.2.5 Cards) to submit CE sentences that have the local agent as the recipient. If the card is intended for the local agent, then the local agent will find the card and read it. If the card contains valid CE, the agent will read the content and react accordingly, for example updating its local knowledge base based upon a tell card. If the card is not intended for the local agent but for another node in the multi node network, then the message will not be read but passed on to the intended agent via policies.

3.2.8 Agents & Policies

As previously mention a CENode instance comprises of a local KB and a local agent which is responsible for updating the KB when valid CE intended for that agent is received. Agents in multi node networks are able to send cards to one another with respect to policies. Policies are instructions, written in CE that, when applied to a node, causes the local agent to communicate with other agents in the network. There are different types of policies:

- Tell

A tell policy inherits from policy and its function is to instruction the local agent to tell the target agent defined by the policy, everything that local agent is told. An example of this would be the following:

- There is a local agent that has the name *'agent1'* and it is told about another agent's ('agent2')
- *There is an agent named 'agent2' that has 'agent2.example.com' as address.*

Now the 'agent1' knows that there is an 'agent2' that has 'agent2.example.com' as address, we can now create a tell policy targeting 'agent2':

there is a tell policy named 'p1' that has 'true' enabled and has the agent 'agent2' as target.

Once this policy has been created, then our local agent, 'agent1', will tell 'agent2' every piece of information that has been told to 'agent1' in tell cards by wrapping the content in a new tell card and HTTP POSTing this to the appropriate endpoint at 'agent2' s host address. As such, 'agent2' needs to be an agent running as a service instance (e.g. via Node.js).

- Ask
    - An ask policy works in almost exactly the same way as a tell policy (with our local agent named 'agent1'): there is an ask policy named 'p1' that has 'true' as enabled and has the agent 'agent2' as target. In this scenario, every ask card sent to 'agent1' will also be sent to 'agent2' using a HTTP POST request. As with targets of a tell policy, target agents of an ask policy must be instances running as a service instance.

- Listen
    - Polls the target agent for instances of 'tell card' sent to 'agent1'. Any cards that are discovered are opened and the content is added to the agent's nodes KB as normal.
    - The target agent must be running as a service (via NodeJs).
    - You should also use a listen policy together with an ask policies. This would a would enable a response to be retrieved from target from the ask policy.

If the node with active policies is not connected the network and therefore not able to communicate with other nodes, then the node will continue to function normally and will attempt to established communication with other nodes once it re-establishes connection with the network.

4 Specification and Design

4.1 Overview

The project consists of three main sections:

- A Program to collect tweets
    - This data can be collected with relative ease using Twitters API, which allows a developer to only retrieve tweets that meet a certain - set terms or within a given geo-spatial region.
- Process the tweets for semantic value
    - Once the data has been collected, it has to be processed in order to enhance the semantic value of the data so that it can be used to provide factual information.
- CENode
    - For a multi node system, one instance of CENode needs to be running as a service (via NodeJs).
    - For this project, I will be using two CENode instances in a client-server model.
    - Store the enhanced data in the server CENode's knowledge base
    - The final stage is the implementation of a web based CENode instance. This would allow the users to query the knowledge base of the server.

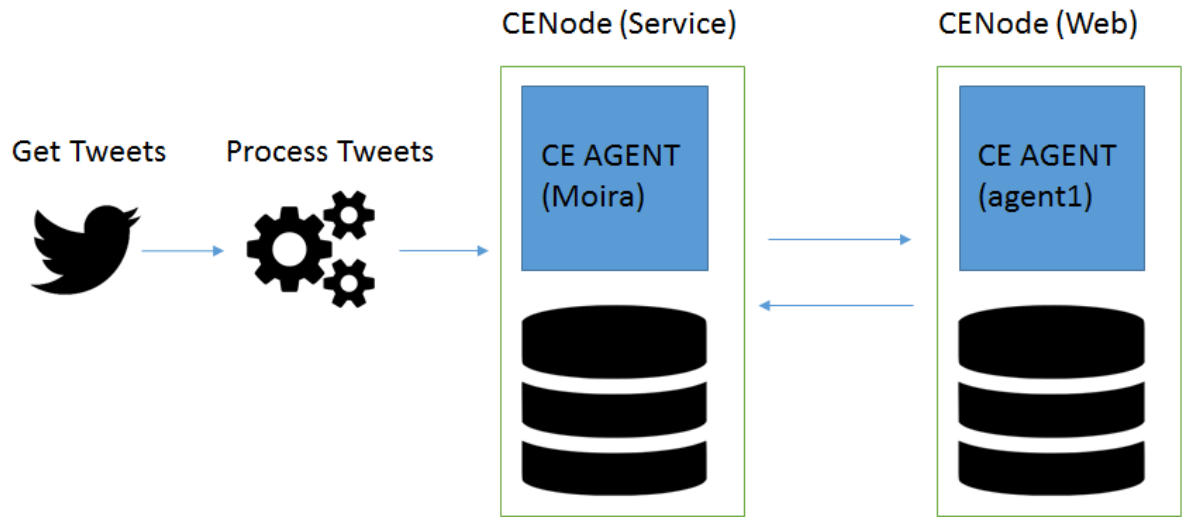

Figure 2 An overview of the purposed system

4.2 Twitter Intake

ReadTweets will use the both Twitter's Streaming API or REST API (depending on requirement) to listen for tweets and it will get these in JSON format as they are loaded. I will be using a hashtag (#) to filter tweets relating to the event I am interested; therefore, the program will need to supply this to the API. I am interested in the text of the tweet and date, so these will be stripped from the JSON that has been returned by the program.

4.3 Process Tweets

4.3.1 Get band name and stage name

The first step for information extraction is to understand about what the tweet is referring to. In a festival setting this is going to refer to an artist/band. Also attendees of the event will need to know at which stage the band is playing and when that is going to happen. The program will take in the tweets from Twitter Intake and process them to extract relevant information. The first function (*getNameExact*) will try and get an exact match, if this fails then it will use (*getNameToken*) to get the name of the band and stage name, I will be using Levenshtein distance. It is a measure of the similarity between two strings. It is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other. I will be tokenising the tweet into individual words and then compares each individual token to an array of band to get the band name and an array containing the names of stages to get the stage name.

4.3.2 Get time

To extract the time, I will be using a simple bag of words approach. For example, if the tweet contains the following phrases "playing now or "on stage now", these indicate that the band is playing at the current time, then I will be using the time that tweet was produced (included in the JSON) as the time for which the band is playing. If the tweet doesn't contain the any of the phrases, then I will attempt to extract the time by using regex to check for any mention of time in the form of 12 hour and 24 hour.

4.3.2 Form CE Sentence

To create a CE sentence to add the node server's knowledge base, I will simply create a string which takes in the band name, stage and time as input and outputs a valid CE sentence that can be successfully parsed by CENode and update the knowledge base.

4.4 CENode

4.4.1 Overview

For this project my aim is to have a central CENode server which reads in the tweets, process them and then stores the knowledge gathered in its KB. To allow user to query this data I will create a web application that takes in user input, evaluates it and then gives the necessary response (Whether it's to update the server's KB or querying the server's KB).

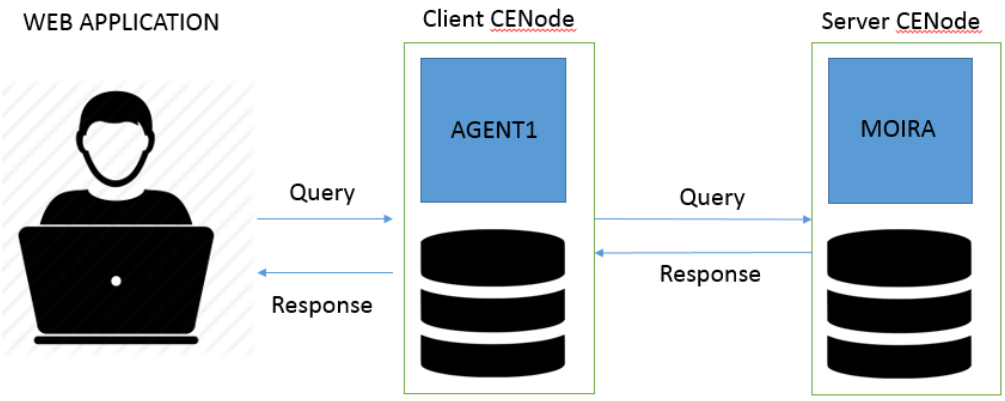

Figure 3 How the CENode instances will function

To handle the networking between the nodes I will be making use of policies (3.2.8 Agents & Policies) and cards (3.2.5 Cards).

- Node Server
  - Tell it about agent1 (web node)
  - Listen Policy – the listen policy will target agent1
- Web Node
  - Tell it about Moira
  - Ask Policy – the ask policy will target Moira
- Listen policies are useful in conjunction with ask policies, since they enable a response to be retrieved from the target of the ask policy.
- This setup will cause 'agent1' to forward all ask cards it receives to Moira and will be able to receive a response from Moira, through the listen policy, once Moira has read and replied to the ask card.

4.4.2 Conceptual Model

It is recommended that a variable MODELS.CORE be passed to the constructor. This model allows the CENode to initialise itself with any key concepts and instances that are required. The core model contains concepts such as the location concept and others that serve as useful parent class when adding to the model.

As we are concerned about a music festival, we can create a model that that will give the node some basic knowledge about the domain. The key to a music festival are the bands/artists that are going to be performing there. Music festival have stages and these are the venue on which the band/artist are going to perform. They also have a scheduled time when they are going to perform, which is a location. Models are simple to create, it's a JavaScript array that contains sentences in CE that are added to the node in order that they are declared in the array.

To model a festival, we are going to have to conceptualise the following:

- Festival
- Band
- Time
- Song
- Stages and their location
- Where the band plays and the time they are performing.

To add these to the node, I will create an array and will use the term conceptualise (see section 3.2.1) to add these statements to the node.

4.4.1 Server

Once the all the relevant information; the band name, the stage they are playing and the time of the performance. I will be forming a CE sentence that will be sent as the content of a tell card to the CENode agent named "Moira" which will be running as a service. If the CE is valid the knowledge base should be updated. Server

Tell - A tell card should be used to tell a particular agent some information. So in this instance it will be used to tell the client node about instances stored in the server's KB.

Listen - listens for any cards that are addressed to it from the targeted agent (agent1). These will be the ask cards that are created when a user queries the client for information.

4.4.2 Client

I will also be implementing a web based node. The function of this node is primarily to allow users to query the server node's knowledge base. To achieve this, I will be making use of policies (3.2.8 Agents & Policies) to achieve this.

Ask Policy - In this scenario, every ask card sent to 'agent1' will also be sent to 'MOIRA' using a HTTP POST request.

To get a response I will need to implement my own function to retrieve the card intended to agent1 with the reply to the ask card. To do this I will need to make use of CENode's API. The agent can be interacted with using these HTTP methods. The GET method can be used to retrieve information from the node server using a given URI and return all cards known by the CENode in line-delimited

CE. Furthermore, you can retrieve cards address to a certain agent, which in this case we are going to be interested in cards addressed to 'agent1'

These cards will then be processed and the relevant information displayed back to the user who asked the question in the first place.

The web application will have the following:

- html file for displaying/taking data
- Main.js ---> main logic of the program
- Cenode.js → cenode source file

HTML

- A means for inputting sentences
- A means for displaying messages from the agent

Main.js (main code) will have the following functionality:

- creates an instance of CENode, continuously runs in the background
- Load models including the all-important ask policy
- The type of CE card generated from the user input is in entirely dependent on the message the user sends to the node.
    - If it's a who/what/where message asking for some information, then it will create an ask card which is sent to node server (Moira) via the ask policy (using POST).
    - Else → natural language to be processed by the node
- Function *pollCards*
    - CEAgents work entirely asynchronously to the rest of the app and the CENode KB itself, therefore we need to create a function that will continuously poll the CENode for any cards that the CEAgent may have written back to us.
- Function *getCards*
    - The aim of this function is going to be to retrieve cards addressed to agent1 from the server's KB.
    - This will be done in the form of a GET request sent to the node server.
    - Then will add these to the client nodes KB to be read by agent in its own time.

4.5 Web App wireframes

So the web app is going to relatively simple as its purpose is to allow a user to send a message to the node and get a response back.

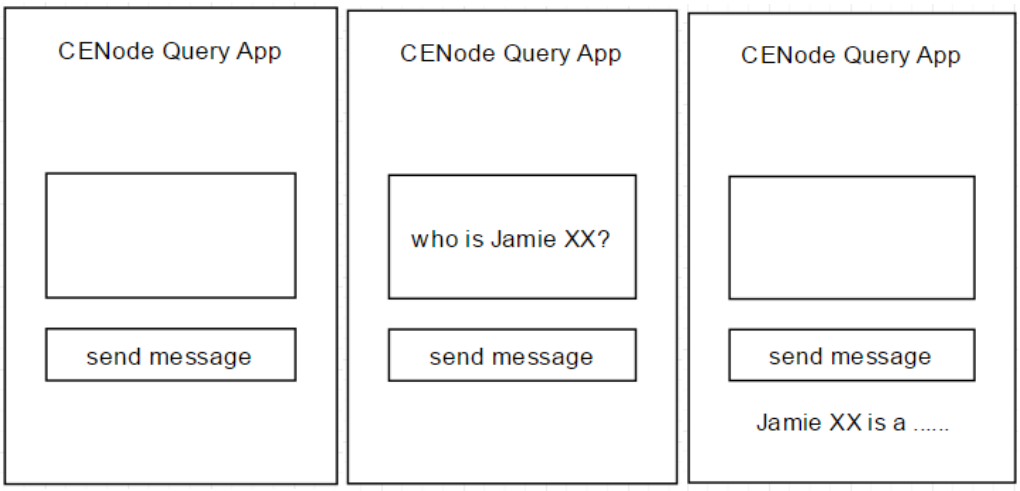

Figure 4 Wireframes showing the site's structure.

As you can see from the wireframe the web app's front end is going to very simple in terms of both design and functionality. Essentially there's going to be a text box for user input, where users can enter the message to send to CENode, with a button labelled "send message" and the response from CENode will be simply displayed below the send message button.

4.5 Data Flow Diagram

The data flow diagram of the purposed system, modelled using Yourdon/DeMarco notation [10]:

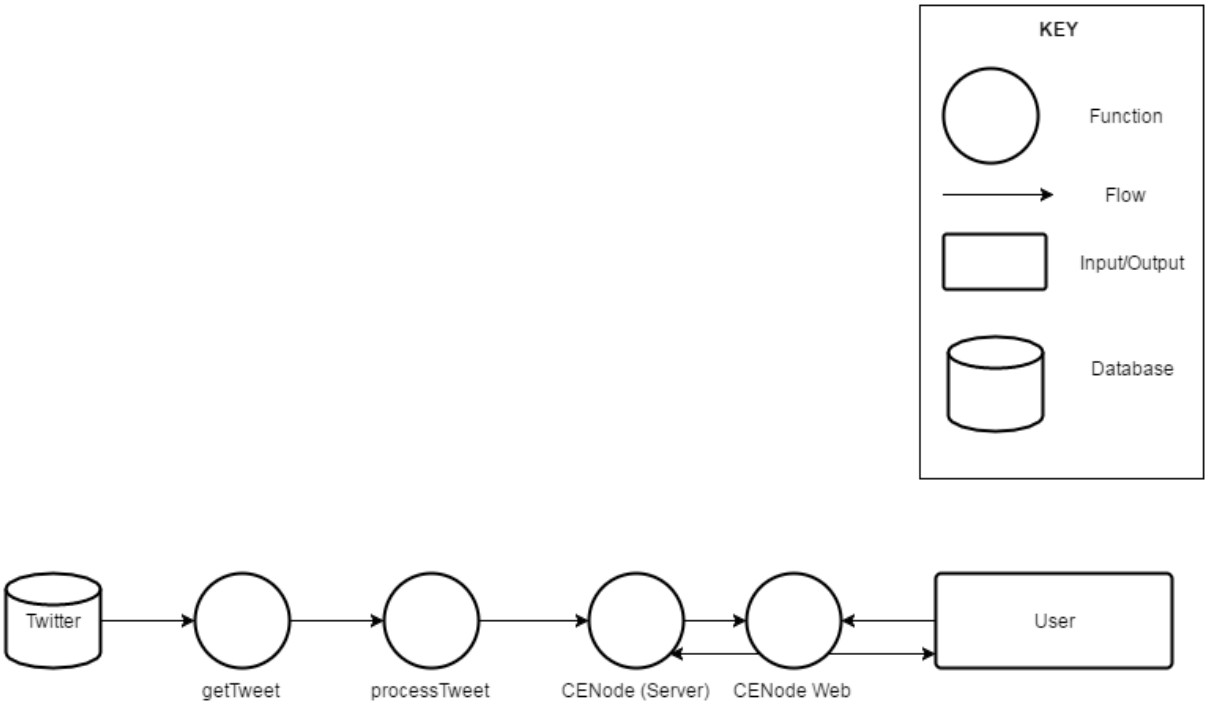

Figure 5 shows the data flow through the system from source to user

4.6 Requirements

4.6.1 Functional Requirements

- Twitter Intake
    - Needs to collect tweets relating to the hashtag specified
    - must be stable and not crash
- Process Tweets
    - Needs to extract the text of the tweet
    - Needs to get band name
    - Needs to get stage name
    - Needs to get time of performance
    - Needs to form CE sentence to add to CENode
- CENode (server)
    - Needs to read CE sentences that are sent to it.
    - If CE refers to a tell card → update knowledge base
    - If CE refers to an ask card → reply with related content
- CENode (client)
    - Needs to take in user input
    - Needs to query the server for information
    - Needs to display this knowledge back to the user

4.6.2 Non-Functional Requirements

- The system should run efficiently
- The system should be platform independent
- The system should be formatted correctly and commented

4.7 Issues & Challenges

4.7.1 Getting Band Name

To achieve this my method was going to carry out named-entity recognition (NER) which aims to label sequences of words in a text which are the names of things, such as person and in this case band/artists name. As I was using a specific domain (music) I will be making use of MusicBrainz, which is a project that aims to create an open content music database.[11] MusicBrainz captures information about artists, their recorded works, and the relationships between them. Recorded works entries capture at a minimum the album title, track titles, and the length of each track. The database can be accessed via an API, I will be using NodeBrainz, which is NodeJs Client that gives full access to the MusicBrainz API (Version 2), and it returns the data in JSON format and allows easy access to the all-important search function.[12] The search function gives a score out of 100 of how closely the term inputted matches an artist. I will be using this to identify the artist in the tweet, if there is one.

This didn't work as valid method for extracting the band name without any previous knowledge meant that I had to tokenise the tweet and create combinations of the tokens and input each of these combinations into the search function of API and then record their scores. The combination with the highest score was deemed to be the band name. The first thing I noticed was that due to the sheer number of artists out there that the search was returning artist matches for tokens of multi word artists. For example, "Foo Fighters" will be tokenised to "Foo" and "Fighters", now both of these returned with a score of 100 (perfect match) as did "Foo Fighters" therefore it isn't accurate. The second problem was with connective words such as "The" and "and" would return matches too. For example, "Florence and the machine" would return a match with 100 for "the machine". Therefore, this method was too inaccurate to use in the program.

4.7.2 CENode Policies

My initial plan was to use policies entirely to handle the networking between the CENode instances but for that to occur each instance had to be running as a service (via NodeJS), which was an oversight on my end. So when an 'ask card' is POSTed to the target, its agent will get round to reading the card in its own time and will write a card back to its own store if the card requires a reply. Therefore, I had to design and try and implement a way of querying the node servers KB and get the cards addressed to the client and add them to its KB.

4.7.3 Web Application

Another challenge I found was the actual web development required. Personally I hadn't done much web development since first year and therefore spent a very large time trying to understand the technologies involved such as NodeJs , Requests etc.

5 Implementation

5.1 Twitter Intake

5.1.1 Approach & Design

The first step of implementation was to implement twitter intake. Twitter has two API as previously mentioned, streaming (real time tweets) and REST. I am aiming to develop two systems to cater for this. One that can take in tweets in real time (as they are posted) and one that can use the search API within the REST series of API's.

To access these API, I will be using Twit which is a NodeJs package which supports both the REST and Streaming API.[13] I will be using the streaming API to collect the tweets in real time and the REST API to collect tweets relating to my planned experiment.

To use twitter API, you have to follow a few steps before you can begin to develop/implement:

- The first step in creating any program which will be using the Twitter API's is to acquire to a developer account. This is a necessary step to make requests for the v1.1 API
- Create an application: Create an application on the Twitter developer site. You need to visit http://dev.twitter.com/apps and click the "Create Application" button. This enables the user to grab a set of unique keys to use for your application. These will allow you to create authenticated requests. These are:
    - The consumer key
    - The consumer secret
    - The access token
    - The access token secret
- Change access level: for this project I will be only reading in tweets therefore I needed to change your settings to Read Only.

5.1.2 Using Twit

5.1.2.1 Streaming

For this project I will be using the public streams available through twitter's streaming API. The streams allow you to access the public data flowing through Twitter. Once applications establish a

connection to a streaming endpoint, they are delivered a feed of Tweets. To achieve this via Twit we will be using the .stream function:

T.stream(path, [params])

- The path is to one of the streaming endpoint that we are planning on using, which in this case is going to be 'statuses/filter'. This endpoint returns public statuses that match one or more filter predicates.
- The params option refers to the parameters for the request. Any Arrays passed in params get converted to comma-separated strings.

For example, to get a list of tweets relating to Arsenal FC, I would use 'statuses/filter' as the endpoint and the hashtag '#AFC' and English on tweets as params. The code would look like this:

```
1. var stream = T.stream('statuses/filter', { track: '#AFC', language: 'en' })
2.
3. stream.on('tweet', function (tweet) {
4.  console.log(tweet)
5. })
```

This would print out the response from the API which is in JSON format. The JSON contains information about the tweet itself such as text, time, if it was retweeted or not etc and it also contains information about the user who posted that tweet.

Extracting information from the JSON.

JSON data is written as name/value pairs.
To access the text of the tweets all we need to do is use the "." notation to access the value stored relating to the name.

Using the example above, to access the text of the tweet:
tweet.text → refers to the text

Getting the time of the tweet to get the datetime:

Tweet['created_at'] → gets the local time when this Tweet was created.

I will be using this for my experiment which will be conducted in real time

5.1.2.2 REST implementation

I also implemented a rest implementation. This was done so that we could get tweets about events after they happened, such as getting all the tweets from a music festival that occurred recently (past 7 days). To access these tweets, we will need to GET any of the REST API endpoints. Twit has a function called get which does this for us.

T.get(path, [params], callback)

- Path
  - Refers to the endpoint that we want to access. When specifying path values, omit the '.json' at the end (i.e. use 'search/tweets' instead of'search/tweets.json').
- Params
  - (Optional) parameters for the request.
- Callback
  - Instead of immediately returning some result like most functions, functions that use callbacks take some time to produce a result, as searching for the tweets and returning take some time, this is why a callback function is required. The callback has
  - function (err, data, response)
    - data is the parsed data received from Twitter

The data is delivered in the form of a JSON object (with the value “statuses”), which contains array of JSON objects of all the tweets that relate to parameters declared when the request was made. To access each individual tweet’s JSON we have to access the array and iterate through each object which represents the tweet.

For example, if we are conducting a real time experiment where we ask people to tweet with a hashtag that we can track (#fyp2016test) then we would use the following code to get the tweets and print out the text from each individual tweet:

```
1. function getTweets(){
2.
3.  T.get('search/tweets', { q: '#fyp2016test', count: 100 }, function(err, data, response) {
4.
5.     //console.log(data)
6.
7.
8.
9.     tweets = data.statuses;
10.
11.    for(var i=0; i<tweets.length;i++){
12.
13.      text = tweets[i].text;
14.
15.      console.log(text);
```

5.2 Processing Tweets

Approach & Design

Looking at the background research done the most sought for information that people would like to know about is regarding timings and location in regards to acts in a large festival such as Glastonbury.

5.2.1 Getting band name & stage name

To carry out the extraction of the band name and stage name, I made use of natural. "Natural" is a general natural language facility for NodeJS. It supports both Tokenizing and string similarity which I need to carry out the extraction of band name and stage name. The following

- Tokenise the tweet

The first step is tokenise the tweet, this done by the tokeniser built into Natural an example is shown below:

```
1. var natural = require('natural'),
2. tokenizer = new natural.WordTokenizer();
3. tokenizer.tokenize(string));
```

- *getNameExact(tweet,option)*
    - This searches for the exact match from the tweet. If its successful it returns the match otherwise *getNameToken()* is run.
- *getNameToken(tweet,option)*
    - is the function that utilises the levenshtein distance algorithm, to see which token is the least different to the array of artist in case of band names or array of stages in terms of getting stage name.

5.2.2 Getting time

- *Get time*
    - If the text contains the words such as “playing now” and "about to go on" then the time of performance is at the time of tweet origin. This can easily be retrieved from the JSON data retrieved from Twitter API.
    - Else regex is used to retrieve the time from the text of the tweet [14]
        - `(?:[0][0-9]|[1][012])(?::[0-5][0-9])?(?:[AP]M|[ap]m)|(?:[0-1][0-9]|2[0-3]):(?:[0-5][0-9])`
        - `(?:[0][0-9]|[1][012])` matches the hour part for `12 hour` format.
        - `(?::[0-5][0-9])` matches the optional minute part (if you ever include) for `12 hour` format.
        - `(?:[AP]M|[ap]m)` matches meridian suffix or you can simply write `(?:[ap]m)` and make it case insensitive using flag `i`.
        - `(?:[0-1][0-9]|2[0-3]):(?:[0-5][0-9])` matches the `24 hour` format.
    - If both of these fail at getting time, then a null statement is returned

5.2.3 Create CE sentence

For a node’s KB to be updated needs to receive valid CE which the node instances will attempt to process it and if necessary update it KB. Therefore, to update the KB I have to form a valid CE sentence containing the band name, the stage where they are playing at and the time of the performance. I achieved this by simply create a string variable which would be sent to the node for processed.

```
1. var sentenceToAdd  = "there is a band named '" + band + "' that plays at the time '
   "  + time  + "' and that has the stage '" + stage + "' as venue."
```

5.3 CENode

5.3.1 Model

The first thing that we are going to need is a model that represents the domain. We are going to have to conceptualise that there is an entity called band, an entity called time, an entity called stage which has a location and that a band has a venue and plays at a certain time. As I am focused on using Glastonbury as an example I will also declare four of the main stages too.

The model that I have created is as follows:

```
var my_model = [
  "conceptualise a ~ festival thing ~ F that is an entity and is an imageable thing.",
  "conceptualise a ~ band ~ B that is a festival thing.",
  "conceptualise a ~ time ~ T.",
  "conceptualise a ~ song ~ S that is a festival thing that has the value W as ~ title ~.",
  "conceptualise an ~ stage ~ L that is a festival thing and is a locatable thing.",
  "conceptualise the song S ~ is played by ~ the musician M.",
  "conceptualise the band B ~ plays at ~ the time T and ~ plays ~ the song S and  and has the stage L as ~ venue ~.",

  "there is a stage named 'pyramid stage' that is in the location 'Main Stages'.",

  "there is a stage named 'other stage' that is in the location 'Main Stages'.",
  "there is a stage named 'west holts' that is in the location 'Main Stages'.",
  "there is a stage named 'the park stage' that is in the location 'Main Stages'.",

];
```

The policies have to be also declared in the models, therefore the web client & node server will have the following also included in their models:

- Web client

```
var streaming_node = [
  "there is an agent named 'Moira' that has 'http://localhost:5555' as address",
  "there is an ask policy named 'p1' that has 'true' as enabled and has the agent 'Moira' as target.",
  "there is a listen policy named 'p2' that has 'true' as enabled and has the agent 'Moira' as target."
];
```

- Node Server

```
• "there is an agent named 'agent1' that has 'localhost' as address",
• "there is an tell policy named 'p1' that has 'true' as enabled and has the agent 'agent1' as target.",
• "there is a listen policy named 'p2' that has 'true' as enabled and has the agent 'agent1' as target."
```

5.3.2 Node Server

CENode instances can be run as a web service by invoking them directly as a node app:

```
$ node cenode.js
```

In these cases, the CEAgent effectively exposes itself to the network and provides HTTP methods to interact with its CENode.

The *getTweet* , *getNameExact*, *getNameToken* and *getTime* functions are located within this file as functions.

5.3.3 Web Node

CENode in the setting of a web application that will allow a user to conduct a simple conversation with a local agent. [15][16]

The fist step is to create an index HTML file. It has two scripts one that refers to CENode Library and the other which refers to main.js which contains the logic for our application.

```
1.  <!DOCTYPE html>
2.  <html>
3.    <head>
4.      <title>CENode WEB</title>
5.    </head>
6.    <body>
7.      <h1>Glastonbury 2015</h1>
8.      <script src="js/cenode.js"></script>
9.      <script src="js/main.js"></script>
10.   </body>
11. </html>
```

Now that we have created a web application, we can now start using CENode. In the main.js the first step is to create an instance of CENode and load it with models that we require:

Our app logic within a file named main.js

- Initialise an instance of CENode
- *var node = new CENode(MODELS.CORE, bands_model);*

This creates an instance of CENode with the following models:

- MODELS.CORE
- Bands_model
- Streaming_node

node.agent.set_name('agent1');
This sets the name of the agent to agent1 which refers to the local CEAgent

The next stage was to create a message facility between the user and the CEAgent:
We are going to have a textbox for input, an area for the messages to be displayed and a button which when clicked takes the message and then processes it.

To achieve this I used simple HTML elements and added them to the index.html

```
<textarea id="input"></textarea>
  <button id="send">Send message</button>
  <ul id="messages"></ul>
```

The next step is declare a name for ourself (so the agent knows whom it is from), retrieve the references responding to DOM elements and create a function that will run when the user clicks on it. This is added to main.js after the node has been declared:

```
var my_name = 'User';

var input = document.getElementById('input');
var button = document.getElementById('send');
var messages = document.getElementById('messages');

button.onclick = function(){
```

The next stage is to create a function which on the user pressing the button takes the message inputted, create a CE card (if who/what/where then ask card otherwise NL card) and send it to the local agent, The card also needs to declare who it is from, so the agent can respond.

```
1.  function send_message(){
2.      var message = input.value.trim(); // CENode seems to need this
3.      input.value = ''; // blank the input field for new messages
4.      if (message == '') return; // don't submit empty messages
5.      var card;
6.      if (message.match(/^(who|what|where)$/)) {
7.          var card = "there is a ask card named '{uid}' that is to the agent 'agent1'
     and is from the individual '"+my_name+"' and has the timestamp '{now}' as timestam
    p and has '"+message.replace(/'/g, "\\'")+"' as content.";
8.      }
9.      else{
10.         var card = "there is a nl card named '{uid}' that is to the agent 'agent1'
    and is from the individual '"+my_name+"' and has the timestamp '{now}' as timestamp
     and has '"+message.replace(/'/g, "\\'")+"' as content.";
11.     }
12.     node.add_sentence(card);
13.     // Finally, prepend our message to the list of messages:
14.     var item = '<li class="'+my_name+'">'+message+'</li>';
15.     messages.innerHTML = item + messages.innerHTML;
16. };
```

My initial method of handling the networking between the two nodes was going to be by using policies however I was not able to get responses back from the server therefore I will be making use of the CENode API. Now the server has an endpoint located at the localhost:5555 and has a get method available that allows you to retrieve all cards intended for a particular agent. So we have to retrieve all cards addressed to ‘agent1’ from the KB of ‘Moira’

To achieve this I will be sending a GET request to Moira (localhost:5555) with the query string (cards?agent=agent1) sent in the URL of the request (http://localhost:5555/cards?agent=agent1), this will retrieve all cards stored in Moira’s KB addressed to agent1 (the web agent). The cards are returned in the form of line-delimited CE, which can easily be separated into individual cards by the use of the .split method into an array of cards. Now responses are in the form of a gist (human friendly response) and this is what we are interested so if the sentence contains the word gist then it is processed else (these are mostly the ask cards) they are not processed.

```
1.  function loadCard() {
2.    setTimeout(function(){
```

```
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (xhttp.readyState == 4 && xhttp.status == 200) {
        //document.getElementById("demo").innerHTML = xhttp.responseText;
        var response = xhttp.responseText;
        var cards = response.split(/\r?\n/);
        var results = [];
        for( var i = 0; i < cards.length; ++i ){
          console.log(cards[i]);
          node.add_ce(cards[i]);
        }
      }
    };
    xhttp.open("GET", "http://localhost:5555/cards?agent=agent1", true);
    xhttp.send();
    loadCard();
  },5000);
}
```

As previously mention CENode work asynchronously and therefore we do not want to want the application to be blocked while waiting for a response, to work around this we need to poll the node continuously to check for any cards that are written back to us.

This code is added to main.js

```
var processed_cards = []; // A list of cards we've already seen and don't need to process again

function poll_cards(){
    setTimeout(function(){
        var cards = node.get_instances('card', true); // Recursively get any cards the agent knows about
        for(var i = 0; i < cards.length; i++){
            var card = cards[i];
            if(card.is_to.name == my_name && processed_cards.indexOf(card.name) == -1){ // If sent to us and is still yet unseen
                processed_cards.push(card.name); // Add this card to the list of 'seen' cards
                var item = '<li>'+card.content+'</li>';
                messages.innerHTML = item + messages.innerHTML; // Prepend this new message to our list in the DOM
            }
        }
        poll_cards(); // Restart the method again
    }, 1000);
}
```

5.4 Issues & Challenges

5.4.1 GIST Response

The cards intended for agent1 from moira were in the form of a GIST card, which cannot be parsed by CENode, therefore unable to added to agent1's KB and subsequently answering the user's query. I spent a lot of time trying to modify the parse_question function in CENode to try to change the way that the responses were stored in moira's KB. After days of debugging, I was able to figure out the solution to this problem, unfortunately I ran out of time and was unable to implement the solution. See futures works (8.1 GIST Response) for the solution

5.4.2 Glastonbury Experiment

Looking back at the implementation of the experiment (7.2 Glastonbury 2015 Experiment), it was the most time consuming part of the project. The first challenge was getting the tweets from twitter. Usually to retrieve past data you would use the REST API but it cannot be used for data that is more than 7 days old. With platforms such as GNIP being for profit, I was therefore forced to look for scripts that would do the job for me; this was difficult, as twitter had changed the way their site worked and therefore most of the scripts I found were not functional anymore. Therefore, I decided to try to write my own using the beautiful soup library for Python. Initially I made some headway and was able to collect a few tweets but not those that required you to scroll down the page.

At this stage I was about to give up and create a pseudo experiment but by some luck I ran across Tom Dickinson's blog post about mining data directly from search and he even had a python implementation! The next stage was to get this mined data into a format that would be parsed by my JavaScript implementation of CENode and the processing of tweets. I decided the best way was to write these to a CSV file. This proved to be challenging as there is an issue that would allow the script to be run only once on the IP and this was so frustrating as I had to send the script to various friends when testing and collecting the data and they would email it back to me.

The next stage was to add the content of the CSV file to the database, which was easily done by me with another python script that I wrote. However, reading in the tweets to the JavaScript was frustrating due to the asynchronous nature of it. Running the processing functions meant that some of the function were carried out on a different tweet then the one intended, this meant I had to use many callback functions, which slowed the program down significantly.

6 Testing

6.1 Overview of Testing

For this project the testing will take the form of unit testing where the smallest testable parts of the overall program, called units, are individually and independently scrutinized for proper operation. It's usually automated but in this case I will be performing this manually. Each unit was tested using own JavaScript program which ran in NodeJs, except for the functions requiring networking which were tested using a web browser. I will be testing against the functional requirements as stated in the design section (4.6.1 Functional Requirements).

6.2 Twitter Intake

The first units to test are the twitter intake functions, which connect to twitter's API and then retrieve the desired tweets in JSON format

6.2.1 REST

REST → the rest API searches the twitter archives over the last 7 days

- Program Name: testREST.js
- Input (tweeted the following):
    - test streaming #fyp2016test
    - This a test of the streaming #fyp2016test
    - This a test of the streaming API #fyp2016test
    - Motorhead is playing at the park stage #fyp2016test
    - Jamie XX is playing at the park stage #fyp2016test
- Expected Result (text of tweets):
    - test streaming #fyp2016test
    - This a test of the streaming #fyp2016test
    - This a test of the streaming API #fyp2016test
    - Motorhead is playing at the park stage #fyp2016test
    - Jamie XX is playing at the park stage #fyp2016test
- Actual Result (JSON format):
    - test streaming #fyp2016test
    - This a test of the streaming #fyp2016test
    - This a test of the streaming API #fyp2016test
    - Motorhead is playing at the park stage #fyp2016test
    - Jamie XX is playing at the park stage #fyp2016test
- Status: Passed

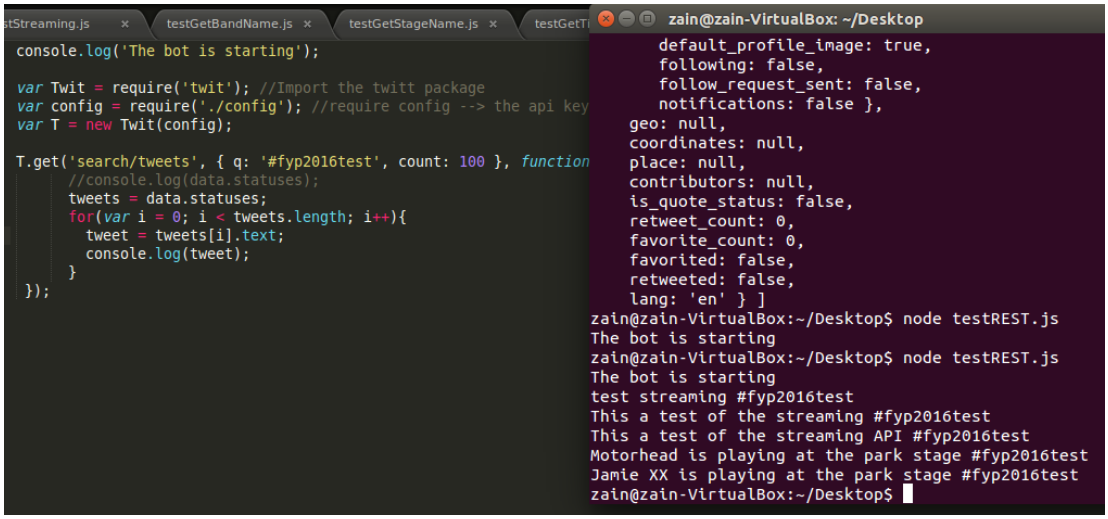

Figure 6 Shows the output when testing the get tweets from REST API functionality

6.2.2 Streaming

Streaming API: get tweets in real time

- Program Name:testStreaming.js
- Input (tweeted the following):
  - Test streaming #fyp2016test (the tweet was made at 9:53 AM on 03/05)
- Expected Result
  - Test streaming #fyp2016test
  - Tue May 03 09:53:00 +0100 2016
- Actual Result
  - Test streaming #fyp2016test
  - Tue May 03 08:53:29 +0000 2016
- Status: Passed (datetime given for +0000 gmt)

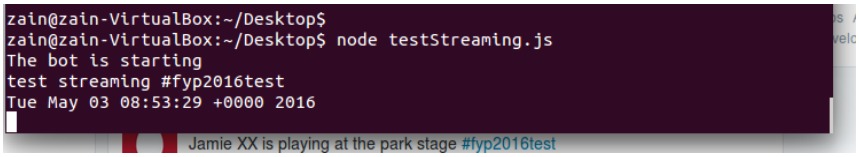

Figure 7 shows the successful retrieval of tweets from the Streaming API

6.3 Process Tweets

Process Tweets: the next stage is to test the all-important information retrieval functions

6.3.1 Get Band Name

- Program Name: testGetBandName.js
- Input: “Massive crowd for the Charlatans opening the Other Stage at 11am #glastonbury”
- Expected Output:
  - Using getNameExact → The Charlatans
  - Using getNameToken → The Charlatans
- Actual Output:
  - Using getNameExact → The Charlatans
  - Using getNameToken → The Charlatans

- Status: Passed

```
zain@zain-VirtualBox:~/Desktop$ node testGetBandName.js
using getNameExact --> The Charlatans
using getNameToken --> The Charlatans
zain@zain-VirtualBox:~/Desktop$
```

Figure 8shows the testing of the get band name functional requirement

6.3.2 Get Stage Name

- Program Name: testGetStageName.js
- Input: “Massive crowd for the Charlatans opening the Other Stage at 11am #glastonbury”
- Expected Output:
  - Using getNameExact →other stage
- Actual Output
  - Using getNameExact → other stage
- Status: Passed

```
zain@zain-VirtualBox: ~/Desktop
zain@zain-VirtualBox:~/Desktop$ node testGetStageName.js
using getNameExact --> other stage
zain@zain-VirtualBox:~/Desktop$
```

Figure 9 shows the testing of the get stage name functional requirement

6.3.3 Get Time

- Program Name: testGetTime.js
- Input: “Massive crowd for the Charlatans opening the Other Stage at 11am #glastonbury”
- Expected Output:
  - using getTime → 11am
- Actual Output:
  - using getTime → 11am
- Status: Passed

```
zain@zain-VirtualBox:~/Desktop$ node testGetTime.js
using getTime --> 11am
zain@zain-VirtualBox:~/Desktop$
```

Figure 10 shows the testing of the get time functional requirement

6.3.3.1 Test using regex

I also tested the regex by using the regex tester on regex101.com

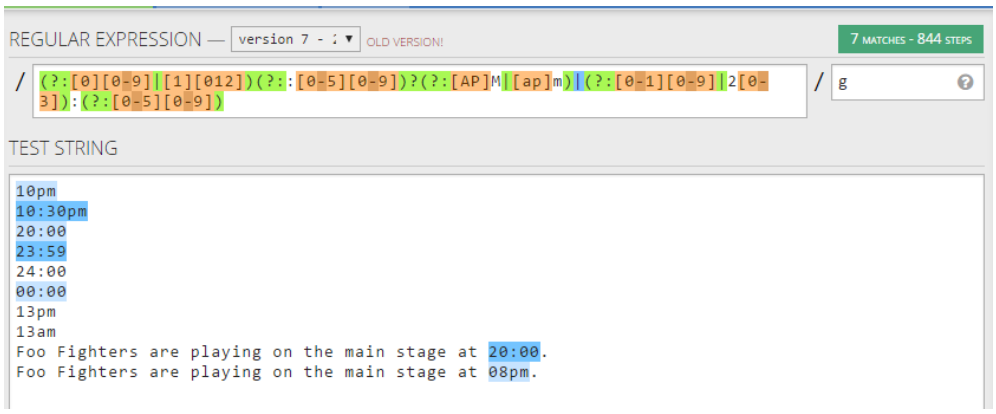

Figure 11 shows the testing of the regex for detecting time

6.3.4 Add CE sentence to node's KB

The next stage is to test the formation of a CE sentence, which can be added to the node's KB. This is done by extracting the band name, stage name and time. Then running .addSentence function on the node with the CE sentence as input.

- Program name: cenode.js (function named: addToNode)
- Inputs:
  - "test streaming #fyp2016test"
  - "This a test of the streaming #fyp2016test"
  - "This a test of the streaming API #fyp2016test"
  - "Motorhead is playing at the park stage #fyp2016test"
  - "Jamie XX is playing at the park stage #fyp2016test"
- Expected Result:
  - Added an instance of Motorhead is playing at the park stage to the nodes KB
  - Added an instance of Jamie XX is playing at the park stage to the nodes KB
- Actual Result:
  - Instance of Motorhead created
  - Instance of Jamie XX create
- Status: Passed

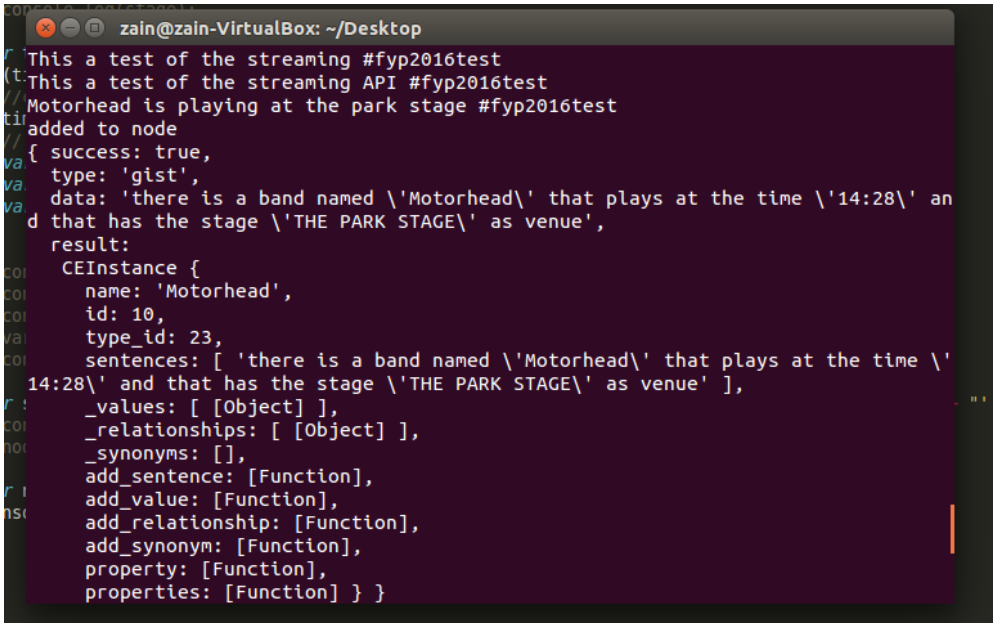

Figure 12 shows the successful addition of the Motorhead instance to the node

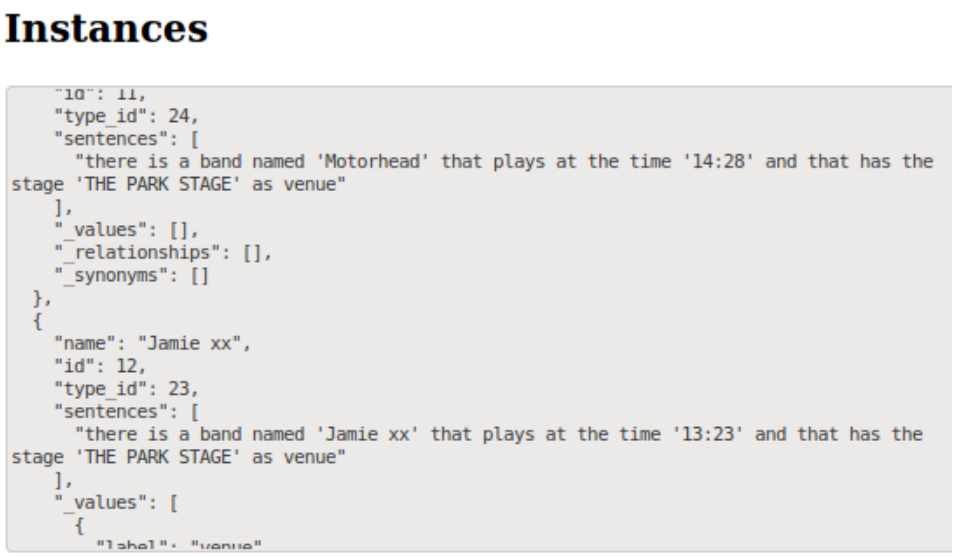

Figure 13 shows the instances of Motorhead and Jamie XX in the node's list of instances

6.4 Web Node Test

The following section is intended to for the testing of the web application and the web instance of CENode.

6.4.1 Take in user input

User input is processed by the node (testing against the web node's local KB)

- Input: who are the foo fighters?
- Expected Output: Foo Fighters is a band. Foo Fighters has the stage 'other stage' as venue and plays at the time '10:30pm'.
- Actual Output: Foo Fighters is a band. Foo Fighters has the stage 'other stage' as venue and plays at the time '10:30pm'.

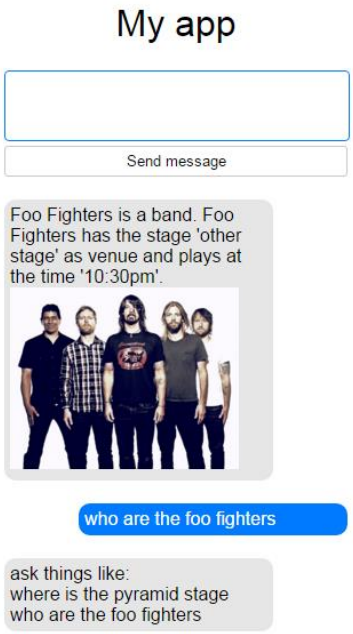

Figure 14 shows the successful test of user input against a demo

6.4.3 Process question

The next functionality is to test is that if the user asks a who/what/where question, then an ask card is created and this is sent to both the local agent (agent1) and an ask sent to the node server (Moira) due to the ask policy declared.

- Function to test: send_message()
- Input: who is Jamie XX?
- Expected results, (expecting two responses):
    - Ask card sent to the local agent (agent1) and response will be unknown as agent1 doesn't know about it
    - Ask card sent to the node server (Moira), which creates a response and stores it in its own KB.
- Actual Results
    - Agent1: "I don't know who or what that is"
    - Moira: Receives the card and creates an answer card and this is stored in its own KB

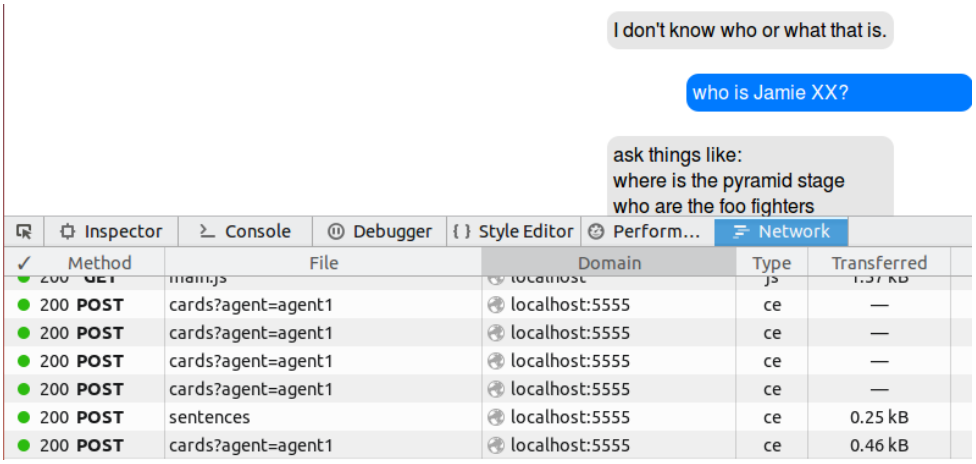

Figure 15 shows that the web application sends the message (Ask card) to the node server (POST --- sentences)

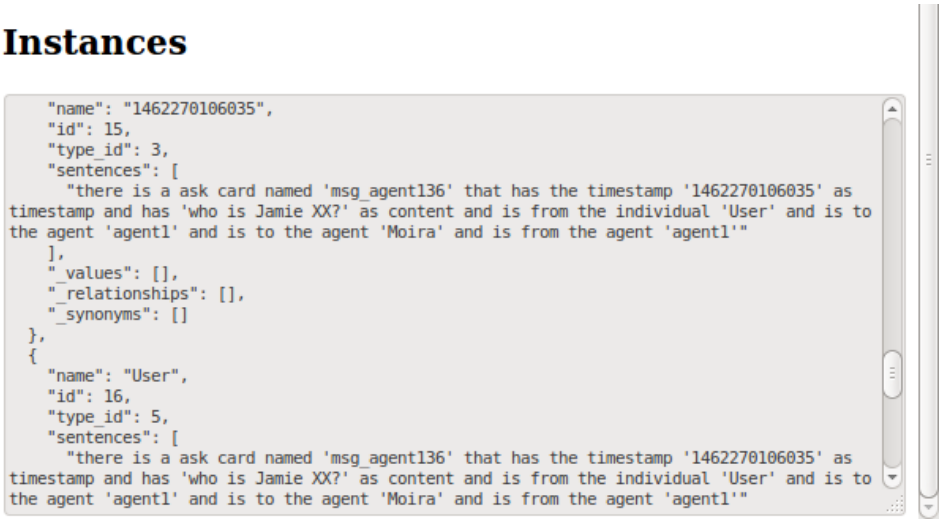

Figure 16 shows the server (Moira) receiving the ask card from the client (Agent1)

Figure 17 shows that server processes the ask card and creates a response (GIST)

6.4.4 Get response from server

The second to last stage of testing the functionality of the system is to retrieve the gist cards from Moira addressed to agent1.

- Input: Get Request (localhost:5555/cards?agent=agent1), we asked it “Who is Jamie XX?”
- Expected Result: Gets just the gist card intended for agent1 and outputs them to the console (testing purposes)
- Actual Result: Gist is outputted to the console
- Status: Passed

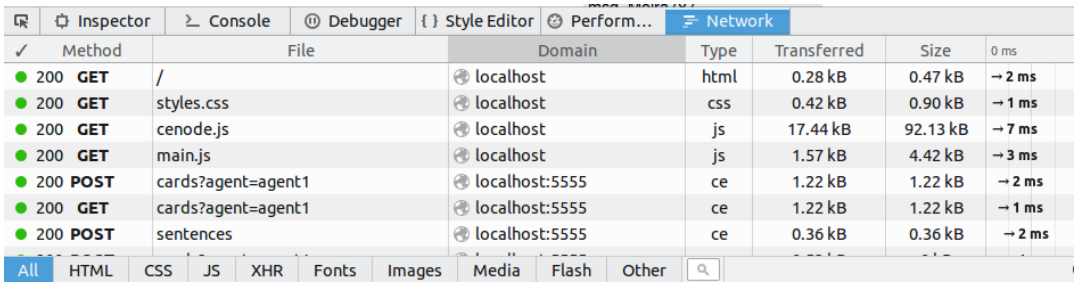

Figure 18shows the web application retrieving the cards from the server via a GET request

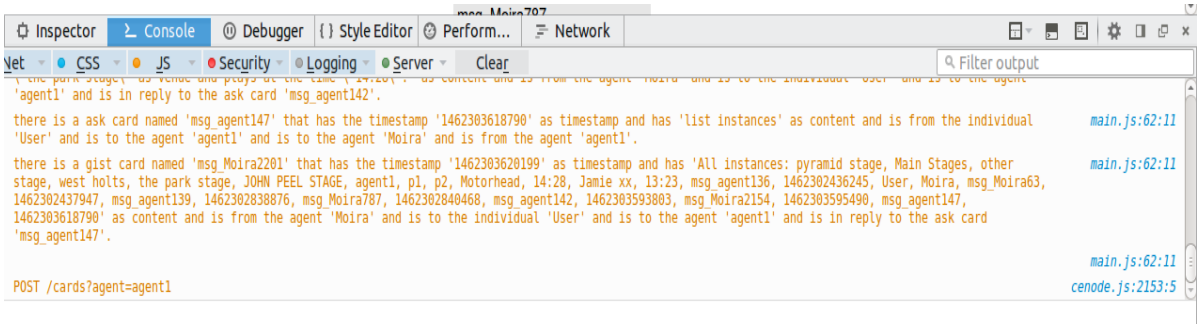

Figure 19 shows the cards being output to the console

6.4.5 Add card to node and display response to the user

- Function to test : poll_cards()
- Input: CE sentence “Jamie XX is a band. Jamie XX has the stage “park stage” as venue and plays at the time “13:23)
- Expected Output: Jamie XX is a band. Jamie XX has the stage ‘park stage’ as venue and plays at the time ‘13:23’.
- Actual Output: N/A
- Status: Failed
- Evidence: N/A (nothing to be displayed, therefore test is assumed to have failed

6.5 Summary of Testing

The unit testing was conducted to see if the functional requirements for the project were met. The parts of the system that worked were as follows:

- Get tweets
    - The program is able to collect tweets relating to the hashtag specified
    - The system was stable
- Process tweets
    - The system was able to extract the text of the tweet
    - The system was able to get the band name
    - The system was able to get the stage name
    - The system was able to retrieve the time of the performance
    - The system was able to form a valid CE sentence that was correctly parsed and an instances of band, stage and time were created.
- CENode (Server)
    - The server was able to receive and process CE that was sent to it
    - The server was able to update it’s KB when it received a tell card

    - The server was able to reply with the correct response (in term of content) but not in the correct format (creates a GIST card instead of a CE card)
- CENode (Web)
    - The web application takes in user input correctly
    - The web application was sccusefully able to query the server for a response to a question asked in the form of sending an ASK card to the server
    - The web application wasn't able to display this knowledge back to the user.

The only functionality which failed (which was the most important out of all) was the system needs to display this knowledge back to the user. The response from the server was in the form of a gist card, which cannot be parsed by CENode and therefore is not added to the node's KB. The card not being added means that the poll_card() function doesn't receive the card and therefore it's not displayed back to the user using the HTML elements. This can be fixed by having the server return a CE card (8.1 GIST Response), which would then will be processed by the node and displayed back to the user via the HTML elements as a result of poll_cards().

7 Results, Experiment and Evaluation

7.1 Final System

The final implementation of the system was able to take tweets in real time/from archives. It was then able to query twitter correctly and retrieve relevant tweets. It was then able to extract relevant information from the tweet and store in its (server's) KB. The web application was able to take in user input and query the server's KB for a response. The server was able to receive this information and create a response with the correct content but in the wrong format (GIST instead of CE), which when the response was retrieved by the client, it couldn't be processed (added to the web node's KB) and therefore wasn't able to be displayed back to the user.

7.2 Glastonbury 2015 Experiment

7.2.1 Aim

The main aim of the experiment was to test the accuracy of the system in terms of how accurate is it at recognising the correct band. The secondary aim of the experiment is to determine whether or not we can gather enough knowledge in a real world scenario.

7.2.2 Design and Implementation

I decided to test the system by using Glastonbury 2015 as the domain. This is because of the large scale of the event, there will be a very large number of people attending (135,000 in 2015). To carry out a version of a controlled experiment, I decided upon using tweets relating to the Friday (June 26, 2015). Another condition that I added was that the tweets be located from Glastonbury, this was done to remove the very large number of tweets that could be categorised as sentimental.

To create the query, I used the advanced twitter search to create a string to query the information I required.[17] The query is as follow "#glasto OR #glastonbury OR #glastonbury2015 lang:en near:"Glastonbury, England" within:15mi since:2015-06-26 until:2015-06-27". The query broken down [18]:

- Hashtags: #glasto , #glastonbury or #glastonbury2015
- Language of tweets: English
- Location: Within a 15 miles radius of Glastonbury

- Time Period: Start of Friday till the start of Saturday

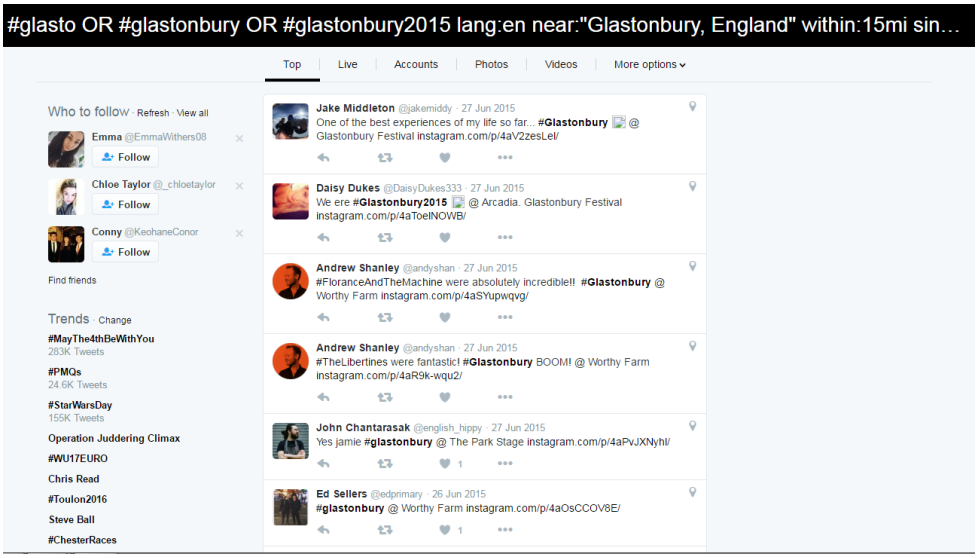

Figure 20 shows the tweets that met the criteria of the search query

Implementation

The tweets and the metadata were retrieved by using a scraper that scrapes directly from twitters search page written in python by Tom Dickinson with a small addition of my own, which was to write the text of the tweet, time and date to a CSV file. [23]

The next stage is to take the data from the CSV file and enter this to a SQL database so it could be read by the system to extract the information from the text and if a valid CE sentence is formed added to the node's KB. This done was in python due to my previous experience in using CSV files and I used SQL again due to past experiences.

The final stage is to retrieve the each tweet (text,date,time) from the database and process them one at a time to extract information. If the tweet contained enough information to form a CE sentence then this was add to the node server.

7.2.3 The Information Gathered and Evaluation

I was able gather 150 tweets that matched my query. These were then processed by the node and of the 150 only 20 were able to be parsed. So they referred to an artist, had an indication of the stage that was the venue for the performance. Having been a small data set I was able to manually confirm that 20 instances should have been added to the KB.

The accuracy of the results was another issue, with the some of the bands were improperly identified.

7.2.4 Evaluation of the Results

Yes you are able to gather knowledge about what's going at a festival at least from a performing references. This experiment showed that if the program was run live at Glastonbury 2015 we would have been to identify which band was playing where and at what time fairly accurately.

7.3 Aims of the finished product from initial plan

In the initial plan I documented that final product must have the following functionality:

- Working knowledge gathering

    - Collect data from twitter
    - Process this data to enrich it semantically
    - Store enriched in a knowledge base

My final system was able to collect data from twitter. Once the data had been collected in JSON format, it was processed to extract information from it and then that information was used to create a CE sentence. The CE sentence was used in conjunction with CENode to store in a KB.

- Working application to access this knowledge
- Mobile Application
    - Develop application that uses the CENode API

I was able to implement a web CENode. The web application provided functionality that allowed a user to input a sentence, which would then be processed by CENode, with the outcome dependant on the users input. So if a user asked a who/what/where question then CENode would send that message to itself and the server for response. However my implementation was unable to display the response from the server to the user due to the response being in the incorrect format (See section for more information).

I also state that the system should be able to

- to support for a single kind of event (e.g. festival or other big public event)

My system was able to support an event of a single type in this case being a festival. This is due to the use of CENode and the model I developed for it. This gave the node some background knowledge and as long as the user provides a list of performers and the venues, it can detect who performing where and at what time.

7.4 Aims of the research from the initial plan

My initial plan was to answer the following question:

- Do people tweeting about a particular event actually reflect what's going on in the real world?

From the initial reports feedback was that this was slightly ambiguous. What the true meaning was that do people tweet about what's going on at the event rather than their reaction to it. The answer to this question is that people tweet more about their reactions to the event and them being at the event rather what's going in the event. This is evident from the Glastonbury 2015 experiment that I carried out. Out of the 150 tweets that were retrieved using my query on 20 contained enough information that could be extracted from the tweet to be added to a KB.

7.6 Use of JavaScript/NodeJS

The use of JavaScript as the main implementation language for the project was in my opinion a suitable approach. NodeJs allowed me to use the node package manager to install libraries very easily and use them without conflicts between the various packages. The wide range of packages also available meant that I was able to use packages to implement the more difficult parts with ease.

7.7 Web Application

The decision I took to implement a web application that allowed a user to query the KB was done to provide an example for the use of the system. There is no point in gathering knowledge about an

event if it's not going to be share. By creating the web application it gives an example of how the knowledge can be used in a real world scenario. For example it can allow users to find out about the status of facilities at a large scale event and maybe even alert the events team about what's going on the ground.

7.8 Summary of Results, Experiment and Evaluation

I created a system that met the aims that I outlined for the product at the start of the project in my initial plan. It's not fully functional due to a minor technical difficult which can theoretically be easily solved given more time. I also stated that the system should be designed in a way that it was able to cater to a certain type of event and I achieved this by modelling a music festival.

I was able to test the functionality of the system in a real world scenario by performing an experiment that used actual tweets collected from Glastonbury at the ground to test the data extraction aspect of my system. It was made to cater to festival given that the user provides band name, stage name the system will then be able to parse tweets and extract data relevant to that particular festival. This was successful as I was able to get meaningful results that were fairly accurate and answering my research question, that tweets are an accurate representation of what's going on the ground.

8 Future Works

8.1 GIST Response

As described in implementation and testing. The system I implemented failed the functional requirement that stated that the system should display the knowledge back to the user. This was due to CENode server responding the form of GIST card, which cannot be parsed by the node to create an instance in it's local KB. To solve this issue, I have identified the following function in the CENode library that needs to be modified. parse_question()to return the instance in CE instead of gist (i.e. call instance.ce, and not instance.gist). That way the information returned by `Moira` will instead be CE and not GIST. That would solve the problem of the response from the server not being displayed to the user because then the card is added to `agent1`'s KB and poll cards will get it and output it eventually.

8.2 Improve band detection

The method for detecting the band names can be improved. Currently it requires the user to specify the band names and my search name function is able to correctly guess the band name to a 4 differences in characters, so it misses out on abbreviations etc.

Another thing that I tried to implement was the ability of using MusicBrainz (music database) to detect the artist. If the function is able to use this instead of a pre-defined list it would then be able to pick up surprise acts which often occur at festivals.

8.3 Modelling of Facilities

A festival such as Glastonbury with over 100,000 attendees must be very hard for the events staff. They have to deal with running facilities such as showers, toilets, food stands and water points. By incorporating this into a model, you could theoretically track the status of the facilities. For example you could have a model stating with the following CE sentence in the KB "there is a toilet named 'T1' that has the status 'busy' and has the location 'area1' as location." This would enable a user to ask

the web application questions like “where is T1” and the events team can find out what facilities are busy by simply inputting “list instances of type busy”.

8.4 Use of other social media sites

Facebook had 1.65 billion monthly active users, which is a lot higher than Twitter’s 305 million monthly active users. This would greatly increase the no of posts that matched my query and therefore the higher of data points would lead directly to a higher amount knowledge gathered. FBgraph is a NodeJs module that provides easy access to the Facebook graph API

Running my Glastonbury experiment, I noticed that many of the post that were geo tagged were result of people posting their images on Instagram. Instagram allows you to add a caption and location to the image a user is posting as well as the ability to share the post on other social media websites such as Facebook and Twitter (where I was able to retrieve these posts). There are several packages available for node that allow access to the Instagram API such as instagram-node.

8.5 CENode Temporal Querying

Temporal querying has not yet been implemented. CENode is still only really a research-grade prototype designed to fulfil a small range of functionality. So for example the end user should be able to ask it questions like `what time are X playing?`. Currently you can get the time by asking it and can get time by asking `"What does Foo Fighters play at?` and it responds with `Foo Fighters plays at the time 10.30pm`. To implement temporal querying you would have to modify the `parse_question` function to handle ‘when’ questions, currently it has functionality to hand what/where/who questions.

9 Conclusion

The aim of the project of this project was to investigate could we gather knowledge about an event as it was happening from social media. To investigate this I chose to investigate Glastonbury, the music festival due to the sheer number of people who attend (120,000 in 2015), which would guarantee people posting about the event on social media. I chose twitter as the means of collecting data due to it’s unrestricted nature (tweets are public by default). As I was attempting to gather knowledge about a music festival, I was aiming to extract the name of the band, the venue they were performing at and the time of the performance and store this in a knowledge base.

Knowledge is useless without a means to model and query it, therefore I made use of CENode, a JavaScript implementation of CEStore, which allows for domain modelling and natural language (NL) to CE interpretation. It allowed me to create model for the domain (music festival) and store the knowledge, from tweets that had been processed, in the form of CE sentences.

Implementation

I made use of NodeJs for implementing this project due to CENode being implemented in JavaScirpt and the availibity of various packages. The implementation was done that two CENode instances were created, one acting as the server and the other as the client. The server took in tweets, processed them and stored them in its knowledge base. The purpose of the client CENode was to take in user input and transfer the query to server which would then respond with an answer to the question and the response displayed back to the user. I was unable to display the message to the user via HTML (able to output to console) due to the the response being in the worng format(GIST instead of CE). This can in be theoretically fixed by altering one function in the CENode library.

I was able to test the information extraction part of the system by conductiong an experimenting that collected made during Friday of the Glastonbury festival by people at the actual event and I was able to gather 150 tweets that met my search criteria, out of which 20 contained extractable information to could be added to the knowledge base. This

The system largely worked well .The ability to extract the performer's name, the stage at which they were playing and the time of the performance worked. The web application which was designed to act as a messaging service between the nodes was able to take into user input and query the server's knowledge base and retrieve the response back but was not able to display this response back to the user via HTML.

To summarise the project overall, I was able to gather knowledge from social media (Twitter) by taking in tweets , processing them and then storing them in a knowledge base and then able that knowledge to be made available to the end user.

10 Reflection

This project has been far more difficult than I had expected. Having previously conducted a research based software development project, I expected that I would be able to follow my plan for the project as stated in my initial plan, thinking that problems would just require a bit of time and research to solve. This was not the case. Some aspects of the project a lot longer than I imagined such as implementation of the Glastonbury 2015 experiment. The collecting, storing and processing of the tweets took me a lot longer than original, which took me a good week to collect and use. Looking back on the project, I think my expectations for myself for this project were, with hindsight, far too ambitious. Using JavaScript as the programming language of choice, which I had a basic understanding and had not worked with extensively with over two years; put me outside of my comfort zone and meant when it came to implementing I had to look up a lot of things and understand how they work before implementing them, which took time. This meant I ate up some of the time intended for report writing and was still attempting to debug the web application while writing the report.

I believe that I have developed my technical skills whilst working on this project. First of all before commencing this project I had very little experience with JavaScript and asynchronous programming. Therefore, I spend learning the basics via the use of online tutorials and learnt about important concepts such as callback functions and how to use GET/POST request properly. Another important skill I believe I have learnt is how to interact with APIs and use them in my work. Previously in my projects, I have always had to implement things myself and working with APIs was different as the responses were not as I would have liked them and had to work around that as well as poor documentation. This skill (working with other people's code) I feel is going to come in handy when I start working as a developer.

Another area I believed I have developed in is my communication skills. Working with a supervisor and Will meant that who did not have a deep understanding of the project meant I had to ask him precise questions as to how CENode worked in order to avoid confusion, which would lead to wasted time. Slack, the team communication tool, which we used during this project worked really well in my opinion. Having all the communication in one place and combined with the ability to integrate GitHub and send each other code snippets meant that it helped to increase my productivity.

Working with a supervisor Alun was something I really appreciated. His knowledge and suggestions when things got tough were helpful. The fact we met regularly helped me to keep track of what I had achieved so far and what I needed to work on. I really enjoyed the fact that no-one was telling me what to do instead just advising me and letting me figure out what I what I wanted to do. This also made me realise the importance of having regular meeting with a supervisor where you review the work you have done, allows to see if the work you have been is correct i..e. does it reflect your aims/objectives.

Although the project wasn't without difficulty I am satisfied with the current state of the system. Although it failed to meet the functional requirement of displaying knowledge back to user, I am pleased because as a single student working on this alone there was only so much I could have done with the time and my ability. The system is able to collect tweets, process them and store them in a knowledge base and can be developed further to do improves its functionality and increase its capabilities. Personally I have developed technical skills and learnt techniques on how to work in a professional manner which will come in handy when I move on to working life.

11 References

References

[1]A. Preece, W. Webberley and D. Braines, "Tasking the tweeters", 2015.

[2]A. Preece, W. Webberley and D. Braines, "Conversational Sensemaking", 2015.

[3]"Twitter", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Twitter. [Accessed: 10- Mar- 2016].

[4]"Social media", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Social_media. [Accessed: 10- Mar- 2016].

[5]"The Boston Bombing: How journalists used Twitter to tell the story", *Twitter*, 2013. [Online]. Available: https://blog.twitter.com/2013/the-boston-bombing-how-journalists-used-twitter-to-tell-the-story. [Accessed: 10- Mar- 2016].

[6]"Intelligence cycle management", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Intelligence_cycle_management. [Accessed: 22- Mar- 2016].

[7]"REST APIs | Twitter Developers", *Dev.twitter.com*, 2016. [Online]. Available: https://dev.twitter.com/rest/public. [Accessed: 19- Mar- 2016].

[8]"The Streaming APIs | Twitter Developers", *Dev.twitter.com*, 2016. [Online]. Available: https://dev.twitter.com/streaming/overview. [Accessed: 20- Mar- 2016].

[9]W. Webberley and A. Preece, *cenode.js documentation*, 2nd ed. 2016.

[10]"Data flow diagram", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Data_flow_diagram. [Accessed: 04- Apr- 2016].

[11]"MusicBrainz - The Open Music Encyclopedia", *Musicbrainz.org*, 2016. [Online]. Available: https://musicbrainz.org/. [Accessed: 20- Mar- 2016].

[12]"jbraithwaite/nodebrainz", *GitHub*, 2016. [Online]. Available: https://github.com/jbraithwaite/nodebrainz. [Accessed: 20- Mar- 2016].

[13]"ttezel/twit", *GitHub*, 2016. [Online]. Available: https://github.com/ttezel/twit. [Accessed: 10- Mar- 2016].

[14]"Get time from text", *Stackoverflow.com*, 2016. [Online]. Available: http://stackoverflow.com/questions/36570600/get-time-from-text. [Accessed: 10- Apr- 2016].

[15]W. Webberley, "CENode API Documentation", *GitHub*, 2016. [Online]. Available: https://github.com/flyingsparx/CENode/blob/master/API.md. [Accessed: 27- Mar- 2016].

[16]"CENode Getting Started", *GitHub*, 2016. [Online]. Available: https://github.com/flyingsparx/CENode/blob/master/docs/getting_started.md. [Accessed: 10- Apr- 2016].

[17]"Twitter", *Twitter.com*, 2016. [Online]. Available: https://twitter.com/search-advanced?lang=en. [Accessed: 10- Apr- 2016].

[18]"Glastonbury Tweets Search", *Twitter.com*, 2016. [Online]. Available: http://tinyurl.com/h5ltfkz. [Accessed: 10- May- 2016].

[19]D. Laney, "3D data management: Controlling data volume, velocity, and variety.", 2001.

[20]"IBM Controlled Natural Language Processing Environment", *Ibm.com*, 2016. [Online]. Available: https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=558d55b6-78b6-43e6-9c14-0792481e4532. [Accessed: 11- Feb- 2016].

[21]M. Srivastava, T. Abdelzaher and B. Szymanski, "Human-centric sensing", *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 370, no. 1958, pp. 176-197, 2011.

[22]M. Osborne and M. Dredze, "Facebook, Twitter and Google Plus for Breaking News: Is there a winner?", 2014.

[23]"tomkdickinson/Twitter-Search-API-Python", GitHub, 2016. [Online]. Available: https://github.com/tomkdickinson/Twitter-Search-API-Python. [Accessed: 14- Apr- 2016].

[24]T. Sakaki, M. Okazaki and Y. Matsuo, "Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors", 2010.