USING TEXTUAL EMOTION EXTRACTION IN CONTEXT-AWARE COMPUTING

TIM DAUBENSCHÜTZ

Bachelor thesis
Institute of Databases and Information Systems
Faculty of Engineering and Computer Science
Ulm University

VERIFIER
Prof. Dr. Manfred Reichert

SUPERVISOR
Marc Schickler

April 2015

VERIFIER:
Prof. Dr. Manfred Reichert

SUPERVISOR:
Marc Schickler

LOCATION:
Ulm

This work is dedicated to my family and friends who made all of this possible in the first place by supporting me morally and financially. Thanks grandpa for pushing me to study Computer Science.

Finally, thanks Robin Kraft, Robin Zöller and Sebastian Erhardt, my dear fellow students.

ABSTRACT

In 2016, the number of global smartphone users will surpass 2 billion. The common owner uses about 27 apps monthly. On average, users of SwiftKey, an alternative Android software keyboard, type approximately 1800 characters a day. Still, all of the user-generated data of these apps is, for the most part, unused by the owner itself. To change this, we conducted research in Context-Aware Computing, Natural Language Processing and Affective Computing. The goal was to create an environment for recording this non-used contextual data without losing its historical context and to create an algorithm that is able to extract emotions from text. Therefore, we are introducing *Emotext*, a textual emotion extraction algorithm that uses conceptnet5's real-world knowledge for word-interpretation, as well as *Cofra*, a framework for recording contextual data with time-based versioning.

ACKNOWLEDGMENTS

First and foremost I would like to thank Elisa Erroi for supporting and motivating me throughout the whole process of writing this thesis. Simultaneously, I would like to give a big thank you to my advisor Marc Schickler for giving me enough leeway to challenge myself. Finally, thank you Alexander Müller and Robin Kraft for being there when I needed your help.

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION

Marc Andreessen, the inventor of the first widely used web browser, once famously said: "*Software is eating the world*". His key argument is that technology companies will disrupt established companies by leveraging their software's benefits. One infamous example, this process can be seen easily is Uber[1] and the taxi industry.
Judging by some companies' monetary valuations – Uber valued $40 billion in December 2014 [21], Facebook's stock is worth $230 billion as of March 2015 [24] – he most certainly is right.
In late 2014, Facebook's mobile applications had approximately 745 million daily active users[2] [20]. Simultaniously, WhatsApp reportedly handles 30 billion messages every day, while counting 700 million monthly active users [18]. These numbers, combined with the fact that Apple recently sold-out an estimated 2.3 million units of their latest product line (the Apple Watch) in about 6 hours [33] and Android's promising sales numbers (its shipments exceeded 1 billion devices sold in 2014) [32], confirm not only that Andreessen's thesis is right, but that software's subsets *instant-messaging* and *smart mobile devices* do too, "*eat the world*".
A recent study of Nielsen found that the average smartphone owner uses approximately 27 mobile apps a month [5]. Though many people use apps like activity trackers, location-based services and life-journals, all this information is hardly ever combined to achieve more advanced tracking results. The underlying problem is that an appropriate platform for storing contextual data does not exist yet. Equivalently, in 2013 SwiftKey[3] users typed an average of about 1800 characters a day on their mobile keyboards [8]. Still, its users never had the chance to profit from a software solution that evaluated their typing behavior. Apart from sentiment analyzing software modules[4], there is no well-performing, accurate and privacy-conform solution to mine personal metadata from text sufficiently, especially none for emotion extraction in particular.
Hence, in this work we address both problems by researching three important domains, namely context-aware computing, natural lan-

1 Uber is a ride-sharing mobile app that lets its users essentially call a cab by clicking a button.
2 Statistics do not include their aquired ventures WhatsApp and Instagram.
3 SwiftKey is an alternative and widely-spread Android software keyboard.
4 Sentiment Analysis is the study of the polarity of words that are used to express attitudes.

guage processing and affective computing.
In addition, we use the acquired knowledge to specify and design a system for recording contextual information historically and for extracting emotion from arbitrary-sized pieces of text. Our findings in context-aware computing will help increase usability and effectiveness of human-computer interfaces [13]. Our system's ability to record and correlate historical data will enable applications to predict future actions [17]. And our reference-implementation will potentially help computational treatment of affect to improve human-computer interaction [36].
In the following section, we introduce the methodology used to research the contents of this work.

1.2 METHODOLOGY AND AMBITION

This work's goal is to highlight problems and challenges in context-aware computing, natural language processing and affective computing. By defining not only the essentials of each respectively, but also by entering the domain sporadically deeper, we want to expose commonly used methods and techniques in all domains to ease future successors' entry-hurdles. Especially the intersection of affective computing and natural language processing potentially poses interesting use cases that could be leveraged to achieve great improvements in consumer-facing context-aware computing and beyond [15].
To prove the practical value of our research and to illustrate its usefulness as well as our ability to develop software, approximately half of this work is a documentation of the reference-implementation that divides itself into the first three phases of the waterfall software development model (requirements, design and implementation).
The next section, briefly covers the context of this work by chapter.

1.3 ROADMAP

In chapter 2, we discuss pervasive computing by enumerating on its four major requirements and highlighting its current problems. Using an example, we define *context*, which is information that can be used to characterize an entity in space and time. Since its broad definition poses a problem for present-day technology we present context-aware computing and introduce a categorization-model, that helps us derive requirements for it.
Subsequently (in chapter 3), we introduce natural language processing and motivate its existence. In defining the term narrowly, we present its tasks, exemplary implementations and highlight its major challenges. We propose six categories of linguistic knowledge and discuss NLP's difficulties. Further, we suggest five commonly used methods in natural language processing to build a foundation for

our later specification.
Chapter 4 introduces the term affective computing. We define the term *affect*, as well as a model called the topology of affective state. Additionally, we present a classification-model, discuss the field's major challenges and define the term *emotion*. To help build specific knowledge about the problem's domain, we introduce four commonly used techniques in textual emotion extraction as well as methods for smoothing their outputs. Since we use the classical waterfall model in development, we represent each phase (requirements, design and implementation) separately as a chapter of this work.
Firstly, we specify requirements (see chapter 5) for an emotion extracting and context recording system, which we divide into the three components Emotext, Cofra and EtMiddleware.
Secondly, in chapter 6 we discuss and design each component separately and present the reasoning behind the choice of tools.
In chapter 7, we document the functionality of each component respectively. We introduce Emotext's text processing mechanism and explain in detail how its emotion extractor generates outputs for its inputs. Additionally, we present Cofra's RESTful web interface and explain EtMiddleware's caching functionality, its usage of smoothing methods as well as its capability of clustering messages.
Finally, we analyze all three phases by comparing the results to the original requirements and explain in retrospective the reasons for the decisions made in the software development process.
We start off by introducing pervasive computing and context-aware computing. Two fields that may grow very large, considering the latest innovations in wearable computing and smart mobile devices.

2 PERVASIVE COMPUTING AND CONTEXT

With the rise of smart mobile devices such as smartphones and tablets on one hand and cloud computing on the other hand consumers have now the computing power of a small supercomputer in their pocket. Today, any smartphone owner can calculate complex equations by simply querying for example WolframAlpha, a computational knowledge engine. Here, all calculations are done in the cloud leveraging the advancing network infrastructure while regarding the limited computational power of a mobile device. Recently, we could also witness an increasing trend in wearable and smart home devices. Internet-connected thermostats, tethered smart watches and fitness bands are just the beginning. With this change in the consumer electronics market, a solid foundation for future pervasive computing has been established. Yet context-aware computing in applications fails to materialize in our everyday lifes. Although rare, there are some strong and production-ready concepts that prove the benefits of context-aware computing. With Google Now leading the way, Google introduced an intelligent personal assistant that employs Google's Knowledge Graph to build more sophisticated search results by calculating their semantic meaning and real-world connections [2]. Google Now does not only predict the users' average commuting duration to work and notifies them about soon arriving parcels, it also suggests job offers from recently visited websites that might fit their expertise. Particular functions of this service, though, are only possible owed to the vast amounts of user data Google collects from its various other products like web search, maps and mail. This shows the difficulties in creating successful und useful context-aware applications, namely collecting and correlating a critical mass of user data for getting a context-aware service to work, and why it yet fails to materialize. In this chapter we introduce the terms *pervasive and context-aware computing*, but also the term *context*.

2.1 PERVASIVE COMPUTING

Pervasive computing, often referred to as ubiquitous computing, describes the seamless integration of devices into users' lives [13]. When first introduced to the term, most people tend to miss the vast implications of this definition. The most common interpretation of this term is that every (man-made) item contains a computer with networking capabilities. This potentially stark magnitude of linked up devices would enable services to collect big amounts of data that could be

used by application developers for artificial decision-making. Especially with the invention of the internet and the recent rise of powerful but small and cheap mobile devices researchers started to envision and define the term more detailed. One of the earliest but most widely acknowledged definition originated from Mark Weiser in 1991. He suggested that every major technology in the future needs to vanish into the user's everyday life to be truly impactful [38]. Despite its simplicity it still influences present definitions but is not fitting for the extent of this work. More comprehensive requirements can be found in works of Satyanarayanan et al. and Henricksen et al. [34, 19]:

- **Disappearance:** Devices disappear from the user's perception entirely and do not demand the user's attention. A quite practical example would be a fridge that captures the quantity of milk in stock and resupplies without the user's interference.
- **Mobility:** Only in a utopian future ubiquitous computing can be in every man-made device. In order to satisfy the *disappearance* requirement the user should at least perceive it that way. This however implies that *uneven conditions*, conditions where a computational device might not be available, need to be masked to convince the user of the ubiquity of the computational environment. This could for example mean that a user's location data is interpolated given the fact that her GPS connection is volatile or inaccurate.
- **Proactivity:** By tracking the user's intentions and actions, a system can learn specific human behavior and make future proposals to the user. At first glance this seems like a decent application for context-aware consumer-facing computing but is, as past projects prove, harder than it might seems. Well designed and timed proposals really have the potential to improve human-computer interaction, while misleading instructions or a poorly recognized context can lead to anger and/or to dangerous situations for the user. Simply, imagine a route planner that notifies the car-driving user too late to take an exit ramp safely.
- **Adaption:** Is the necessity of change when there is a significant mismatch between supply and demand. Here the term adaption describes the computational ability to adjust to more relevant data that is usually derived from the user's context. Take again a route planner that distributes cars to different routes in a city to avoid congestion.

While requirements, visions and specifications of pervasive computing are widely available and discussed frequently, the concept still faces tough problems that need to be overcome. One being the high fragmentation in the mobile devices and operating systems market today that is accompanied by a big variety of proprietary protocols and

tools. For pervasive computing to flourish, protocols must be open, extensible, accessible to everyone and free. Ease of usage and distinctive appearance enable wide-spread participation and development. Unfortunately, the status quo in 2015 is quite contrarian to this vision: A recently released smart watch may only be usable if connected to a smart phone of one certain type (Apple's iOS for example), while most cars integrate support only for another mobile operating system (Google's Android for instance). With the increasing number of participants in the numerous smart-markets, the situation will get worse, since proprietary software is commonly used as a unique selling proposition. The vision of pervasive computing in stark contrast describes the interconnectivity and communication of all electronic devices. Nevertheless, seamless integration of every electronic device is a challenge that can only be achieved through time, standards and a unifying middleware [34].

Another problem the concept faces is that pervasive applications and devices need to become more flexible and *target-oriented*[1]. With subsiding cloud computing prices, service providers are now able to "waste" computing power to improve the user's experience [34]. In turn, mobile computation can be outsourced to the cloud allowing for the creation of even smaller, lighter and more durable mobile devices [34]. This creates an environment where smart mobile devices can be produced cheaply, with little to no computing power but networking capabilities that can then be used for a greater purpose in the cloud. Finally, every pervasive mobile device is just an array of sensors with networking capabilities sending a constant stream of data to the cloud. In a nutshell, pervasive computing in today's world can be broken down into cheap, mobile devices that are connected to the cloud making them hereby interconnected.

Once those problems are overcome and the technology is ready, pervasive computing offers a broad array of possibilities that can improve the user's everyday life. While today's systems are poor at capturing the user's intent [34], future applications will help humanity to overcome the rapidly increasing phenomenon of *information overload* by leveraging the cloud's computing ability to compress complex data into chunk-sized, comprehendible information-bites [38]. Still, this can only be achieved with subsiding cloud computing prices, open protocols and cheaper, smaller, lighter and more durable mobile devices [34].

In the next section, we are introducing the term context, as well as context-aware computing and highlight both their relations to pervasive computing.

1 Target-oriented describing the contraction between the user's input and output.

2.2 CONTEXT

When using the word *context* outside of the computer science domain, the term is often used for describing *"implicit situational information"* in social interactions with other human beings [11]. We use both the word but also the concept to reference other situations in social interactions with other human beings. Given its vague but broad terminology in the computer science domain, the term unfortunately has a long history of attempted and failed definition by researchers. Many of them first referred to it as location or identities of nearby people and objects. Other definitions were done by enumeration [11, 17], reciting every circumstance that applies. Since there is still no definition that can live up to the already introduced concepts and specifications, we introduce the term by giving a practical example:
Two persons are waiting for a bus at the station. Multiple lines stop at the station and also multiple lines lead to both persons' final destinations. A bus arrives and so one person asks the other: "This one or the next?". While a computer algorithm might have understood their speech and would have been able to track their location, it would not have been able to determine what the person asking the question was trying to say. The sequence "This one or the next?" would not enable the algorithm to calculate helpful information without knowing that both are talking about their decision which bus to take. So to participate in the decision-making process, the algorithm first would need to know the contextual information that both persons are waiting together for a bus. Still, with this information at hand, making the decision might be influenced by further factors including:

- Wether both persons are in a hurry and how long the first bus takes compared to the second one?
- How many people are on the first bus and if it would therefore be more pleasant taking the second one?
- What the weather is like? It might be freezing cold so that they would probably prefer taking the first bus.

Special about these factors is that they are - in contrast to for example the age of both persons, their gender or their nationality - contextually important because by using them, a relevant solution to both persons' best intentions can be computed. By using a temperature sensor and a third-party service for obtaining the number of passengers on the bus, what the weather is like and maybe even what clothes (warm or cold) both persons are wearing the algorithm could compute a sufficient conclusion that might support or even replace the decision-making of both actors.
Based on this example we see that understanding the speech and language is equally significant as knowing information that can be used

to describe the context of a person. Generally speaking only information that can be used to characterize an entity in space and time is context [17]. Additionally, an entity can be a person, place or any object that can be considered relevant to the human-computer interaction [17]. Another good rule of thumb to decide wether or not a piece of information is context, is to ask ourselves if it can be used to define the situation of an participant in an interaction [17]. This shows that context can grow very rich and include multiple layers of physical, physiological and emotional state but also a person's personal history [34]. Because most of these states need to be handled separately, researchers [13] propose that context needs to be segregated into multiple dimensions. More specifically, they suggest that context can be differentiated between:

- **External context** - Can be physically measured, examples are:
  - Location;
  - time;
  - season; and
  - movement;
  - (room) temperature;
  - brightness.
- **Internal context** - Is specified through the user, examples are the user's:
  - Emotional state;
  - focus of attention;
  - orientation; and
  - work context;
  - identity;
  - goals.

Quite different to these proposals, Dey et al. [17] suggested about six years earlier that context-aware applications are using the five *W questions*, known from journalism. Specifically, the *who's*, *where's*, *when's* and *what's* of entities to determine *why* a particular situation is happening. This proposal implies that context can be divided into five different categories, but Dey et al. go even further by classifying context into *primary* and *secondary* context, the so called *levels of context*. Identity (who), location (where), time (when) and activity (what) are primary context level types while secondary context level types are information that can be indexed by primary context. Take the extraction of a number from a phone directory as an example: In order to retain this task's primary context the entity's identity is required, in this case his forename and surname. Dey's definition might be sufficient to narrow down a situation's horizon, but is generally inadequate when used in a computational situation. Here, data and its statistically correct correlation process is far more important than to link specific data to W questions. At this stage in history, big data is practically not interpreted individually but only by its result's statistical relevancy. As we will learn in the next section, there are several steps to be reached, before being able do that.

2.3 CONTEXT-AWARE COMPUTING

Context-aware computing is a superordinate of pervasive computing (see section 2.1) and can be described as the process of collecting *contextual information*, information that can be used to describe the situation through automated means and use this information to enrich the software around it [13]. Although Baldauf et al. [13] suggest that systems need to be able to adapt their operations *without* the user's intervention, hence suggesting that context-aware systems need to act *proactively* towards the user's actual intent, we hold the opinion that by satisfying the mentioned definition of context-aware computing, more generic and usable requirements can be elaborated. For the sake of completeness though, we hereby mention Dey et al.'s proposal [17] to specify context-aware applications by their *required* features. These can be broken up into three categories:

- *presentation* of information and services to the user;
- automatic *execution* of a service; and
- *mapping* of context to information.

Undoubtedly, an application could easily categorized using this model, but, as we will debate later on, it's assumption that *execution of service* needs to be automatic, as well as that information only needs to be *mapped* to context are quite unclear. Also, as we will learn later on, context *presentation* can have multiple manifestations. Take for instance, Dey et al.'s example [17] of entering a room with multiple accessible printers. In a completely context-aware future your mobile phone would access the printers' networks automatically and choose - in case the user would actually need one - a printer accordingly to the file the user wants to print (text, image, ...). Using Dey et al.'s categorization for specifying the application we can clearly see that it matches. However, a mobile phone that would access the printer's networks automatically, but show a list of all available printers in order to let the users choose manually according to the file they want to print would still match this definition. Especially in this case, the question which implementation is more mature in term of contextual usage arises. To solve this problem we propose a model that divides context-aware computing into four strategies by particularly looking at how they *acquire* their information and how they *process* it. By definition, an application is composed of an automatic or manual information acquisition and information processing. Despite the simplicity of this definition, we need to further discuss the terms *manual* and *automatic* to avoid ambiguity. In this work the term *manual* is used as a synonym to do something by hand, not using machinery or electronics to fulfill a function whereas *automatic* means the direct opposite, namely using machinery or electronics to fulfill a

function. Irrespectively of its automation degree (automatic or manual), a process still needs to be initiated somehow, which is why the initialization of a process must not be used to categorize the process's degree. Further, *manual information acquisition* means to enter values into a specific computer interface by hand while *automatic information acquisition* means that those values are collected through varied sensors. An information handling process is classified as *manual information processing* if it requires additional user input to complete, while a *automatic information processing* one does not. By constructing a 2x2 matrix, as can be seen in figure 1, we can develop four major *context-aware computing strategies*:

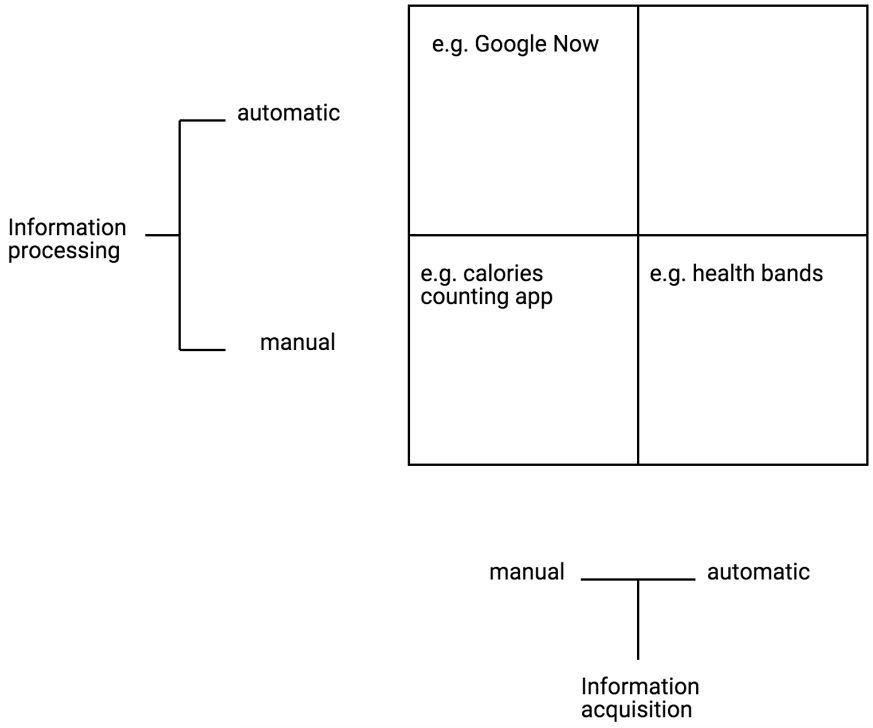

Figure 1: Schematic illustration of the four major context-aware computing strategies.

- **Manual processing - Manual acquisition:** Can be found in the bottom left corner of figure 1. Applications of this strategy can often be found in our everyday lives. Take for an example a calories counting app that needs manual input after every meal. Rarely information is processed to a greater degree than summing up the calories of all meals and showing the user his remaining calories for the day.
- **Manual processing - Automatic acquisition:** Can be found in the bottom right corner of figure 1. Applications of this kind can also be found in our everyday lives, especially with the latest

hype around health tracking wearable devices. These devices automatically track the user's activity by using a variety of sensors while analysis of this data is rarely done extravagantly.

- **Automatic processing - Manual acquisition:** Can be found in the top left corner of figure 1. Despite the fact that these types of applications can eventually be found in your everyday lives, they are far more rare than ones from the more primitive strategies. One famous example of such an application is Google Now, an intelligent personal assistant that is able to learn, recommend and adapt on its own and hardly needs his user's manual input.

- **Automatic processing - Automatic acquisition:** Can be found in the top right corner of figure 1. As of today, these applications are not yet to be found in our everyday lives since they would require the application to be completely decoupled from a user's manual inputs.

In our model the maturity degree of applications can easily be determined according to the automation utilization of an application. Of course, automatic information processing and acquisition is far more advanced than its manual pendant. While *manual information processing and acquisition* can theoretically already be done using pen and paper, *automatic information processing and acquisition* needs complex algorithms that may not yet exist. For both in-between strategies can be said, that information processing and acquisition are subject to the commutative property, which is why the maturity degree of automatic information acquiring but manual information processing applications and manual information acquiring but automatic information processing applications are equal.
Having defined this model, we can now easily derive requirements for context-aware computing, namely *adaption* and *disappearance*.
Clearly, *adaption* is a requirement as it can be used synonymously to automation. The reason *disappearance* is a requirement for context-aware computing is due to the fact that automation, and in this special case, information processing and acquisition can only perceived as automatic processes if they disappear from the user's perception [34, 13].
Having gathered all this information about pervasive computing and context, we can now devote our attention to two equally important fields that need to be studied to understand this works purpose. In the two following chapters, we firstly introduce *natural language processing*, the extraction of information from natural language, and further embellish it by introducing *affective computing*, the computational discipline of emotion extraction and expression.

3

NATURAL LANGUAGE PROCESSING

As the quantity of user-generated content on the internet has risen exponentially over the last ten years, users find themselves increasingly overwhelmed by the amount of information that is presented to them. This phenomenon helped information extraction to its most recent Renaissance becoming an expanding field in research [7].
Other parties' opinions have always played a crucial role in decision-making. In fact *social decision-making* has now become the de-facto standard. According to a study with more than 2000 American adults, 81% stated that they have at least once used the internet for researching a product online. 32% of those surveyed also stated that they have provided rating on a product, service or person at least once [30]. This can be seen as an indicator that commercial decision-making has progressively gone social. Nowadays, many big online retailers have a social rating system for their products (see amazon.com or walmart.com). There are rating platforms for nearly every aspect of our lives, such as movies, restaurants, travel guides, tech guides, or even universities. Still, with more and more information being generated on the world wide web every day, it becomes increasingly difficult for users to briefly extract relevant information. This is the reason why natural language processing has had such a surge of interest with the rise of the internet [30].
For instance take the process of buying a digital camera. Instead of reading through a five page long review for it, the user would rather like to know wether its lens holds the manufacturer's promise or not. In this use case, *natural language processing* can really make a difference. One example this has been done is Microsoft's Bing Shopping platform [1]. An algorithm extracts affect-containing[1] words as well as feature-representative[2] words from documents and classifies the gadgets' features depending on the polarity of those affective words. A frequent occurrence of "*bad*" and "*lens*" together in different reviews then allows the algorithm to draw the conclusion that the quality of the camera's lens is perceived as insufficient by the majority of the customers. All gathered information can then be presented to the users beforehand, to enable them to extract information more easily and finally accelerate their decision-making process. Whilst the given example appears to be quite useful, its core functionality is derived rather from sentiment analysis than from natural language process-

1 Words that contain an emotional meaning.
2 Words that refer to one of the camera's features.

ing. To be able to make this distinction however, we need to define the term natural language processing further.

3.1 DEFINING THE TERM NATURAL LANGUAGE PROCESSING

We have been using the achievements of natural language processing unconsciously for years. Ever since in 1976 a Canadian computer program has been used to generate unedited weather reports to the public in French and in English [22]. Nowadays we conveniently use voice recognition, auto completion and text-to-speech, as they are built into nearly every smart mobile device or desktop computer. Natural language processing, a subordinate of *speech and language processing*, features a broad array of tasks, such as:

- natural language understanding and generation;
- speech recognition;
- machine translation;
- discourse analysis; and
- sentiment analysis.

Therefore it is often described as an intersection of computer science, artificial intelligence and linguistics, with the goal of letting computers execute useful tasks involving human language [22]. Today, implementations of these tasks can be seen in various popular applications:

- Apple's Siri[3] as an example of a **conversational agent**. An application that communicates with humans in natural language;
- Google Translate as an example of **machine translation**. An application that is able to translate complete documents on its own [3]; and
- WolframAlpha as a web-based **question-answering** engine. An application that can be queried using natural language [10].

What makes the implementation of these tasks challenging, is that in order for them to complete they need *language-specific knowledge*. Other than in regular data processing, where it is generally sufficient to define one set of rules per application, rules for processing a language can not be derived from another language and require deep structural knowledge. To express this difficulty, Jurafsky et al. [22, 9] exemplarily suggest the unix tool `wc`[4].
While it is fairly easy to use it for counting bytes or lines in a file,

3 An intelligent personal assistant similar to Google Now.
4 wc is a tool for counting byes, words and lines in text files.

counting words is definitively a non-trivial task. This is, as we addressed already, due to the fact that we need language-specific knowledge to accomplish the task. One could argue that whitespace is sufficient, but given the English language's characteristics of abbreviating possessive personal pronouns ("Wendy's", "Moe's Burger Bar") and modal verbs ("can't", "don't", "I'd", "I'll") this simple rule will not yield sufficient results.
The proper name "San Fransisco" allows us to dive even deeper into the problem. Despite the fact that it consists syntactically of two words, semantically it refers to one concept, namely the famous city. With languages differing in rules, it becomes apparent that they may also differ in processing-effort. Take the Turkish language for an instance: There, every verb can be causativized multiple times (*you cause A to cause B to... do C*), creating an even larger quantity of words to memorize. This makes the language harder to parse in comparison to English [22]. Luckily, language-specific knowledge can help us overcome these hurdles. Jurafsky et al. split it into six categories and called it *linguistic knowledge* [22]:

- **Morphology:** The study of meaningful components in words. One of the most conventional use cases of this is the recognition of plural forms in nouns. While in most cases this seems like an easy task (*door* and *doors*), it can get challenging considering every exception existing in a language (*fox* and *foxes*).

- **Syntax:** The structural relationship between words. Its importance is enforced when linked information is extracted from a sentence or document. Take for instance the following sentence: "*I don't like Brad Pitt, but I do like some of his latest flicks*". Here "*his latest flicks*" is used as a reference for *Brad Pitt's movies*, because they were already mentioned in the main clause.

- **Semantics:** Can be divided into two sub categories, one being *lexical semantics* which is the study of the meaning of all the words (for example *"flicks"* being a synonym for *movies*), while *compositional semantics* explain the meaning behind word constructs (*"latest [flicks]"*, where *latest* has a contextual component, in this case *temporal*, to it).

- **Discourse:** The study and analysis of linguistic units that composed amount to humanoid dialogues. If someone replied to the previous statement about *Brad Pitt's movies* with: *"No! Those are garbage"*, then *"Those"* is used synonymously for Brad Pitt's movies but this time beyond the context of a single sentence and person.

- **Pragmatics:** The study of how language can be used to achieve goals. Using natural language, actions can be demanded ("*Open*

the door!"), statements can be made ("*The door is open.*") and information can be requested ("*Is the door open?*").

- **Phonetics and Phonology:** The study of linguistic sounds, specifically how words are pronounced and how they are realized acoustically.

Despite the diversity in these tasks' nature they all commonly aim to resolve one main problem in language processing: reducing *ambiguity*. Ambiguity is the uncertainty about an attribute of any concept, idea, statement and so on and means in our case, that for one input there are varied alternative structures that can be interpreted from it. Jurafsky et al. [22] express this problem in a fitting example. The english sentence "*I made her duck*" can, depending on its context, have several meanings. They can be:

1. I cooked duck (a waterfowl) for her (meaning a meal).
2. I caused her to duck (the physical exercise).
3. I waved my magic wand and turned her into a duck (the waterfowl).

This difficulty arises due to the fact that the word *duck* can either be a verb (to lower one's body) or a noun (the swimming and flying animal). Therefore, we call *"duck"* a *morphologically* and *syntactically* ambiguous word. Further, the word *"make"* is semantically ambiguous, as it can mean *to make*, *to create*, *to cook* or *to enchant* somebody or something [22]. There are several methods and techniques to approach this ambiguity in natural language processing which we are going to introduce in the following section.

3.2 METHODS AND TECHNIQUES USED IN NATURAL LANGUAGE PROCESSING

With natural language processing being an intersection of computer science, artificial intelligence and linguistics, it features a big quantity of techniques to achieve functionality, most of which we cannot cover in this work. Instead we will focus on methods and techniques that are applicable for emotion extraction. At first, we need to refine our knowledge about words and sentences.
To begin with, a *lemma* is a word that bundles all its enhancements to a single word. Take for instance all conjugations of a word. *Sing*, *sang*, *sung* are called *wordforms* and are bundled by their lemma *sing* [23]. Then there are *types* and *tokens*. Types are elements of a dictionary, while tokens are an entity in a sentence. In the clause "*He heard his phone ring*" there are five different types, while there are only four different tokens. "*He*" and "*his*", though different words, have the same

meaning because they call out the same entity, namely the man who heard his phone ring, which is why they are considered to be a single token but two different types [14]. Luckily, this also gives a definition for our previous problem with the proper name "San Fransisco". In short, it consists of two types, but only one token.
When processing a document, the first task is to analyze it syntactically and, subsequently, semantically. This is why we begin with introducing the method of *word segmentation*.

3.2.1 *Word Segmentation*

Word segmentation, often referred to as *tokenization* is the task of separating out words from a text corpus. As we already learned, segmenting words on the appearance of a *space* or *punctuation* character does not yield satisfying results. We mentioned the proper name "San Francisco", still there is more like for example abbreviations (*O.K.*). The issue becomes even more apparent in other languages such as French where *cliticization*, the blend of an prefix and a stem[5] ("*l'ordinateur*", english: the computer), is commonly used [14]. The German language also features obstacles. Here, nouns can be chained together and allow the speaker to compose very long words ("*Donaudampfschiffahrtsgesellschaftskapitänskajütenschlüssel*", english: Danube Steamboat Shipping Company Captain's cabin key). Word segmentation is usually achieved using regular languages in combination with language-specific rules.

3.2.2 *Word Normalization*

Once we've successfully segmented our words, we proceed with normalizing them. Basically, normalization features the *splitting of words*, the *compression of singular and plural forms* and the *matching of words* that *semantically refer to the same entity*. This process is technically called *word normalization* or *stemming* and is done using morphology [23]. As we mentioned in section 3.1, morphology is the study of meaningful components in words. It explores the way in which words are built up from meaningful, smaller chunks. Those are called *morphemes*. Generally, word segments can be divided into two types of morphemes called *stems* and *affixes*. While some words can contain more than one affix, every word can have only one stem. Take for example the word "*foxes*". It consists of a stem ("*fox-* ") and an affix ("*-es*"), while the word "*rewrites*" has the stem "*-write-*" and two affixes ("*re-*" and "*-s*") [23]. We can also see that affixes can have different features. In particular, there are four ways affixes can appear in words [23]:

5 We will be covering *stems* and *prefixes* (*affixes*) in the next section.

1. **Prefix:** Precede the stem (**un**fair)
2. **Suffixes:** Follow the stem (fox**es**)
3. **Circumfixes:** Encompass the stem, like the german word **ge**sag**t** (en. said). Note that *rewrites* is not classified as a circumfixed affix as it consists of two morphemes of different origin. The prefix "*re-*" indicates a repetition of a task and the suffix "*-s*" expresses a conjugation in the third person.
4. **Infixes:** Split the stem (legen - **wait for it** - dary)

An algorithm for word normalization features knowledge of these language-specific grammatical rules. One of the most widely used algorithms for stemming is the one created in 1980 by Martin Porter [4]. Here a finite state machine uses a formal language to convert for example conjugated affixes into their basic forms again [23]. It contains rules like these:

$$\text{IES} \rightarrow \text{I} \tag{1}$$

$$\text{I} \rightarrow \text{Y} \tag{2}$$

The word "*libraries*" would first be matched by the first rule (1) to "*librari*" and subsequently by the second one (2) to "*library*". As we will find out later, stemming does not yield perfect results throughout and is, in use cases like ours, neither necessary nor improves emotion extraction sufficiently.

3.2.3 *Negation Handling*

Detection and handling of negation in natural language processing is another important task, given the fact that negation words have a profound influence on the result of textual semantic extraction of information (in sentiment analysis for example) [30]. As with every technique introduced so far, negation detection requires sufficient language-specific knowledge. It needs to be aware of how the language can be used syntactically and semantically to negate meaning in documents. Considering the english language as an example, detection of negating words ("*not*", "*don't*", "*never*") is easily achieved using pattern matching. The more challenging task is to detect subtle negation like irony or sarcasm [30].
An easy, but commonly used method to manage negation is substitution. An algorithm iterates over every sentence and negates certain words if the context of the sentence requires it. More precisely, it

prepends the word "NOT-" to every word between the causative negation and the next punctuation. So for example in the sentence: "*[...] didn't like this movie, but [...]*", the negation-word "*didn't*" is detected and every word between it and the next comma are adjusted, converting the sentence to: "*[...] didn't NOT-like NOT-this NOT-movie, but [...]*". This way, *"NOT-like"* will be analyzed instead of "like", hence the sentence's meaning remains semantically correct. The method was first proposed by Das et al. [16] and is still commonly used throughout various text processing algorithms.

3.2.4 *Stop Word Removal*

Words that carry a low information content and are only present in a sentence for syntactical reasons are commonly called *stop words* [26]. Natural language processing, as well as other closely related disciplines focus therefore mostly on accurate ways to remove them from a document efficiently. Makrehchi et al. [26] suggest that there are two categories of stop words:

- **General stop words:** Words that fit the above definition of being used for syntactical reasons. Some examples are:
  - *Articles:* "the", "a", "their", ...;
  - *Conjunctions:* "and", "or", ...; and
  - *Negations:* "not", "never", ...
- **Domain-specific stop words:** Words that have, according to Makrehchi et al. [26], no "*discriminant value*" within a specific context. The inflationary usage of the word "*learning*" in the domain of education for example makes it a stop word. In computer science, however, "*learning*" is more of a keyword, he suggests.

Problematic is the fact that, stop words occur disproportionately. As we will find out in further chapters though, textual extraction of emotions can be quite computation-intense, which is why stop word removal should be considered an important requirement later on.

3.2.5 *Text Classification*

The task of text classification is essential to natural language processing, as it features a broad array of applicable use cases. Some writers even call it one of the "most fundamental" technologies in most natural language processing applications [30]. In our everyday life this can be witnessed most easily in applications that feature spam detection, language identification and sentiment analysis [23]. In particular, classification is the assignment of a class for a certain document. The assigned class can thereby only be chosen from a fixed *set of clasess*.

The easiest way to do text classification is to use a *bag-of-words model* [14]. In disregard of grammar and even word order, a document is parsed by simply counting all occurrences of words in a key-value object. Consider these sentences in listing 1:

Listing 1: Two exemplarily sentences describing Bob's and Alice's feelings towards theatres and movies.

```
Alice and Bob like to go to the movies.
Alice also likes to go to the theatre too.
```

The resulting key-value object can be seen in listing 2:

Listing 2: Both sentences from listing 1 as a list of occurences.

```
{
  "Alice": 2,
  "and": 1,
  "Bob": 1,
  "like": 2, // stemming: "likes" ==> "like"
  "to": 3,
  "go": 2,
  "the": 2,
  "movie": 1, // stemming: "movies" ==> "movie"
  "also": 1,
  "theatre": 1,
  "too": 1
}
```

In essence, the frequency of words within a document is measured. Generally, not all occurrences of every word are counted, but only the ones that match specific criteria. Extraction of attitude can be done by determining the frequency of certain biased adjectives and nouns. The resulting key-value object is then compared to a set of classes, to calculate the class resembling the closest. We learned in the introduction of this chapter about Microsoft's Bing Shopping platform. It uses exactly this procedure to determine a product's features and ratings. Clearly, the bag-of-words model is one of the easiest approaches, whereas more sophisticated approaches, like *Naive Bayes*, use statistical models [14]. But since the introduction and explanation of these methods would push the boundaries of this work much to far, we will not cover them any more detailed.

4 AFFECTIVE COMPUTING

So far, we discussed *context-aware computing* and *natural language processing*. As we learned in chapter 2, every information that helps characterize a person in a specific situation is essentially *contextual* information. We also learned in chapter 2, that context-aware computing combines manual or automatic information acquisition and processing. Eventually, we will learn in this chapter, that affective computing is actually an implementation of context-aware computing. As its more generic superordinate, it requires the acquisition of information firstly. In our case this is done using techniques from natural language processing to extract emotions from text. Secondly, it also features a reactionary component: Emotion expression an equivalent to context-aware's information processing.
This chapter, however, strongly focuses on defining the term *affective computing* properly, gives insights into what emotions actually are and introduces multiple methods that can be applied for emotion extraction. It will not cover emotion expression, as this would break the mold of this work.

4.1 DEFINING THE TERM AFFECTIVE COMPUTING

Affective computing is the study of perception and expression of emotions and mood through computers [31]. Like context-aware computing (see chapter 2.3), affective computing can also be divided into four categories, according to Picard et al. [31]:

1. A computer is not able to perceive or express emotions;
2. A computer can express emotions, but not perceive such;
3. A computer cannot express, but perceive emotions;
4. A computer can both express and perceive emotions.

Picard et al., amongst others [31, 30], argue that affective computing, if defined and implemented correctly, may one day enhance human-computer interaction and also the ability for computers to make decisions. Especially in context-sensitive[1] scenarios, affective applications may recognize the user's emotions appropriately for the application to automatically adapt to this change of temper. If however, computers obtain the ability to act affectively, then Picard et al. [31] argue that their emotional state should be observable. They enhance this

1 Scenarios where context-aware computing can yield benefits for users.

thought by splitting up affective computing's field into two major tasks:

- **Emotion recognition**: The study of extracting human emotion through multimodal[2], channels; and
- **Emotion expression**: The study of expressing emotion computationally to one or multiple human beings.

Imagine an application that reads a patient's stress and depression levels and automatically adjusts the brightness of the lights in its room accordingly. While this certainly is not the most stunning use case and does in fact not express emotion directly, it provides at least affective computing's right to exist in consumer electronics and medical computing. Yet it also shows that affective computing's integration into our every day lives, is right at its beginnings. Not only is this the case as emotions are poorly understood, but also because *emotion recognition* must first deliver sufficient results to be used further in *emotion expression* [12, 15]. As of today, emotion recognition, also referred to as *human emotion recognition*, is executed insufficiently [15] and considered to be a hard problem [sic]. An influencing factor for this is that the task is multimodal. Even more problematic is the fact, that mood classification in text is done equally insufficiently by humans comparing them to computer algorithms [28]. Mishne et al. [28] only comment speculatively on the reasons for this and name problems like: inadequate task briefings and document size, but most importantly the high subjectivity of the task itself. Some researchers [31] shift the blame for this on emotion theory. They claim that emotion theory is still far from being complete, so how can affective computing be. The problems are apparent. For affective computing to prevail, large and qualified sets of data are required to enable machine learning algorithms to study sufficiently [15]. In addition, it doesn't stop there. As we already mentioned in the previous section 3.2.5, classification's accuracy depends on the issue's domain. Everyone who has traveled exceeding the borders of his continent knows that emotion and mood are culture-specific. This obviously adds not only to the complexity, but also to the duration of extraction and expression and therefore increases the cost of creating sufficient labeled data sets for affective machine learning operations [15]. Troubling to emotion extraction in text are findings of researchers that suggest that only 4% of words in documents have emotional value [15]. This is further confirmed by some works that claim to return insufficient results from extraction of emotions through text [15, 28]. Still, affective computing is an important future topic in computer science.
But what exactly do the terms *affect* or *affective* actually describe?

2 Communication is carried out over multiple channels of perception (even synchronously in some cases). For instance: Text, audio, video, ...

Scherer et al. [35] answer this question by proposing the *topology of affective states*, a model that classifies affect, the psychological state of a living being, into five distinguishable categories:

- **Emotion:** Is the evaluation of an event (e.g. *fear, joy, angry, ...*);
- **Mood:** Is a ongoing, arbitrary subjective feeling (e.g. *gloomy, ...*);
- **Interpersonal stances:** Are emotional stances towards another person in a specific interaction (e.g. *friendly, supportive, warm, cold, ...*);
- **Attitudes:** Are enduring dispositions towards objects and persons (e.g. *loving, liking, hating, desiring, ...*); and
- **Personality traits:** Are stable, sometimes even inherited, behavioral features (e.g. *reckless, hostile, anxious, ...*).

Especially in the context of text extraction this distinction is important because the extraction of *an enduring disposition towards objects and persons* (attitude) from text or speech is called *sentiment analysis* and lives in its very own field. Admittedly, most methods and techniques for extraction intersect with those used in emotion extraction [37, 12]. Especially, when using text as a major resource, both sentiment analysis and emotion recognition rely heavily on natural language processing. Still, the goal of affective computing is not only to recognize emotion, but to express it as well. As we do not want to miss out on methods that could possibly help us reach our goal, we will exemplarily introduce a concept taken from sentiment analysis that can be transferred to emotion recognition. It is initiated by Pang et al. [30]. He suggests that attitudes can be segmented into levels. His basic thesis, however, is that every text document, independent of its size contains a quantity of sentiments. This group of dispositions towards objects and persons are called *local sentiments* and form in total a *global sentiment* of the document. With this definition at hand, it becomes clear that therefore attitude can vary depending on position, document size and their entities they relate to. Take for instance a movie review. Although the review might overall contain clearly negative sentiment towards the film, individual performance of actors for example can still be rated positively. Because emotions are much more short-lived than attitudes, evaluations of events tend to overlay each other. Therefore, the tracking of position and time frames of emotional outbursts can be beneficial in the task of emotion recognition and determination. Also, recording emotion magnitude can be favorable, as we will see later on. One case where an ordered and weighted set of emotion records can be particularly helpful, is when trying to analyze so called *meta-emotions*. We call them meta-emotions because they are composed of multiple, more basic emotions and the context (in this case the meta-information) they are embedded in [31].

However, before we can elaborate further on emotion processing from text, we first need to introduce the term *emotion* properly.

4.2 DEFINING THE TERM EMOTION

Emotions weaken our ability to make rational decisions. It is not a coincidence that this saying is commonly believed, especially by people that have to make tough decisions everyday. That is, as emotions are a guiding force in our perception and attention [31]. This is given the fact that the cortex[3] and the limbic system[4] are connected through a reciprocal relationship, whereas the limbic system can be viewed as a filter or preprocessor for incoming perceptual information. The connection between both systems is so strong, researchers argue that an existence without experiencing emotions is not possible, because emotions are literally pervasive [31]. In addition, they claim that inside the brain, the process of thinking and feeling is the same. They go even further arguing, that the quantity of emotions involved in making decisions can either damage or evolve the process, but with certainty alter it. This implies that a thought or decision can never be made without disregard of feelings, ergo a rational decision can never be truly rational.
Even more striking is the fact that emotions also influence our outer appearance, namely physical state. This can easily be observed in real life. A person's physical appearance and behavior tends to differ greatly when they received compliments a moment ago compared to when they have just been insulted. Conversely, tension can of course also come from the inside. Sweaty hands, crying and nervousness in the voice, are just a few examples of the fact that there is a tight coupling between physical state and affective state and that every emotion has a unique somatic response pattern [31]. Of course, this knowledge is interesting as it would allow the unambiguous and accurate extraction of emotions through, say physical sensors [31]. Unfortunately, Picard et al. [31] claim that not every affective state can be observed through measurable functions. Instead they argue emotions can on the one hand be *consciously experienced*. Those are called *emotional experiences* and can not be observed from another person for example. And that on the other hand emotions can also be experienced in revelation, meaning that they can be observed from another person for example. These kind of emotions are called *emotional expressions* [31].
When it comes to classification of emotions, researchers' theories distribute quite differently. Basically it can be differentiated between 2 to 20 different emotions, where the most common ones are anger, fear, sadness, happiness and surprise [12]. We mentioned earlier, that from

3 The cortex is the most outer layer of our brain.
4 The limbic system is the seat of memory, attention and emotion.

these so called *basic emotions* more advanced versions of emotions can be composed. These *meta-emotions* are not only created in the intersection of multiple emotions but also in the variation of magnitude [25]. Examples of both can be seen in:

- **Frustration:** a low magnitude anger; or
- **Horror:** a high magnitude fear; or
- **Relief:** essentially fear followed by happiness.

Implementing a text processing emotion algorithm, this knowledge can be used beneficially, as it enables us to extract not only basic emotions but also meta-emotions. The most interesting correlation of data however does not result in combining emotional information reflexively, but in combination with contextual information.

4.3 INTERACTION OF EMOTION AND CONTEXT

We learned that emotions are a guiding force in our perception and attention and that this results in blending the processes of thinking and feeling. In addition, we learned (in chapter 2) that a person and its state can - at least theoretically - completely described by recording all its contextual information. Figuratively speaking, a person's active or passive and physical or non-physical actions always influence their chronological, contextual state. While states seem to change in form and content, their individual history remains untouched and can only be adjusted by signing over, not by changing specific information in history. Take a burn victim for example. Under the assumption that it will never forget the situation that led to the circumstances as well as the resulting consequences, this contextual and historical information can not be changed. Instinct and strongly negative connotations to fire as well as eventual scars remain and can only be overcome making more positive experiences with fire or even undergoing psychological or hypnotical therapy. The conclusion we draw from this is that certain states of context result in certain emotions and that in return certain emotions cause context-state changing actions. Therefore we propose, that under the assumption that emotions can accurately extracted and a sufficient amount of contextual data is recorded for a specific person, computational prediction of a person's actions but also its emotions is possible.
Later in this work we will introduce two tools for exactly this use case. One for recording person-based, contextual information, and another one for extracting emotion from text. For the moment however, we first need to introduce methods and techniques for extracting emotion textually.

4.4 EMOTION EXTRACTION FROM TEXT

Social interaction is a key part of human life. As we mentioned earlier, decision-making requires a critical mass of emotion and thereby social value for it to be successful. Hence, Liu et al. [25] suggest that successful social interaction is actually successful affective interaction. When studying human-to-human and human-to-computer interactions, researchers [29] found out that computer users can interact most naturally when being social and affectively meaningful. This implies that increasing the emotional value of an human-computer interface, increases the value of the interface itself [25]. Additionally, gathering useful information through these interfaces may improve cognitive and emotion theory and improve our understanding of human-health and well-being [31]. But, in pursuit of those goals we need to start at the very beginnings, namely emotion recognition. Digitalization of life altered our social and professional conditions immensely. We chat, write emails, take notes on our phone, tablets and laptops and even program collaboratively through the internet. To express our thoughts, ideas, plans but most importantly emotions however, we still rely on one major concept: text. While technology for extracting emotions through physical means might already be possible, practically doing so is another story. Text input in contrast, is ubiquitous in our lives. Why waste energy and resources on new sources, when perhaps the most intimate one is already available. This is the reason we devote this work to the extraction of emotion from text.

4.4.1 *Methods for Extracting Emotion from Text*

To begin with, we introduce four commonly used methods to solve the problem. Whilst every method involves a basic preprocessing with some of the previously introduced natural language processing techniques, all take very different approaches at their core. Yet, there is an considerable number of hand-crafted models and approaches, mostly taken by researchers, for this task that we will most likely not introduce.

4.4.1.1 *Keyword spotting*

Keyword spotting is the easiest and most approachable method. Similar to the bag-of-words-approach used in sentiment analysis, keyword spotting is based to counting the frequency of feature-containing[5] words in a document. While it is certainly the most economic method[6], it has an unsurprisingly low accuracy, especially when used for documents that contain large amounts of negations and sarcasm [25]. Be-

5 feature, because keyword spotting can be applied in many areas, not only affective computing or sentiment analysis.

6 It neither takes a lot of time, nor a lot of computation time.

cause it relies mostly on surface features, many sentences containing strong sentiments or emotions are not labeled correctly. A section in a perfumes guide (Luca Turin and Tania Sanchez, *Perfumes: The Guide*, Viking 2008.) taken from Pang et al. [30] illustrates this problem beautifully: "*If you are reading this because it is your darling fragrance, please wear it at home exclusively, and tape the windows shut*". No obviously negative words appear, still the sentence is loaded with emotion and attitude, strongly advising the reader never to use this perfume ever. While such sentences tend to be used rather rarely in every day life, their existence is not refutable, which ultimately makes keyword spotting an insufficient approach for extracting emotions from text [25].

4.4.1.2 *Lexical affinity*

Lexical affinity, when compared to keyword spotting, is a slightly more sophisticated method. Basically, lexical affinity still uses keyword spotting as a basic approach for extraction. What sets it apart is that words get assigned a probabilistic affinity. For example the word "*accident*"'s affinity probably tends to consist of the emotions "*sadness*" and "*anger*". Usually, this knowledge is extracted from linguistic corpora and either assigned by hand or with the use of special computer algorithms [25, 22]. However, similar to keyword spotting, this approach also operates on word-level which leads to decreased accuracy when processing sentences containing negation. Examples of this are: "*I've successfully avoided a car accident*" and "*I met my girl friend by accident*". While the human reader will spot the affinity to *luck* and *joy* with ease, an algorithm might rather assign the sentences the emotions *sadness* and *anger*, as *accident* is in both cases the most emotion-rich word.

4.4.1.3 *Statistical Natural Language Processing*

An even slightly more sophisticated approach for emotion extraction from text can be implemented using *statistical natural language processing*. Other than both previously mentioned methods, statistical NLP does not necessarily operate on the word-level and is therefore not as surface-reliant as keyword spotting and lexical affinity. Conversely, the units of text that are used for training can be arbitrary in size, but need to be plenty and labeled accordingly [25]. Only then statistical NLP can achieve competitive accuracy. The benefits of this approach are that document features like the valence of keywords as well as punctuation and word frequencies can be taken into account. Still, keywords need to be obvious for the classifier to tag them correctly.

4.4.1.4 *Usage of Semantic Networks*

The so far introduced methods for extraction are fairly common as they are used quite a lot in closely related areas as for example in sentiment analysis. Still, words can have tremendous emotional power,

especially embedded in a personal or historical context ("*Your sister died in a car accident*"). We also learned, that sentences do not necessarily need to contain obvious negative words to be negative themselves. But then how do we recognize these language constructs properly? One way of doing this is to use semantic networks and the resulting real-world knowledge about the inherent affective nature of words. Liu et al. [25] use this technique to analyze text on sentence level and later apply real-world knowledge to it. They use a network called *OpenMind*[7] and claim that their approach is superior to keyword spotting and lexical affinity, as it does not rely on surface features and cannot be tricked by structural language features as negation. Further Lui et al. [25] argue that statistical models are effective but require to much input to be used for casual human-to-human interaction. According to them, using the semantic networks approach is superior, especially when applied to short documents.

4.4.2 *Smoothing outputs*

At this point, we filtered our text inputs, removed all stop words and choose a method for extracting emotions. Optimally, our application now returns a word- or sentence-based emotion-expressing vector. Still, these results can processed further to increase they statistical accuracy, as Liu et al. [25] claim. They propose so called smoothing methods to do so. The most easiest way to do smoothing is *interpolation*. Given three sentences, the middle one being neutral, the two encompassing being angry, the middle one can then be interpolated into a slightly less angry sentence. A related method is *decay*. It can be implemented by transferring affective state from its source sentences to subsequent sentences. Leveraging the differentiation of global and local affective states (we introduced this concept in section 4) in documents, another smoothing model can be introduced, called *global mood*. Paragraphs for example can be classified by using smaller chunks of local affective state. To achieve an even better result interpolation and decay can be used. Finally, there is the concept of *meta-emotions*. As we learned in this chapter, emotions relate reflexively to themselves in a compositional nature. We quoted that there are between 2 and 20 emotions of which most of them are composed of other emotions. An example we introduced was the emotion *relief*, a state of *horror* followed by *joy*. Under the assumption that we can extract and detect basic emotions through the means of a semantic network, it should therefore also be possible to smooth the resulting vector with knowledge of an emotion's compositional nature.

7 As of the 8.4.15, all websites related to the OpenMind innitiative are down.

5 REQUIREMENTS

So far, we introduced *pervasive computing* and *context*, *natural language processing* and *affective computing*. Because the conceptual introductions to these have been sufficiently discussed and elaborated on, we can now devote our attention to describing requirements for an emotion recognizing and context-aware application.

The system in question needs to solve two basic use cases:

1. It should be able to **extract emotions from text**
2. It should allow for **recording arbitrary contextual data**

Since these use cases involve rather different tasks, we would prefer to implement them in separate components. As we will find out though, not all requirements can be satisfied in strictly separating concerns in the above described way. Therefore a third use case arises:

3. It should allow **the communication of the two components described in 1. and 2.**

To make referencing easier, we give each of the mentioned use cases a component name:

1. **Emotext**
2. **Cofra**
3. **EtMiddleware**

The following sections are separated requirements listings for all three components. All three sections are written independently from their future implementation and only specify *What* must be integrated, but not *How*.

5.1 REQUIREMENTS FOR EMOTEXT

5.1.1 *General Requirements*

1. Emotext is a module-like component, that can easily be integrated into other applications as a third-party dependency.
2. Emotext features a generic and easy-to-use interface.
    a) Its functionality is loosely coupled and allows for a granular usage of specific methods.

   b) Its functionality is well documented.
   c) Its methods and variables are named predictably.
3. Parameters, known prior to runtime, can be configured without deep technical knowledge, as they are not hardcoded into the actual source code.

5.1.2 *Requirements for Natural Language Processing*

1. Emotext allows adjustment for other languages, though processing English documents is sufficient.
2. Emotext features functionality for segmentation as described in section 3.2.1.
3. Emotext features functionality for word normalization (also referred to as *stemming*) as described in section 3.2.2.
4. If desired, Emotext is able to remove stop words as described in 3.2.4.
5. Any natural language processing functionality implemented by Emotext is connectable.

5.1.3 *Requirements for Emotion Extraction*

1. Emotext implements functionality for extracting emotions from texts of any size as described in section 4.4.
2. Emotext is able to calculate an unknown word's emotional meaning on its own, without manual labeling.
3. Parameters for the process of extraction are adjustable without deep technical knowledge.

5.2 REQUIREMENTS FOR ETMIDDLEWARE

1. EtMiddleware features functionality for caching Emotext's text processing results persistently and dependent on all non-deterministic parameters introduced in section 5.1.3 1.a).
2. EtMiddleware features functionality for smoothing text processing results as described in section 4.4.2.
   a) At least one of the described methods in section 5.1.3 for smoothing results is implemented.
   b) EtMiddleware provides an easy way for developers to enhance functionality, by providing straightforward ways for integration and extensibility.

c) Smoothing functionality is implemented in a way that can be reused (or easily adjusted) on a word, sentence- and document basis.

3. EtMiddleware provides an interface for clustering messages based on their submission time.

   a) Any potential configurable parameters must be adjustable without deep technical knowledge.

5.3 REQUIREMENTS FOR COFRA

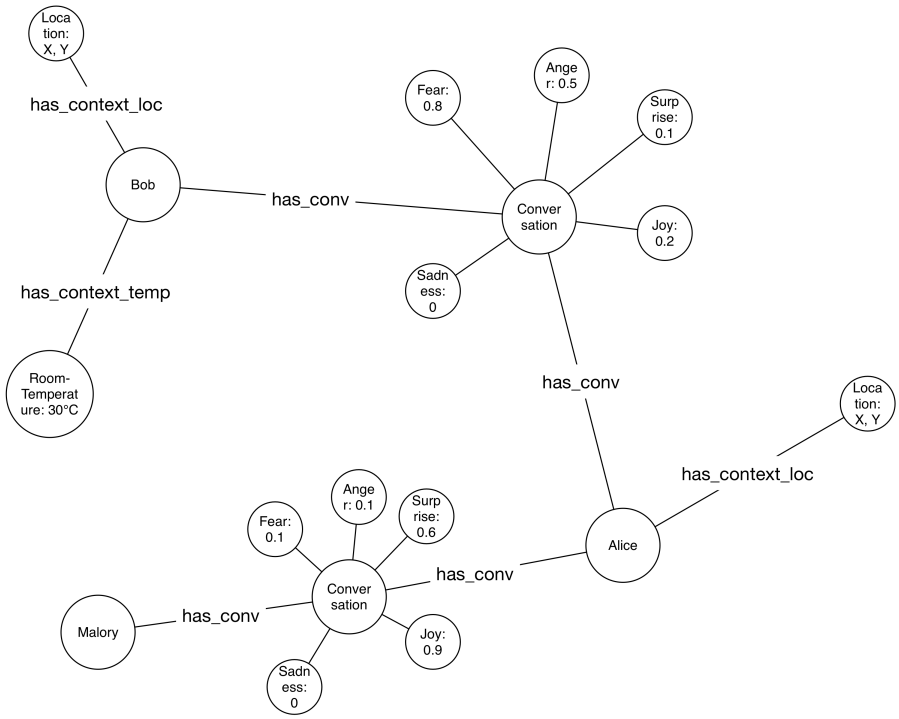

Figure 2: A simplified representation of Cofra's desired graph-like data structure. Time-based versioning was left out for simplification.

1. Cofra allows for storing contextual data as described in section 2.2 and figure 2 persistently.

   a) Its data structure allows for saving data in a generic-as-possible way by mapping real-world entities to multi-purpose key-value-like objects.

2. Cofra provides an easy-to-use RESTful interface for manipulating and populating its data.

   a) Contextual data is connected to a central entity, e.g. a person.

   b) Contextual data is allowed to be missing, removed or manipulated without removing any historical data or changing the data's inherent structure.

3. All data is versioned and immutable.
   a) If the user requires to manipulate existing data, a copy is created, mutated and saved accordingly (marked as a new version).
   b) Any change (manipulation, creation, deletion) is a creation of an entity's new state.
   c) All entities' states are recoverable and discoverable.
4. Cofra's data is easily queryable.
   a) Existing data's historical states are discoverable.
   b) An entity's different states are comparable through an interface.
5. Any communication between a user and Cofra is encrypted using latest standards.
6. The users' data is dereferenced from their real-world identities (*pseudonymization*).

6 DESIGN

In the last chapter, we discussed the requirements for Cofra, Emotext and EtMiddleware. In short, we discussed all required implementation-independent requirements by articulating *what* functionality must be provided but not *how* it should be provided. We choose this procedure because it creates a more generic wording for what we want to achieve and it enables us to focus on the *how* afterwards, in the design part. In the following sections, we are going to discuss decisions that have been made during the design and implementation phase of the project and the reasons behind them.

6.1 THE CHOICE OF TOOLS

When it comes to the choice of tools, programming can be compared to a craft. We have to choose a software development method, an environment, an IDE, a programming language and modules that we would like to use to ease the challenges that we face. In many instances, however, this choice is made passively and without technical reasoning. Generally speaking, people rather chose the known, than the unknown. But especially when designing and implementing software this can slow down the creation process and lead to difficulties in later stages.
Knowing this pitfall, we made our design decisions based on professional arguments, which we introduce in the following sections.
In contrast to the sections of the last chapter, in this one we will not always differentiate between Emotext, Cofra and EtMiddleware. We will however tag individual sections if they reference design decisions specifically made for either one. As for the next section, we will talk about emotion extraction techniques used in Emotext.

6.1.1 *Emotion Extraction Techniques*

Earlier in this work (see chapter 4.4.1) we discussed commonly used techniques in textual emotion extraction. In detail, we mentioned four different ones:

- **Keyword spotting:** Counting all occurrences of feature-containing words
- **Lexical affinity:** Assigning words a probabilistic affinity
- **Statistical Natural Language Processing:** Using the valence of keywords as well as punctuation and word frequencies

- **Semantic Networks:** Defining the emotional value of a word by finding their relation to emotions through real-world knowledge

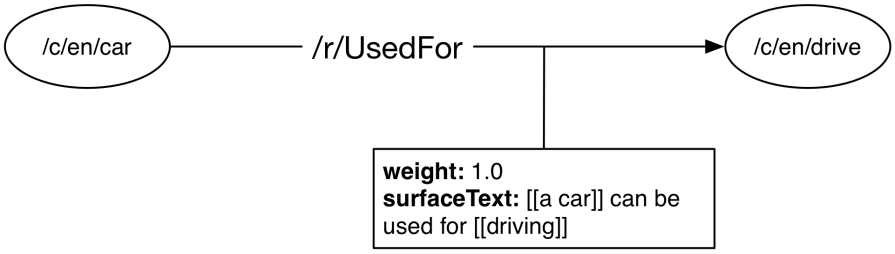

Figure 3: A simplified representation of conceptnet5's internal data structure per word per link. Every word can have any number of links going in and out.

In our requirements specification (see section 5.1.3) we did intentionally not discuss a choice in technique, as this choice is not only a conceptual but also a technical one.
What we did mention, however, was the required functionality we would like to achieve. As a reminder, we want Emotext to be able to extract emotional features from words by looking at their connections to emotion-related words. In essence, this means that statistical natural language processing as well as keyword spotting are insufficient for this task. Further, we specified that Emotext needs to be able to extract information from a document of any size. Though Lexical affinity could be used in this context, finding a sufficient lexicon for this case turned out to be difficult. Instead, we found an advanced semantic network, supported by MIT[1], called *conceptnet5*. Essentially, conceptnet5 is a database controller written in Python that provides a RESTful web interface for looking up language specific concepts behind words. Internally, every word has a list of closely related words connected to it. As seen in figure 3, every link between two words is named, giving the pair a more advanced meaning. In the mentioned figure 3, the concept of *driving* a car is expressed. This approach creates a graph structure that can, though links are directed, be traversed in any direction, circularly, from any word.
In addition, conceptnet5 is provided free of charge, is open source software hosted on Github, has sufficient documentation and promises to grow in scope since new data is populated on regular basis by various contributors. Eventually, we chose to use conceptnet5 in Emotext, hence a semantic network for extracting emotions from text.
Furthermore, Conceptnet5 is implemented using the programming language Python. Which leads us to the choice of programming lan-

1 Massachusetts Institute of Technology

guage, we discuss in the next section. It addresses all three components.

6.1.2 *The Programming Language*

In the last chapter, we briefly described two of our main use cases, namely the extraction of emotions from text and the recording of arbitrary contextual data. In chapter 4 we learned that in doing so, we first need to process the text using methods of natural language processing. We have also learned that there are plenty of techniques for emotion extraction from text (see section 4.4.1) but decided to use conceptnet5, a semantic network, to do so.
The decision, made in the design process, came down to choosing a language that fulfilled the following requirements:

- **Quality of string implementation:** As it is an important task in natural language processing
- **Availability of NLP[2] libraries:** Should provide a basic set of tools for the most commonly used techniques we described in section 3.2
- **Enforcement of modularization practises:** Should encourage the programmer to structure functionality into separate modules
- **Ease of usage:** Should neither have syntactical nor semantical overhead in expression of commands. Instead it should provide the programmer with a high number of easy-to-use and well-documented libraries.

Though unfamiliar with Python at that time, we chose it as it complies with all of the enumerated requirements. It features a wealth of integrated libraries (including a sufficient string implementation, but also a lot of database drivers), has a comprehensive natural language processing library, called Natural Language Toolkit [6], and promotes a modular and agile approach to software development. In addition, it provides a direct way to interact with conceptnet5 and can be run on any major operating system.
Having chosen our programming language we can now take a step further in discussing Cofra's database and data structure.

6.1.3 *The Database*

Similarly to programming languages, database systems come in various forms and functions. While most claim to be general-purpose,

2 Natural Language Processing

they are not. In the previous chapter we described Cofra's database requirements:

- **Timebased versioning:** A data structure's contents need to be versionable over time
- **Adjustability and extensibility:** Data structures must be easily adjustable and new fields should be added effortless
- **Immutable data:** Committed data should not be directly mutable via an update for example. Instead, a copy should be created and mutated (versioning)
- **Interconnected data structures:** The database should allow for the creation of any desired data structure. Particularly tree and graph structures as well as reflexive data structures should be supported
- **Key-value storage capabilities:** Key-value storage functionality would be beneficial
- **Query interface:** The database should provide an interface that allows for easy accessibility to specific data and should allow users to correlate data right out of the box

Despite our researching efforts, we did not find a database software that sufficiently satisfied all our requirements simultaneously. When it comes to *timebased versioning*, *immutable data* and *key-value storage capabilities*, a technique called *event-sourcing* sounds promising though. In essence, event-sourcing is the recording of all changes to an application's state as a sequence of events. Instead of updating our persistent data by sending events to various controllers, we save every event itself persistently. By reiterating over all changes, we can then describe the application's state at any given time and also find the reasons for state transitions [27]. Though this technique sounds promising and appropriate for our use case, there are not many well documented, supported and advanced databases that would allow the implementation of Cofra without compromises. Simply implementing an event-sourcing database ourselves, was not possible neither, as it would have required a great deal of effort, especially making the process transactional. Eventually, we chose the relational database system PostgreSQL, as it offers a strong query language, satisfies all structural requirements, offers key-value storage capabilities and allows us to create versioned data structures that are – at least seemingly – immutable as we will see in chapter 7. Prior to this however, we introduce Cofra's relational data structure and object-relational mapping.

6.2 DATA STRUCTURE AND OBJECT-RELATIONAL MAPPING

Now that we selected our database and programming language, the next essential step is to create a data structure that fits the earlier mentioned requirements (see chapter 5.3). As a reminder, this task was particularly delicate since the data structure should allow the recording of arbitrary contextual data persistently. In short, the requirements are as follows:

- Recorded data is always connected to a central entity.
- All recorded data be represented in a graph like-structure.
- The data structure should support time-based versioning and promote immutability.
- Entities may be interconnected through contextual data (this implies that entities can share contextual data amongst themselves).
- Contextual data can be missing or changed, but not changed structurally.

With these requirements at hand, we can now introduce Cofra's data structure and its object-relational mapping.

6.2.1 *Data Structure*

Cofra's persistent data structure consists of two tables: A *persons* and *contexts* table. As seen in figure 4, a person has a tuple of integers as its primary key, compound from `id` and `timestamp`. A person can also have a string `name`. The attribute `modified` is not modifiable as it is set automatically by a SQL-trigger on every update. A person can have an arbitrary number of contexts related to it. In contrast, a context

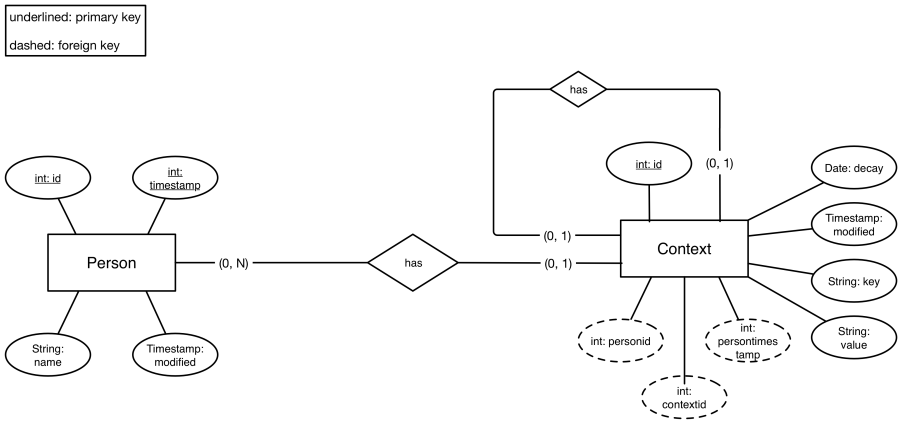

Figure 4: Cofra's data structure in min-max notation.

can (at least for now) either have `personid` and `persontimestamp` or only `contextid` defined. This means that it can either have a person as its parent or another context. For a context to relate reflexively[3], it has an integer `id` and a foreign key `contextid`, that can be used to refer to another row in the table. Furthermore, it has the strings `key` and `value` for content storage, a non-configurable attribute `modified` and a special attribute called `decay`.
Since contextual states of persons are almost never steady, there must be a way for contexts to be deleted after a certain timespan. By setting the `decay` attribute a context can be deleted on a specific date. Cofra detects exceeded `decay` dates automatically, creates a new version and deletes the decayed context from the new version.
Now that we discussed Cofra's data structure, we can introduce the mapping between database relations and Python objects.

6.2.2 *Object-relational Mapping*

Mapping this kind of relational structure is fairly straightforward. As seen in figure 5, every entity is of the type `GraphNode`. A `GraphNode`

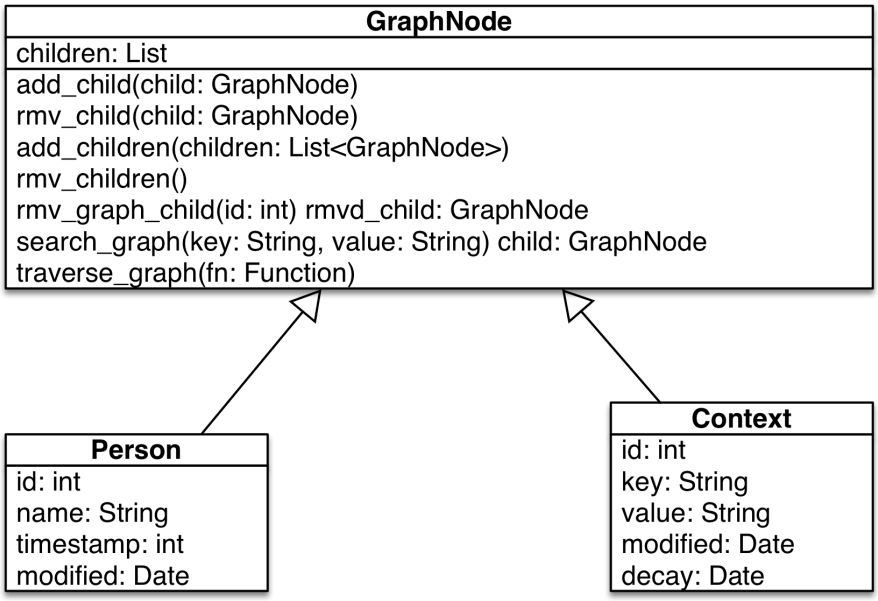

Figure 5: Cofra's object-relational mapping explained in a class diagramm. `Person` and `Context` are simple `GraphNodes` with individual properties that do not implement any methods themselves.

can have an arbitrary number of children and implements methods for manipulating those. Exemplarily, multiple children can be added using `add_children` and individuals can be removed using `rmv_child`. Children's children can be removed using `rmv_graph_child`. Chil-

3 Meaning that it links to another context in his own table.

dren with a specific `key-value` combination can be looked up using `search_graph`. Additionally, children can be manipulated in-place with an anonymous function passed to `traverse_graph`.
The actual entities `Person` and `Context` implement the same attributes, their relational pendants also do and inherit all functionality and attributes (in this case only the list attribute `children`) from their parent `GraphNode`.
Now that we made all necessary decisions concerning the systems setup, we briefly explain its architecture as whole again.

6.3 THE ARCHITECTURE

The system we designed and implemented in this work basically consists of three major components (also seen in figure 6):

- **Cofra:** A RESTful web interface that allows users to record contextual data in a versioned, event-sourced fashion.
- **Emotext:** A Python module that extracts emotions from arbitrary text.
- **EtMiddleware:** Enables communication between Cofra and Emotext.

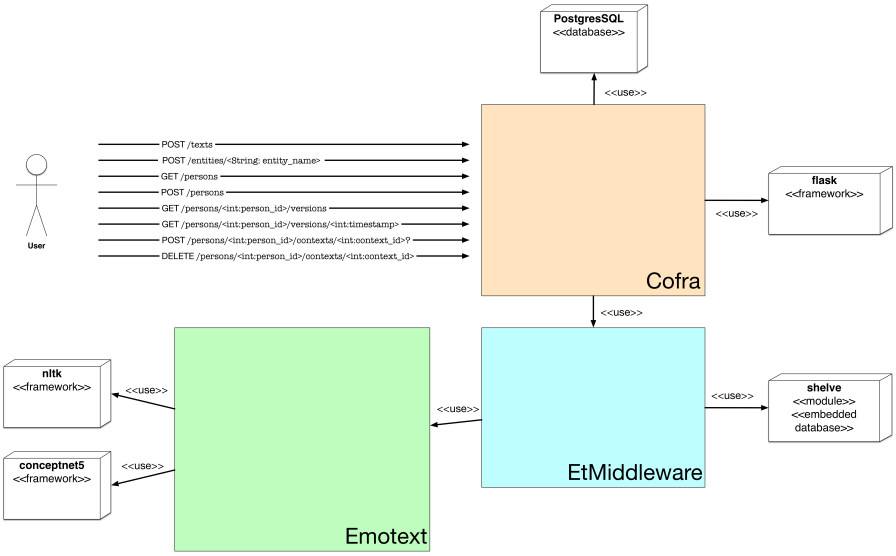

Figure 6: The system represented as a simplified UML class diagramm.

Each of these three components connect to various third party dependencies. They are[4]:

- **PostgreSQL:** Is an advanced object-relational database management system.

4 Descriptions taken from the libraries' Github pages.

- **flask:** Is a small web framework for getting started quickly.
- **shelve:** Is an embedded[5] database and comes with Python's standard library.
- **nltk:** Or Natural Language Toolkit, is a suite of Python modules for research and development in natural language processing.
- **conceptnet5:** Is a word-based common-knowledge database written in Python.

Now that we designed all necessary parts of our system, we can focus our attention on the actual implementation, which we will cover in the next chapter. In a nutshell, we will discuss the core functionality of all three components.

5 A database that is tightly integrated into the application.

7 IMPLEMENTATION

In this chapter we will introduce implementation-specific details for all of our three major components (Cofra, Emotext and EtMiddleware). Since we completed the last chapter by discussing the systems architecture, we will start off with exposing it further. As can be seen

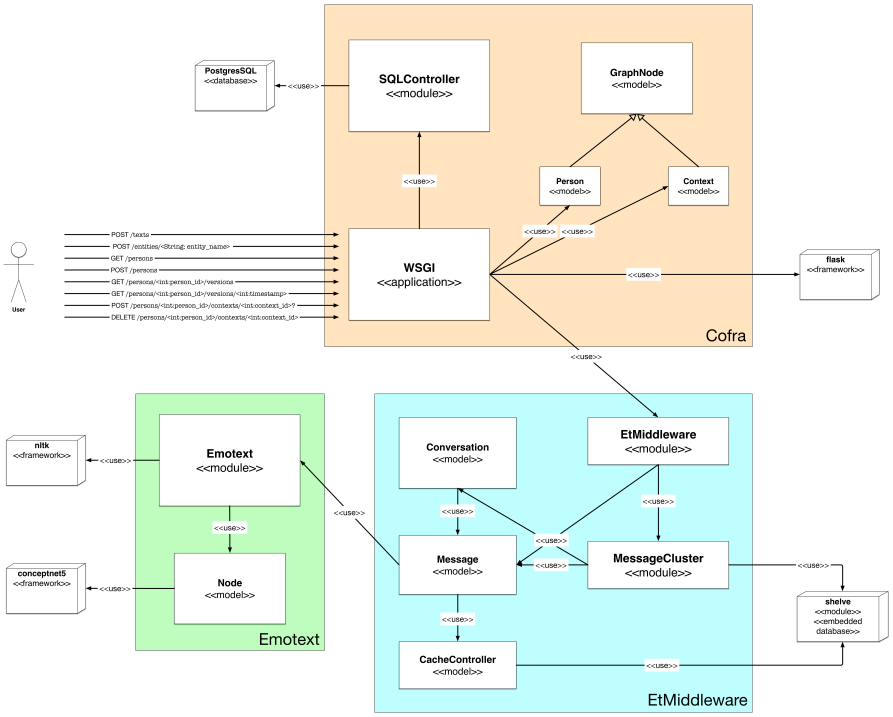

Figure 7: The system devided into three major components including a simplified version of their classes and modules.

in figure 7, we enriched our previously introduced graphic (see figure 6) quite a bit. In particular, we added all modules and classes to the components, to give a more advanced view of the system. To start with, we will take a closer look on Emotext.

7.1 EMOTEXT

Emotext is a module written in Python that uses conceptnet5 and Natural Language Toolkit (nltk) to extract emotions from text. Internally, the module itself hosts another batch of modules. The file structure can be seen in listing 3:

Listing 3: The internal file structure of Emotext's source code.

```
emotext
    | apis
      | concept_net_client.py
      | text.py
    | models
      | models.py
    | utils
      | utils.py
    | requirements.txt
    | settings.py
```

Every `.py` file hosts its own Python module by default. In detail, `concept_net_client.py` communicates with conceptnet5 via HTTP using a method called `lookup`. `text.py` implements the main functionality of the module, namely text processing and emotion extraction. Though, the actual graph search for conceptnet5, that uses the conceptnet5-client lookup functionality, is implemented in `models.py` The module `utils.py` only features one method for processing a string, hence does not require any closer discussion. Instead we will take a closer look at `text.py`.

7.1.1 *Text Processing*

Inside of `text.py` there is a function called `text_processing` (see listing 4). This function is one of the core functionalities of Emotext, as it bundles most `nltk`'s features. For execution, a string of text, as well as three booleans must be passed. The booleans determine how the string should be processed. Particularly, their functions are:

- `remove_punctuation`: Determines wether or not punctuation shall be completely removed from text. We use a simple tokenizer that uses regular expressions. Filtering for `\w{2,}` removes all words smaller than double digits that are not in the range of the alphabet. For instance, punctuation characters, numbers and small words.

- `stemming`: In chapter 3.2, we already talked about stemming. As a reminder, stemming or word normalization is the process of feeding back a morphed word to its origin ("*libraries*" → "*library*"). Popular stemmers use syntactical methods[1] instead of semanticals, which is why it is important to provide the option to turn them off completely, as they will produce rather unsatisfactory results in some cases.

1 This means that the stemmer is using a language-specific rule to process a word instead of using a language- and word-specific rule. For instance: "*smiles*" → "*smile*" but "*foxes*" → "*foxe*" (wrong) instead of "*fox*" (right).

- `remove_stopwords`: Stop words are words that do not contain any valuable information for the text processing algorithm (see section 3.2.4). In our case, we use `nltk`'s standard corpus of English stop words for removal. Examples of such words are: "I", "me", "my", "myself", "we".

Though already enabling plenty of functionality, not everything the function does can be connectible via a boolean. For example, sentences will inevitably be tokenized using a punctuation tokenizer as can be seen in the code snippet 4 (lines 3-5).

Listing 4: A simplified version of the `text_processing` function inside of `api.text.py`.

```
1  def text_processing(text, remove_punctuation=True, stemming=True,
       remove_stopwords=True):
2
3      # Tokenize from sentences to words
4      sentence_tokenizer = PunktSentenceTokenizer(PunktParameters()
           )
5      sentences = sentence_tokenizer.tokenize(text)
6
7      # Remove all non alphabetic words smaller than 2 digits
8      if remove_punctuation:
9          punct_rm_tokenizer = RegexpTokenizer(r'\w{2,}')
10         sentences = [punct_rm_tokenizer.tokenize(s) for s in
               sentences]
11
12     # Remove all stop words, based on nltk's corpus
13     if remove_stopwords:
14         sentences = [[w for w in sentence if not w in stopwords.
               words(language)] for sentence in sentences]
15
16     # Stem words, or at least make them lower case
17     if stemming:
18         stemmer = SnowballStemmer(language)
19         sentences = [[stemmer.stem(w) for w in sentence] for
               sentence in sentences]
20     else:
21         sentences = [[w.lower() for w in sentence] for sentence
               in sentences]
22
23     return sentences
```

Once `text_processing` was fully executed, our original sentences passed in will return as an array of potential feature-containing words that we can now submit to the emotion extraction algorithm.

7.1.2 *Emotion Extraction*

Also implemented in `text.py` is a method called `build_graph`. It is essentially the core of Emotext's emotion processing algorithm. It takes four arguments:

- `token_queue:` A list of words we want to extract emotions from. These words must be passed in as a list of `Node` objects.
- `used_words:` A list of words we already traversed in the process. This is needed because `build_graph`'s search algorithm operates breadth-first on a graph. Therefore we need to avoid cycles and previously processed words of same depth.
- `emo_vector:` A boilerplate Python `dictionary` which we will use to save the found emotions for a specific word. It serves as a future reference to our firstly passed in word, as `build_graph` is implemented recursively.
- `depth:` The levels of children `build_graph` traversed.

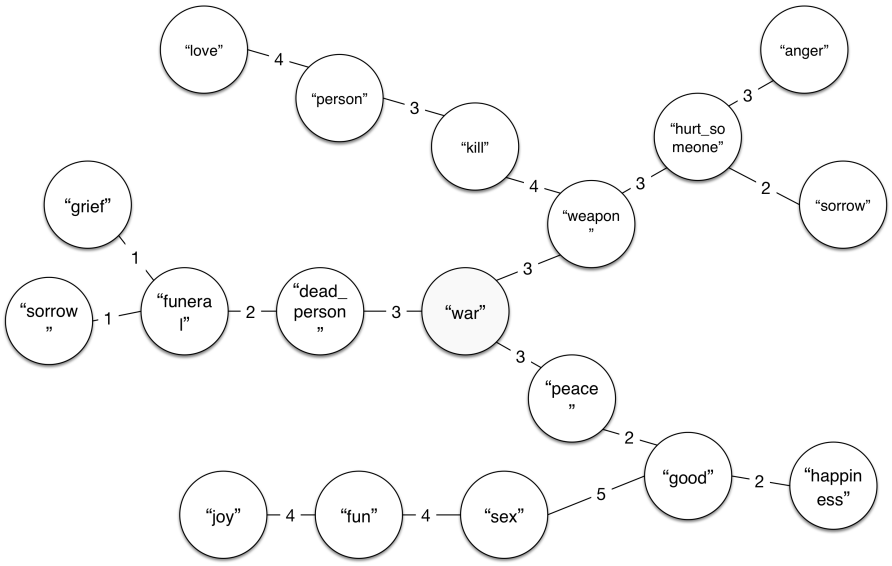

Figure 8: A simplified version of conceptnet5's graph structure based on the word "*war*". In our case, links are bidirectional and weighted by cohesiveness.

To begin, we cast a `string` word to a `Node` object, put it in a list and submit it as `token_queue`. We also pass in an empty list, since no words have been traversed yet (`used_words`), a boilerplate emotion-vector as well as the depth 0.
The idea is to traverse conceptnet5's graph breadth-first starting at a

particular word and find all shortest paths of its surrounding emotions. Emotions are found by looking up the word "*emotions*" on conceptnet5 and by filtering its edges by the link attribute "*IsA*". In figure 8, the surrounding emotions of "**war**" are "*grief*", "*sorrow*", "*love*", "*anger*", "*happiness*" and "*joy*". To accelerate lookup times, we preprocessed all "*isA*" relationships of "*emotions*" and hardcoded them in `config.cfg`, hence they can easily be adjusted by the user. Given the fact that conceptnet5 almost contains 50 gigabytes of data, traversing is a computationally intensive task that we need to limit somehow. Essentially, with enough time or enormous computing-power any connection between two words could be made. However, we only want closely related emotions. Therefore, Emotext provides three ways of limiting a word's lookups:

- `MAX_DEPTH`: Is the maximum traversal distance `build_graph` is allowed to go.
- `MIN_WEIGHT`: As seen in figure 8, links between nodes are weighted. `MIN_WEIGHT` is the minimum weight `build_graph` is allowed to look upon.
- `REQ_LIMIT`: Communication between Emotext and conceptnet5 is done via HTTP. `REQ_LIMIT` limits the number of edges that are returned upon a lookup. Without an HTTP request limit, conceptnet5 usually sends up to 50 related nodes. As this slows down both conceptnet5's lookup times as well as transfer times to Emotext, limiting this parameter to a small number can improve Emotext's per-word processing time greatly.

Once all lookups were successful and `build_graph`'s limiting conditions are satisfied, we take every path that was found in the process, traverse it back and calculate a score for each emotion and path. The presented score formula in figure 7.1.2 shows that we favor short distances instead of long ones, and high numeral links instead of low ones.

$$S_{p(l)} = \sum_{i=1}^{l} \frac{w_i}{i \cdot l^2}$$

In this formula, $S_{p(l)}$ is the score a single path achieves for l, the length of the path p. w_i is the weight of a specific link in the path at index i. By dividing the link's weight through the index i, multiplied by the path's squared length l, each term becomes quadratically smaller. The evaluation of single weights w_i, however, follows the harmonic series.
If we apply this formula to our previously introduced example in figure 8, the score of the path between "*war*" and "*sorrow*" (specifically: $war \rightarrow dead\_person \rightarrow funeral \rightarrow sorrow$) $S_{p(3)}$ would look like this:

$$S_{p(3)} = \frac{3}{1 \cdot 3^2} + \frac{2}{2 \cdot 3^2} + \frac{1}{3 \cdot 3^2} = \frac{13}{27} \approx 0.481$$

In comparison, the score of the longer path between "*war*" and "*joy*" $S_{p(5)}$ produces:

$$S_{p(5)} = \frac{3}{1 \cdot 5^2} + \frac{2}{2 \cdot 5^2} + \frac{5}{3 \cdot 5^2} + \frac{4}{4 \cdot 5^2} + \frac{4}{5 \cdot 5^2} = \frac{112}{375} \approx 0.299$$

Yet, there is another problem we're facing, which are the multiple occurrences of different paths to the same emotions. As can also be seen in figure 8, "*sorrow*" can either be reached by traversing:

$$war \rightarrow weapon \rightarrow hurt\_someone \rightarrow sorrow$$
or
$$war \rightarrow dead\_person \rightarrow funeral \rightarrow sorrow$$

While we could now easily figure out which path is the strongest, we do not want to chose between them, excluding all of them but one. Instead, we would rather - since this is a frequent occurrence - like to add these connections to our score calculation as a word that has many different paths to a specific emotion is likely to be more closely related to it than a word that does not. To do this, we gather the scores for each emotion in a dictionary. If multiple paths for an emotion are found, we simply add them up. For our example in figure 8, the resulting dictionary looks like listing 5.

Listing 5: Calculated absolute scores for the example given in figure 8.

```
{
    "joy": 0.299,
    "sorrow": 1.055, // 0.481 + 0.574
    "grief": 0.481,
    "love": 0.437,
    "anger": 0.611,
    "happiness": 0.518,
}
```

Once we're done with this, we calculate the percentage of each emotion in the dictionary. This is achieved by dividing all values of the dictionary through the sum of all values (which is 3.351).
In listing 6, this method has been applied.

Listing 6: Related emotions for the word "*war*", described in percentages.

```
{
    "joy": 0.08,
    "sorrow": 0.31,
    "grief": 0.14,
    "love": 0.13,
    "anger": 0.18,
    "happiness": 0.15
}
```

Subsequently, this dictionary is returned as an HTTP response to the client, and Emotext's task is done.
Now that we showed the operating principles behind Emotext, we can devote our attention to Cofra's RESTful web interface and EtMiddleware's additional functionality.

7.2 COFRA

In comparison to Emotext, Cofra's architecture is slightly more advanced. It integrates two third-party modules (flask and PostgreSQL), which are respectively integrated in `WSGI` and `SQLController` (both of which can be seen in figure 9). `WSGI` implements a RESTful web interface for interacting with Cofra, while `SQLController` is a Python class for handling database queries. `GraphNode` as well as `Person` and `Context` compound a simple model structure that mimics (as we already saw in section 6.1.3) the database's tables. In this chapter, we introduce Cofra's RESTful interface and how it can be used to manipulate and populate data.

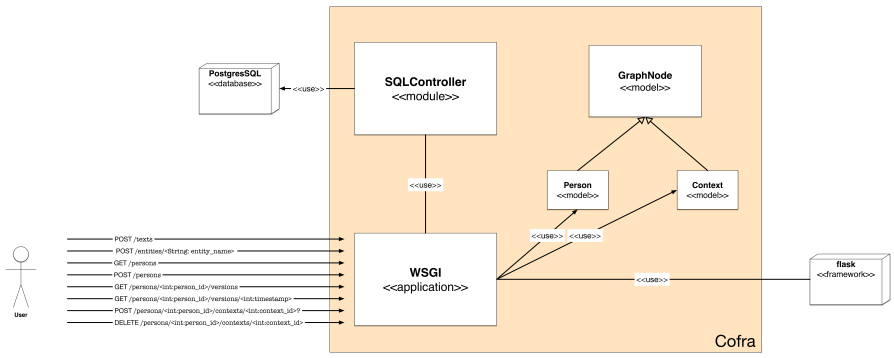

Figure 9: A striped version of the overall system's architecture, only showing Cofra's relevant components.

7.2.1 *The RESTful Web Interface*

In section 6.2, we already introduced Cofra's data structure and object-relational mapping. For reminder, we do not want to update our data by changing already persistently saved bytes on our database. Instead we want, if an update is inevitable, to add our newly discovered data to the latest version's copy and save it afterwards. To keep it simple, we decided that an entity and its context can only be enhanced through the RESTful web interface one object at a time.
Exemplarily, for creating a new person we send an HTTP request like the one seen in listing 7.

Listing 7: A `POST` request for creating a new person.

```
POST /persons HTTP/1.1
Host: http://localhost:3000
Content-Type: application/json

{ "name": "Bob" }


HTTP/1.1 201 Created

{
    "name": "Bob",
    "id": 0,
    "timestamp": 0,
    "modified": 2015-01-13T15:32:43.450686Z,
    "children": []
}
```

The server will return the request after creating a new row. As can be seen, the returned object not only features the `name` of the person, but also its last `modified` date, an `id` and a `timestamp`. Either by requesting the resource `GET /persons` or `GET /persons/<person_id>/versions` we can see the newly created person's latest states.
Now that we have successfully created a new person, we can enhance its current state by adding contextual information over time. There are two routes for doing this:

1. `POST /persons/<int:person_id>/contexts`
2. `POST /persons/<int:person_id>/contexts/<int:context_id>`

Using the first one, an HTTP request looks like the one shown in listing 8. As we can see, the body's dictionary only requires the property `key` to be defined, still `id`, `modified` and `children` are added automatically by the server. Also, notice that the `timestamp` attribute of Bob got incremented while his `id` stayed untouched. Purposely, we

created `Location` without a `value`, as it simply does not have any. Instead, its actual value is derived from its children, as we will see next.

Listing 8: A `POST` request for adding contextual information to a person.

```
POST /persons/1/contexts HTTP/1.1
Host: http://localhost:3000
Content-Type: application/json

{ "key": "Location" }


HTTP/1.1 201 Created

{
    "name": "Bob",
    "id": 0,
    "timestamp": 1,
    "modified": 2015-01-13T15:35:33.456586Z,
    "children": [{
        "id": 0,
        "key": "Location",
        "modified": 2015-01-13T15:35:34.656236Z,
        "children": []
    }]
}
```

In another request (see listing 9), this time using the second route, we add a property called `Longitude` to `Location`. Note, that again `timestamp` was increased and `id` remained the same. Requesting `GET /persons/1/versions`, still all versions of Bob can be inspected. For deleting either `Location` (and all its children) or `Longitude`, the HTTP `DELETE` method can be applied to both previously introduced routes as well. It removes the nodes in question and creates a new version of Bob with an once more incremented `timestamp`. Again, all previously generated versions remain available, as there is no way of deleting something completely.

Finally, there is another component we introduce, called EtMiddleware. Previously, we described it as a component for enabling Emotext's and Cofra's communication. As we will find out in the next section, EtMiddleware additionally implements functionality that could

have only implemented using the users' acquired data collectively.

Listing 9: A `POST` request for adding contextual information to an existing context.

```
POST /persons/1/contexts/1 HTTP/1.1
Host: http://localhost:3000
Content-Type: application/json

{
    "key": "Longitude",
    "value": "10.4541194"
}


HTTP/1.1 201 Created

{
    "name": "Bob",
    "id": 0,
    "timestamp": 2,
    "modified": 2015-01-13T15:37:10.749253Z,
    "children": [{
        "id": 0,
        "key": "Location",
        "modified": 2015-01-13T15:37:10.952745Z,
        "children": [{
            "id": 0,
            "key": "Longitude",
            "value": "10.4541194",
            "modified": 2015-01-13T15:37:11.102655Z,
            "children": []
        }]
    }]
}
```

7.3 ETMIDDLEWARE

As we mentioned already in Requirements and Design, Emotext is implemented as a Python module and Cofra is implemented as a standalone application. Because it promotes reusability, we explicitly specified their independency. However, there still needs to be a way to interact with Emotext RESTfully, but more importantly, there is functionality that can only be implemented leveraging both. Use cases like:

- correlating users' emotions and contexts; or
- clustering users' messages to conversations; or
- smoothing emotion extraction outputs; and

- caching emotion extraction outputs;

are only possible using both technologies. Figure 10 shows all relevant pieces of EtMiddleware. Its most integral part is the module `MessageCluster`.

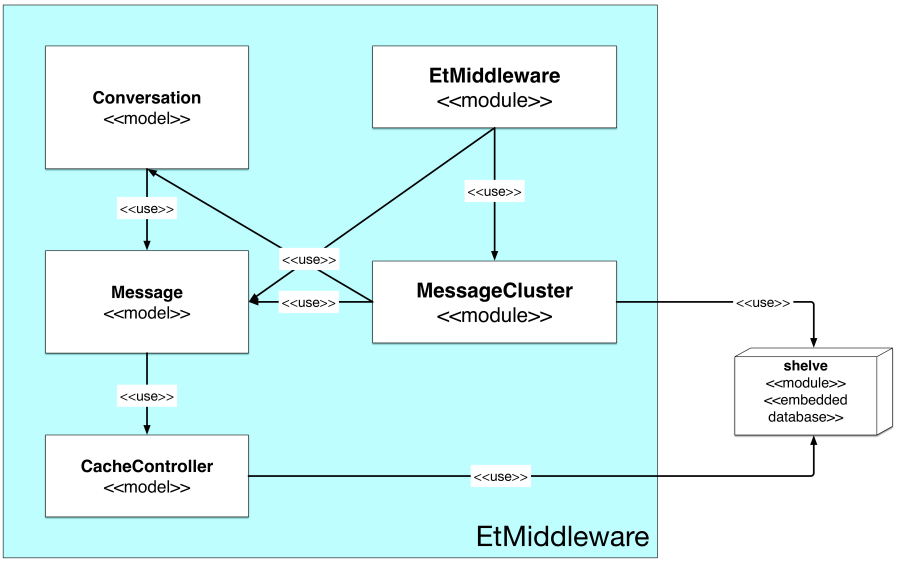

Figure 10: A striped version of the overall system architecture, only showing EtMiddleware's relevant components.

7.3.1 *Message Clustering*

Already, we discussed the importance of social interaction in our everyday lives (see Affective Computing's section 4.4) and specified that Emotext must be able to extract emotions from any document independent of size (see Design's section 6.1.1). This specification is important as Emotext should be applicable in widely spread communication tools like instant messaging, where short documents are the default and messages are not necessarily clustered into conversations. Hence, EtMiddleware implements an interface that clusters messages according to their arrival date. The flow chart in figure 11 explains the basic process. `MessageCluster`, a standard Python class, is initialized with a numeric threshold time in seconds. When the first message is added to `MessageCluster`, a counter is started that keeps track of the time between the latest two consecutive messages. If the counter's time exceeds `MessageCluster`'s threshold time, a conversation is considered to be over.
Hence, `MessageCluster.to_conversation` is called and its embedded database, as well as its counter, are reseted. As long as the counter does not exceed the threshold, the submitted message is just added to the database. Otherwise, the process will reset itself and start all

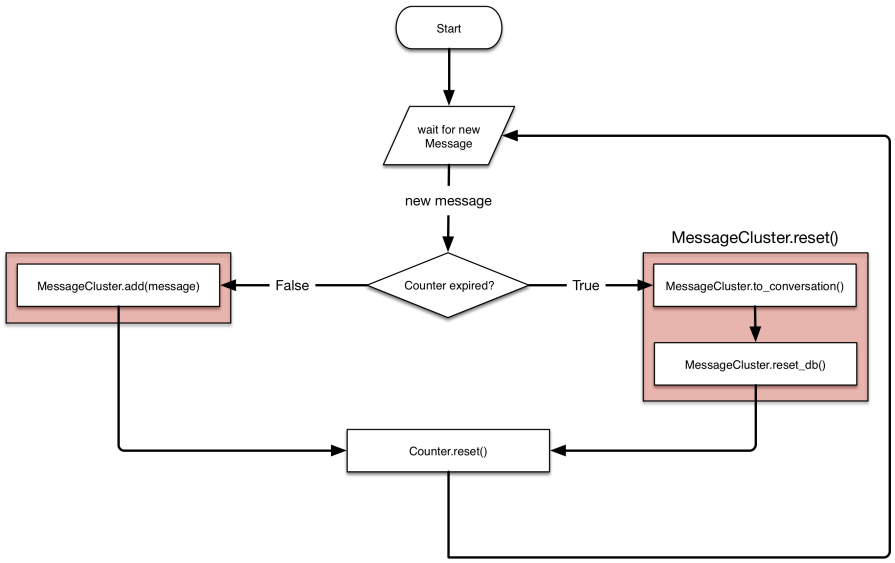

Figure 11: A flow chart describing `MessageCluster`'s methology of grouping messages.

over again.
Another functionality that can only be implemented in EtMiddleware is the caching of words that have already run through Emotext, in order to constrain its computationally intensive nature.

7.3.2 *CacheController*

A few sections earlier (see section 7.1.2), we learned that emotion extraction from text is a computation consuming task. However, given the same limiting parameters (`MAX_DEPTH`, `MIN_WEIGHT` and `REQ_LIMIT`) it is also a deterministic one. Hence, `CacheController`, also a standard Python class, can be initialized using all three limiting parameters and passed into `Message.to_emotion_vector`. For every token passed in `to_emotion_vector` checks for a cache-hit and returns the responding vector or `None`[2]. Of course, if a token does not produce a cache-hit it is processed and added to the cache. This procedure greatly reduces server load and overall lookup times.
Potentially, `CacheController`'s functionality could be outsourced into its own package, saving Emotext's results persistently.

7.3.3 *Smoothing Methods*

In section 4.4.2, we introduced methods for smoothing Emotext's outputs. Exemplarily, we implemented a word-based interpolation method into the `Conversation` class. As can be seen in lisiting 10, it

2 The Python equivalent of Java's `Null`.

calculates the average values for a token's resulting vector by looking at its left and right neighbors.

Listing 10: A simplified version of the interpolation method used in `Conversation`.

```
def word_interp(words):
    interp_words = list()

    for i, w in enumerate(words):
        prev_word = words[i-1]
        if i == len(words)-1:
            next_word = words[0]
        else:
            next_word = words[i+1]
        if prev_word is not None and next_word is not None:
            interpolated_w = self.interp_emotion_vector(prev_word
                , w, next_word)
            interp_words.append(interpolated_w)
    return interp_words
```

For even better results, interpolation (and generally smoothing methods) could also be implemented on a sentence, conversation and even document level.
In the next section, we will discuss and rate the achievements of this work.

8 DISCUSSION

This chapter evaluates and discusses the results of this work. Prior to rating our implementation results, we will analyze the process in retrospective and highlight failures, achievements and things we have learned.

8.1 RETROSPECTIVE

Prior to writing this work, I hardly knew anything about context-aware computing, natural language processing and especially nothing about affective computing. In the beginning, the idea was to write a sentiment analyzing mobile application to help treat mentally ill patients. One idea was to adjust a room's temperature or lighting dependent on a patient's mood. In theory, patients would install an alternative keyboard on their mobile device that would record (with the approval of the patient of course) their written communication. Soon though, while defining and writing the chapter about context-aware computing bigger problems emerged. Specifically, context's generic definition helped surface some of them. While nowadays most applications inherit their state directly from the most recent pieces of information, in context-aware computing time and therefore historical data is equally important as current data. In addition, contextual data can not be jammed into a static data structure, relational databases these days promote. Instead, a data structure must provide a way to model real-life situations freely at runtime. Furthermore, it needs to promote version control and help the user query historical and present data in an understandable fashion. Ultimately, I did not find any appropriate database system that could satisfy all these requirements, which lead to the implementation of Cofra.
Similarly, sentiment analysis did not fit the goals I wanted to achieve. The idea was to somehow classify a user's affect using their their textual input, not being sure yet how this could be done. Only after quite a lot of research in natural language processing and sentiment analysis I found the field of affective computing. While it most certainly served my needs, namely recognizing *emotions* (and not *attitudes*, for example), there are hardly any resources on extracting emotions from text. Moreover, graph-searching a semantic network is rather a novelty. The most expedient work for this was probably Lui et al's "*A model of textual affect sensing using real-world knowledge*" [25]. Their approach consists of the conversion of a semantic network's information into a affect-measuring lexicon of words. Not only did this lead to the

discovery of conceptnet5, it also introduced to me the idea of smoothing methods.
The Python programming language turned out to be equally inspiring. It is extremely easy to learn, comes with a gigantic toolkit of embedded ready-to-use libraries and strongly promotes modularization. Particularly pleasing was that it is self-documenting. Per convention, any function is ought to have a so called *doc-string*, a multi-line comment that describes its functionality. In spite of using various third-party libraries, it never lead to reading an extensive documentation. Conversely, I found myself often searching for function definitions in the actual source code. Surprisingly though, this was fun, effective and by no means difficult. In every instance, external code was easily understandable, sufficiently documented and ultimately contributed to a better understanding of the Python language on my part. Additionally, the lack of exceptional programming language constructs[1] lead to a lot of bug free code and satisfying debugging sessions. Test-driven development, again, turned out to be the right programming method to implement backend software and especially helped refactoring major parts of the system.
The most essential lesson learned from writing this thesis though, is that provided with motivation, time, great advisors and the right understanding of how to examine a topic, any subject is worth researching but most importantly *learnable*.

8.2 EVALUATION

In this section, we rate the implementation against all the enumerated requirements (see chapter 5). Ratings are represented as percentages, critical decisions are annotated in the comment column. To ease comprehension, we segmented the requirements rating equivalent to the grouping used in chapter 5 and provided links for reference.

8.2.1 *Emotext*

This section compares Emotext's requirements to its actual implementation. Specifically, requirements are grouped by their field of expertise.

1 Consider Javascript's "*Falsy Values*" or Objective-C's agressive garbage collection one of them.

8.2.1.1 *General Requirements*

Link	Name	Rating	Comment
5.1.1.1	Modularization	80%	Can be integrated as third-party dependency, but is quite large (conceptnet5 is 50GB).
5.1.1.2	Interface	90%	Granularity. Self-documented Python classes, predictable function declarations and variable names.
5.1.1.3	Configuration	100%	All parameters are adjustable in `config.cfg`

Table 1: Rating for Emotext's general implementation

8.2.1.2 *Requirements for Natural Language Processing*

Link	Name	Rating	Comment
5.1.2.1	Localization	50%	Though possible, results in other languages are inaccurate as conceptnet5's database for them is considerably smaller. English-specific NLP techniques used (besides `nltk`).
5.1.2.2	Segmentation	100%	Segmentation on sentence and word level, language independent because `nltk` is used. Punctuation characters and non-words are removed.
5.1.2.3	Normalization	70%	Language independent implementation through `nltk`. Stemming decreases accuracy as it is done syntactically.
5.1.2.4	Stop Words	100%	Language independent stop words removal through `nltk`
5.1.2.5	Granularity	80%	Punctuation removal, stemming and stop words removal is connectable.

Table 2: Rating for Emotext's natural language processing implementation

8.2.1.3 *Requirements for Emotion Extraction*

Link	Name	Rating	Comment
5.1.3.1	Document Size	80%	Extraction is done on word level. So even tiny documents can be processed sufficiently. More comprehensive levels (sentence-based e.g.) can easily be implemented.
5.1.3.2	Technique	100%	All common English words are processed and yield corresponding emotions.
5.1.3.3	Configuration	100%	All parameters are adjustable in `config.cfg`

Table 3: Rating for Emotext's emotion extractor implementation

8.2.2 *EtMiddleware*

In this section we discuss EtMiddleware's implementation compared to its requirements.

Link	Name	Rating	Comment
5.2.1	Caching	100%	Word-based caching algorithm, dependent on configurable parameters, persistently stored in embedded key-value storage.
5.2.2	Smoothing Methods	75%	Message-based interpolation is possible. Enhancements are effortlessly implementable. Interpolation was not implemented in a reusable way.
5.2.3	Message Clustering	100%	Message clustering to conversations is possible. All parameters are adjustable in `config.cfg`.

Table 4: Rating for EtMiddleware's implementation

8.2.3 *Cofra*

In this last section, we compare the requirements to our actual implementation.

Link	Name	Rating	Comment
5.3.1	Genericity	70%	Complex data structure modeling is possible, e.g. trees and graphs. However, the application logic only supports trees, no graphs. Connections between persons in SQL are possible, not in in application.
5.3.2	Interface	100%	Flexible RESTful web interface for manipulation of data.
5.3.3	Immutability	100%	All data is versioned. In the application logic, data is treated as immutable. Historical states are explorable.
5.3.4	Interface	60%	Data is only queriable for the user through the RESTful web interface. Queries in SQL are possible, though complex (e.g. recursive statements). Data is not comparable.
5.3.5	Encryption	0%	Was not implemented, though in an actual deployment scenario `flask` could be easily adjusted to using HTTPS for example. Database is not encrypted as well.
5.3.6	Pseudonymization	100%	Tracked users can choose any name that is not already used. No meta-data (like the user's IP address for instance) is tracked.

Table 5: Rating for Cofra's implementation

9 CONCLUSION

The goal of this work was to use textual emotion extraction in the field of context-aware computing. To achieve it, we chose to study three research subjects. Though seemingly unrelated judging by their names and attempts, context-aware computing, natural language processing and affective computing result in an interesting intersection, ready to be exploited by specifically designed software.

In chapter 2, we learned that for context-aware computing to function appropriately, its requirements for adaption and disappearance must be fulfilled and enough data to study the user's environment must be present. We learned that there are multiple dimensions of context, namely external (physically measured) and internal context (specified through the user, not physically measured). As our initial goal was to use text-extracted emotions in context-aware computing, these findings helped us enhance the idea further. Essentially, we developed a software sensor to detect a user's emotions and later use them as an internal context dimension in context-aware computing. To properly implement this, we researched natural language processing (see chapter 3) and affective computing (see chapter 4), in order to find commonly used methods in both domains.

Over the course of the requirements and design phase (chapters 5 and 6), we found that there was no plug-and-play solution present for saving and handling contextual data. Hence, we did not only specify requirements for Emotext, our text-processing component, but also for Cofra, a contextual data management framework. To reiterate on this: Emotext is a Python module that is able to extract emotions from any word, using conceptnet5, a word-based graph database for real-world knowledge. Cofra is a Python application that can be used to record arbitrary contextual data in a graph-like structure.

Subsequent research showed that Emotext's outputs can further be polished by applying smoothing methods on them and by specifying the application's use case closer to a certain domain. One example, we showed this exemplarily was with the implementation of a message clustering algorithm that can be used to group incoming instant-messages in a larger context, namely a conversation. In this case, we adjusted Emotext's functionality to be used a social-communication context.

Due to the limited scope of this work, we were not able to fulfill all imposed requirements appropriately. Requirements like localization, negation handling and stemming turned out to be quite hard problems, as they require deep linguistic knowledge. In addition, Cofra's

makeshift implementation is subpar. In order to record and correlate contextual data successfully, a fact-based, versioned, event-sourced database with a strong query language that allows for complex real-world-like entity-relationship models is required, yet not available currently. Unluckily, we were also not able to apply all this functionality to a real-world use case. Still, we can think of plenty examples were they could potentially flourish. As mentioned in chapter 4.3 for instance, the records of emotions and contextual data could be used as already labeled data for machine learning operations to predict either a users future change in state or - as a result of a change in state - its future emotions. Considering the recent hype in wearables and smart mobile devices in users' homes and pockets, Cofra could be used as a recorder for contextual data. More exciting, though, is the idea of using Emotext to predict a stock's market by analyzing certain social media streams. Hence, we would love to see the great potential of these applications being enhanced and used.

LIST OF FIGURES

LISTINGS

LIST OF TABLES

BIBLIOGRAPHY

[1] Bing blog: Bing shopping launch. http://blogs.bing.com/search/2011/03/01/bing-feature-update-searching-for-a-good-deal-new-natural-language-capabilities-in-bing-shopping-understand-prices . Accessed: 19-03-2015.

[2] Google I/O Day One: Google continues attacks on Apple, Amazon. http://www.cnet.com/news/google-io-day-one-google-continues-attacks-on-apple-amazon/. Accessed: 16-12-2014.

[3] Google Translate. https://translate.google.de. Accessed: 27-04-2015.

[4] Martin Porter's website. http://tartarus.org/~martin/PorterStemmer/. Accessed: 27-04-2015.

[5] Smartphones: So many apps, so much time. http://www.nielsen.com/us/en/insights/news/2014/smartphones-so-many-apps--so-much-time.html. Accessed: 17-04-2015.

[6] Natural Language Toolkit. http://www.nltk.org/. Accessed: 28-04-2015.

[7] Sentiment symposium tutorial. http://sentiment.christopherpotts.net/. Accessed: 14-12-2014.

[8] The world according to SwiftKey. http://swiftkey.com/en/blog/the-world-according-to-swiftkey. Accessed: 17-04-2015.

[9] Unixtool wc man pages. http://unixhelp.ed.ac.uk/CGI/man-cgi?wc. Accessed: 27-04-2015.

[10] WolframAlpha. http://www.wolframalpha.com. Accessed: 27-04-2015.

[11] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[12] Cecilia Ovesdotter Alm, Dan Roth, and Richard Sproat. Emotions from text: machine learning for text-based emotion prediction. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 579–586. Association for Computational Linguistics, 2005.

[13] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[14] Dan Jurafsky, Christopher Manning. Natural language processing.

[15] Taner Danisman and Adil Alpkocak. Feeler: Emotion classification of text using vector space model. In *AISB 2008 Convention Communication, Interaction and Social Intelligence*, volume 1, page 53, 2008.

[16] Sanjiv R Das and Mike Y Chen. Yahoo! for amazon: Sentiment extraction from small talk on the web. *Management Science*, 53(9): 1375–1388, 2007.

[17] Anind K Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[18] Stuart Dredge. Whatsapp growth continues with 700m users sending 30bn daily messages. http://www.theguardian.com/technology/2015/jan/07/whatsapp-growth-700m-users-facebook. Accessed: 17-04-2015.

[19] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Pervasive Computing*, pages 167–180. Springer, 2002.

[20] Facebook Inc. Mobile daily active users (mobile daus). http://files.shareholder.com/downloads/AMDA-NJ5DZ/3907746207x0x805520/2D74EDCA-E02A-420B-A262-BC096264BB93/FB_Q414EarningsSlides20150128.pdf. Accessed: 17-04-2015.

[21] Chris Johnston. Ubers value more than doubles to $40bn after investors back fundraising. http://www.theguardian.com/technology/2014/dec/05/uber-value-doubles-after-fundraising. Accessed: 17-04-2015.

[22] Dan Jurafsky and James H Martin. *Speech & language processing*. Pearson Education India, 2000.

[23] Daniel Jurafsky and James H Martin. *Speech and language processing, 2nd edition*. Prentice Hall, 2008.

[24] Tomi Kilgore. Facebooks stock-market valuation tops $230 billion. http://www.marketwatch.com/story/facebooks-stock-market-valuation-tops-230-billion-2015-03-20. Accessed: 17-04-2015.

[25] Hugo Liu, Henry Lieberman, and Ted Selker. A model of textual affect sensing using real-world knowledge. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 125–132. ACM, 2003.

[26] Masoud Makrehchi and Mohamed S Kamel. Automatic extraction of domain-specific stopwords from labeled documents. In *Advances in information retrieval*, pages 222–233. Springer, 2008.

[27] Martin Fowler. Event Sourcing. http://martinfowler.com/eaaDev/EventSourcing.html. Accessed: 01-04-2015.

[28] Gilad Mishne. Experiments with mood classification in blog posts. In *Proceedings of ACM SIGIR 2005 Workshop on Stylistic Analysis of Text for Information Access*, volume 19, 2005.

[29] Clifford Nass, Jonathan Steuer, and Ellen R Tauber. Computers are social actors. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 72–78. ACM, 1994.

[30] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.

[31] Rosalind Wright Picard. Affective computing. 1995.

[32] Rob Price. Android just achieved something it will take apple years to do. http://uk.businessinsider.com/android-1-billion-shipments-2014-strategy-analytics-2015-2?r=US, . Accessed: 17-04-2015.

[33] Rob Price. The apple watch is already wiping the floor with the entire smartwatch market. http://uk.businessinsider.com/apple-watch-kgi-securities-2-million-ming-chi-kuo-smartwatch-android-wear-pebble-2015-4?r=US , . Accessed: 17-04-2015.

[34] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

[35] Klaus R Scherer. Emotion as a multicomponent process: A model and some cross-cultural data. *Review of personality & social psychology*, 1984.

[36] Carlo Strapparava and Rada Mihalcea. Semeval-2007 task 14: Affective text. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, pages 70–74. Association for Computational Linguistics, 2007.

[37] Carlo Strapparava and Rada Mihalcea. Learning to identify emotions in text. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1556–1560. ACM, 2008.

[38] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

ERKLÄRUNG

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm,

Tim Daubenschütz