CEBOT: ACCESSING KNOWLEDGE IN SLACK VIA NATURAL LANGUAGE

By

ANDREW DAWSON

Supervised by Prof. Alun Preece. Moderated by Prof. Paul Rosin

Submitted to the Faculty of
Cardiff University
School of Computer Science and Informatics
in partial fulfilment of
the requirements for
the Degree of
Bachelor of Science in Computer Science
6th May 2016

1 Abstract

The growth of the internet and increasing globalisation has spurred development in remote collaboration. We investigate the leading cloud based collaboration software and determine that it contains a wealth of information that could be exploited to improve a team's productivity and transparency. We propose a natural language based conversational approach to discover hidden trends within the collaboration environment and to support Human Computer Collaboration (HCC).

The approach uses a Controlled Natural Language (CNL) and external Natural Language Processing service (NLP) to parse a user's Natural Language (NL) input, convert it to CNL, query the NL knowledge base (KB) and output it in a suitable graphical or conversational based style.

We create a platform that successfully integrates a NL KB with a collaboration environment, and demonstrate examples of Human to Machine, Machine to Human and Machine to Machine services. The platform is designed in such a way as to be highly extensible and we go on to identify areas for immediate extensibility including relevant research and existing techniques in the NLP area that could be exploited in future iterations.

2 Acknowledgements

I would like to express my deepest gratitude to Professor Alun Preece for his help and guidance throughout this project.

I would also like to thank the Dave Braines, Will Webberly and Anna Thomas for providing their invaluable Slack team collaboration data and for their feedback on the system.

3 Table of Contents

4 Table of figures

5 Introduction

In the last few years Slack (a cloud based collaborative working environment) has created waves in the software industry, not just for its explosive growth but also for the benefits that teams using it are experiencing. Slack comes with the Slackbot, a simple chatbot analogous to a much less irritating Microsoft Clippy, designed to provide help to new users and reminders to existing users.

Slack is a rich source of social and collaborative data however much of it is hidden from the user in the general 'noise' of the conversation. Our aim is to take the data that Slack already stores about a team and analyse it to uncover hidden high level trends, such as which team members collaborate most frequently with one another.

To allow the team to access these insights we will integrate CEnode with Slack via their custom integrations so that Slack teams can benefit from conversational interaction with a natural language knowledge base.

We evaluated several different methods of using custom integrations to implement our knowledge base, eventually designing Cebot, a bot user powered by Slack's API. Cebot uses an external service to greatly expand CEnodes current Natural Language Processing abilities. This service allows him to effectively support very accurate Human to Machine communication.

The complex queries that we could now ask Cebot to perform became too convoluted to effectively display using Slacks default text message formatting. We used an external service to dynamically create and host interactive graphics, supporting effective communication between the machine and the user.

We assessed the accuracy and usefulness of NLP on historic user messages, attempting to extract a summary of concepts discussed and sentiment displayed over a specific timeframe. From this we determined areas which could benefit from further research, and an approach that could provide a simple summary of the most important points from a selection of user's messages.

6 Background

6.1 The wider context

Slack is an online messaging service for teams that share a common goal or interest. Since its launch in late 2013 it has generated significant 'buzz' and not just in the computing industry. Headlines such as "Slack, the Office Messaging App That May Finally Sink Email" [Manjoo 2015] and "Beyond email: could startup Slack change the way you work?" [Kiss 2016] are common place when searching for Slack news. In May last year Slack became the fastest startup to reach a $2billion valuation [Baer and Gould 2016]. Slack is "already being used by 77 percent of Fortune 100 companies in some way" [Kahn 2016], along with prestigious companies such as Harvard and NASA so it is clear that it must be more than just the hype that has drawn users in.

Slack functions in a similar way to Internet Relay Chat (IRC), infact the first version was based on a modified IRC client. Teams are made up of individuals that are closely linked in some way, in most cases these will be individuals that all work for the same company. Each team is provided with a web address to access Slack e.g www.cetesting.slack.com. Communication between members on a team largely takes place on 'channels'. Teams create channels to organise collaboration and communication between members, so in a small game design company there may be channels for marketing, human resources and level design. Inside a channel users communicate by posting messages, sharing files, creating 'posts' (Documents hosted by Slack that are editable in a similar way to Google Docs) etc. Users can also communicate via Direct Messages (DM's) for one to one private conversations.

So what makes it so popular? First of all it's free for an 'unlimited' number of users (the limit appears to be 8462 users [Kim 2016]). The signup takes less than two minutes and no special software or applications are required (although there apps for nearly all major platforms), which means that individual business areas can start using Slack without having to wait for their IT department to configure it. The user interface is also very 'clean' and responds quickly to input. An example of the user interface can be seen in Figure 1.

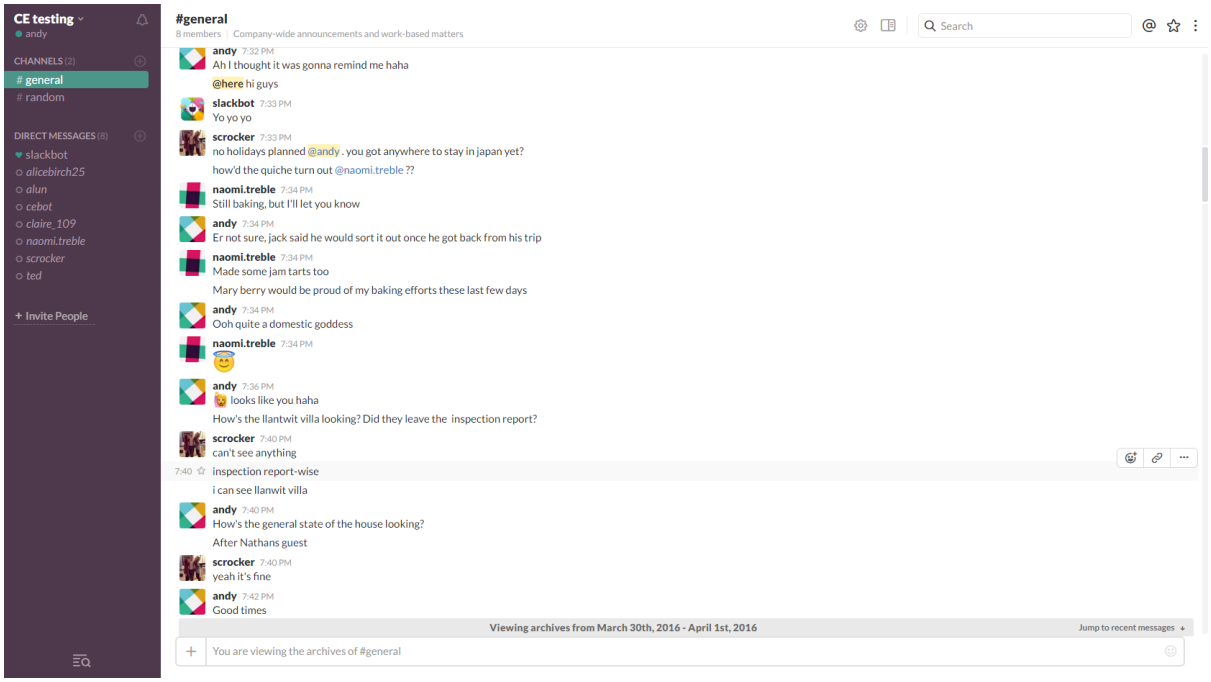

Figure 1 The Slack environment as viewed through their website

The main reason for Slacks success however is the change in culture in teams that use it. To gather evidence of the anecdotal benefits they were hearing about, Slack conducted a survey of over 1000 administrators/owners of paid Slack teams. In addition to the results shown in Figure 2 80% of owners thought Slack had increased transparency in their team, 79% thought Slack had improved the culture of the team and 62% thought it had made finding information easier [Slack 2015].

Figure 2 Benefits of using Slack as reported by their users Source (Slack Survey Results – July 2015)

The most negative statistic to come out of the report is that 1 in 4 people thought that Slack had not made finding information any easier

6.2 The problem

Slack is great for collaboration and as a record of communication among a team, but as their own report shows search is an area that needs improvement. It's often hard to find specific things (documents especially) and even harder to aggregate bits of information to make sense of what's going on in the environment: who talks to whom, about what, when, and what the tone is, etc. This problem is aggravated on a busy channel because, similar to Facebook Messenger, the Slack UI only loads the most recent messages. Scrolling up above a certain point triggers loading of the next batch of messages, which then reorganises the message window. In a busy team trying to find out what happened last week can be a

frustrating experience. There are also situations where you know a specific person uploaded a file but you can't remember what it was called, or someone talked about a particular subject but you can't remember who. This makes the information particularly hard to find using Slacks existing search, it has essentially become lost in the 'noise' of the channel. This problem is compounded by the high numbers of messages that Slack processes, our testing channel contains more than 1500 messages, nearly all of which are from 1 person.

6.3 Why address this problem?

Slack an excellent tool and its popularity is growing, paid users more than tripled in one year [Constine 2016]. Slack's API's are also very well documented and they actively support the community in developing helpful extensions to the Slack environment. It could be a really fantastic knowledge management tool if only there was a way to unlock what's going on at a "higher level", aggregating conversational data to find trends that would go unnoticed at a lower level and remain lost in the noise of the conversation. Slack contains a rich dataset on how humans collaborate on specific tasks which we can exploit to try and add value to the service. Can we make it better as an environment for knowledge management, tapping into both *explicit* knowledge such as what people post and talk about, and *tacit* knowledge such as people's social networks and mood.

6.4 Stakeholders within the problem area

Small to medium sized Slack teams who are actively focusing on one particular area/problem in detail. The teams will probably use Slack as their primary method of electronic communication and if they are separated by location maybe even their primary method of overall communication. Not only is the system more useful on an engaged team sending hundreds of messages a day, a more 'reserved' team who only use it infrequently will not suffer the problem we have identified, so it is those teams who we will be focusing on supporting.

6.5 Theory associated with the problem area

CNLs are typically a part of the solution to the broader problem of Human Computer Collaboration (HCC). HCC is collaboration "involving at least one human and one computational agent" [Terveen 1995]. As [Terveen 1995] notes HCC draws on many disciplines as varied as linguistics and psychology. Its two main disciplines are Artificial Intelligence (AI) and Human Computer Interaction (HCI). He states "From AI, it draws knowledge representation and reasoning techniques and a commitment to formal computational modelling. From HCI, it draws interaction and information presentation techniques and an awareness of the asymmetry in the abilities of people and computers. " In our system the AI area is represented by CEnode which will store our model of the Slack environment and the knowledge we have gathered from it. The HCI portion will be based on what we as developers can build within the restrictions of the Slack environment, at the very least we should be able to support plain text and pictures. We need to be mindful of the ways in which humans prefer to receive data for example, people tend to associate bar charts with comparisons of individual data points and line charts with showing a trend based on the collection of all those data points [Zacks and Tversky 1997].

The users of our system will already be communicating with each other in natural language conversational format. To support the principles of human-machine interaction we should allow our users to communicate with the system in the same way as they would

communicate with each other. This approach has been made feasible thanks to lengthy research in this area by the AI community [Allen et al. 2001] and the mainstream popularity of speech recognition software in mobile phone operating systems (Siri, Google now, and Cortana). Not only are these systems easy to use, users report a pleasurable feeling when their needs are met by a well-designed system [Weiss et al. 2015]

Despite the success of these platforms a NL conversational interface remains a hard problem, one that we hope to make easier through the use of a Controlled Natural Language (CNL). "CNLs are engineered languages which use a selection of the vocabulary, morphological forms, grammatical constructions, semantic interpretations, and pragmatics which are found in a natural language such as English." [Whyner et al. 2009]

CNL's are typically used in computer science as a way to make a language easier for a computer to parse whilst still being easily comprehended by a human speaker of that language. This is achieved by basing the controlled language on a restricted model of the original language, reducing the complexity and ambiguity that is found in many natural languages.

ITA Controlled English (CE) is a type of CNL. "It is unambiguous by using only words that are defined by the analyst as part of a conceptual model. CE therefore serves as a language that is simultaneously understandable by human and computer system, thus facilitating the communication between them." [Mott 2010]

The machine-processability of CNLs makes them well-suited to support interaction between humans and computers [Preece et al. 2014]. Using a CNL over say SQL provides two main benefits to the user. Firstly as CEnode is based on their natural language the time required to learn it is minimal compared to the complexities of formal programming languages. Secondly the underlying data is stored in the same CE format, meaning if a user can 'talk' to the machine they can also 'listen' and understand its response [Whyner et al. 2009].

CEnode is a lightweight JavaScript implementation of CE. CEnode is standalone and does not require a web connection. CEnode can be used to build an application that stores its information in a natural language repository called a knowledge base (KB). This KB can be added to and queried using CE, and responses can be provided in CE which are easily comprehensible by users. This means that non-technical users of the application can access information only previously available to those with expert knowledge in technical languages such as SQL.

CEnode's KB is made up of 'concepts'. These concepts make up the ontology for the KB. For example the following sentence creates the teacher concept as a subtype of person: `conceptualise a ~ teacher ~ F that is a person.` [Mott 2010]

A record of an individual concept is called an instance. This sentence: there is a teacher named Frank will create a new instance of the teacher concept called Frank. Properties can be added (if they have been defined in the concept first) with a sentence such as `: the teacher Frank has 'Butcher' as lastName.`

CE also supports relationships and logical rules between concepts. We can use all of these functions to represent knowledge taken from the Slack environment in a similar conversational style to the one that Slack users will already be using in their teams.

6.6 Constraints on the approach

The system must allow a slack team member to ask questions about the slack environment from within Slack and receive a result in the same Slack channel the question was asked.

6.7 Existing solutions relevant to the problem area

As mentioned above we want to implement a conversational approach to accessing the hidden knowledge located in the Slack environment and we want this approach to be well-integrated into the Slack environment itself. There should be no disconnect between the user communicating with team members and communicating with our system. Google and Siri are powerful, easy to use and popular NLP systems so they make good models for us to follow but neither are integrated in the Slack environment nor can they give us access to its hidden knowledge.

The Slack App Directory was launched in December last year, with 150 apps [Yeung 2015]. As the Slack user base has grown, so has its App Directory. Searching for relevant solutions in this problem area using this directory was difficult. The site looks similar to Apples App Store but whereas that includes pictures and videos, Slack's directory just includes a short summary, which is often not entirely clear.

For example the app 'Relay' description begins with 'Stream important user engagement activity directly to your team's Slack channels' [Slack 2016a]. But this isn't measuring Slack user engagement it is measuring consumer engagement for a service called Stripe and simply displaying that in Slack.

There are apps that encompass some of what we are trying to achieve or include features we would like to emulate. For example Ideabot. Users of this app can type `/idea [your idea here]`. They can then search for their own or others ideas by typing `/idea –search [keyword]`. This app includes a search function but only for its own limited list, not for the actual messages sent on the channel [Slack 2016b].

The app 'Awesome' includes a /recap command. This is a weekly highlight reel of what's important to your team. The problem with this approach is that users must manually 'highlight' what is important themselves, Awesome just keeps track of the list to show you at the end of the week [Slack 2016c].

'Nurtz' is an application that is described as "Get a good human professional editor who will swiftly refine and polish your text without having to leave your Slack window." Users can submit a section of text, for example an email, and the bot will proofread it. It promises an almost real time response which indicates that it may be a computer system using NLP to check grammar, punctuation etc. but the first line of the description indicates that the editor will be a human. Either way as it is not an open source solution we can't actually use it to help achieve our goal [Slack 2016d].

6.8 Methods and tools used

Slackbots – A node.js module that exposes the Slack Real Time Messaging (RTM) and Web API's. See section 7.9

Wit.ai – A Natural Language Processing (NLP) service used to export structured data from sentences. See section 9.4

Plotly – A website for creating and hosting interactive web graphs and graphics. See section 9.5
AlchemyAPI – A service based on Watson that uses NLP to perform functions such as sentiment analysis. See section 9.6

6.9 Research questions

Can a conversational agent equipped with a knowledge base of data gathered from a Slack environment address the above problem? What are the challenges in terms of human-machine, machine-human, and machine-machine communication in the Slack environment? What benefits are conferred by the use of controlled natural language as a basis for knowledge representation in this context?

7 Selection of Approach

7.1 Creating the experimentation environment

Being unfamiliar with both Slack and CEnode it was important to set up experimentation environments for both that could be used to gain a greater understanding of the underlying features they supported. CENode and its documentation were downloaded from www.cenode.io. At the time this website had a demo page that you could use to add concepts to the KB and it would display both instances and concepts in text boxes as they were added. This page was also downloaded and examined so that it could be reverse engineered and hosted on our own server. Once the page was loaded and the concepts displayed we could be sure that the server was setup correctly and CENode was working just as the developers intended it, as they had designed the demo page.

This setup also had the benefit of being able to see the changes to the KB in real time as sentences were being submitted, without having to deal with the formatting issues that are apparent when accessing through the command line. Example CE from the documentation was used as a baseline of what inputs should be accepted by the default system and to get to grips with the formatting of CE sentences.

Once the example CE produced the expected results it could then be substituted for our own, which experimented with more complex concepts and relationships and gave us insight into the more implicit rules of the language. For instance the KB must be 'built' in order. For example `conceptualise a ~ teacher ~ T that is a person` will not be accepted unless you have first conceptualised a person with `conceptualise a ~ person ~ P that is an agent` or similar.

Once it was apparent that CEnode was working properly on the server we began to create the Slack testing environment. The private Slack team CEtesting (www.slack.com/cetesting) was created with the purpose of enabling safe experimentation. All the basic functionality that Slack offers to a standard slack user was tried; uploading files; posting messages; 'reacting' (attaching a small emoji to another users message to display your reaction to that message) to messages etc. Not only would this build familiarity with the Slack web client it should also create at least one instance of each type of functionality in the channels history. The history was then downloaded, Slack provides this in the same JSON format it requires of its developers (see Figure 3), which gave us a starting point to reverse engineer default functions or message formatting programmatically at a later date.

```
"reactions": [
  {
    "name": "heart",
    "users": [
      "U0K2F3HL4",
      "U0W9H4QPQ"
    ],
    "count": 2
  }
]
```


Figure 3 Typical extract from a Slack channel history showing 'reactions' to a posted messsage

To integrate CE into the Slack environment we need a way for CENode to read the users questions and respond to them appropriately. There are a number of ways for developers to get data into and out of the Slack environment. Slack calls these methods 'integrations'.

- Slash commands
- incoming webhooks
- outgoing webhooks
- Bot Users
- Slack App (a bundle of some or all of the above)
- Real Time Messaging API (RTM API)
- Web API

These integrations don't have to be used in isolation, for example a Bot User may use the Real Time Messaging API to monitor events happening in a channel.

7.2 Slash commands

These are messages that begin with a forward slash. They are a staple of text based message systems and are used to tell the environment to treat this message as a request from the user, rather than a piece of communication between users. For example `/away` will change your 'presence' to away.

Slack comes with 34 built in slash commands and will display them when a user types `/` into the chat box. Further commands can be added by the team by creating a slash command on the integrations page or may be included in an App that has been added to the team.

Slash commands follow this format `/[command] [criteria]`. So a command to check the traffic conditions in the city of Cardiff might look like `/traffic Cardiff`. The text immediately following the command will be sent to an external url via HTTP POST, along with information such as the channel name, userID and command used.

Slash commands are the easiest way to get a query from a user out of the slack environment but their simplicity comes at the cost of functionality.

7.2.1 Evaluation

The slash command being the simplest of the integrations seemed like a good place to start. The slash command tutorial takes you through the process of setting up the command 'isitup'. This command connects to www.isitup.org and checks the online status of a website

provided by the user criteria section of the command. For example /isitup google.com would return the status of google.com.

The tutorial outlines the basics of setting up a PHP script to receive data from slack in a JSON string, using cURL to send/receive data from a website and then using that data to provide a response to the Slack user. This tutorial was used as the basis of a /ce command, setting up a test script in PHP and hosting it on a free web service along with CEnode.js. The script was simple, receive the POST data from Slack, save it in variables, and initialise CENode, as shown in Figure 4

As the script was in PHP and CEnode is in Javascript including it in the PHP file requires <script> tags. Viewing the PHP file in a web browser produced the text indicating that CENode had been initialised. The slash command is supposed to post output that has been echoed in the file. However when called from within Slack using the slash command, Slack instead posts any text that is not PHP as you can see in Figure 5.

Figure 4 High level diagram of the approach using a slash command

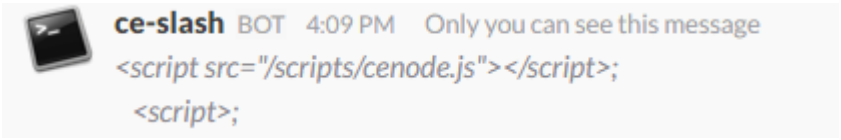

Figure 5 The erroneous output from this approach

7.2.2 Suitability for project

Unsuitable for the main approach as slash commands just send the contents of that message plus its metadata to an URL. Which means CEnode would have no idea of the contents of the rest of the channel. It would have no awareness of the environment; topic of channel; users in the channel etc.

It could be used to ask questions of CEnode if some way around the issue of JavaScript code being included in the response was found

7.3 Incoming webhooks

incoming webhooks are setup and triggered by the developer. The developer sends a HTTP POST request to an address the incoming webhook provides with a JSON object as the payload. Slack receives the request, takes the text key from the object and outputs it as a message to your users. Other key values can be sent with the object to set various parameters, such as changing the channel the message is posted to from the option that was chosen when the incoming webhook was set up.

7.3.1 Evaluation

incoming webhooks are not setup with a specific trigger they are simply a way of sending data into Slack. Slack has a second tutorial that details using a slash command to trigger a

file, searching Wikipedia using the WikiAPI and sending its response via an incoming webhook. We thought we could use this technique to solve the previous issue with slash commands. The slash command would be triggered by the user, sending the appropriate data to my server. A script would receive that data and send it on to another file that had initialised CEnode (so as to avoid the slash command immediately returning any value that isn't PHP). The second file would use an incoming Webhook to send the CE response back to Slack, which would then be posted on the same channel, appearing to the user as a direct response from the slack command (see Figure 6). However this approach did not work as the slash command still printed out an invalid message containing the code used to call the incoming webhook.

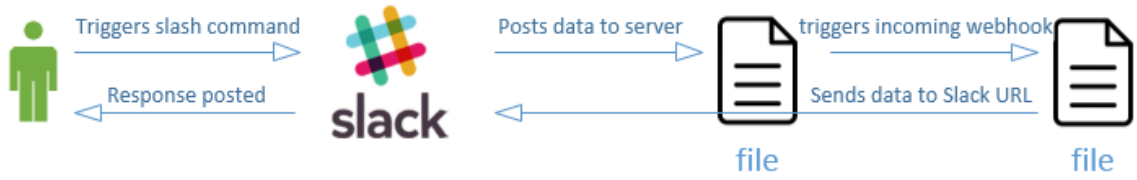

Figure 6 High level diagram of the approach using a slash command and incoming webhook

7.3.2 Suitability for project

Cannot be used on its own for the main approach as it just deals with importing data into Slack. It could be used as an easy way to post messages if we can find a suitable integration to trigger it.

7.4 Outgoing webhooks

These are almost identical to slash commands, with a few key differences. They only work on public channels. Instead of being activated by `/[command]` they have two ways of being started one of which is optional, if the other is present:

- The message is in the channel specified when the outgoing webhook was setup
- The message contains the trigger word(s)

For example if you wanted to keep track of a channels messages you could set up an outgoing webhook with the specified channel #General and no trigger word. Then every message sent in that channel will be posted to the web address you specified in real time, along with that messages metadata.

7.4.1 Evaluation

The next approach was to use an outgoing Webhook instead of a slash command. The outgoing Webhook would be triggered by a user sending a message that included the term 'Cebot:'. It would then send the data to another file which would activate an incoming webhook to send the response to Slack (see Figure 7). As outgoing Webhooks do not post a response themselves this approach should avoid Slack printing any code. Both parts worked in isolation, however the outgoing Webhook would not trigger the incoming Webhook file when they were both used together.

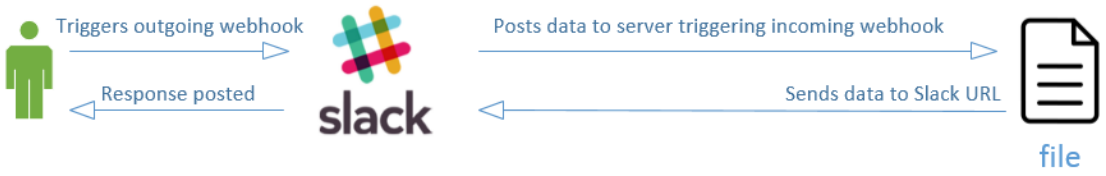

Figure 7 High level diagram of the approach using an outgoing webhook and incoming webhook

7.4.2 Suitability for project

An outgoing Webhook could be set up to trigger anytime a message is sent in a particular channel. In this way they could be used to record the ongoing activity of the channel in the CEnode KB but it would still suffer from the same lack of situational awareness as the slash command. Could be useful as part of a larger system together with other integrations.

7.5 Real Time Messaging API

The RTM API allows a developer to connect directly to a Slack team. Once connected they will receive all events (such as a user joining the team or someone sending a message) as JSON objects in real time. The developer can use the RTM API to send simple messages to a slack team by sending JSON over the websocket connection. These messages cannot contain any special formatting or attachments however.

7.5.1 Evaluation

The RTM API is more complicated than the other solutions and works in a different way as there is no actual set up on the Slack side apart from generating a token. I had hoped to avoid the complexities of Slacks RTM API and Web API for the prototype, as this was supposed to be a quick proof of concept and I didn't need all of their functionality to prove this concept. However if the project was to succeed past the prototype stage, these APIs would be required for CEnode to gather enough information about the environment to provide useful analysis anyway.

The help documentation for Slacks APIs suggest using one of their community built modules for handling API calls between the developers program and Slack. We decided to try a module built in Node.js because it is JavaScript and it has an excellent package manager that makes installing extensions and keeping them up to date easy. We can use the connect() function of the module to set up a Websocket connection between Slack and our system. We can then use an event listener to respond to events as they happen in the Slack environment.

7.5.2 Suitability for project

Suitable for the main approach with some major limitations. The RTM API provides us with an up to date model of the Slack environment. When it connects to Slack it sends us team, channel and user values. It can also monitor what is currently happening in the Slack environment sending events to our system as they happen.

However it does not deal with historic events and messages. It is also limited to posting plain text messages. Both of these issues will severely limit the functionality of our system if we design our main approach around the RTM API

7.6 Bot Users

Bot Users are designed to mimic real users. As such they have many of the same attributes as a real user (an entry in the team directory, profile picture, online status etc). They also can carry out many of the same actions as a real user such as direct messaging or mentioning someone. Bot Users are controlled by the developer using Slack's RTM API or one of their other API's.

7.6.1 Evaluation

Bot users can be used to provide a more conversational approach to our system and can provide it with some personality. As Bot Users have no functionality on their own, there is very little setup and no downside to using them.

7.6.2 Suitability for project

Suitable as part of the main approach when implementing one of the API's to provide functionality.

7.7 Web API

The Slack Web API allows a developer to access nearly all of Slacks functionality programmatically. There are over 90 methods that the developer can use to do things like open a multiparty direct message or archive a channel. The developer sends a HTTP GET or POST request to https://slack.com/api/[METHOD], where [METHOD] is the name of the method you want returned, along with any parameters the method requires. A JSON object is returned by Slack containing the requisite data.

7.7.1 Evaluation

The Web API has some methods that will be very important to the functionality of this project. The channels.history method in particular will provide the system with the data it needs to fill the KB. The RTM API allows a developer to post messages with less delay but the chat.PostMessage method will allow the system to provide clear and detailed answers using rich text messages, advanced formatting and attachments.

7.7.2 Suitability for project

Unsuitable for the main approach as it cannot monitor activity in Slack in real-time. Could be used with an outgoing Webhook to record messages sent in a channel but you would still miss out on events.

7.8 Slack App

A Slack App is used to take custom integrations that you have made for your team, and make them available for everyone. Developers can put Bot Users, incoming webhooks and Slash commands together to create 1 easy to install application. Unlike custom integrations created for 1 team, Slack Apps require submission and approval by Slack before anyone can use them. Slack Apps Bot's can use the RTM API but require additional permissions to access the Web API.

7.8.1 Evaluation

This approach falls outside of the scope of the project. It is undoubtedly the best approach to create an easily distributed 'product' and should be noted for anyone wishing to expand upon this project although Bot Users that are part of an App are restricted from using many methods including channels.history

7.8.2 Suitability for project

Unsuitable due to timescales involved / uncertainty surrounding approval process and not having access to the full list of methods

7.9 Conclusion and the prototype

Many of the problems encountered when researching the integrations were probably due to the use of two different programming languages, one of which is client side and one server side. Slacks tutorials were written in PHP and we had more experience using PHP so it made sense to start there. Once I delved into the CEnode code however I realised that I needed to fully focus on Javascript in order to get the most out of this project.

It is obvious that none of these integrations would completely satisfy all of our requirements for this system. We needed the RTM API to tell the system about events currently occurring in the Slack environment and we needed the Web API to retrieve events that had occurred before the system had been started. We decided to take a hybrid approach and construct the system using a Bot User, the RTM API and the Web API.

The Slackbots node.js module allows a bot to connect to Slack using the RTM API, which is the quickest way to send and receive data from Slack. It also includes a function that supports posting messages via the Web API and all the added functionality that provides, from a developers standpoint this is the best of both worlds. Using a Slackbots tutorial provided us with a framework to create a working prototype. We instantiated CEnode and loaded the KB. When a question was directed at our bot we used the CEnode function `ask_question()` to query the KB for the answer. Once CEnode responded the Web API was used to post a message to the channel the user asked the question in (see Figure 8). This proved definitively that communication between Slack and CEnode was possible and satisfied my criteria for a basic prototype.

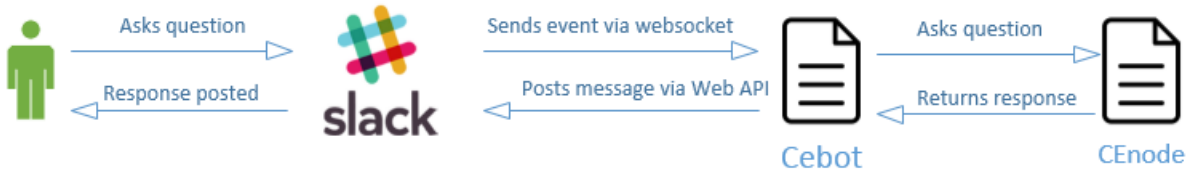

Figure 8 High level diagram of the design of the prototype

8 Approach

8.1 Modelling the Slack environment

The prototype system used example CE that is found in CEnode documentation. This was to ensure that any errors were caused by the implementation of the prototype and not by invalid CE. For the core system it was time to create a model for the KB that would describe the Slack environment. Exploring Slack's API gives a good indication of how Slacks own developers model the Slack environment. When the RTM API connects to Slack the first thing it receives is a list of objects that describe the current slack team (channels, users, groups etc.) From this data we constructed the model seen in Figure 9.

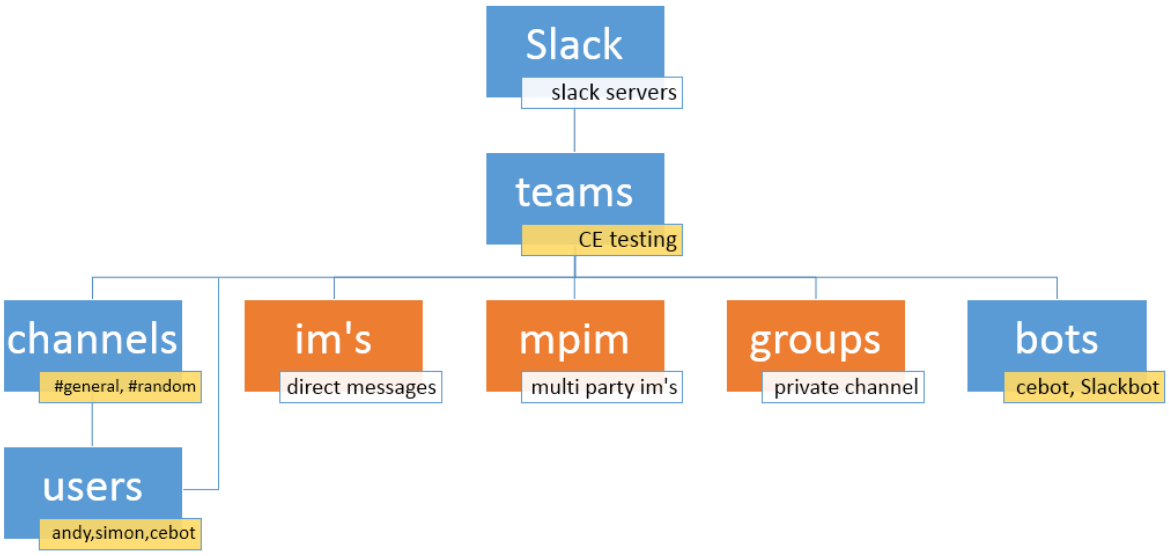

Figure 9 Basic model of the slack environment. Objects in orange are private therefore not accessible by Cebot. Labels in white are descriptive, those in yellow illustrate example data for the object

This model was a good starting point but it included data that we couldn't access and data that was redundant for our purposes. We identified a subset of the model that was manageable to implement and sufficiently rich in potential interations, by focusing on 'visible' elements of the environment such as users, channels, messages etc. We also removed duplicate or redundant data such as the bots category as these also appear in the users category with a is_bot flag.

The objects that are private and have been removed from the model (im's, mpim, groups) are all essentially channels and could be added at a later date as a child concept.

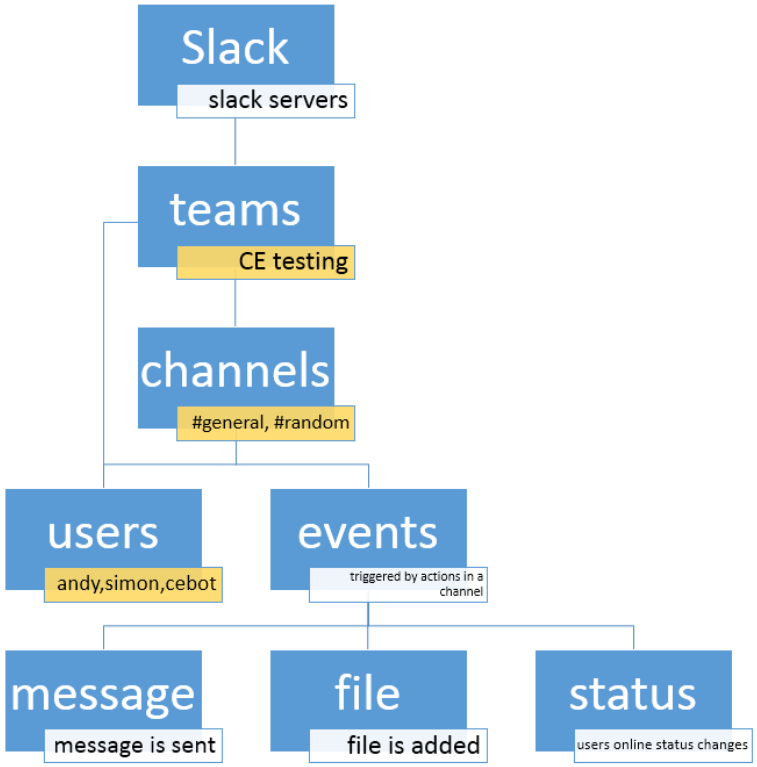

Figure 10 Further refined model of the Slack environment that includes events

This new model includes events Figure 10, which we used to update the KB with real time activity from the channel. The activity we were most interested in included messages being sent, files being uploaded and users coming online/going 'away'.

We identified the properties that needed saving from the data that we could get from Slack's API's (see Table 1).

Table 1 Shows useful values extracted from Slack API

Message	File	Status
Sender(Slackuser)	Uploader(Slackuser)	Changer(Slackuser)
Channel	File name	New status
Message contents	File title	
Mentioned(Slackuser)	File type	
Timestamp	URL	
	Timestamp	

From there we developed the concepts that CEnode will use to have a basic understanding of the Slack environment. The order of these sentences is important as CEnode will reject a concept if it is based on a concept it does not yet have stored in its KB.

```
conceptualise a ~ slack thing ~ S that is an entity
```

```
conceptualise a ~ team ~ T that is a slack thing and has the value V as ~ teamName ~
```

```
conceptualise a ~ slackuser ~ U that is a slack thing and has the value V as ~ slackname ~ and has the value V as ~ firstname ~ and has the value V as ~ lastname ~ and has the value V as ~ truename ~ and has the value V as ~ userid ~

conceptualise a ~ channel ~ C that is a slack thing and has the value V as ~ channelName ~ and has the value V as ~ created ~ and has the slackuser U as ~ creator ~ and has the value V as ~ channelid ~ and has the value V as ~ topic ~ and has the value V as ~ purpose ~
```

These four statements set up the 'top' of the CE model. Words surrounded by tildes are essentially variable names. Everything is based on the abstract concept 'slack thing'. 'Slackuser' and 'channel' are the two most important concepts in this system and their length reflects that.

```
conceptualise an ~ event ~ E that is a slack thing and has the value T as ~ timestamp ~

conceptualise a ~ presence change ~ P that is an event and has the slackuser U as ~ changer ~ and has the value V as ~ new presence ~

conceptualise a ~ message ~ M that is an event and has the slackuser U as ~ sender ~ and has the channel C as ~ channel ~ and has the value V as ~ messagetext ~ and has the slackuser U as ~ mentioned ~

conceptualise a ~ file ~ F that is an event and has the slackuser U as ~ uploader ~ and has the value V as ~ slackname ~ and has the value V as ~ fileName ~ and has the value V as ~ fileTitle ~ and has the value V as ~ fileType ~ and has the value V as ~ permalink ~",
```

These four statements are used to represent the real time events that occur within a channel. The concepts previously established are used here which is why the order of these sentences is so important.

```
conceptualise the slackuser U ~ is a member of ~ the team T

conceptualise the slackuser U ~ is a member of ~ the channel C

conceptualise the channel C ~ exists for ~ the team T and ~ membership includes ~ the slackuser U
```

This CE is used to establish relationships between the team, its channels and its slackusers. A user can be a member of both the team and multiple channels at the same time.

8.1.1 Issues/Challenges/Problems encountered

Confusingly many of the concepts that Slack refers to in its documentation are just different names for existing concepts. For example users talk in channels. A Group is the same as a channel except that it is private. A private conversation between 2 individuals is called a Direct Message in the Slack client. In the data provided by the Slack API it is called an IM. The Slack API also refers to MPIM, Multi Person Instant Messages, which would be the same as a Group but without a Topic or Purpose.

Originally `~ membership includes ~` was expressed by the term `~ has member ~` but CEnode kept rejecting it. This is because CEnode has restricted words that have specific meanings, in this case the word 'has' is used to indicate that a concept should store a particular value e.g `and has the channel C as ~ channel ~`. As developers we need to keep this in mind when naming our concepts, relationships and values.

8.2 Design of core system

As it should be apparent from our testing with the initial prototype due to our inexperience with Slack, CEnode, Javascript and Node.js we needed to develop a highly iterative approach to the design of this system. As the design and testing progressed we expanded our knowledge of each of these systems which allowed us to develop new features or improve the efficiency of the system. As such the system went through a number of quite substantial design changes that allowed us to make the most of the technologies invlovled.

The core design, and each subsequent design relies upon three main parts:

8.2.1 Slackbots module

This is responsible for direct communication with Slack. Cebot inherits the functions found in this module. When Cebots run function is called this module connects to Slack's servers using the RTM API and a token that is unique to each slack team. This opens up a websocket between Slack and the module. Cebot can then listen to this websocket to hear events sent by Slack (such as a user sending a message) and decide how to respond to those events. Cebot could use this websocket to send messages back to Slack however the RTM API only supports basic text. A better choice would be to use the Slack Web API which allows the developer to use full text formatting and attachments.

8.2.2 Cebot.js

This file acts as the intermediary between Slack and CEnode. This file is responsible for monitoring events happening in the Slack environment. It also instantiates CEnode with the Slack model and asks it questions. Think of it as a translator that converts Slack JSON payloads to valid CE and vice versa.

8.2.3 CEnode.js

This file contains the entirety of CEnode, a lightweight alternative to the CE Store. It is responsible for managing the KB and parsing CE sentences and it has a number of useful functions for dealing with user input and responding appropriately.

8.3 Basic design concept

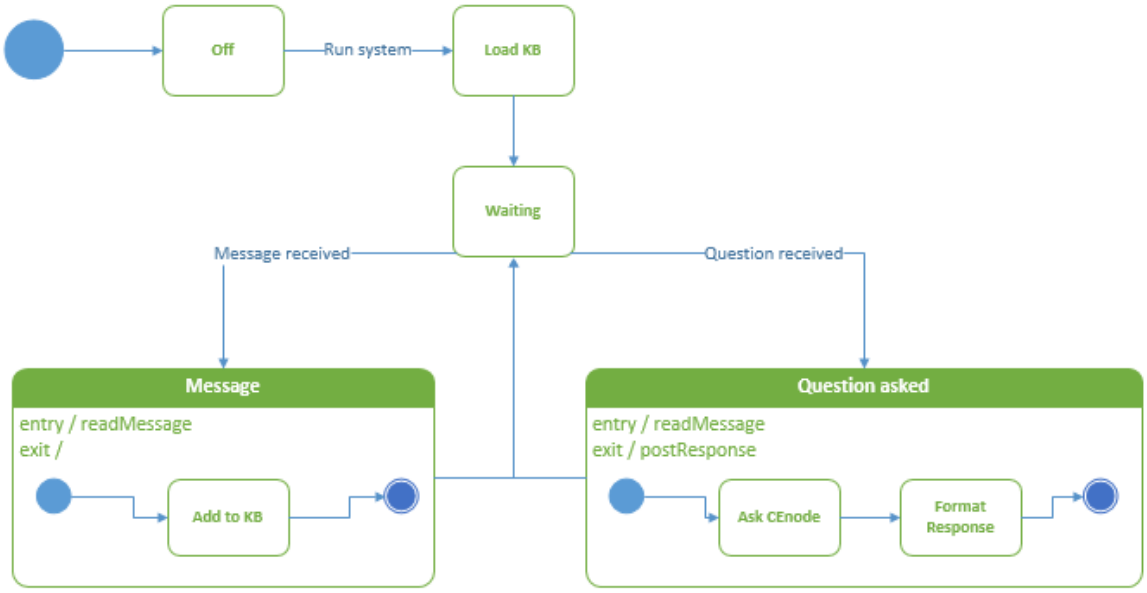

Figure 11 High level state machine diagram of the core concept

The overall design concept is simple as can be seen in Figure 11. When the system is started it will connect to Slack via the RTM API using the team's Slack token. The token is a standard feature of most API's and acts similarly to a username and password, telling the system which teams data it should send and ensuring that only authorized developers have access. Once started the RTM API will provide data on the team, its channels, its users etc. It will also provide an address that the system can then use to open a websocket connection. Using the data provided by the RTM API the system will then convert the team, channels and users to valid CE and store them in the KB. (see red section in Figure 12)

The system is almost ready to respond to a user's queries. The RTM API data includes things like the names of channels and their members but not what has happened on that channel. To get this the system will use the Web API method channels.history(). This returns all of the messages sent in the channel, starting with the latest, in blocks of 100. The system will go through each message, converting the JSON data into valid CE and storing it in the KB. When it is done with the current block it checks for the presence of the 'has_more' flag. If this flag is true then the system will call the API method again with latest parameter set to the timestamp of the latest message entered. It will keep looping in this way until it has recorded all of the messages sent on the channel. (see purple section in Figure 12)

The system is now in an idle state. It is listening to activity occurring on Slack using the websocket connection. When an event is received several checks are made. First the system checks to see if the event is actually a message. Slack uses the 'message' type for a variety of user actions that are not relevant to us. If it is a `presence_change` type we can use this to record a change in the user's online status, the system will convert to CE and log it in the knowledge base, if not then it continues with its checks. Then it checks if the event is from a public channel, if a user DM's Cebot we do not want to record this in the KB. It also ignores any messages sent by Cebot itself, in order to avoid loops.

The system determines if the message is a question to Cebot or communication between users on the channel. Users can question Cebot by typing a message beginning with 'Cebot:', any text after the colon is then treated as a question/request. If the message is not for Cebot then it is converted to CE and stored in the KB.

If it is a request for Cebot then the system will remove the calling parameter from the message text (Cebot:) and send the entire message object to CEnode. We can then use the `ask_question()` function to query the KB for the answer to the users query. Depending on the response from CEnode the system will use the Web API to post either the answer or a suitable failure message. (see yellow section in Figure 12)

The system then returns to its idle state, waiting for further activity on Slack.

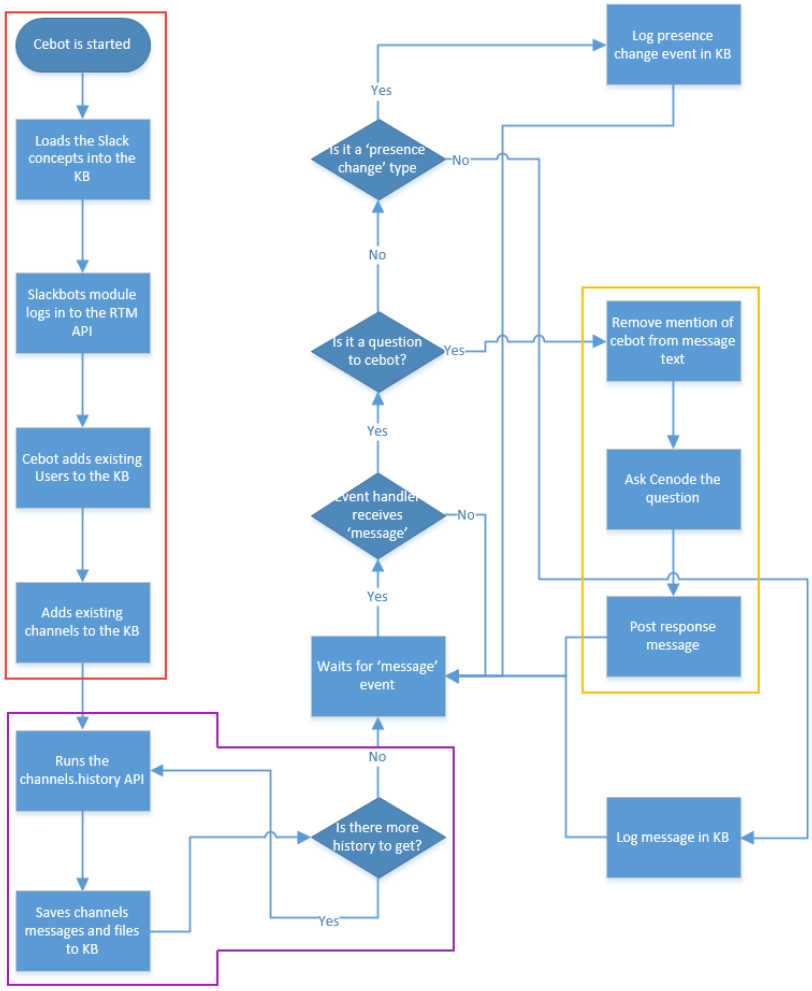

Figure 12 Basic flow diagram of the core system

9 Implementation

9.1 Core system

9.1.1 Naming instances in CE

CEnode requires each instance of a concept to have a unique name. We decided to use Slacks internal identifier for a slack object as the name of the corresponding CEnode instance see Figure 13. This choice is slightly counter to the philosophy of using human readable names for instances however as the user would not being seeing raw CE we thought it would not harm the users experience. From a developers standpoint this made coding both Slack and CEnode far less confusing as you know that user U0K2F3HL4 in Slack will have an instance named U0K2F3HL4 in CE.

```
CEInstance {
  name: '1456939912.000007',
  id: 123,
  type_id: 33,
  sentences: [ 'there is a message named \'1456939912.000007\' that has the slac
kuser \'U0K2F3HL4\' as sender and has the channel \'C0K2MLQNA\' as channel and h
as \'test <@U0LSH0B7E> <@USLACKBOT>\' as text and has \'1457546213043\' as times
tamp and has \'U0LSH0B7E,7\' as mentioned' ],
```

Figure 13 Example of a message saved as an instance in the KB

9.1.2 Timestamps and Moment.js

We decided to use the messages 'ts' property as its name. Slack ensures that these timestamps are unique so that messages can be displayed in order. However Slack doesn't handle timestamps in a consistent way. While handling objects returned from Slack it became clear that some timestamps were strings and some were integers. Slack's Github page for their API documents talks about the reasons behind this.

As timestamps were so important for this system we decided to look for a module to specifically handle them. There are ways of dealing with UNIX timestamps using Javascripts Date object but they aren't very elegant and with the issue of some being strings and others being integers a module that could handle both would be ideal. We settled on Moment.js a lightweight module entirely dedicated to parsing, formatting and displaying time and date objects in javascript. One of its most useful functions is the ability to handle date logic such as subtracting a certain number of days from a date, even handling situations which cause the date to 'rollover' to the previous month.

9.1.3 Slack mentions

As messages are broadcast to the channel there is no 'To' field as you would expect in other methods of communication, such as email. It would be difficult for a machine to determine who the intended recipient(s) are in the context of a general chatroom without this information. Humans can use grammatical forms, past experience, current conversational context and personal knowledge of the other members of the conversation and even then it is still sometimes hard for them to tell who a message is directed at.

Thankfully we can use mentions to partly solve this problem. A key part of using Slack is '@'ing other users. This works in much the same way as it does on Twitter/Facebook etc. When the user types a message they can include the @ symbol and Slack will present them with a list of the channel's users they can choose. When the message is sent users who have been 'mentioned' will receive a notification. Therefore they are often used to indicate you

want a specific user to see something E.g ‘hey @martin, im going to need you to work on Saturday.’ When a message is sent that includes a mention Slack changes the @ to this format <UserID>.

One of the advantages of CEnode is that only the name is required to save an instance of a concept. It also allows you to save multiple values for the same ‘field’. This makes it easy to save multiple mentions by just looping through the message and adding more CE to the end of the sentence E.g. `there is a message named '1358547726.000003' that has the slackuser 'U2147483896' as sender and has the channel 'C2147483705' as channel and has 'test' as messagetext and has '1358547726.000003' as timestamp and has 'U1234567890' as mentioned and has 'U0987654321' as mentioned`

This means that a single concept can be used to save valid CE in multiple different situations.

9.1.4 Online status and ‘presence change’

The ‘presence_change’ event was to be used to log when a user’s online status changed. In this way it was hoped we could build up a log of when a user typically logged on in the morning, went ‘away’ during the day (possibly for lunch etc) and when they finally logged off in the evening. However it is not uncommon for a Slack user to be part of multiple teams. The Slack client provides an easy way to switch between teams in its UI see Figure 14.

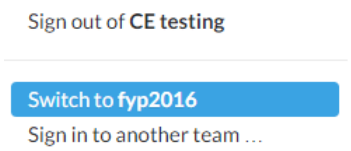

Figure 14 Slacks web UI for quickly switching between teams

When a user switches team and then switches back too many ‘presence_change’ events are triggered. This makes the data stored in the KB unreliable. (see Figure 15)

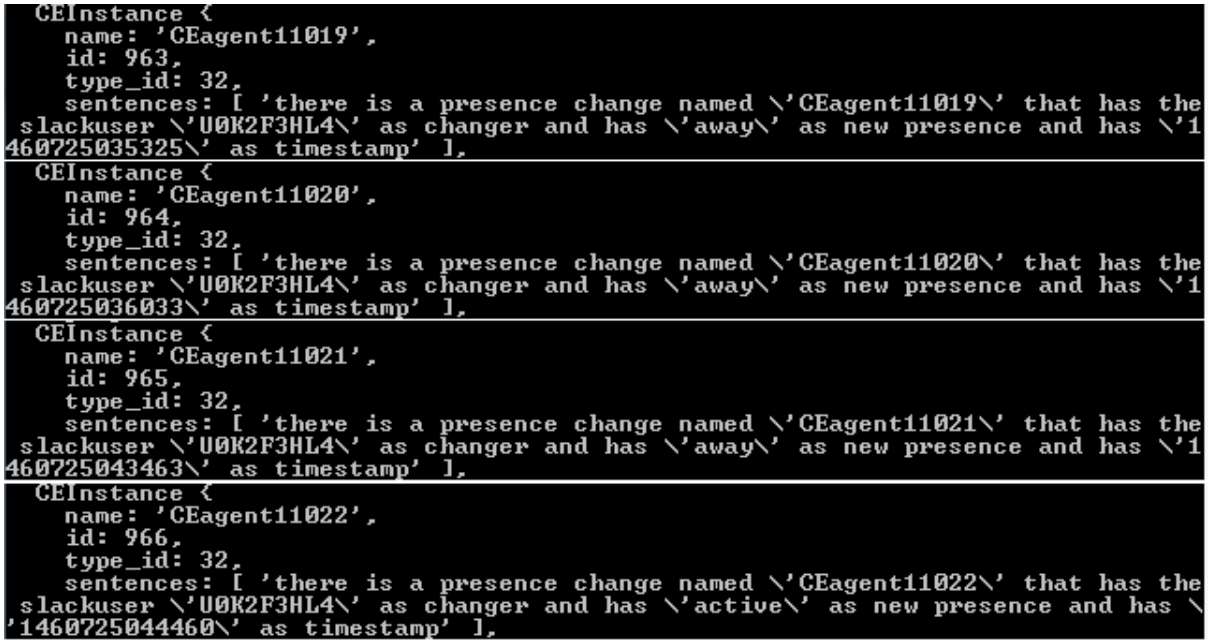

Figure 15 Example of presence change event being triggered multiple times for a single team switch

Another issue with using presence_change is that these events are not included in the channel history. As there is currently no way to save CEnodes KB and the KB can't load these events from the channels history it means that they are lost every time the system is shut down.

Because of these issues it was decided to instead use the timestamps of a user's sent messages to determine when they were online. This provides a more accurate description of when they are actively involved in the Slack team's discussion as opposed to merely just being online.

9.2 Issues with default CEnode

With the core system up and running a design issue soon became apparent. We had incorrectly assumed that CEnode would be able to provide answers to a user's questions if they were presented to it in natural language. This is important because we couldn't use two of CEnodes most helpful user interface functions, 'autocorrect' and CE 'spellchecking'. Other apps developed using CEnode have had control of their UI which means that CEnode can help the user by suggesting what the next word should be based on concepts and relationships already stored in the KB or by indicating where a word is incorrectly spelt. This both helps the user learn the format of valid CE and reduces the frustration of forgetting what a concept is called or how it is spelt. Using Slacks developer tools it is not possible to change the default Slack client UI to allow these features. It is possible to build an entirely new Slack client but this would be far too ambitious with this projects timeframe.

Another reason that CEnodes ability to understand NL is important is that CEnode is designed to allow users to change the conceptual model of the KB and add instances to the knowledge base whenever they send a message to CEnode. This would prove very disruptive to our system as we have already provided it with a model of the Slack team as Slack itself sees it, so we can be sure it is a true representation. If a user started altering that model by adding additional concepts or perhaps adding an instance of a user that doesn't

actually exist, not only would the system provide incorrect data when questioned it would most likely crash when it tried to access objects through the Slack API that are not real. So it is necessary to switch users away from typing CE and instead use NL, which is how they expect to interact with a chat bot anyway.

Because CEnode was designed as a lightweight alternative to the CE Store its ability to understand user questions is restricted and the answers it provides are limited. CEnode can respond to user questions asked in these formats:

- (who / what) (is / are)
- (who / what) (is / are) (an / the)
- List (instances of type / all instances of type) or List (instances)
- Where (is / are) (an / the)
- (who / what) is (in / on / at)

For examples of the first three question formats see Figure 16. The last two questions are location based and not applicable for this system.

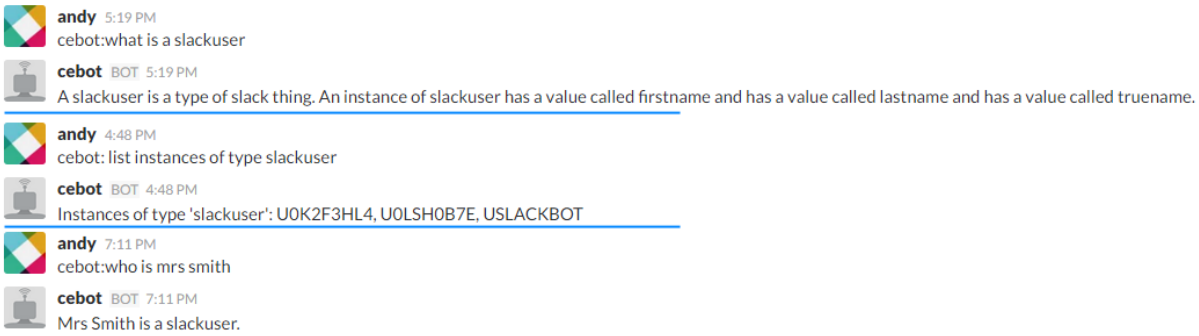

Figure 16 Output supported by inbuilt CEnode query function. You can see the drawback of using Slacks internal ID's as names for CEnode instances. Blue lines have been added for clarity

As you can see from [the figure above] the responses CEnode gives to user questions are limited. This is fine for the basic questions it supports but it will not be suitable for the more advanced questions this system will need to be able to answer to prove useful to the Slack team. The response 'Mrs Smith is a slackuser' for example is correct but it is missing information that the KB has available about Mrs Smith, her first name for instance. You can pre-empt further questions by providing information that is likely to be wanted by the user in the original question. The reason CEnode does not do this is because it has one function 'parse_question' designed to handle all of its supported question types and therefore it must be as generic as possible. The response to 'what is a slackuser' in [figure whatever] is a good example of this.

9.3 Expanding CEnodes functionality

We needed to expand CEnodes current parse_question function to be far less strict in the format of questions that it will accept. Investigation into the default parse_question function revealed that it uses a series of regular expressions to try to match user input with a format it can understand. It can then use those same Reg Ex to remove the 'who is' part of the question, leaving just the 'criteria' of the question.

```
name = t.match(/^(?:\bwho\b|\bwhat\b) (?:is|are)(?: \ban?\b | \bthe\b | )([a-zA-Z0-9_ ]*)/i)[1].replace(/\?/g, '').replace(/'/g, '');
```

This Reg Ex requires that the word ‘who’ or ‘what’ is at the start of the text. Then there is an optional word of either ‘is’ or ‘are’ followed by an optional word ‘an’ or ‘the’. It then accepts any combination of characters or numbers. It ignores the case when matching and replaces all appearances of question marks or single quotes.

We initially tried to expand this method to include a question such as “who does andy talk to the most” or common variants of. It quickly became apparent that coding a single Reg Ex to handle even a few variations was not working. We then tried using multiple regular expressions, one for each way the question could be worded. Not only was this a very ‘messy’ approach from a coding standpoint, it still required the user to format their question in a certain way, which left them wondering why one question was accepted but a very similar question would be rejected.

We spent a small amount of time researching text based adventure games that used to be popular when computers graphics capabilities were very limited. However it transpired that most used a very rigid system similar to the regular expression system already used. Users would have a limited range of commands such as ‘pick up’, ’talk to’ or ‘use’ and the game would simply try to match their input to a list of available commands.

While searching for regular expression alternatives we discovered a company who has a regular expression AI that creates a regular expression based on a training set you provide. You can then provide an unseen testing set to judge its effectiveness. This AI was very good for ‘tradiational’ Reg Ex such as postcodes, telephone numbers etc but it could not deal with sentence structure.

9.4 Core system + Wit AI [H>M]

It was obvious that regular expressions weren’t going to be suitable for handling users conversational queries but the last approach led us to explore the viability of training an AI. It was then that we stumbled upon WitAI (Wit). This is a service designed to allow app developers to send it their user’s freetext or speech input and receive back a JSON payload with Wit’s guess.

Developers create ‘intents’ for their application. Intents are used to indicate what the user wants your system to do. To support the question “who does andy talk to the most” we created the intent “mentions”. Before the intent can be used you must train it with at least three examples of how your users will express this intent. In reality it takes around 20 example sentences before Wit’s confidence rises to a level suitable to use in an application.

Intents support the use of ‘entities’. Entities allow you to extract useful data from a sentence and there is a list of built in entities such as ‘contact’ which will try to find any names in a sentence and return them in an array. Entities also help Wit to distinguish between intents, if a user has submitted a sentence with a name in that could match multiple intents, but only one of the intents uses the contact entity, then Wit will raise the confidence value for that particular intent.

Developers cannot alter built in entities but they can train them. For example by default contact will only extract simple names such as ‘simon’. A developer can submit a sentence such as “who does simon1 often talk to?” and then manually assign the contact entity to ‘simon1’. Once the developer has submitted enough examples Wit will learn that numbers are acceptable in the contact entity. Using this process we taught Wit to recognise Slack usernames that can contain special characters as valid for the contact entity.

Developers can also build custom entities. Building a custom entity works in a similar way to creating an intent. You give the entity a name and then create an output field that you want it to produce, for each of these fields you supply values that should trigger that output. We created the custom entity FileType to support users filtering file uploaded by file type. The output ‘spreadsheet’ will be triggered by any one of these words:

- Spreadsheet
- Spreadsheets
- XLS
- ODT
- XLSX
- XLSM

Custom entities can be ‘open’ or ‘closed’. In an open entity Wit will suggest new values for that entity based on user input. If Wit finds a word it doesn’t recognise in the place it was expecting to find an entity it will return that word as a suggested value. This approach provides a good level of accuracy for some entities but for others, such as our FileType entity, it is better to close the entity and handle no value being returned in the systems code.

Another useful feature Wit provides is the ‘inbox’ (see Figure 17). Every sentence sent to Wit through its API by our system is saved in the inbox. Developers can see the intent Wit thought was most applicable to this question, the confidence of that guess between 0 and 1 and any entities Wit captured. Developers can also view the JSON payload the application would have received which is useful for debugging. If Wit made the right guess then we ‘validated’ the sentence and it became part of the training set. If Wit made the wrong choice of intent or extracted the wrong value for an entity/missed a value then we can correct its choices and then validate the sentence. If the sentence is completely unrelated to questions we want the system to support then we can archive it for later or delete it.

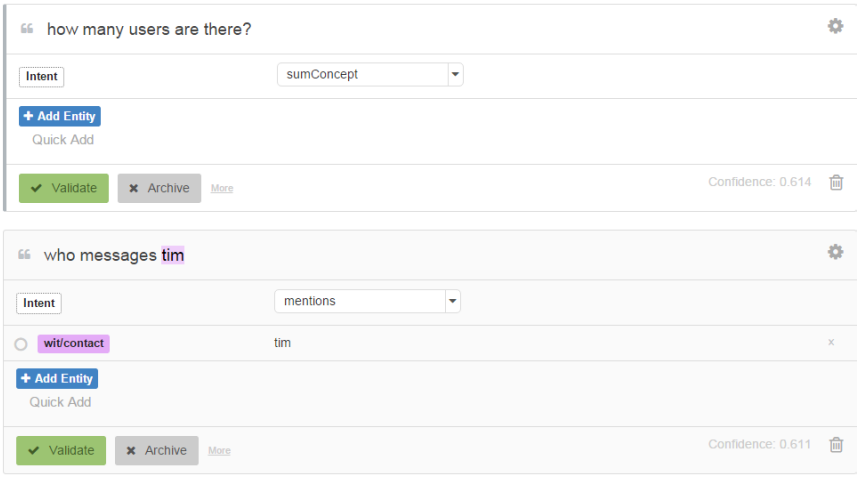

Figure 17 An example of entries in the Wit inbox

Using WitAI means that the system can support a user asking questions in a conversational manner. The user does not need to learn CE as this can easily be handled by the system behind the scenes. Users do not need to remember which static questions they can ask or how those questions should be formatted. This greatly reduces the barrier to entry for users and allows more complex queries to be asked that can be filtered by multiple different variables.

9.4.1 Drawbacks of using Wit AI

The Wit service is an online only service. This requirement is not a drawback for this system as Slack requires the user to be online anyway. However we have introduced another external service and therefore we are relying on something we have no immediate control over. If the Wit servers are unresponsive for whatever reason this will have a major negative effect on the functionality of the system. There is some form of redundancy, the system will still check user queries against CEnodes built in functions, but the more complicated questions will not be answerable.

- The API can change at any time
- There is limited control over built in entities.
- You cannot set up custom entities using algorithms or regular expressions.
- Might struggle with too many intents that use similar wording and entities

9.5 System + Plotly [M > H]

Now that the system could understand more complex user input, more complicated questions could be asked of it. Which meant more complicated answers had to be given. For the question “who does andy talk to the most?” we could return the answer “Ted”. This would be a correct response but it is not very helpful to the user. Given this answer the user will most likely ask “how many messages has andy sent to ted?” or the inverse “who does ted talk to the most?”/ “how many messages has andy sent to ted?” By providing a more helpful answer in the first place we can reduce four separate queries to just one. There are a number of benefits to this approach:

- Reduced number of API calls made to Wit AI lowers the amount of bandwidth required by the system
- Users are not frustrated by having to repeat basically the same query multiple times

- Reduced number of calls to the CEnode KB means a quicker system overall
- Not having multiple intents with similar sentence structure means that Wit is more accurate in choosing the correct intent

The Wit service helps us here as well. It sends user input to our inbox in the order it was submitted, which makes it easy for us to see when a user's query was not fulfilled to their satisfaction and which questions they then went on to ask to find their desired information.

When trying to answer the question "when is andy often online" we came across a problem. Initially we had 2 responses to this question. One returned the days that Andy was online and the number of messages he had sent that day. From this a human can easily extrapolate the day Andy is typically the most active. The other response returned the times that Andy was most active by listing the number of messages sent in certain time periods for example '6-9am'.

Using the approach above it became obvious that users would want both sets of information given to them and so combining them into one query made sense. Slack posts support plain text being formatted in italics, bold or code. They can include hyperlinks, emojis or 'attachments'. The closest thing Slack provides to a table is the 'field' attachment that allows 2 pieces of data to be placed side by side, giving the impression of columns. This query however was too convoluted to be displayed as text, it would take significant effort for the user to decipher it as you can see from Figure 18. This query type is perfect for displaying in a graph and we could use the Slack API to upload a picture file as a response to the users query but we would need a way of creating dynamic graphs.

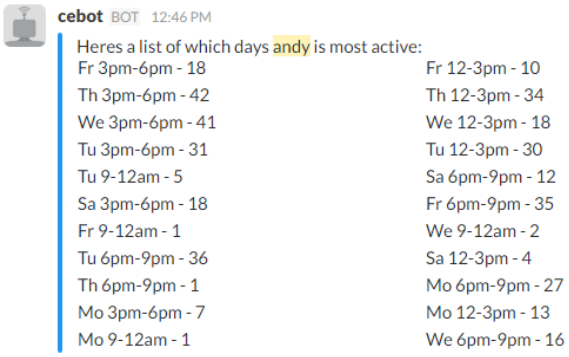

Figure 18 An example of an early version of the 'activity' function output

We immediately thought of Google Charts. These have been a popular choice for developers for many years and the API is well supported and documented. We managed to create a test graph using JavaScript and things appeared to be going well. However Google Charts work by using the browser to render the data into a chart format. The output from the API is a hyperlink that tells the browser engine what to draw in a new window, this is not a typical hyperlink, it is hundreds of characters long for a simple graph. This looks very messy in a Slack message but can be hidden by posting with this format <URL|Text to display>. The problem is Slack decides what is and isn't an URL and there is no way to force the API to create an URL when you post. Slack says "URL detection is performed by the server. We do this so that URL detection (a non-trivial operation) is performed consistently across multiple clients" [Slack 2016e].

There also doesn't appear to be a way of using the Google Charts API to save a created chart as a picture.

So we needed a new service that could create a dynamic chart in Node.js and supported saving the chart as a picture. This is when we started using Plotly. Plotly offers a free graph hosting service for public graphs and is used by large organisations such as the Washington Post. Their tutorials for JavaScript offer no explanation just the code required to replicate their sample graph. They often just didn't work and a significant amount of time was spent learning the intricacies of the API and how to format the graphs properly.

The API creates an URL to the graph that Slack recognises but by default it does not save the graph as a picture. This does not matter however because we can use a Slack formatting technique called 'unfurling'. This makes the Slack servers go to the link destination and return a summary as an attachment to our post. In Plotly's case Slack returns a picture of the graph that can be enlarged for users to look at without having to leave the Slack service

A free Plotly account allows a developer to host an unlimited number of public graphs and one private graph. Originally we were using the API to overwrite the one private graph every time the function was called although this caused a bit of confusion when multiple people ask for graphs in a short amount of time. It also stops the unfurl feature from working. Slack will only unfurl each link once even though the data at that link has changed. Therefore we changed the system to increment the URL every time the graph is created, meaning the previous URLs still work and so does the unfurling.

Plotly supports many different types of graphs from simple bar charts or line graphs to histograms and heat maps. Unfortunately getting the API to create these graphs and display them properly is quite a challenge. The 'team statistics' query was supposed to produce a horizontal bar chart that showed Slack users on the Y axis, messages sent on the bottom X axis and files uploaded on the top X axis. No matter how the bars were grouped, stacked or overlaid Plotly would either use the bottom X axis for the files bar or use the top axis and refuse to display the messages bar. After searching and failing to find any examples of a horizontal Plotly bar chart with 2 axis we came to the conclusion that this type of graph could not be made.

If the user chooses to visit the Plotly link they will find a fully interactive graph. Holding the mouse over the graph shows the relevant data points and labels. Users can zoom and pan around the graph to highlight data subsets or they can view and export the underlying data the graph is based on.

Plotly allows us to communicate a complex set of data to human users in a way they can easily understand.

9.5.1 Drawbacks of using Plotly

The downsides of incorporating Plotly are similar to those of using Wit but now we are relying on an external service to display an answer instead of understanding a question. As a fallback measure we could display the answer in text format although this might not be much more helpful to the user than an error message.

- Creates good looking charts but formatting is very difficult using the API
- API is poorly documented for Node.js

- Free account only supports 1 private graph
- Question of data privacy as graphs are available to all of those with the URL
- Some graphing formats are impossible to create
- With at least one graph the picture that Plotly generates is incorrect, the interactive graph is correct however. As seen in Figure 19 the bars do not correspond to any labels on the x axis.

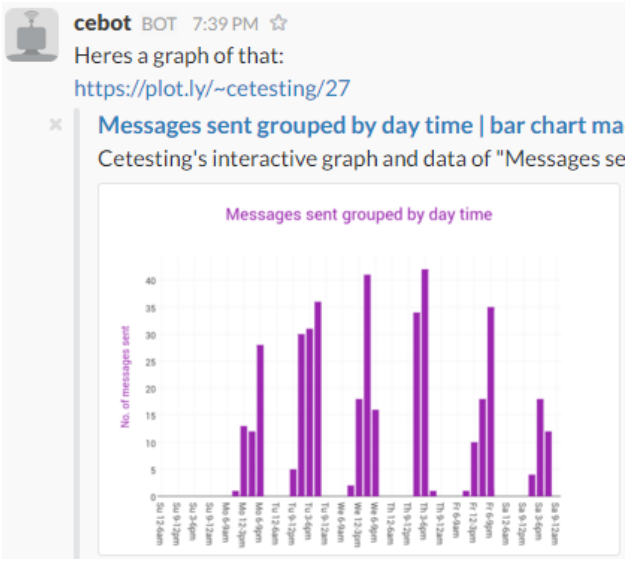

Figure 19 The activity output when 'unfurled' by Slack shows an incorrect bar chart

9.6 System + AlchemyAPI [M > M]

Functionality wise we were starting to run out of useful commands that could be built into the system. As Slack is fundamentally just a messaging service most of the data that can be obtained from the service deals with messages. Messages themselves can be boiled down into 4 main variables

- Who sent the message
- When they sent the message
- Who they mentioned (if applicable)
- What they said

So far our functions had avoided the last one. There is good reason for this as parsing the content of a user's instant messages and providing accurate information about them is a very difficult problem. When messages are delivered in plain text even humans can misconstrue the other persons tone. An email that the sender thinks is politely worded may come across as 'bossy' by the recipient. Emoji's and internet slang have grown as a way to alleviate this problem and are now commonly used even in the workplace. In fact Slack has recently introduced animated Gifs as a further way of displaying user emotion [Slack 2016f].

We began by testing a module called Sentiment 'that uses the AFINN-111 wordlist to perform sentiment analysis on arbitrary blocks of input text' [Sliwinski 2016]. The AFINN-111 wordlist is 'a list of English words rated for valence with an integer between minus five (negative) and plus five (positive)' [Nielsen 2011]. Using Sentiment we can compare sentences submitted by our users to the wordlist and retrieve a score based on how many

negative/positive words occur in that sentence. We can also add to the wordlist terms that may be common to our organisation or to Slack, or we can override a value for a word in the existing wordlist.

Using this approach we can create a query such as “how is andy feeling today?” that will retrieve all the messages that Andy has sent from the KB and run them against the wordlist to calculate his total ‘sentiment’ score. This is quite a basic approach however as it doesn’t tell us anything about what Andy was actually saying, it uses a list that was last updated in 2011 and all it provides us with is a number.

We wanted to add the functionality for users to be able to see a summary of someone’s messages, for example which keywords have been used. This could be very useful in a business environment where a manager could quickly see a summary of what his team had been talking about that day/week/month. It could also highlight areas where team members have been talking about similar ideas, which could lead to new collaborations.

A popular algorithm for extracting keywords from a block of text called tf-idf which means Term Frequency – Inverse Document Frequency. This algorithm works by first removing words common to the language, in English for example this would remove words such as ‘and’. It then counts the number of times each word appears in the remaining text block, removing any words that do not achieve the minimum frequency measure set by the developer. The remaining words are compared to the ‘corpus’. The corpus is a selection of documents that are representative of a typical block of text in some way. This could be a default set of documents representing the English language or it could be more specialised, Computer Science journal articles for instance. Comparing to the corpus increases the accuracy of the algorithm by controlling for keywords that are common in that environment.

Natural is a Node.js module that supports tf-idf, along with many other natural language functions. We started testing this algorithm using messages that had been sent by a user and comparing that to a corpus of messages that had been sent by every other user. We eventually abandoned this approach as the tf-idf function in Natural does not return a list of keywords, instead it requires a list of keywords and returns the documents where those keywords most frequently appear.

We needed a more intelligent solution. Everyone remembers IBM’s Watson supercomputer winning Jeopardy but since then his team have been busy expanding his functionality to include areas as diverse as modelling the interactions between different types of drugs. Watson uses the AlchemyAPI (Alchemy) to analyse webpages and return things like sentiment and high level concepts. IBM have spun this off as a separate service available to developers. Alchemy uses data mining and deep learning to scan millions of web pages, blog posts and news articles

Alchemy allows developers to run up to 1000 API calls a day on their free service. Their API’s are well documented and easy to understand. Each function they offer is split into a separate API call such as alchemy.keywords or alchemy.concepts. This means that the developer can pick and choose what data they want to receive. We ended up creating a sort of asynchronous waterfall function whereby each API call would be made after the return of

the last one, ensuring that we had all the necessary data before formatting a message and returning it to the user.

We decided on the most useful and accurate API functions and implemented those in our system. We chose keywords, entities (named people, places or things), high level concepts and document sentiment. We concatenated all of the messages the specified user had sent in the specified time period, inserting a full stop at the end of each message as these are often forgotten by users and the use of sentences make a large difference to the results returned by Alchemy.

The Alchemy service is designed to analyse web pages and news articles where the text is more structured and they can use objects such as hyperlinks and pictures as contextual cues to the underlying message of the text. As such it isn't really intended to be used in this way but we had some encouraging initial results from out tests so we decided to use it.

9.6.1 Drawbacks of using Alchemy

- Data privacy concerns
- Cannot change the sentiment values of words manually
- 'Concepts' returned are usually very inaccurate

9.7 Final system

Included below are examples of the functionality the final system supports.

9.7.1 Help

Figure 20 shows the help function, designed as a tutorial for new users. It uses a wide range of Slacks formatting and emoticons to make this large chunk of text easier to decipher.

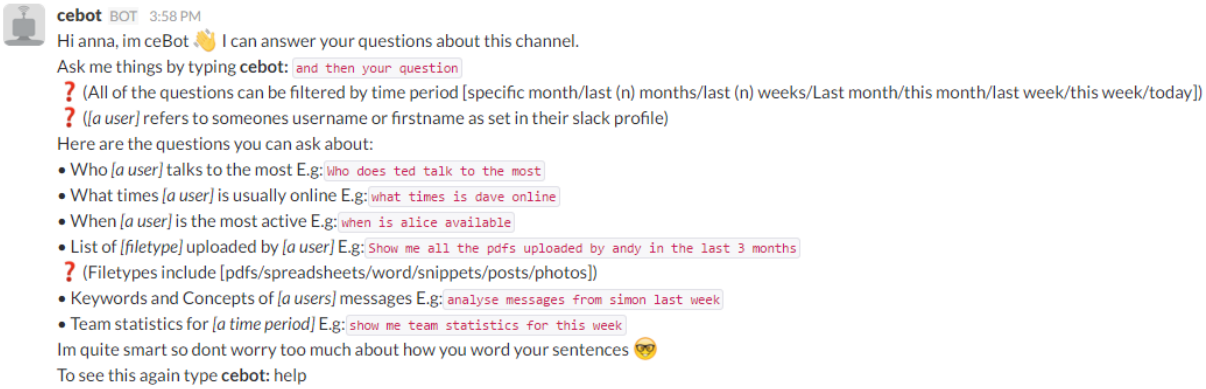

Figure 20 Cebot's help output displayed in a Slack channel

9.7.2 CEnode querying

Figure 21 shows the basic query functionality (who/what is) provided by CEnode. In this example the #general channel was asked about. This response shows the downside of using internal Slack ID's as names for CE instances, that they negatively affect a human's ability to easily comprehend CE sentences.

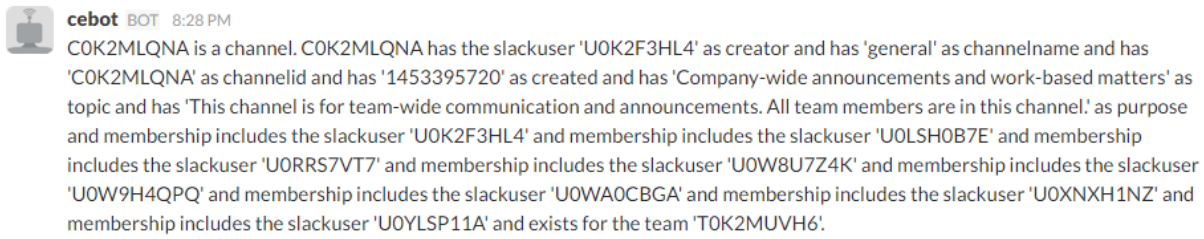

Figure 21 The output produced when Cebot is asked 'what is C0K2MLQNA'

9.7.3 Team connections

Figure 22 shows who a team member has been talking about and who has been talking about them.

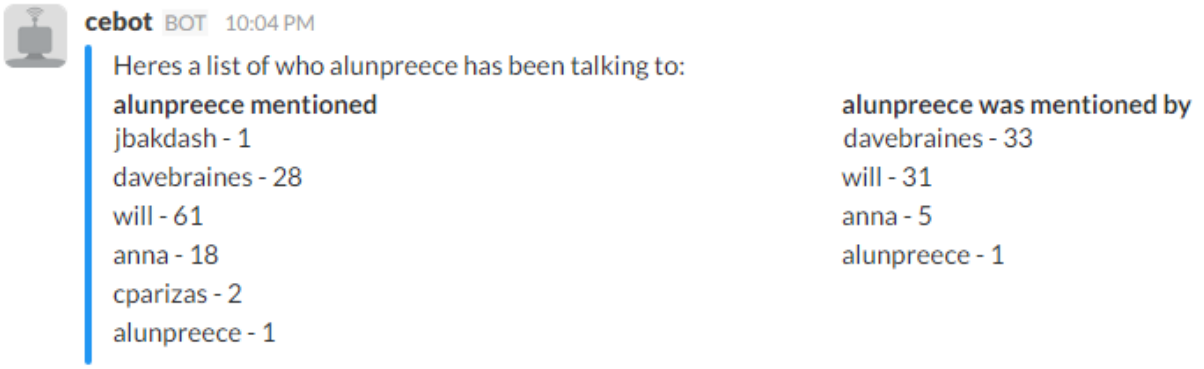

Figure 22 An example of the output produced when Cebot is asked 'who does alun often talk about?

9.7.4 Online times

Figure 23 shows when a user is most often active, based on the total amount of messages they have sent split between time periods. This tells us that 3-9pm is the best time to talk to this user as that is when they are most active

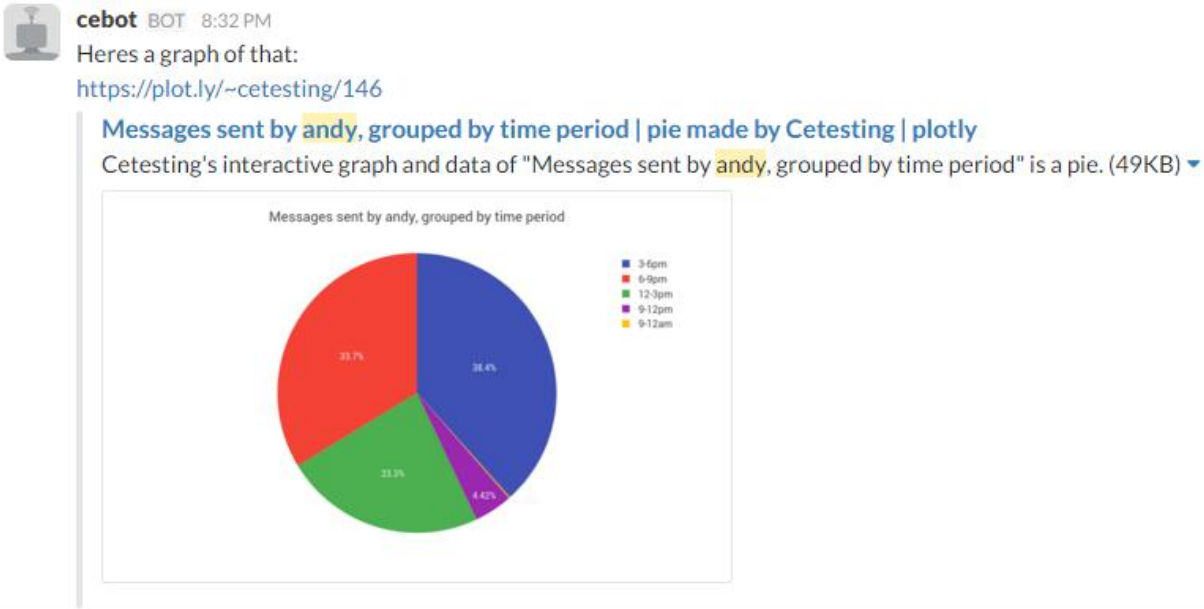

Figure 23 An example of the output produced by the question 'What times is andy often online?'

9.7.5 Active periods

Figure 24 shows the total number of messages a user has sent split between days of the week and time periods, this tells us that 3-6pm on a Tuesday is the optimum time to collaborate with this user.

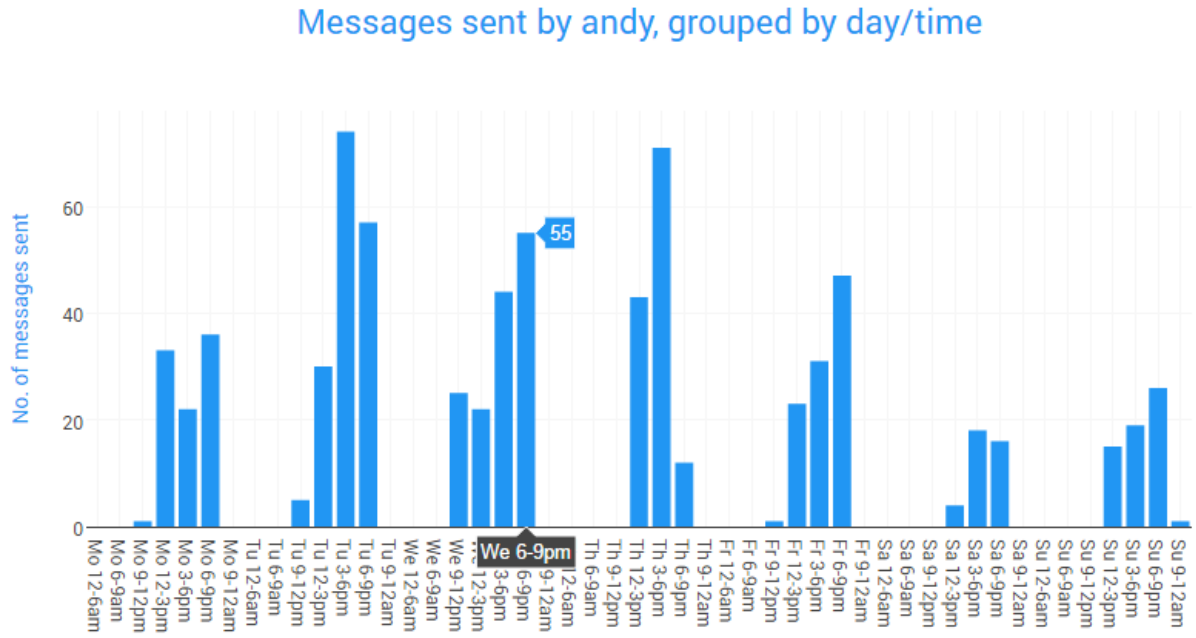

Figure 24 The view from the Plotly website of the output created by the question 'When is andy busiest?'

9.7.6 File log

Figure 25 shows a list of uploaded files with hyperlinks to their content. A broad range of filetype criteria can be used (along with the standard username and time criteria) to quickly find the files a user needs.

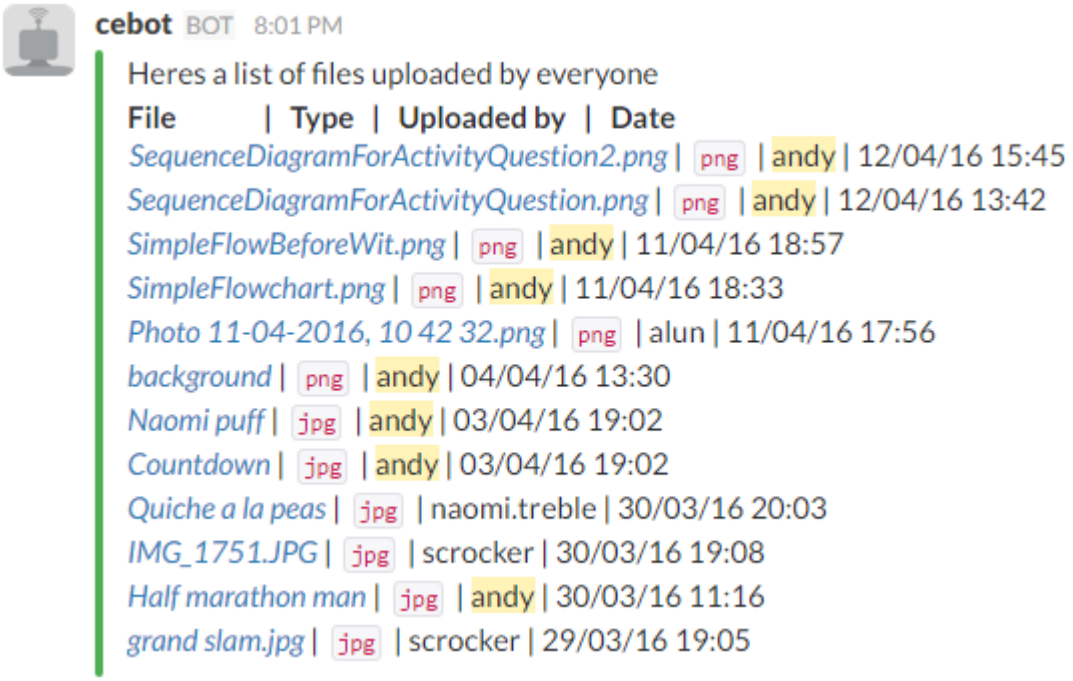

Figure 25 An example of the output produced when Cebot is asked to 'List pictures that have been shared'

9.7.7 Sentiment analysis

Figure 26 shows the extracted keywords, entities and concepts from a block of text. As you can see the entity extraction is very accurate, even identifying the CE Store and CEnode. This function is a good basis for summarising a user's messages within a time period, for example a user could use this to receive a summary if they have missed a days communication.

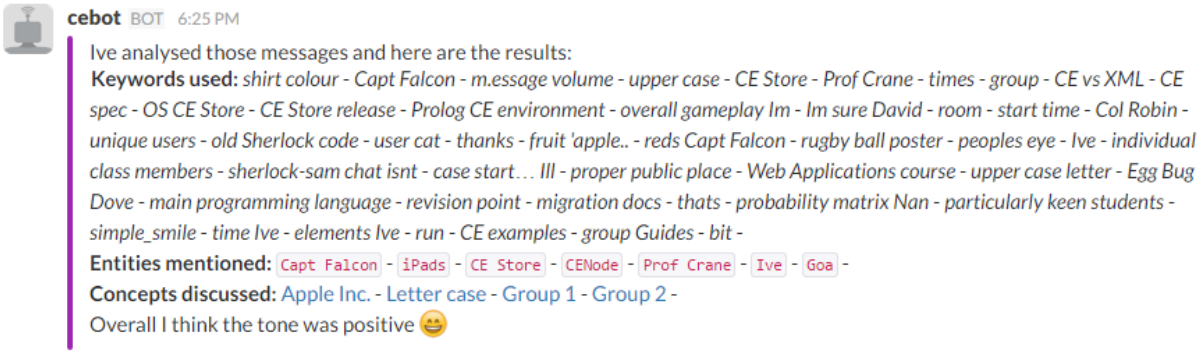

Figure 26 An example of the output produced when Cebot is asked to 'analyse messages from will in december'

9.7.8 Team statistics

Figure 27 shows a breakdown of the channels communication for a time period. This can highlight problem areas, for example a team member may be contributing less because they do not feel that their contributions are being listened to or one member may be dominating a channel making the others feel uncomfortable etc.

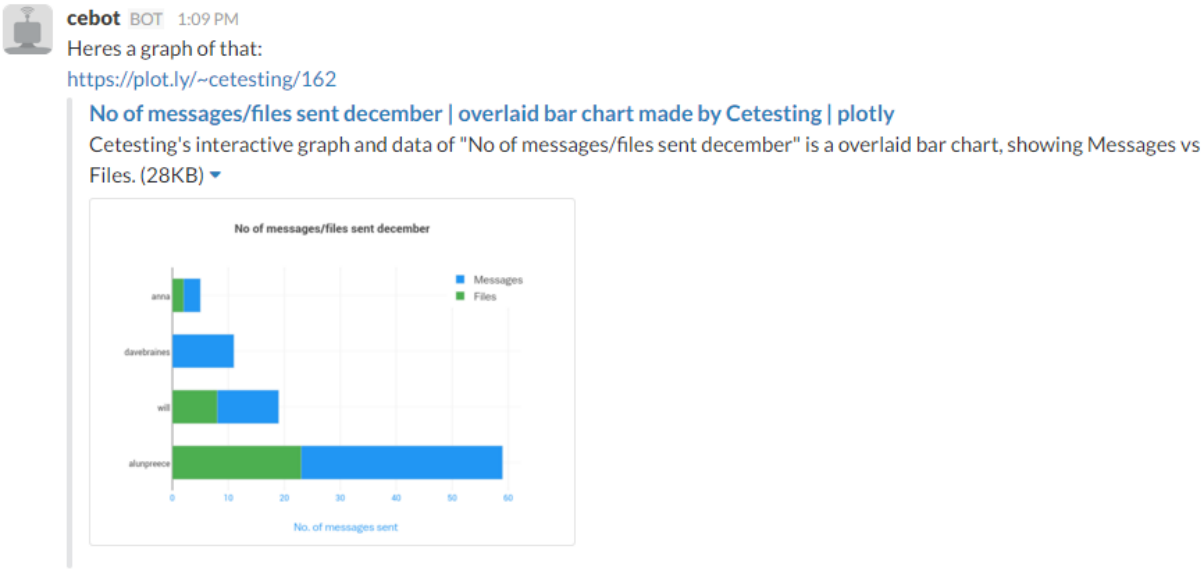

Figure 27 An example of the output produced when Cebot is asked 'show me the team statistics for december'

9.7.9 Final system diagrams

Figure 28 shows a basic overview of the final system, with the 'loading' of the system in the far left column and then the 'loop' as the system carries out its task and then returns to waiting for a new Slack event to be triggered. Figure 29 shows a sequence diagram of system when a question is asked that requires a response from Plotly. The diagrams

produced to answer the other questions supported would look very similar to this one except Plotly would either not be used or it would be replaced by the Alchemy service.

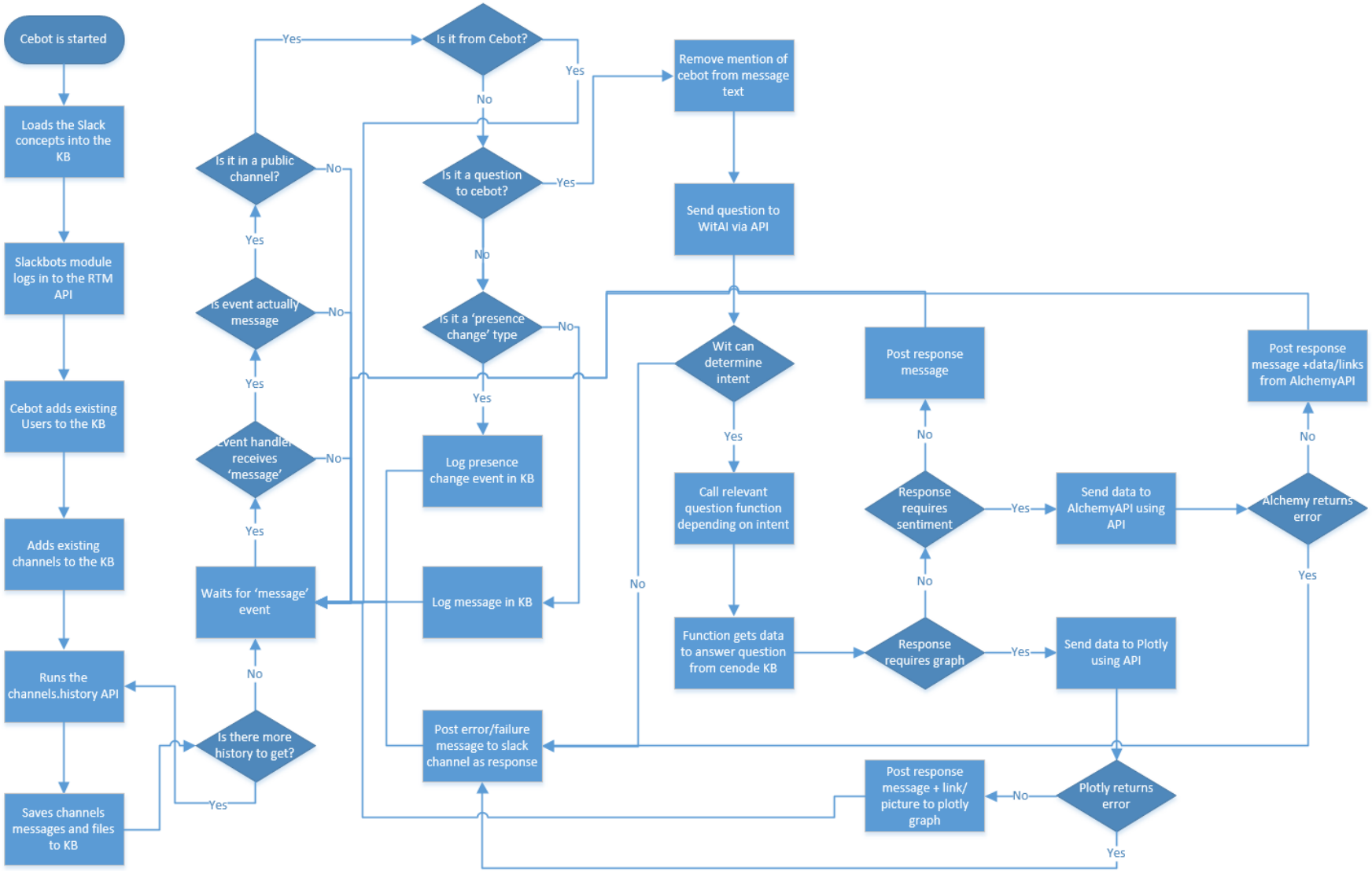

Figure 28 Flowchart of the design of the final system

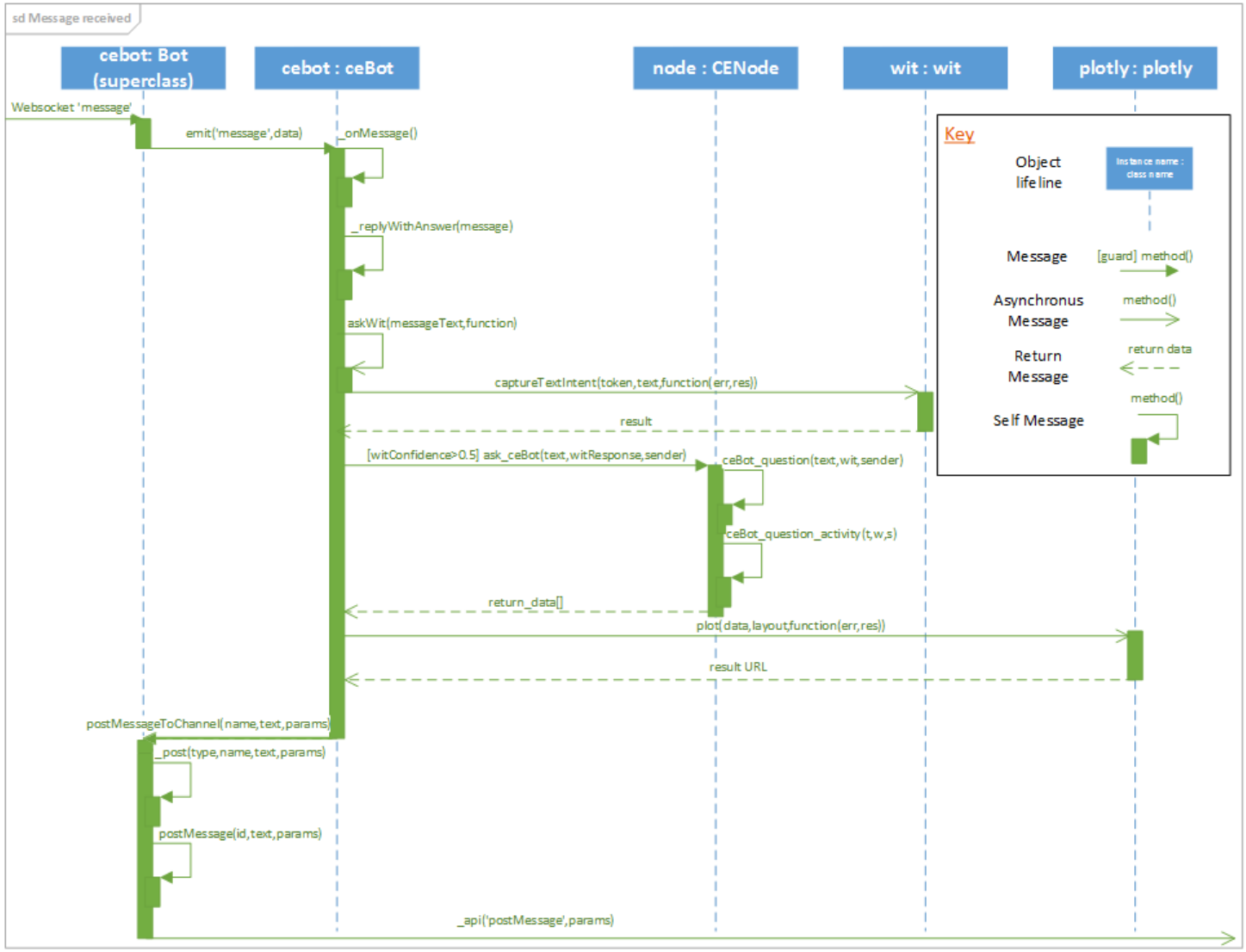

Figure 29 UML sequence diagram showing a question that requires an output produced by Plotly, based on the principles discussed at http://www.ibm.com/developerworks/rational/library/3101.html

10Testing

10.1 Testing method

The testing of the system was divided into two main sections, analogous to import and export. The thinking behind this was the system would never produce the correct output if the input was at fault. It would be easy for missing data to go undetected when there are thousands of messages being imported so it was crucial that import was tested first.

Throughout the development of this system function testing was used extensively, usually the functions were tested in separate files to ensure that the function operated as it should independent of outside interference. It would then be incorporated into the main system and further tests would be run to certify that it did not have an adverse effect on any other areas of the system.

The testing detailed here will be test cases based on the domain testing format. It would be infeasible to test all the possible values of a given variable so we will divide the tests into different groups and choose a few representative values from each group [Kaner 2016].

10.1.1 Storing the Slack environment in the KB

The first section was further split into two parts as the each required the use of different techniques.

1. Storing historical events
2. Storing events in real time

The first part uses a recursive loop alongside the Web API to download and store everything that's happened on that channel before Cebot was started. This part will test the system's ability to convert Slacks objects into valid CE and store them in the KB as part of their respective concepts. The history from six separate channels was tested each with differing amounts of active users, files and messages to simulate a wide variety of Slack activity. Four of these channels are based on exports from a functional Slack team

The second part uses the RTM API to monitor events that are happening on the channel in real-time. This part will test the capability of the system when responding to Slack events. The system was tested against events it was designed to respond to and events it was not to judge its ability to handle the majority of functionality Slack currently supports.

10.1.2 Extracting information from the KB

The second section was split into three parts that better reflected what each part of the systems underlying goal was. A regular test case testing the sentence "who does andy talk to" could fail to successfully retrieve the answer from the KB because the CE generated was incorrect, making the result of the test a failure. However the hard problem of interpreting the user's natural language query might have been successfully achieved and therefore not reflected in the failure result.

1. Understanding user input – Human to Machine
2. Querying the Controlled English KB – Machine to Machine
3. Outputting the response – Machine to Human

Part 1 tested the intents entities by choosing boundary values. The intents themselves had been tested at every stage of the development process the average length of the 5 main intents training set was now 54 sentences. As such we already knew that the NLP area of the system was very accurate and it was becoming difficult to word sentences in a way it had not seen before whilst making sure the sentences were still grammatically correct. Each of the entities are based on lists so the boundary values were a value that wasn't in the list and a null value. A successful test indicates that null values and values not in the list have been ignored and typical values have been successfully extracted.

Part 2 tested the ability of the system to convert the user's natural language sentence into criteria, querying the KB using that criteria and obtaining the correct answer for that sentence. Each intent has a corresponding function and it was the accuracy of these functions that was tested in this part. The system was tested based on data from a subset of the channels tested in the Storing historical events section as we have manually counted the correct number of messages etc for those subsets. A successful test will be one where the system has returned an object/objects containing the correct data as it relates to the question.

Part 3 tested the systems integration with its external services, namely the Slack Web API, the Plotly API and the Alchemy API. Each of the tests carried out in Part 2 were carried out in Part 3 but the results focused on the output visible to the Slack user and what they would expect to see vs what was actually output.

A successful test will be one where the system has posted a response in the format that we intended it to, without missing data or layout issues. Note that Part 3 does not have to post 'accurate' data to pass the test if it has been passed inaccurate data by Part 2, it just has to format it correctly.

10.1.3 Overall user experience

The system was further tested by two groups of users, novice users who had no previous experience with Slack and expert users who had all been using slack for at least 1 year. The novice group was invited to chat on the CEtesting team. These initial conversations helped to populate the KB with real data. Once we thought enough data had been logged they were told to type 'Cebot:help' and read the instructions Cebot posts. No further instruction was given as we observed the users testing the system.

The expert users were invited to a new Slack team created based on exports from a Slack team they had worked on previously. This meant that the system could be tested with real data without any chance of damaging the data. They were given the same instructions as the novice group.

10.1.4 The contact entity

The system relies upon the contact entity being able to recognise English names from a variety of sentences and as such this was tested separately. The entity was tested against 10 unseen English names taken from a list of the most common baby names in England for boys, swapping any names that occur in the training set with the next name on the list [Office for National Statistics 2015]. It was then tested against the list of girls names in ascending order of popularity with the least popular names being first on the list. It was also tested against 10 unseen valid Slack usernames, the accuracy of the system when tested

against usernames is expected to be far lower than when tested against proper English names.

10.1.5 The sentiment question

This function was tested in the same way as the others, did the function get the necessary data from the KB in order to produce the correct answer. However this raises an interesting question as to what the 'correct' answer actually is when it comes to judging sentiment or finding concepts in a section of natural language text. Humans often have problems judging sentiment from text alone and this can cause issues in an office environment where email is heavily relied upon, for example [McAndrew and De Jonge 2010] found that messages written in the third person were perceived as angrier than those written in the first person.

To test the accuracy of the various areas that the system uses (entities, keywords and sentiment) we devised a test that centred on a block of conversational text. If the text contained two people conversing then the block would be split in two and run once for each 'side' of the conversation, in order to mimic how the function would be used in Slack. If a side had less than 80 words then it was ignored. We also needed conversations that the majority of humans would agree were positive or negative to remove as much subjectivity as possible. The easiest place to get conversations that meet all these criteria is from popular films. We could have chosen popular books but the data for these is not as readily available, whereas film scripts have clearly labelled dialogue. Any narration or stage direction was removed, leaving just the characters dialogue.

We took the results from the API of the top 5 most relevant results in each category (relevance as given by the API is a value between 0 and 1), relevance values below 0.5 were ignored to mimic the way the function is set up for Slack.

The same text was then given to 8 volunteers who were told to highlight entities (named people, places or things), the top 5 keywords or phrases they thought were most relevant, the overall sentiment of the text (positive, neutral, or negative) and a sentiment score between 10 and -10. Entities were chosen if more than 60% of volunteers selected that word(s) as an entity. Because of the variety of keywords/phrases that could be picked by volunteers, any that had been picked by 3 or more people were chosen. If that did not result in 5 keywords/phrases being chosen then the list was filled in descending order of the most popular single keyword chosen.

The API sentiment score was multiplied by 10 to match this format for comparison. They were not asked to list concepts as the concepts that Alchemy returns are based on entries in online databases such as DBpedia so it would not be feasible to compare the results.

11 Results and Evaluation

11.1.1 Storing historical events

Table 2 Results of testing the storing of Slack channel data in the KB

Channel	No. of users	No. of messages	No. of files	Boundary value	Passed
#random	1	4	0	Files, Users	TRUE
#random (ITA)	4	3	1		TRUE
#general (ITA)	4	40	6		TRUE
#cenode (ITA)	4	80	7		N/A
#cenode (ITA) - August	2	2	2		TRUE
#sherlock (ITA)	8	626	130		N/A
#sherlock (ITA) - January	2	13	0	Files	TRUE
#general	7	999	22		N/A
#general - January	1	0	1	Messages, Users	TRUE

The results in Table 2 show each of the channels that the system was tested against. There is no way in vanilla Slack to have Slack calculate the number of messages or files in a channel, therefore testing that the KB was a true reflection of the Slack channel meant manually counting each message, file and user. For this reason the rows in grey were ignored, instead a single month's worth of data from that channel was used (the rows directly below grey rows). Months were chosen to highlight boundary values, these were the lowest values available as the limit to the number of users (9000ish), messages (10000) and files (based on data allowance instead of quantity) was too high to test. The system had no issues with any of these tests.

11.1.2 Storing events in real time

Table 3 Results of testing real time events

Event	Pass	Error code
channel_created	Yes	
channel_rename	Yes	
channel_deleted	Yes	
dnd_updated_user	Yes	
file_deleted	Yes	
file_change	Yes	
message	Yes	
pin_added	Yes	
pin_removed	Yes	
reaction_added	Yes	
reaction_removed	Yes	

Table 3 shows the results of testing various Slack RTM events on the system. None of these events had any negative effect on the system, any it is not explicitly coded to handle it would just ignore. This does raise an issue with CEnode when representing the Slack environment however. CEnode does not currently support the removal of instances, so if for example a channel is deleted there is no way to update the KB to reflect that. If the system is going to be expanded to fully model the Slack environment then this functionality must be added to CEnode.

11.1.3 Understanding user input

Table 4 Results of domain testing entities

Boundary		Test cases					
Variable	**Condition**	**1**	**2**	**3**	**4**	**5**	**6**
timePeriod	not in list	"moctober"					
	Typical		"january"				
	Null						
hours	not in list			"days"			
	Typical		"times"				
	Null						
fileType	not in list				"psd"		
	Typical					"excel"	
	Null						
duration	not in list						"28 parsecs"
	Typical					"28 days"	
	Null						
Expected result		Pass	Pass	Pass	Pass	Pass	Pass
Actual Result		Pass	Pass	Pass	Pass	Pass	Pass

Table 5 Results of domain testing intent

Boundary		Test cases		
Variable	**Condition**	**1**	**2**	**3**
Intent	not in list	"what is the weather like in wales"		
	Typical		"show me a list of word files uploaded by dave"	
	Null			
Expected result		Pass	Pass	Fail
Actual Result		Pass	Pass	Fail

The test cases shown in Table 4 and Table 5 are designed to test the boundaries of the NL service. It was expected that any value not in a list or any null value would be ignored and that was the case. The only test to fail was the test that involved sending Wit a null value.

This somewhat likely possibility had been overlooked during the design and should be remedied.

11.1.4 Querying the Controlled English KB and Outputting the response

Table 6 Results of testing the systems accuracy when querying the KB and displaying the results of those queries

Channel	No. of users	No. of messages	No. of files	Mentions	Boundary value	Query Pass	Output Pass
#random	1	4	0	0	Files, Users, Mentions	TRUE	TRUE
#random (ITA)	4	3	1	knolleary - 2/1		TRUE	TRUE
#general (ITA)	4	40	6	davebraines - 12/13		TRUE	TRUE
#cenode (ITA)	4	80	7			N/A	N/A
#cenode (ITA) - August	2	2	2	will - 1/1		TRUE	TRUE
#sherlock (ITA)	8	626	130			N/A	N/A
#sherlock (ITA) - January	2	13	0	anna - 1/1	Files	TRUE	TRUE
#general	7	999	22			N/A	N/A
#general - January	1	0	1	0	Messages, Users, Mentions	TRUE	TRUE

Table 6 shows the results of tests designed to judge the robustness of the systems queries and its outputs. The data for these tests is the same as the data from Section 11.1.1 to ensure that any errors picked up here were not caused by an incorrect import. To test the 'mentions' function we picked one user and manually counted how many times they mentioned other users (the first number) and how many times they were mentioned (the second number). The system did not encounter any problems whilst performing these tests. However we did spot an issue with the output of the file function. It appears as though the output is truncated after 13 rows of files so asking for a list of all files in either the #general or #sherlock channels will result in a failed output. This should be remedied in a future version.

11.1.5 Overall user experience

We have no formal results for this section. Anecdotally novice users said the system was very easy to use but as they had never used Slack before they were unsure of how useful it would be. Experienced users also found the system very easy to use and were much more interested in the data extracted about themselves and their team. We had originally planned to carry out a formal user evaluation involving the user completing multiple tasks,

both with vanilla Slack and with Cebot. Unfortunately due to problems with the Wit service and other development issues, time constraints would not allow us to design an unbiased test and we were not able to undertake this evaluation. We believe we would have seen positive results from this evaluation.

11.1.6 The 'contact' entity

Sentence submitted: "how many times has [name] talked to me?"

Table 7 Results of testing the ability of the system to extract common names

Name	Captured	Intent correct	Confidence	Incorrect value
oliver	No	Yes	0.996	
jack	Yes	Yes	0.998	
harry	Yes	Yes	0.999	
jacob	Yes	Yes	1	
charlie	Yes	Yes	0.978	

Only the first 5 boys names are shown in Table 7 for the sake of brevity, we ran this test for 5 further boys names and 10 girls names and every single name was captured with an average confidence of 0.97.

Even though this is an unseen question that is not validated and is infact deleted before the next submission the confidence value that Wit returns is rising steadily. This implies that the Wit AI is learning from the sentence submissions, which makes testing difficult. If Wit is using pattern recognition to guess that any word in that area of the sentence is a contact then all of this method of testing may be fruitless.

To determine this we tested the alternative approach with 5 unseen sentences and the next 5 unseen boys names from the list.

Table 8 Results of testing to determine if Wit solely uses pattern matching

Sentence	Captured	Intent correct	Confidence
who does noah speak to?	Yes	Yes	0.998
who mentions joshua often?	Yes	Yes	0.998
who does alfie talk about?	Yes	Yes	1
who chats to muhammad	Yes	Yes	0.981
who is chatting to henry a lot?	Yes	Yes	0.952

The results in Table 8 suggest that Wit is not just using sentence structure to determine names. They are probably checking against a list of common names as this is a computationally easy task, and then using the pattern matching to increase accuracy (avoiding words that are also common names such as 'will') and capture obscure names.

This alternative approach also presents issues, the least of which is that we would require a minimum of 20 new sentences (40 if we still wanted to test the less popular names). We also would not be able to tell if the Wit system was not detecting a name because the sentence structure was so unfamiliar or because it just didn't recognise the word as a viable

name. We also cannot reuse a name in case it recognises it as a contact based on previous appearances. Ultimately this approach is unfeasible as many of the intents already have 40 + training sentences and if Wit is remembering deleted sentences then we cannot ensure that our testing set is actually unseen, making the data produced unreliable.

11.1.7 Sentiment analysis

Table 9 Results of humans extracting entities vs automatic extraction

Text	Questionnaire responses	Alchemy Service	Fuzzy match
1	The Emperor, Luke, Obi-Wan, galaxy	Luke	25%
2	Scotland, Petticoat Lane, Nedry	Nedry, Scotland	67%
3	Alan, John, Lex, Tim	John, Alex, Lex, Tim	100%
4	Arendelle	Arendelle	100%
5	Anna	Anna	100%
6	Mankind		0%
7			100%
8	Lieutenant Weinberg, Marines, Santiago	Santiago, Lieutenant Weinberg	67%
9	Andy, Buzz, Space Ranger	Andy	33%
10			100%

As you can see from Table 9 on the whole entity extraction was very accurate. Every entity Alchemy extracted had also been picked out by the volunteers. It is difficult to tell whether the entities it missed were because it couldn't detect them as entities or whether they do not fall under the developer's definition of entities. The documentation simply lists entities as named People, Places or Organisations which is the same definition that was given to the volunteers. However it is clear that some volunteers took a very broad interpretation of this definition as they highlighted words such as 'you' and 'I'.

Table 10 Results of humans extracting keywords vs automatic extraction

Text	Questionnaire responses	Alchemy Service	Fuzzy match
1	destiny, power, father and son, father, galaxy	destructive conflict, combined strength, father, Search your feelings, galaxy	40%
2	fleas, spared no expense, flawless, creation	fleas, Clown fleas, high-wire fleas, Petticoat Lane, flea circus	20%
3	illusion, power, never had control, dying, people	Flea circus, illusion, power, mistake, people	60%
4	forever, together, storm, winter	time, eternal winter, winter weather, deep snow, cause	20%
5	alone, fear, free, life, safe	Anna, gates, fool, sun, curse	0%
6	independence day, 4th of july, freedom, right to live, fight	largest aerial battle, Good morning, petty differences, new meaning, common interests	0%
7	bare necessities, life, necessities of life, worries, forget	bare necessities, simple bare necessities, prickly pear, Old Mother Nature, life	40%
8	handle the truth, luxury, responsibility, defending, truth	greater responsibility, punch line, Santiago, luxury, truth	60%
9	cool, forget about me, toy, greatest, help	Buzz, Buzz Lightyear action, Space Ranger, toy, COOL toy	40%
10	slippers, trouble, alive, dog, pleasure	nice little dog, good time, good little girl, unexpected pleasure, slippers	60%

Table 10 shows the comparison between human keyword extract and Alchemy's. We believe these results are very positive, despite the low matching percentage. Many of these texts were hundreds of words long and yet most of the time Alchemy identified 2 out of 5 most important keywords to human volunteers. Even in areas where it did not match any keywords, text 6 for example, it still extracted very important phrases from the text and provided a good summary of text that was 160 words long.

Figure 30 shows the sentiment score given by humans and alchemy to each piece of text. A positive score indicates a 'positive' sentiment and a negative score indicates a 'negative' sentiment. Ignoring the height of the scores (as Alchemy appears to be adverse to high scores) Alchemy gave the same overall sentiment as humans in 6 out of 10 texts, which is slightly better than chance. However as the error bars indicate, the volunteers were very divided on the sentiments of the texts, with scores varying widely in some cases. This demonstrates that even humans have struggle to identify sentiment from text and the difficulty of this problem area. These results can be improved in the Slack environment through the frequent use of emoji's which can be passed to the Alchemy service as plain text e.g ':grin:' or ':joy:'

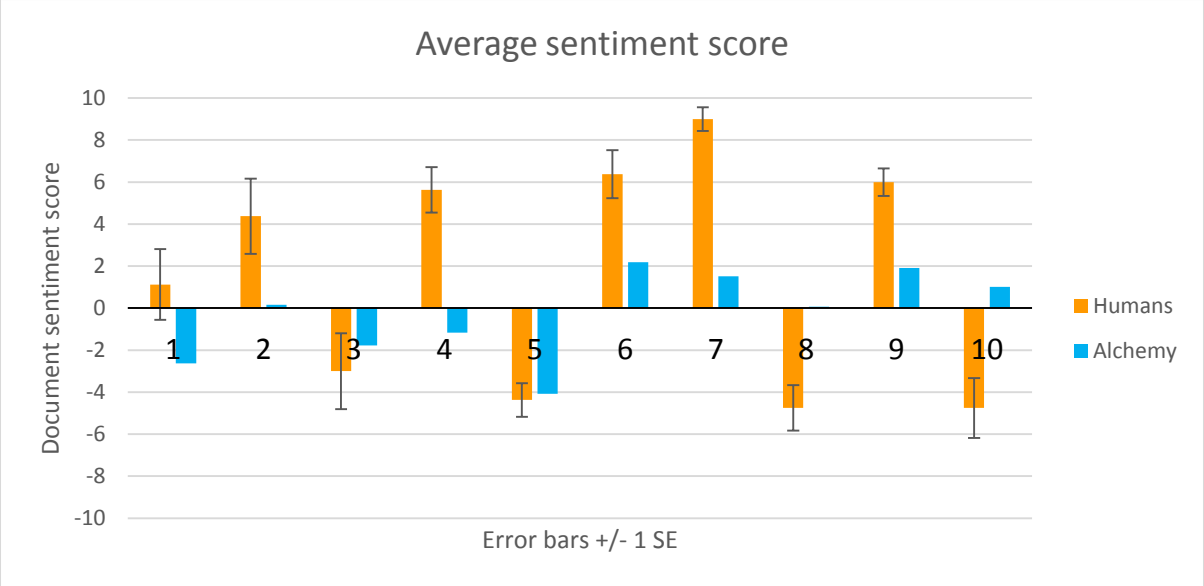

Figure 30 Graph showing the average sentiment score determined by volunteers vs the sentiment score returned by Alchemy

Obviously there are a number of problems with this test, the main being that dialogue from films is not often a representation of real life dialogue. Further the human reviewers may be judging their evaluation of the text on their memory of the film, if they have seen it. However it is a relatively good starting place to begin looking at ways to improve sentiment analysis for the system.

11.2 Strengths

NLP aspects of the system are very strong which has led to a dramatic increase in the functionality of CEnode. Previous CEnode apps have hard coded specific questions into the app in order to provide a response to sentences other than the supported 'who/what is'. As demonstrated above this system can respond to a wide variety of conversational styles and will continue to learn from future user input. Supporting a wide variety of sentence structures was important from both a user interface and a functionality perspective. As sentences are not hard coded the system can easily be expanded with new queries simply by creating a new intent.

The queries are proven to be both reliable and accurate. In addition to the formal tests above the system was tested hundreds of times a week whilst it was being developed. The way that the functions have been developed means that the system is easily extensible. Future functions can filter the two most useful criteria, date and user, from wit sentences with only a few lines of code.

Results are returned very quickly. Results not requiring external services to output i.e Plotly or Alchemy are near instantaneous. Those that do require external services are still returned in under five seconds, which is important for the user experience, especially as there is no 'loading' UI.

One of the main advantages of NL is that it can be used by complete novices. There is no interface to learn your way around or syntax to remember, all the user needs to know is what types of things they can ask about.

The CE that the results are based upon can be examined by users. Cebot keeps a log of the last query and the underlying data from the KB that it is based on and can post it as a 'snippet' in a channel. This means that users can verify the information they are receiving, which is especially helpful if they encounter any erroneous outputs. Because the data is stored in CE it is easy for a human to read and it is presented in a conversational style.

The system has effectively used data mining to provide insights that are not possible solely using Slack. The system was originally intended to improve Slack's search providing a whole range of criteria for sorting through the data that is stored in a Slack channel. However Slack has been making rapid improvements to its UI and search functionality so we decided to focus on extracting trends and hidden knowledge from the environment. All of our functions provide answers that a human user cannot easily retrieve from Slack, in fact this caused us problems when it came to testing as the only way to verify the number of times a user had mentioned another user was to manually count them, which quickly became infeasible with normal channels.

11.3 Weaknesses

In the middle of formally testing the system the Wit service stopped providing outcomes. The website was still accessible and the service continued to respond to API calls but both the website and the API would not return any outcomes. As you can see in Figure 31 a response was still being returned by the outcomes array was empty.

Figure 31 The screenshot on the right shows the expected JSON response from Wit. The screenshot on the left shows the Wit website, highlighted in red is the response it provided

The outage was caused by a major update to their service introducing an error, this error lasted for 4 days. This was catastrophic to the functionality of our system. It would still respond to "[who/what] is a [concept]" type questions because they use CEnodes built in fucntions. Any other type of question would crash the system. We designed the system to cope with the Wit service being entirely unreachable, in which case it would return an error, or not understanding the question at all and returning 'unknown'. The API documentation does not mention a scenario where Wit would only provide a partial response and we had not foreseen it.

The loss of functionality was so great that I had effectively introduced a single point of failure into the system. We talked in the approach about the downsides of using 3rd party systems and the lack of control you have over them but at some point, especially with an internet dependant system, you will always be relying on a 3rd party, whether that is Slack itself or the server provider who hosts your system. All you can do as a developer is to ensure that there are sufficient backups and redundancies to handle foreseeable threats to the system. Unfortunately there does not seem to be a way to extract the data from Wit in order to back it up. A future project should look at ways of mitigating this vulnerability.

Another weakness was discovered when the system was evaluated using a real Slack team. The system crashed when queried about that channels data. The error was traced back to a message instance that had been created called 'concept'. It should not have been possible for an instance to be named that as all message instance names are based on the Slack timestamp for that message. The 'concept' instance also had no values assigned to it and appeared to be a duplicate of the instance preceding it in the KB.

This instance was created because the message that the slack user sent included the phrase "the message concept". One of our concepts is named 'message' so it appears CEnode was treating this phrase as a request to store a new instance of this concept and naming it 'concept'. We believed that by having the system construct the CE it would not be possible for slack users to introduce new concepts or instances and that those instances would only be created in response to events occurring in the Slack environment.

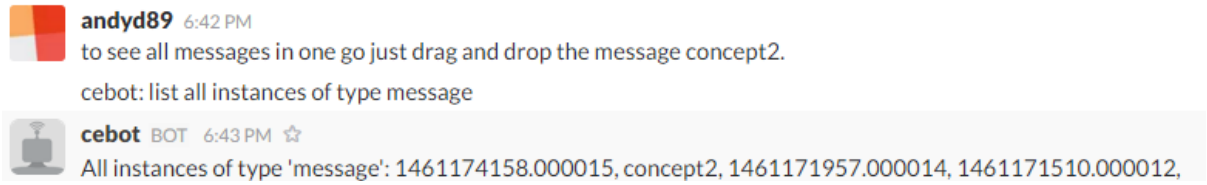

Figure 32 A screenshot that shows a message that creates the error and the erroneous instance in the KB

Here is an example of the CE we generate to store a message like the one shown Figure 32.

```
there is a message named '1461173208.000015' that has the slackuser
'U1273U1RV' as sender and has the channel 'C2147483705' as channel
and has 'to see all messages in one go just drag and drop the
message concept2' as messagetext and has '1461173208.000015' as
timestamp
```

We had incorrectly assumed that purpose of the apostrophes that surround variables was to tell CEnode not to process those words as CE. The way it actually works is described as "The incoming text is first split into phrases (multiple sentences), sentences, clauses (within a sentence, e.g., separated by commas or semicolons) and words. The agent then operates at a sentence level and scans the words left to right looking for matches to the CE model (details below). For each word (from left to right), the agent looks in the knowledge base to find any matches against the CE names of the concepts/instances/properties plus any synonyms defined (using CE)." [Preece et al 2014]

CEnode was designed from the bottom up to allow users to add multiple concepts and instances from a single sentence submission.

```
relationship_facts_singleword = t.match(/(?:\bthat\b|\band\b|) (?!\bhas\b)([a-zA-Z0-9 ]*) the ([a-zA-Z0-9 ]*)/g);
synonym_facts = t.match(/is expressed by '([^'\\]*(?:\\.[^'\\]*)*)'/g);
```

Figure 33 Screenshot showing the CEnode code that created the erroneous instance

In this case the words were captured by a regular expression looking for relationship facts in the submitted sentence, as shown in Figure 33. When the function could not find an instance to match that relationship it created an instance.

In our system this behaviour allows code injection, whereby a user exploits the systems design to inject unwanted and potentially damaging code. These types of attacks are common against JavaScript based systems and developers can use built in functions to sanitize inputs that are at risk. There are two obvious solutions to this problem.

The first is to change the names of our concepts to words that would not commonly be used in conversation. This approach may be simple but it is not very secure. For example there are around 1.5 billion messages sent on Slack each month [Smith 2015]. The odds may be low that someone will send a message containing an obscure concept name but once they do the system will not successfully run until that message is deleted.

The second option is to create a ‘blacklist’ of values that should be stripped out or altered in some way so as to not be recognised by CEnode. We had already implemented a similar list to strip out characters such as speech marks that cause errors when CEnode parses them. However this should be expanded to dynamically pull the concept names from the KB and alter an occurrences of those words before they are passed to CEnode.

12 Future Work

Through the course of this project we have realised many ideas that we have not been able to bring to fruition or even experiment with. The ideas below are listed in ascending order of difficulty.

The keywords supplied by Alchemy can be used with a tf-idf algorithm to create a more readable summary for the asker by returning full sentences along with the keywords. Each message sent by a slack user is input as a document for the corpus. The algorithm will then return a score based on how many times each keyword was mentioned in each document. Returning the top 5 documents will give the asker a summary of the 5 most important sentences determined by keyword. This could be used as an easily digestible end of day report for a line manager of software development team for example.

The system is modelled on a subset of the actual Slack environment. This model could relatively easily be expanded to encompass the entire Slack environment with enough time. Many of the missing objects are based on existing objects, for example the system can already process some events but others, such as a user updating their Do Not Disturb information, are ignored. Similarly, Channels are implemented but Direct Messages are not as they were not needed to meet our aims, however we are sure being able to message Cebot in private would provide value to the user.

Effectively modelling the Slack environment would require changes to CEnode however. CEnode currently does not support the removal of instances from its KB. This would need to be implemented so activities like channel deletion and message editing would not cause the KB to be based on inaccurate data.

The ways in which the system provides a response to the user can be expanded. Graphically there are many tools that we could use to make it easier for the user to parse the data at a glance. One of the graphics we were experimenting with was a heatmap for user activity. The API code for heatmaps is quite complicated so we did not have time to produce this unfortunately. Figure 34 shows an example of the type of heatmaps Plotly supports.

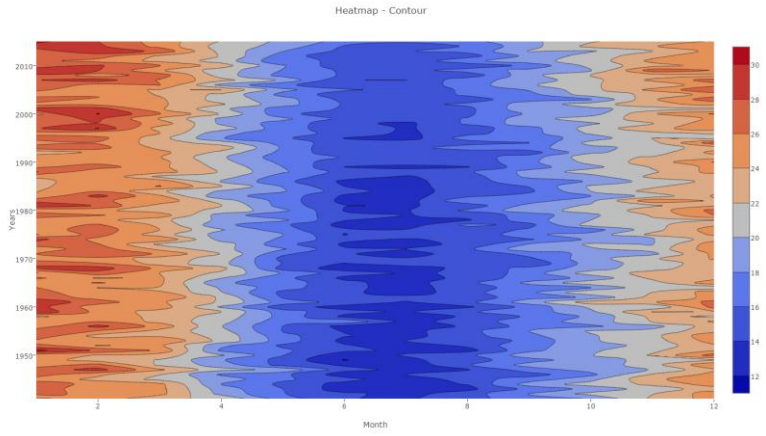

Figure 34 Plotly heatmap Source: Plot.ly (https://plot.ly/3/~MattyK8/)

Table 11 Simple heatmap showing number of messages sent by a user on days within time slots

	12-6am	6-9am	9-12pm	12-3pm	3-6pm	6-9pm	9-12am
Monday	0	0	1	32	22	36	0
Tuesday	0	0	5	30	74	57	0
Wednesday	0	0	25	22	44	55	0
Thursday	0	0	0	43	71	12	0
Friday	0	0	1	23	31	47	0
Saturday	0	0	0	4	18	16	0
Sunday	0	0	0	15	19	26	1

Table 11 shows an example heatmap table made in Excel. With this format you can instantly see at a glance that Tuesday and Thursday between 3 and 6pm is when this user sends the most messages and that there is no point messaging them anytime between 9pm and 9am because they will not be online.

In a similar vein the number of times users have mentioned each other could be changed from a text based response to a social graph, with nodes representing users and the size of the edges increasing based on how often those users talk to each other. We could also use a similar graph for keywords and concepts, with the size of the nodes representing the amount of time that user has used those keywords/concepts.

Currently the system supports the Ask/Tell relationship with CEnode in the following ways. Every message that is directed at Cebot is treated by CEnode as an ‘ask’ that requires an answer. Every event that occurs in Slack and needs to be recorded in CEnode is treated as a ‘tell’ whereby we are updating the knowledge base by telling it something. Currently the users cannot ‘tell’ CEnode things directly, this occurs behind the scenes.

Implementing this feature would open up a large chunk of functionality particularly in the knowledge management area. For example a user could tell Cebot that they are interested in a particular subject. Cebot would then DM that user every time someone references that subject or it could provide a summary of messages, similar to an RSS feed. Users could make this task easier on the system by using hashtags e.g ‘since the last build the loading bar is now flickering #UIbug’. This would need to be handled carefully to ensure that users do not inadvertently change an aspect of the KB that should be determined by the Slack environment, for example the channels that exist for this team. Doing so would certainly crash the system and could prevent it from starting again if it continues to try to load the incorrect instance into the KB.

Inversely Cebot cannot ‘ask’ the user questions. This functionality would be easier to implement in a DM between the user and Cebot as in a public channel other users could accidentally respond to Cebot or the actual user could mistakenly answer Cebot when trying to respond to a colleague. This could be used to ask a user what they are working on that day when they log in for example. Each time a user completes a task they could tell Cebot about their new task. Their project manager could then ask Cebot for a summary of this information to keep track of how the project is progressing. Cebot could ask a user about their particular skills and experience. It may have already created a model based on what

that user has been discussing with other users. It could ask for their LinkedIn profile to further build up this model. It could then be used as a skills map to help locate expert users within the Slack team. In both scenarios we would also need to be careful not to create a Microsoft Clippy situation whereby instead of aiding users we end up irritating them [Veletsianos 2007].

In this project we used Wit to solve the AI hard problem of NLP in our CEnode app. However it is clear that CEnode itself needs support for processing a wider range of user queries. Wit was a good solution for this system as the system requires internet access by virtue of the fact that it has to interact with Slacks servers. CEnode however is designed to be lightweight and not require internet access. A whole project could be organised around examining Wit and its competitors, picking desirable features, investigating how they work and ultimately trying to recreate them for CEnode.

Much research has been performed in the area of speech recognition and determining intent from said speech. The task of incorporating intent classification into CEnode is made easier by virtue of not needing to transcribe spoken speech into text. [Gorin et al, 1997] details a method of focusing on the salient grammar in the submitted sentence and then mapping those fragments to a machine action. [Gorin et al, 2002] discusses an approach of using clustering algorithms to group similar salient phrases to increase accuracy in classifying intent.

[Gupta et al, 2006] takes a different approach than Gorin's work and instead attempts to extract a user's intent rather than the machine action that should take place. This means that the model can be applied to many different applications, rather than just the one it was specifically built for. They also identify an approach that minimises the requirement for humans to label large amounts of training data, which will be important if intent classification is to be included in CEnode

[Sarikaya 2008] combines three methods to also significantly reduce the amount of effort required to label testing data and then devises a method to use existing data sets found on the Web to build reliable language models when there is limited relevant domain data.

[Young et al, 2010] describes the improvements found when using a statistical approach to intent classification. Their probability based model shows good results when compared to traditional deterministic rules for interpreting dialogue.

Obviously the accuracy of a lightweight, offline system would be inferior to the Wit system that can use data mining and rules based on results from thousands of applications hosted on its servers but the benefit of it being offline and independent from a 3rd party should outweigh the downsides.

The sentiment area of this project could also use some work. Alchemy performed well in extracting keywords from a block of text but the sentiment score was often wrong and the concepts were comically bad, with the Billy Joel song 'We didn't start the fire' making frequent appearances when we asked for a summary of users feelings. This is another area of ongoing research and could be spun off into a new project, comparing current algorithms and solutions and possibly combining the best aspects into a new solution, preferably

something that can be modified by the users to better match the problem their team is trying to solve.

13 Conclusions

In this dissertation we have identified a problem with Slack, namely its users dissatisfaction with search and its inability to provide a high level view of what is occurring within a Slack channel. We proposed a solution to this problem based on combining a Slack integration with a Controlled Natural Language knowledge base. We first demonstrated that such a solution was technically feasible and then created a prototype, meeting our first aim.

During the course of implementing the solution we discovered that CEnode was unable to provide the NLP that we required to parse a user's input into an actionable intent. We researched various methods of doing this and chose a method that could be implemented within the timeframe. We demonstrated that this method successfully supports Human to Machine communication using a wide variety of Natural Language sentences and can reliably extract two important types of criteria.

We realised that Slacks formatting options were not always acceptable to display the results of more complex queries to the user. To get around this restriction we used a 3rd party service to host our dynamically generated charts and graphics. Through various formats including graphs, rich text and 'snippets' we supported Machine to Human communication.

We investigated NLP solutions for extracting keywords and sentiment from a collection of users messages. We successfully implemented a service based on Watson and judged its effectiveness. This service supported Machine to Machine communication, providing analysis on the data in our knowledge base. We found its entity and keyword extraction to provide useful results but its sentiment analysis was slightly disappointing.

We produced a fully functional, stable Slack bot and proved that it can successfully parse a wide variety of users natural language input. Using a Controlled English knowledge base we were able to analyse thousands of events quickly and accurately, exposing trends in collaborative working that would have gone unnoticed in vanilla Slack. Users can also extract the raw data in a Controlled English format to analyse themselves.

In conclusion the main outcomes of the project are (1) a prototype platform integrating a Natural Language knowledge base with Slack, and examples of Human to Machine, Machine to Human and Machine to Machine services; (2) tested on a real team with positive initial feedback; (3) identified areas for immediate extensibility including relevant research and existing techniques that could be exploited. This project provides a solid foundation for further projects to be based upon, especially in the areas of intent sentence classification, keyword extraction and sentiment analysis where we have identified approaches that should be investigated further.

14 Reflection on Learning

This project has been a valuable learning experience. I now have proven experience in a number of areas from GitHub to NodeJS to JavaScript. I am sure this project will serve as important evidence for future employers, not just as evidence of the technical skills I have learnt but also of my ability to manage a project independently, to tight timescales and under stressful circumstances.

I am not the most organised person and with a project of this size it was very important that meticulous notes were kept. I used OneNote as it is backed up to the cloud and can search text within screenshots. Each weeks work was logged on a separate pad and detailed notes were taken, along with links to helpful resources. This made constructing the final report much easier however there are still some ways that this could have been done better.

Storing knowledge by weeks was great for showing what had been achieved in that period but is less useful when you need to find a particular error message for example. What I should of done is made extensive use of the 'tagging' system that OneNote has. Notes can be tagged with labels such as 'bug', 'toDo', 'Important' and it also supports custom tags. Tags can then be summarized and searched. Spending longer properly categorizing my data would have made collating the final report even easier and it is definitely something that I will start to do as I enter the working world.

Throughout this project I was careful to ensure that I had suitable backups of all important work, atleast one physical copy and one stored off site in the cloud. However the work that I had put into training Wit AI could not be backed up, there was no export option. When the service went down I initially assumed it would be up again within a couple of hours. The outage lasted four days and it was one of the most stressful periods of my life. I had no idea when service would be restored, I had just begun to carry out formal test cases and to my horror discovered that the system would not function at all. To ensure that this time was not wasted, I designed a method to test the effectiveness of the Alchemy service. This test was quite rushed as I was panicking that I was losing time and not making progress. The initial idea for this project was to carry out user evaluation tests to see whether the system could reduce the time taken to carry out various tasks within Slack. I am very disappointed that I did not have time to undertake these assessments as I think they would have shown that the system could have a measurable impact on a user's ability to manage knowledge in the Slack environment. This experience will definitely make me less reliant on 3rd party services in the future, especially for important tasks.

15 Table of Abbreviations

AlchemyAPI (Alchemy)

Application Program Interface (API)

Artificial Intelligence (AI)

Controlled Natural Language (CNL)

Direct Message (DM)

Human Computer Collaboration (HCC)

Human Computer Interaction (HCI)

Instant Message (IM)

Internet Relay Chat (IRC)

ITA Controlled English (CE)

Knowledge Base (KB)

Multi Person Instant Messages (MPIM)

Natural Language (NL)

Natural Language Processing service (NLP)

Real Time Messaging (RTM)

WitAI (Wit)

16 References

Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L. and Stent, A. 2001. Toward Conversational Human-Computer Interaction. AI Magazine (4). Available at: http://www.aaai.org/ojs/index.php/aimagazine/article/viewFile/1590/1489 [Accessed: 6 May 2016].

Baer, D. and Gould, S. 2016. Here are the 20 fastest companies to reach a $2 billion valuation [Online]. Available at: http://www.techinsider.io/fastest-companies-to-reach-a-2-billion-valuation-2015-5 [Accessed: 6 May 2016].

Constine, J. 2016. Slack's growth is insane, with daily user count up 3.5X in a year [Online]. Available at: http://techcrunch.com/2016/04/01/rocketship-emoji/ [Accessed: 6 May 2016].

Gorin, A., Abella, A., Riccardi, G. and Wright, J. 2002. Automated natural spoken dialog. Computer 35(4), pp. 51-56. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.9469&rep=rep1&type=pdf [Accessed: 6 May 2016].

Gorin, A., Riccardi, G. and Wright, J. 1997. How may I help you?. Speech Communication 23(1-2), pp. 113-127. Available at: http://disi.unitn.it/~riccardi/papers/specom97.pdf [Accessed: 6 May 2016].

Gupta, N., Tur, G., Hakkani-Tur, D., Bangalore, S., Riccardi, G. and Gilbert, M. 2006. The AT&T spoken language understanding system. IEEE Transactions on Audio, Speech and Language Processing 14(1), pp. 213-222. Available at: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1561278&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1561278 [Accessed: 6 May 2016].

Kahn, J. 2016. Slack Targets Salesforce, Oracle With Productivity Aim [Online]. Available at: http://www.bloomberg.com/news/articles/2016-04-27/slack-targets-salesforce-oracle-with-productivity-ambitions [Accessed: 6 May 2016].

Kaner, C. 2016. What Is a Good Test Case?. Ph.D. Florida Institute of Technology.

Kim, E. 2016. Startup founder claims $2.8 billion startup Slack is misleading people about its free 'unlimited' plan [Online]. Available at: http://uk.businessinsider.com/slack-free-unlimited-plan-has-limits-2015-6?r=US&IR=T [Accessed: 6 May 2016].

Kiss, J. 2016. Beyond email: could startup Slack change the way you work? [Online]. Available at: https://www.theguardian.com/technology/2016/mar/25/slack-butterfield-emoji-chat-nasa-harvard-silicon-valley [Accessed: 6 May 2016].

Manjoo, F. 2015. Slack, the Office Messaging App That May Finally Sink Email [Online]. Available at: http://www.nytimes.com/2015/03/12/technology/slack-the-office-messaging-app-that-may-finally-sink-email.html?ref=technology&_r=2 [Accessed: 6 May 2016].

McAndrew, F. and De Jonge, C. 2010. Electronic Person Perception: What Do We Infer About People From the Style of Their E-mail Messages?. Social Psychological and Personality

Science 2(4), pp. 403-407. Available at: http://spp.sagepub.com/content/2/4/403 [Accessed: 6 May 2016].

Mott, D. 2010. Summary of ITA Controlled English. Available at: https://www.usukita.org/papers/5658/details.html

Nielsen, F. 2011. AFINN [Online]. Available at: http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010 [Accessed: 6 May 2016].

Office for National Statistics, 2015. Baby Names in England and Wales, 2014.

Preece, A., Braines, D., Pizzocaro, D. and Parizas, C. 2014. Human-machine conversations to support multi-agency missions. SIGMOBILE Mob. Comput. Commun. Rev. 18(1), pp. 75-84.

Sarikaya, R. 2008. Rapid bootstrapping of statistical spoken dialogue systems. Speech Communication 50(7), pp. 580-593. Available at: http://www.sciencedirect.com/science/article/pii/S0167639308000447 [Accessed: 6 May 2016].

Slack, 2016a. Awesome [Online]. Available at: https://slack.com/apps/A07V4HDSP-awesome [Accessed: 6 May 2016].

Slack, 2016b. Ideabot [Online]. Available at: https://slack.com/apps/A0A6LHSPK-ideabot [Accessed: 6 May 2016].

Slack, 2016c. Message Formatting | Slack [Online]. Available at: https://api.slack.com/docs/formatting#linking_to_urls [Accessed: 6 May 2016].

Slack, 2016d. Nurtz [Online]. Available at: https://slack.com/apps/A0K4NMT5F-nurtz [Accessed: 6 May 2016].

Slack, 2016e. Relay [Online]. Available at: https://slack.com/apps/A0H7JRUUS-relay [Accessed: 6 May 2016].

Slack, 2015. Slack Survey Results - July 2015.

Slack, 2016f. Using Giphy with Slack [Online]. Available at: https://get.slack.help/hc/en-us/articles/204714258-Using-Giphy-with-Slack [Accessed: 6 May 2016].

Sliwinski, A. 2016. sentiment [Online]. Available at: https://github.com/thisandagain/sentiment [Accessed: 6 May 2016].

Smith, C. 2015. 29 Interesting Slack Statistics [Online]. Available at: http://expandedramblings.com/index.php/slack-statistics/2/ [Accessed: 6 May 2016].

Terveen, L. 1995. Overview of human-computer collaboration. Knowledge-Based Systems 8(2-3), pp. 67-69.

Veletsianos, G. 2007. Cognitive and Affective Benefits of an Animated Pedagogical Agent: Considering Contextual Relevance and Aesthetics. Journal of Educational Computing Research 36(4), pp. 373-377.

Weiss, B., Wechsung, I., Kühnel, C. and Möller, S. 2015. Evaluating embodied conversational agents in multimodal interfaces. Computational Cognitive Science 1(6).

Wyner, A., Angelov, K., Barzdins, G., Damljanovic, D., Davis, B., Fuchs, N., Hoefler, S., Jones, K., Kaljurand, K., Kuhn, T. and Luts, M., 2009. On controlled natural languages: Properties and prospects. In Controlled Natural Language (pp. 281-289). Springer Berlin Heidelberg.

Yeung, K. 2015. Slack hits 2M daily active users, launches third-party App Directory, $80M developer fund [Online]. Available at: http://venturebeat.com/2015/12/15/slack-hits-2m-daily-active-users-launches-third-party-app-directory-80m-developer-fund/ [Accessed: 6 May 2016].

Young, S., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B. and Yu, K. 2010. The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management. Computer Speech & Language 24(2), pp. 150-174. Available at: http://dl.acm.org/citation.cfm?id=1621240 [Accessed: 6 May 2016].

Zacks, J. and Tversky, B. 1997. Bars and Lines: A Study of Graphic Communicatio. Stanford University.