Karlsruhe Institute of Technology

Coreference Resolution for Software Architecture Documentation

Bachelor's Thesis of

Quang Nhat Dao

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Anne Koziolek
Second reviewer:	Prof. Ralf Reussner
Advisor:	M.Sc. Jan Keim
Second advisor:	M.Sc. Tobias Hey

03. February 2022 – 03. June 2022

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Abstract

In software engineering, software architecture documentation plays an important role. It contains many essential information regarding reasoning and design decisions. Therefore, many activities are proposed to deal with documentation for various reasons, e.g., extracting information or keeping different forms of documentation consistent. These activities often involve automatic processing of documentation, for example traceability link recovery (TLR). However, there can be problems for automatic processing when coreferences are present in documentation. A coreference occurs when two or more mentions refer to the same entity. These mentions can be different and create ambiguities, for example when there are pronouns. To overcome this problem, this thesis proposes two contributions to resolve coreferences in software architecture documentation.

The first contribution is to explore the performance of existing coreference resolution models for software architecture documentation. The second is to divide coreference resolution into many more specific type of resolutions, like pronoun resolution, abbreviation resolution, etc. For each combination of specific resolutions, we will have a specific approach.

To evaluate the work of this thesis, we will first look at how the approaches perform for coreference resolution for software architecture documentation. For this, Hobbs+Naive, a combination of Hobbs' algorithm and naive non-pronoun resolution, achieves an F1-score of 63%. Meanwhile StanfordCoreNLP_Deterministic, a deterministic coreference resolution system of Stanford CoreNLP, scores 59%. Then, we want to see how well the approaches resolve coreferences for a specific activity, which is TLR. StanfordCoreNLP_Deterministic reaches an F1-score of 63%, while Hobbs+Naive reaches 59% for this aspect. Lastly, because coreferences from pronouns are one the biggest issues for TLR, we also evaluate how the approaches perform for pronoun resolution. In this case, the combinations with Hobbs' algorithm as pronoun resolution model achieves an F1-score of 74%, whereas StanfordCoreNLP_Neural only achieves 71%. To conclude, the combination approaches perform better for coreference resolution for software architecture documentation. Moreover, they perform better than the existing model approaches at pronoun resolution for TLR. Nonetheless, the existing model approaches are superior when it comes to coreference resolution for TLR.

Zusammenfassung

In der Softwareentwicklung spielt die Softwarearchitekturdokumentation eine wichtige Rolle. Sie enthält viele wichtige Informationen über Gründe und Entwurfsentscheidungen. Daher gibt es viele Aktivitäten, die sich aus verschiedenen Gründen mit der Dokumentation befassen, z. B. um Informationen zu extrahieren oder verschiedene Formen der Dokumentation konsistent zu halten. Diese Aktivitäten beinhalten oft eine automatische Verarbeitung der Dokumentation, z. B. Traceability Link Recovery (TLR). Bei der automatischen Verarbeitung kann es jedoch zu Problemen kommen, wenn in der Dokumentation Koreferenzen vorhanden sind. Eine Koreferenz liegt vor, wenn sich zwei oder mehr Erwähnungen auf dieselbe Entität beziehen. Diese Erwähnungen können unterschiedlich sein und zu Mehrdeutigkeiten führen, z. B. wenn es sich um Pronomen handelt. Um dieses Problem zu lösen, werden in dieser Arbeit zwei Beiträge zur Koreferenzauflösung in der Softwarearchitekturdokumentation vorgeschlagen.

Der erste Beitrag besteht darin, die Leistungsfähigkeit bestehender Modelle zur Koreferenzauflösung in der Softwarearchitekturdokumentation zu untersuchen. Der zweite Beitrag besteht darin, die Koreferenzauflösung in viele spezifischere Arten von Auflösungen zu unterteilen, wie die Pronomenauflösung, Abkürzungenauflösung usw. Für jede Kombination von spezifischen Auflösungen haben wir einen spezifischen Ansatz.

Um die Arbeit dieser Abschlussarbeit zu evaluieren, werden wir uns zunächst ansehen, wie die Ansätze für die Koreferenzauflösung in der Softwarearchitekturdokumentation abschneiden. Hier erreicht Hobbs+Naive, eine Kombination aus Hobbs' Algorithmus und naiver Nicht-Pronomen-Auflösung, einen F1-Score von 63%. StanfordCoreNLP_Deterministic, ein deterministisches System zur Koreferenzauflösung von Stanford CoreNLP, erreicht dagegen 59%. Dann wollen wir sehen, wie gut die Ansätze die Koreferenzen für eine bestimmte Aktivität, nämlich TLR, auflösen. StanfordCoreNLP_Deterministic erreicht einen F1-Score von 63%, während Hobbs+Naive 59% für diesen Aspekt erreicht. Da Koreferenzen von Pronomen eines der größten Probleme bei TLR sind, bewerten wir schließlich auch, wie die Ansätze bei der Pronomenauflösung abschneiden. In diesem Fall erreicht die Kombination mit Hobbs' Algorithmus als Pronomenauflösungsmodell einen F1-Score von 74%, während StanfordCoreNLP_Neural nur 71% erreicht. Zusammenfassend lässt sich sagen, dass die Kombinationsansätze eine bessere Leistung bei der Koreferenzauflösung in der Softwarearchitekturdokumentation erbringen. Außerdem schneiden sie bei der Pronomenauflösung für TLR besser ab als die bestehenden Modellansätze. Nichtsdestotrotz sind die bestehenden Modellansätze bei der Koreferenzauflösung für TLR überlegen.

Contents

List of Figures

List of Tables

1. Introduction

When it comes to developing software, especially a large software system, software architecture plays an important role [20]. Software architecture describes the software system as a set of smaller components [15]. These components relate and work with each other to perform the main functionalities of the system. This partitioning of the system into smaller parts allows many people to cooperatively and productively develop the system [15]. Moreover, a good software architecture also helps improve many desired qualities like performance, accuracy, security, etc [15]. Given the use of software architecture, it is crucial to document it so that it can be easily comprehended and shared [15]. Beside development, software architecture also supports other activities such as deployment and maintenance [29]. Therefore, it is vital to document the software architecture to provide these activities with necessary information [29].

Documenting software architecture is essential to capture all reasoning and design information behind it [28]. These information can also describe domain or technical aspects of the software. Overall, there are two forms of software architecture documentation: informal documentation and formal documentation. On one hand, informal documentation can be written in natural language or sketched in order to be easily accessed and used [28]. An example of informal documentation would be a natural language text describing a software architecture. On the other hand, formal documentation is written in modelling languages, e.g., UML. Thus, it requires technical knowledge to read and understand formal documentation. In spite of that, this provides benefits such as supporting tracking changes in software architecture, early evaluation and simulation for performance prediction [29].

Many activities involve automatic processing of the documentation. For example, natural language processing techniques are used to detect the uncertainty cues in software architecture documentation [43]. However, a phenomenon can happen in the documentation that can prove problematic for many of those activities. That phenomenon is coreferences. A coreference can occur, when two or more mentions refer to the same entity. There are two forms of mentions: indirect and direct mentions. Coreferences from indirect mentions can cause ambiguities in the documentation. One activity that is much affected by these ambiguities is traceability link recovery (TLR).

To highlight the problems of coreferences, TLR is explained as follows. Both informal and formal documentation describe the architecture in one way or another. Along the course of development, changes can be made to either or both of the documentation. Such changes can make two documentations no longer consistent to each other. Hence, any changes made in one form should also be considered for the other to avoid deviation. Deviation can result into missing or outdated documentation or inconsistency between artifacts [28]. If left unchecked, the deviation can cause confusion and lead to deficiencies in the final product [42]. The below excerpt and Figure 1.1 are examples of informal and

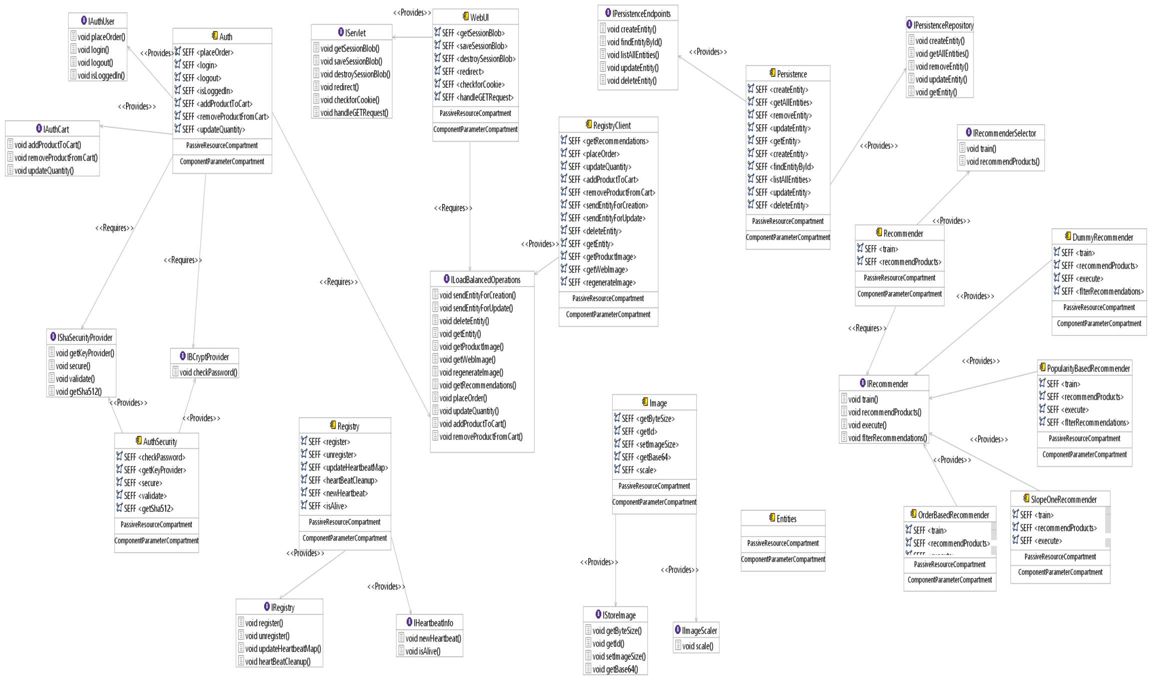

Figure 1.1.: Formal documentation of the case study TeaStore from the SWATTR project [1]

formal documentation. They both describe the software architecture of an application named TeaStore [31].

> The Persistence service provides access to the data persisted in the relational database back-end. It maps the relational entities to the JSON entity objects passed between services using the EclipseLink JPA ORM mapper. It features endpoints for general CRUD-Operations (Create, Read, Update, Delete) for the persistent entities.

We take a modification in the informal documentation as an example. Assuming that *The Persistence service* is to be deleted, not only its mentions in informal documentation should be removed but also those in formal documentation. Therefore, it is helpful to keep track which elements in which documentation are related. One way of doing that is by creating a form of connection between the two forms of documentation. For example, we can establish a link between a mention in informal documentation and another mention in formal documentation. Thus, when we change a mention in any of the two forms, we can see if it is linked to any other mention, and make appropriate adjustment provided that mention exists.

Hence, TLR was proposed to tackle this problem. TLR is the task in which the dependencies and relationships between software artifacts are identified [32]. Particularly, it creates links between elements from different artifacts. The links are called tracelinks. In this thesis, we will constrain the software artifacts in TLR to informal and formal documentation. In this regard, a tracelink is created when an element in formal documentation is mentioned in informal documentation. For the previous example, we can create tracelinks between each mention of Persistence service in the informal documentation to the model element Persistence in the formal documentation. For direct mentions, we only need to find an element in formal documentation with the same name and create a tracelink. Nonetheless, we may want to focus more on indirect mentions, since their references are obscure to us at first glance. This makes tracelinks between them and the model elements even more interesting and important. Take the following example:

> The RecipeManagement class is responsible for managing recipes. It has methods for creating, reading, updating and deleting recipes.

It is obvious that the word *RecipeManagement* mentions an element named RecipeManagement. Therefore, if there is a RecipeManagement model element in the formal documentation, we should create a tracelink between it and the word *RecipeManagement*. Besides, we should also create a tracelink between the word *It* in the second sentence and the RecipeManagement model element because it refers indirectly to the RecipeManagement model element. Because *RecipeManagement* and *It* are mentioning the same element, we have a coreference. Due to the ambiguity caused by this coreference, or in other words because TLR does not know which entity *It* refers to, the tracelink can not be created. Thus, we have to resolve the coreference, i.e., find out which entity the mention refer to. Coreference resolution is the task of gathering mentions into equivalence classes [16]. The mentions in a class all refer to the same entity. Furthermore, coreference resolution is an important part of many higher level natural language processing (NLP) tasks that demand NLP understanding such as document summarization, question answering, and information extraction [2].

Without coreference resolution, TLR is almost certain to miss tracelinks between indirect mentions and the model elements, because it does not know which entities the indirect mentions refer to. Missing these tracelinks can lead to indirect mentions not being tracked. As a result, if models in formal documentation change, deviation can occur and lead to inconsistent artifacts, although keeping artifacts consistent is one of many main responsibilities of TLR. Thus, it is safe to say that coreference resolution is an important part of TLR. Not only TLR, but also other activities that process software architecture documentation can be affected by the ambiguities caused by coreferences.

Notwithstanding, coreference resolution for software architecture documentation has not yet received full attention. Hence, this thesis focuses on tackling coreference resolution in software architecture documentation. For that, a number of approaches will be proposed. It will be looked into how these approaches perform for coreference resolution for software architecture documentation. Moreover, we will see how well the approaches solve the problems that coreferences cause in TLR. To be more specific, we will see how our approaches resolve coreferences resulted from mentions, that should be linked to the model elements in formal documentation by TLR. Additionally, coreferences caused by pronouns also pose as one of the biggest issues in TLR. Therefore, we will also look into the performance of our approaches as pronoun resolution models.

First, Chapter 2 describes the terms and definitions that lay the foundation of this thesis. After that, Chapter 3 covers the works that are related. These works can be used to compare against the work of this thesis. Subsequently, Chapter 4 explains the approaches to the problem. Chapter 5 evaluates the results of the approaches. Last but not least, Chapter 6 concludes this thesis and presents potential future work.

2. Foundations

This chapter provides a general overview on terms and definitions that lay the foundations of this thesis. First, Section 2.1 describes the two forms of documentation that represent software architecture. Then, Section 2.2 explains the concept of traceability link recovery. Next, Section 2.3, 2.4, 2.5 in turn clarify the terms coreference resolution, pronoun resolution and string metric algorithms. These are important to the understanding and implementation of this thesis. Last but not least, Section 2.6 introduces the natural language processing frameworks that support the implementation of this thesis.

2.1. Software Architecture Documentation

There is no fixed definition of software architecture. However what most definitions have in common is that software architecture describes the system under development as a set of smaller components, that work together to fulfill the main functionalities of the system [15]. The reason for this division is that it allows members of a team or teams to work together regardless of temporal or geographical boundaries to develop the software system [15]. Software architecture provides many benefits for development, e.g, understanding of the software system, reusability, construction blueprints, communication, etc [23]. Moreover, numerous essential qualities of the software like performance, accuracy, and security are enabled by software architecture [15]. Apart from development, software architecture also provides necessary information for other activities like deployment and maintenance [29]. With the said use and benefits of software architecture, it is without a doubt important to document the software architecture to share it across team members and better maintain it should the need arise [15].

Software architecture documentation can be categorized in two forms: informal documentation and formal documentation. Informal documentation can be written in natural language for the sake of readability and ease of formulation. One clear example of informal documentation is texts. The following illustrates one of such.

> QuickMeal has the following classes: User, Recipe and Ingredient. User has an id, user name, password, name, list of their own recipes and list of their favorite recipes from other users. A recipe has an id, name, tag, list of ingredients, and cooking time. An ingredient has an id and a name...

The other form of documentation is formal documentation. Different from informal documentation, the formal documentation is written by modeling languages. One example of a modeling language is UML, which is mainly used for visualization of design of system [17]. Therefore, reading and composing this form require special knowledge of the software

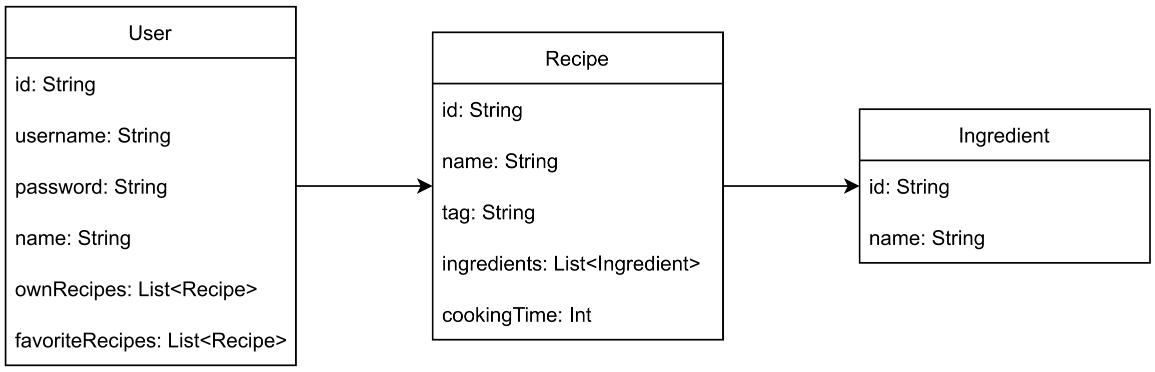

Figure 2.1.: An example of model elements

system and the modeling language itself. Some modeling languages like Palladio also allow with the help of simulation to early predict the performance or reliability of a software system [40]. Figure 2.1 illustrates an example of formal documentation. The model elements in this example describe the software architecture more or less like the one by the previous example of informal documentation.

As we can see from the two examples, informal and formal both describe the same software system, just in different ways. Thus, it is important to keep the two forms consistent, otherwise functionalities will be missed or wrongly implemented in the final product [42]. Hence, the following Section 2.2 will present a solution to such problem. This is also an example of an activity that processes software architecture documentation.

2.2. Traceability Link Recovery

Traceability link recovery (TLR) is a task in which links are established between elements in different software artifacts [38]. The created links are called tracelinks. In the case of software architecture, a tracelink is created when an element in the formal documentation is mentioned in the informal documentation. This plays a decisive role in keeping the consistency between the forms of documentation, because we can determine which elements are not mentioned and as a result which information are outdated or contradicted [28]. Moreover, TLR can make improvements to the maintainability and reliability of software [32]. With a large enough project, manually linking components can be very laborious and error-prone. Thus, this furthers the need of automatic and accurate TLR.

Figure 2.2 illustrates the linking of components from a textual document to a class diagram with TLR. The document in the left side mentions an element named user. On the right side, the class diagram User also happens to exist. Therefore, we can create a tracelink between them.

Now the clients demand that a user should no longer has a username. We can change the informal documentation, and use the created tracelink to find the corresponding model element. Then we adjust the model element so that the two artifacts stay consistent to each other. We can also make changes in the model elements and use the tracelinks to

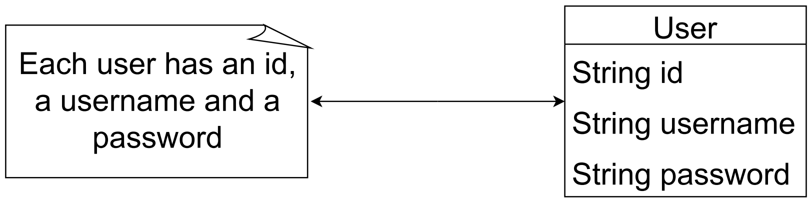

Figure 2.2.: An example of TLR linking components from a textual document and a class diagram

find where the model elements are mentioned in the informal documentation and make appropriate adjustment.

2.3. Coreference Resolution

Coreference resolution is one of the important topics in the computational linguistics. It is the task of determining which mentions in a document are jointly making reference to the same entities. The following example illustrates an example of coreference resolution.

> John said: "Today, I went to a super market and bought a watermelon. It was delicious"

In this small text, *John* and *i* are both referring to the same identity, which is a person named John. On the other hand, *a watermelon* and *it* are referring to the delicious watermelon that John bought from the supermarket. It might seem easy, but there are many other grammatical rules that make implementing an algorithm for coreference resolution even more difficult. An example of such rule is gender nouns. The word "she" might refer to a woman or other feminine nouns like mother, actress, or ship while the word "he" is more likely connected with masculine nouns like father, husband, and rooster. This requires the algorithm to be able to distinguish such cases. Moreover, there are instances in which coreference resolution must depend on the context underlying the text under examination as shown in the following example.

> The parents let their children go to the party because they did well on the exams.
>
> The parents let their children go to the party because they wanted time for themselves.

In the first sentence, *their children* and *they* are more likely to refer the children of the mentioned parents. That is because of the context *they did well on the exams.* Now that the context has changed to *they want time for themselves* in the second sentence. The likelihood that this action belongs to the parents is higher. Thus, *The parents* and *they* may corefer.

In the previous examples so far, we see that one of two coreferencing mentions is a pronoun. In such cases, the problem of coreference resolution can be characterized as pronoun resolution. Pronoun resolution will be discussed in the next section. However, it is not always the case.

In the text below, *Microsoft Word* and *the software* corefer to each other. Such problem requires resolution for non-pronouns. One way to tackle this problem is depicted in Section 2.5.

> Microsoft Word is a word processing software developed by Microsoft. It was first released on October 25, 1983. The software is written for many platforms like Microsoft Windows, macOS, and Unix etc.

Because coreference resolution plays an important role in many high level natural language processing tasks [2], many approaches to the problem have been made. These approaches can generally be categorized into deterministic, mention-pair, mention-ranking or entity-based algorithms. Each of these categories has its advantages and disadvantages. For example, deterministic approach has the lowest F1-score but it does not require data sets to train the models like those of mention-ranking approach. The concrete models will be described in Chapter 3.

2.4. Pronoun Resolution

Pronoun resolution can be considered as a part of coreference resolution. Therefore, coreference resolution models should theoretically be able to resolve pronouns. Besides, there are algorithms that resolve pronouns exclusively. One prominent example is Hobbs' algorithm.

Hobbs' algorithm is an algorithm for pronoun resolution. Like coreference resolution, pronoun resolution finds out which two mentions are coreferencing to each other. Nonetheless, one of the two mentions must be a pronoun. Hobbs' algorithm is built on a simple tree search procedure defined by terms of depth of embedding and left-right order [33]. In a nut shell, it traverses the constituent tree upward and leftward from the pronoun. Along the way, it will propose noun phrases. These noun phrases are potential coreferencing candidates for the pronoun. The detailed steps of Hobbs' algorithm can be found in [26].

2.5. String Metric Algorithms

When we resolve two non-pronoun mentions, there are two ways. The first way is taking into consideration the context behind them. The second way is resolving them based on their similarity. We will focus more on the second rather than the first way, because enabling a computer program to analyze contexts is no easy task. Therefore, this section will focus on the string metric algorithms. These algorithms measure the distance or inverse similarity between strings.

Levenshtein distance is a string metric algorithm to measure the difference between two sequences. It calculates the smallest number of single-character edits that are required

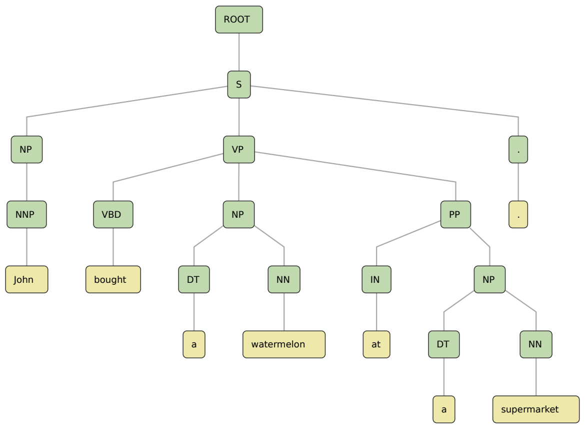

Figure 2.3.: A constituent tree generated from a sentence by Stanford CoreNLP

to change one word into the other. The strings do not have to be the same length. For example, the levenshtein distance between the word *books* and *back* is three.

Hamming distance also calculates the minimum characters between two strings that are required to transform one word to another. Unlike Levenshtein, the two words must have equal lengths. For instance, the hamming distance between the word *book* and *back* is two.

2.6. Natural Language Processing Frameworks

Coreference resolution is a well-known natural language processing (NLP) task. Therefore, many NLP frameworks provide it. Moreover, if we want to develop our own coreference resolution models, we may have to rely on other NLP tasks, such as tokenization, lemmatization, sentence splitting, etc. Thus, we can use available NLP frameworks to support us. Not only they may provide us with their own coreference resolution models, but they also support necessary NLP tasks for coreference resolution. This allows us to speed up the development by focusing more on the main logic of coreference resolution. There are already many NLP frameworks available like openNLP [3], LingPipe [4], etc. However, we only concentrate on the frameworks that the implementation of this thesis uses.

The first framework is Stanford CoreNLP [5]. The most work of this thesis is supported by Stanford CoreNLP. It supports various NLP tasks, e.g., tokenization, constituency parsing, parts of speech etc. Most notable among them is coreference resolution. Stanford CoreNLP provides three different modes of coreference resolution, namely neural, statistical, and deterministic. We will discuss more about them in Section 3.1. Each of the NLP tasks requires a certain set of annotators. These annotators are required to set up a pipeline. The NLP tasks are then performed on an input document by having the pipeline process the document. Stanford CoreNLP supports a variety of programming languages, but most supported is Java. Figure 2.3 illustrates Stanford CoreNLP performing constituency parsing on a sentence. The result is a constituent tree, which can be used by Hobbs' algorithm to resolve pronouns.

The second framework is spaCy [6]. This framework is developed for Python. NeuralCoref 4.0 is an extension of spaCy and it comes with coreference resolution [7]. The details of NeuralCoref 4.0 will be explained in Section 3.1. With all the important terms and definitions explained, we can move on to work, that are related to this thesis.

3. Related Work

This chapter describes works that are related to this thesis. First, Section 3.1 lists the coreference resolution models in the natural language processing community. Next, Section 3.2 explains a number of work that use coreference resolution in traceability link recovery. Last but not least, Section 3.3 details activities that process software architecture documentation.

3.1. Coreference Resolution Models

In this section, we will discuss a number of coreference resolution models. Some of them are used in the implementation of this thesis. Commonly, the average F1-score metric based on three evaluation models is used to indicate the performance of a coreference models. The evaluation models are MUC, BCUBED and $CEAF_{\phi}4$. MUC and BCUBED will be described in Chapter 5.

NeuralCoref 4.0 is a pipeline extension for spaCy. It is composed of two sub-modules. The first one is a rule-based mentions-detection module. The module adopts SpaCy's tagger, parser and NER annotations for the identification of a set of likely coreferencing mentions. The second one is a feed-forward neural-network computing a coreference score for each pair of possible mentions [7]. Figure 3.1 shows a result of NeuralCoref 4.0 resolving coreferences. We can see that it gives a score of 2.74 for the pair of mentions *a watermelon* and *it*.

We have stated in Section 2.6 that Stanford CoreNLP provides three different modes of coreference resolution. There are deterministic, statistical and neural systems. Deterministic system is a multi-pass sieve rule-based coreference system based on the work described in Lee et al. [34] and Raghunathan et al. [39]. It has the lowest F1 score (49.5 and

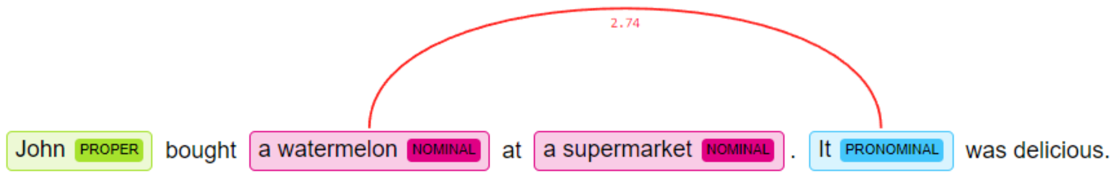

Figure 3.1.: An example of NeuralCoref 4.0 resolving coreference

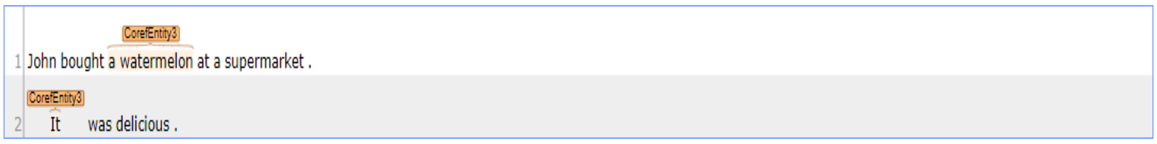
CorefEntity3
1 John bought a watermelon at a supermarket .
CorefEntity3
2 It was delicious .

Figure 3.2.: An example of Stanford CoreNLP resolving coreference

based on the CoNLL 2012 evaluation data) among the three modes. Statistical system is a mention-ranking model. It uses a large set of features and iterates through each mention in the document and may create a coreference link between the current one and a previous mention in each iteration. It has a F1-score of 56.2 and the fastest resolution time. Different from the previous two systems, neural system is a neural-network-based mention-ranking model. It has the highest F1-score (60) but the slowest resolution time out of the three [5]. Also for this model and all the models below, the metrics are evaluated on the data of the CoNLL-2012 shared task [27]. Figure 3.2 shows a result of Stanford CoreNLP resolving coreferences.

Kantor and Globerson (2019) is a coreference resolution model that is based on entity equaliziation [27]. According to Kantor and Globerson, it is important to capture properties of entity clusters to resolve coreferences. Hence, to achieve this, they approach the problem with an entity equalization mechanism. Equalization in this context means to represent each mention in a cluster by approximating the sum of all mentions in the cluster. The model has the average F1-score of 76.6.

Joshi et al. (2019a) present a coreference resolution model that uses BERT [24]. The abbreviation stands for Bidirectional Encoder Representations from Transformers. It was designed by Devlin et al. to "pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers"[21]. The F1-score of this coreference resolution model is higher compared to the model from Kantor and Globerson based on the OntoNotes (+3.9) and GAP (+11.5) benchmarks. On average, the model has the F1-score of 76.9.

Joshi et al. (2019b) is different from the previous version. It uses SpanBERT instead of BERT [25]. SpanBERT is a pre-training method designed and improved in representing and predicting spans of text. It is an extension of BERT in a sense that it masks contiguous random spans instead of random tokens. Then, it trains the span boundary representations in order to predict the entire content of the masked span. Therefore, the need to rely on the individual token representations within the masked span is removed. Compared to its predecessor, the model has a higher average F1-score (79.6).

Xu et al. (2020) is a coreference resolution model that is supported with higher-order inference (HOI) [46]. Moreover, they propose an end-to-end coreference system as well as four different HOI approaches. These approaches have been adapted by many recent coreference resolution models. They are attended antecedent, entity equalization, span clustering, and cluster merging. This model has the average F1-score of 80.2.

s2e+Longformer-Large is a model proposed by Kirstain et al [30]. It does not depend on span representations, handcrafted features, and heuristics. Therefore, it has a smaller memory footprint. Furthermore, the coreference resolution model competes on par with the then standard model and also is simpler. s2e+Longformer-Large has a slightly higher F1-score than that of Xu et al. (80.3).

Currently, the most recent coreference resolution model is wl-coref + RoBERTa [22]. Previous models greatly depend on span representations in order to create coreference links between word spans. The number of word spans is $O(n^2)$ and the number of potential links is thus $O(n^4)$ with n being the length of input text. This requires a variety of pruning techniques to make computation of these models feasible. However, wl-coref + RoBERTa considers coreference links between individual words instead of word spans. Hence, the complexity is reduced to $O(n^2)$ and the pruning of mentions is no longer necessary. Later, the word spans can be reconstructed by the model. This model has the highest average F1-score (81.0).

3.2. Traceability Link Recovery

Coreference resolution plays an important role in traceability link recovery (TLR). Therefore, many approaches put great stress on the performance of coreference resolution. Because this thesis not only focuses on coreference resolution for software architecture documentation but also for TLR, a relation to the following works can be made.

Keim et al. discuss the topic of trace link recovery for software architecture documentation [29]. They propose the framework SoftWare Architecture Text Trace link Recovery (SWATTR) for tracelink recovery. A tracelink is created between a sentence and a model element if the element was referred in the sentence. This does not exclude coreferences like *It*. Currently, the function of coreference resolution is disabled due to low accuracy.

Schlutter and Vogelsang describe how to improve trace link recovery using semantic relation graphs and spreading activation [41]. In their work, they reuse their natural language processing (NLP) pipeline with many techniques such as part-of-speech tagging, lemmatizing (morphological analysis), dependency parser (grammatical structure), SRL and coreference resolution. Coreference resolution is applied to the semantic relation graphs. For example, if a phrase is a pronoun, the vertex of the representative is added instead of the vertex of the phrase.

Zhang et al. introduce an ontological approach for the semantic recovery of traceability links between software artifacts [48]. In their work, coreference resolution is used to find references, each of which involves a single entity being referred to with different textual descriptors, along with pronominal references.

Arunthavanathan et al. focus on adding an NLP support to extend their tool SAT-Analyzer [14]. SAT-Analyzer stands for Software Artefacts Traceability Analyzer. The tool uses Standford CoreNLP to resolve coreferences in order to extract the artifact elements from the requirement document. Stanford CoreNLP was described in previous Section 2.6 and its coreference resolution in Section 3.1.

3.3. Software Architecture Documentation Processing Activities

Beside traceability link recovery, there are many other activities that also process software architecture documentation. Coreferences in the documentation may affect the performance of those activities. Therefore, the usage of coreference resolution for those activities can be considered.

Shumaiev and Bhat propose the use of natural language processing techniques for the detection of uncertainty cues in software architecture documentation [43]. They analyzed three documentations and retrieved a variety of uncertainty cues. Based on them, they hypothesized how the communication about the software architecture with the stakeholders can be improved with on-time uncertainty detection.

According to Su, a software architecture documentation contains architectural information, that needs to be structured as chunks in order to improve the use of the documentation [44]. A chunk is a group of related information. The work aims to identify and create useful chunks of architectural information in software architecture documentation. Moreover, the development of a tool to create and use this knowledge is also focused on. The approach is to capture and save the exploration paths of users' documentation and the related meta data. The paths are then analyzed to look for common patterns that can be grouped into chunks.

Architecture recovery focuses on providing a high level abstraction of a system that uses the architectural elements to represent how it functions and interacts. This abstraction, or architecture, makes it easier to understand the program and supports all the phases of software life cycle. Chardigny and Seriai propose the use of the intentional architecture of a system in order to boost "the adequation between the resulting software architecture and the architect's expectations without requiring more human expertise" [19]. The intentional architecture represent the system as how its designers imagined. The authors present in their paper a semi-automatic process for intentional architecture recovery from what the expert recommend and the available documentation.

4. Approach

The goal of this thesis is to resolve coreferences in software architecture documentation. Therefore, we create a number of Java libraries to do that. In this chapter, we will look into the approaches that will be taken to create these libraries. First, Section 4.1 describes the general idea to tackle the problem. Then, Section 4.2 presents the architecture of the Java project to implement the libraries. Next, Section 4.3 explains the terms that are most important to the domain of the implementation. Last but not least, Section 4.4 explains details that are important to the implementation of the libraries.

4.1. Overview

One viable way of resolving coreferences is using already existing models implemented for coreference resolution. These models are for example Stanford CoreNLP, NeuralCoref 4.0, etc. To a certain extent, we can use these models to resolve coreferences. However, if we break down coreference resolution, we will find that it consists of many more specific forms of resolution. For instance, coreferences can come from pronouns, two identical mentions, two nonidentical but coreferent mentions, or abbreviations (KIT and Karlsruher Institut für Technologie). Instead of using a coreference resolution model to resolve all types of mentions, we can combine many models and algorithms, and use specific ones to resolve specific types of mentions. Not only does this improve the performance of coreference resolution, but also the level of customization of the libraries. For example, if model A resolves pronouns better than B and B resolves abbreviations better than A, we can delegate pronoun and abbreviation resolution accordingly. Furthermore, if our documentation has no pronouns, we can use a trivial pronoun resolution. It will simply skip the resolution and save the calculation time. Resorting to multiple resolutions also boosts the extendibility of our Java libraries. In the future, if there is a better resolution model for some specific types of mentions, we can swap this model in and leave other models intact. With that said, the calculation time can significantly rise, because some or all of the models may have to run through the documentation at least once.

To summarize the idea, there are three main objectives to fulfill. First, we have to integrate the already existing coreference resolution models through adapters. We also need to create an interface, that these adapters have to implement. This will allow third party users to use the adapters without having to know about their implementation details. Second, we have to break down coreference resolution into more specific forms of resolution. In this thesis, coreference resolution is broken down to pronoun and non-pronoun resolution. Of course, the non-pronoun resolution can be further broken down into many other sub-categories like personal names, abbreviations, etc. Nonetheless, we may want to keep it as general as possible so that our approach can theoretically cover all cases of coreferences. For each of the two specific resolutions, we will have a corresponding interface to expose its functionalities. The coreference resolution adapters obviously have to implement both interfaces, because coreference resolution is supposed to resolve any types of mentions. Lastly, we need to combine the two resolutions to resolve coreferences. Thus, we have to create another interface to use the pronoun and non-pronoun interfaces. This interface in turn extends the coreference resolution interface to provide necessary functionalities. Then, we can choose a pair of adapters from pronoun and non-pronoun resolution to form a new library.

Our goal after all is to create Java libraries for coreference resolution. With the general idea already identified, we can move on to specifying the architecture of our Java project. The next section will go into more details about the architecture and the components inside it.

4.2. Architecture

Figure 4.1 illustrates the architecture of our Java project. The concrete coreference resolution Java libraries are created by either extending the abstract class *CoreferenceResolver* directly (with the exception of *CombinableCoreferenceResolver*) or indirectly through the abstract class *CombinableCoreferenceResolver*. *CoreferenceResolver* makes sure that all concrete classes can provide the third party user with methods and properties necessary for coreference resolution and retrieving the results. For example, users can initialize an instance of *CoreferenceResolver* with an input text, call method *resolveCoreferences* to resolve coreferences, and use getter methods for properties *mentions*, *links*, *chains* to access the results. *StanfordCoreNLPCoreferenceResolver* and *NeuralCoref* are the directly extending classes of *CoreferenceResolver*, and will be more thoroughly discussed in Section 4.4. *CombinableCoreferenceResolver* implements the method *resolveCoreference* of *CoreferenceResolver*. First, it populates the resolvers with results of resolution by calling their methods *resolve<type of mention>s*, e.g., *resolvePronouns*. Then it delegates the resolution of each mentions by calling the method *resolve<type of mention>*, e.g., *resolveNonPronoun*, of the resolver responsible for these mentions. In the current architecture, we have two interfaces for specific resolvers, namely *PronounResolver* and *NonPronounResolver*. The pronouns will be resolved by *PronounResolver* and non-pronouns by *NonPronounResolver*. The implementing class of *PronounResolver* is *Hobbs* and of *NonPronounResolver* are *Naive* and *Levenshtein*. They will be explained in more details in Section 4.4. Beside the input text, the concrete classes of *CombinableCoreferenceResolver* will be initialized with implementing classes of *PronounResolver* and *NonPronounResolver*. For instance, *HobbsAndNaive* is initialized with an input text, an instance of *Hobbs* and an instance of *Naive*. It will resolve pronouns with Hobbs' algorithm and resolve non-pronouns with the naive non-pronoun resolution. Directly extending classes of *CoreferenceResolver* also implement *PronounResolver* and *NonPronounResolver* and therefore are eligible for combination. Thus we can have libraries such as *HobbsAndStanford* and *HobbsAndNeuralCoref*. Theoretically, the concrete classes of *CombinableCoreferenceResolver*, e.g., *HobbsAndStanford*, can be combined as well. However, that is unnecessary, since their abilities to resolve specific mentions

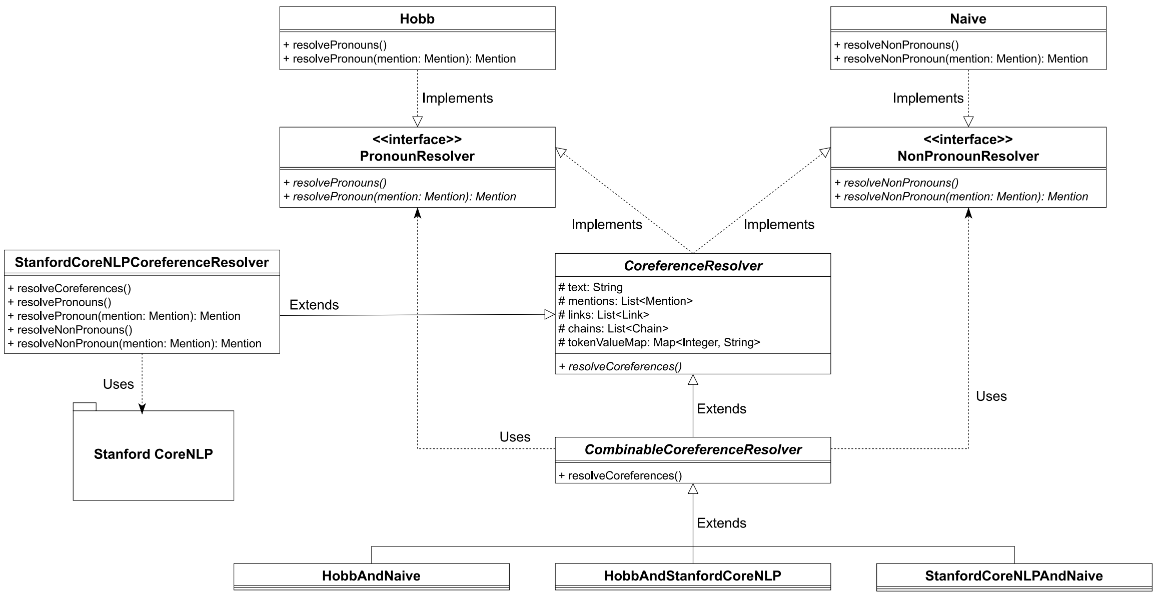

Figure 4.1.: Architecture of the Java project

are already provided by specific resolvers implementing interfaces *PronounResolver* and *NonPronounResolver*.

We have mentioned but not yet explained properties *mentions*, *links*, and *chains*. They are of classes *Mention*, *Link*, and *Chain* to model the respective terms. Hence, Section 4.3 will provide definition for important terms of the domain.

4.3. Data Definition

The domain of our Java project contains some terms that are important to the understanding and implementation of the project. In this section, the most important terms will be explained in detail.

Because coreference resolution is the task of finding mentions that corefer to other mentions, we therefore need to be able to model mentions. Before we can model a mention, we have to be able to model smaller units. These units are single words or punctuation marks in a text. In order to do model them, we can tokenize the text. Tokenization is a natural language processing (NLP) task supported by many NLP frameworks. The results are in-memory data models providing different properties of the units such as starting position, ending position, original text, etc. Our interest however is the IDs of the tokens. They are the indexes of appearance in the tokenized text. Below is a text, in which each token is assigned with its ID directly below.

```
One of the main components of Media Store is a server-side
0   1  2   3    4          5  6     7     8  9 10
web front end , namely the Facade component , which
11  12    13  14 15     16  17     18         19 20
delivers websites to the users and provides session
21       22       23 24  25    26  27       28
management . To meet the user authentication requirement ,
29         30 31 32   33  34   35             36          37
registration and log-in have to be offered .  To this end ,
38           39  40     41   42 43 44      45 46 47   48  49
the Facade component delivers the corresponding registration
50  51     52        53       54  55            56
and log-in pages to users .
57  58     59    60 61    62
```

A mention is a noun phrase that refers to an entity in a text. Entities can be identical, so some mentions can have identical string representation. Therefore, we need to find a way to model each of the mentions uniquely. One possible way is using the IDs of the first and the last tokens. For example, the mention *One of the main components* can be modeled with indexes 0 and 4. If a mention is a single word, the first and the last token are identical. For instance, the mention *websites* can be modeled with index 22.

Links are the results of coreference resolution. When two mentions are found to corefer to each other, we can create a link between them. That link connects from the mention that appears earlier to the mention that appears later in the text. To model a link, we can use the two mentions that the link is made of. An example of a link is the link between mention *the Facade component* (16 to 18) and *the Facade component* (50 to 52).

Another result of coreference resolution is clusters. Each mention relates to a cluster. Cluster is a set of mentions that the relating mention corefers with, including itself. For example, mention *the Facade component* (16 to 18) relates to the cluster containing itself and mention *the Facade component* (50 to 52).

Links and clusters are two different results of coreference resolution. The former emphasizes strict order of mentions while the latter does not. We need both of them for the evaluation later. Nonetheless, we do not have to return both of them to users as coreference resolution results. We can resort to another form of data called chains. Chains are sets of coreferent mentions like clusters. However, the mentions in a chain are sorted by the order of their appearance. That way chains can still model clusters and a link can be implicitly modeled by two consecutive mentions in the containing chain. As a result, we can return our results in a more memory-saving manner. Although our main result of coreference resolution is chains, links and clusters are still provided for the convenience of evaluation in Chapter 5. Figure 4.2 illustrates links, clusters and chains and how they are constructed from mentions. The chain combines the links and the cluster as one form of data.

Our main focus is still the coreference resolution Java libraries. In the next Section 4.4, we will look into the important details regarding the implementation of the libraries.

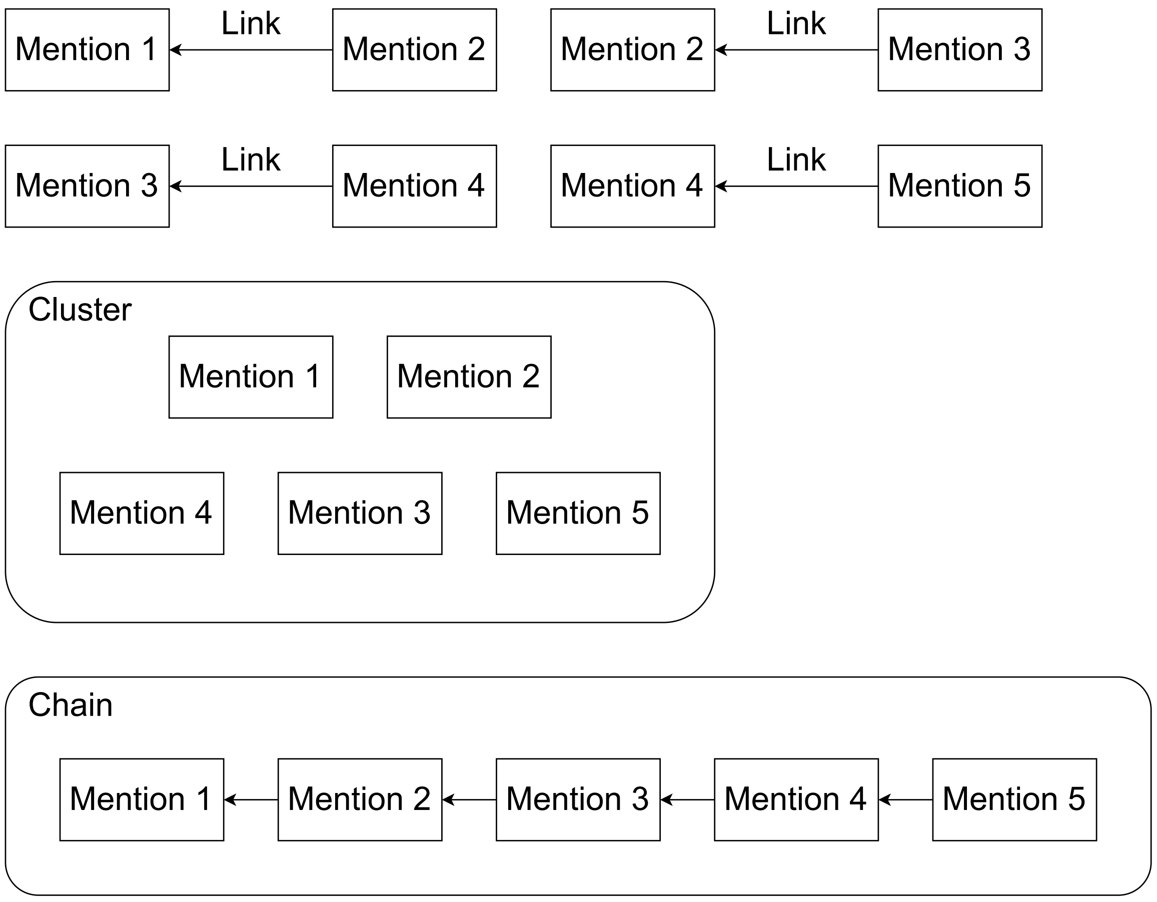

Figure 4.2.: Links, clusters, and chains as coreference resolution results

4.4. Implementation Details

In this section, we will take a look at details that are important to the implementation of the coreference resolution Java libraries. In our implementation, there are two types of libraries. They both will resolve coreferences by finding the coreferent mentions for each mention in the text. They will return the results as links, clusters and chains. Their differences are the fundamental way they are structured to resolve coreferences. The first type of libraries are those that directly extend the abstract class *CoreferenceResolver* to integrate coreference resolution models. They resolve coreferences with the help of the underlying coreference resolution models.

There are two concrete classes of the first type, *StanfordCoreNLPCoreferenceResolver* and *NeuralCoref*. These classes derive from the already existing coreference resolution models, StanfordCoreNLP and NeuralCoref 4.0. StanfordCoreNLP is a well-known natural language processing (NLP) framework [5]. It supports a lot of NLP tasks such as tokenization, sentence splitting, lemmatization, etc. Among those tasks is coreference resolution, hence the reason why we choose this framework. The way it works is that we have to specify a number of annotators to configure a pipeline. These annotators are required for the pipeline to perform specific NLP tasks. The pipeline then performs those NLP tasks on input documents. For coreference resolution, there are certain annotators that are required. Moreover, StanfordCoreNLP provides three different modes of coreference resolution, namely neural, statistical, and deterministic. Each of them may require different or additional annotators and options to configure the pipeline. The core idea behind the implementation of *StanfordCoreNLPCoreferenceResolver* is to set up the pipeline for coreference resolution, run the input text through the pipeline, retrieve and convert the results into our desired format, chains.

NeuralCoref 4.0 is an extension of spaCy, which is in turn a NLP framework. Furthermore, spaCy is developed for Python instead of Java like StanfordCoreNLP. This makes it difficult to directly integrate Neural Coref 4.0 to our Java project and use it. To use NeuralCoref 4.0 to resolve coreferences, we can start a remote server. This server will set up the pipeline for coreference resolution. The Java library *NeuralCoref* then makes a HTTP POST request containing the input text. The server upon receiving the request runs the attached input text through the pipeline and sends the results back as JSON data format. The Java library *NeuralCoref* then parses the JSON to reconstruct the chains.

The second type are those that indirectly extend *CoreferenceResolver* through *CombinableCoreferenceResolver*. They employ multiple specific resolvers to resolve coreferences. *CombinableCoreferenceResolver* delegates the resolution of specific types of mentions to specific resolvers. In our implementation, we have resolvers for pronouns and non-pronouns. For pronoun resolution, we use the Hobbs' algorithm. The approach for the libraries using Hobbs is first constituency-parsing all the sentences in the input text. Figure 4.3 illustrates an example of constituent tree created from a sentence. Each yellow tag represents a token and each green tag represents the grammatical role of one or a group of tokens.

Then, for each sentence we apply Hobbs on every pronoun contained in that sentence. The Hobbs' algorithm consists of nine steps which basically starts from the pronoun, traverses the constituent-tree derived from the sentence containing the pronoun, and proposes noun phrases (mentions) along the way. However, Hobbs' algorithm did not

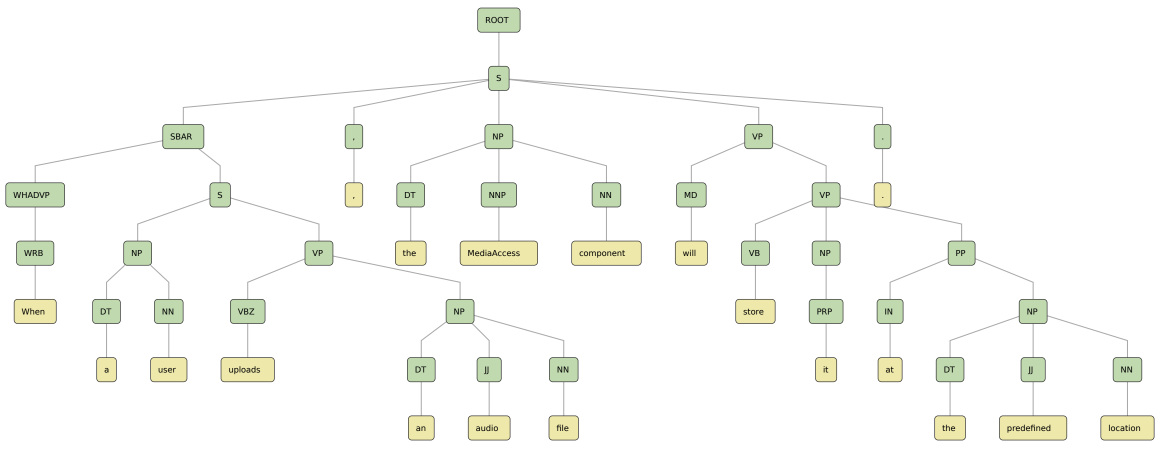

Figure 4.3.: Constituent tree from a sentence

mention which noun phrase to choose out of the proposed noun phrases. In the previous example 4.3, noun phrases *a user*, *an audio file*, and *the MediaAccess component* will be proposed by Hobbs' algorithm.

Hence, we can use a ranking system to score the proposed nodes and choose the nodes with the highest score. One of such ranking system is proposed by Lappin and Leass [33]. In this paper, salience factors are introduced to score the proposed nodes. For example, noun phrases that are head nouns will be given 80 points. Those that are not will be given 0. An example of head nouns is *a bag of candies* in *I have a bag of candies.* . A counterexample would be *candies* because it is contained in *a bag of candies*. Nonetheless, those factors are not enough to differentiate between plural or single nouns or nouns with different genders. Thus, we can extend that ranking system. We can do that by looking whether the pronoun and the proposed noun phrase have the same state of plurality or the same gender. For plurality checking, we can see the tags that Stanford CoreNLP gives the nouns after constituency-parsing. The tags will be NNS for plural nouns or NN for single nouns. For gender nouns, Stanford CoreNLP also provides gender identification. If the pronoun and the proposed noun phrase do not share the same state of plurality or gender, the noun phrase will be excluded from scoring and therefore from choosing.

For non-pronoun resolvers, we have two libraries, *Naive* and *Levenshtein*. *Naive* is implemented as a naive non-pronoun resolver. It checks for equality and links them if identical. However, there are also mentions that are not identical but still coreferent to each other. For this, *Levenshtein* uses Levenshtein distance to calculate the degree of similarities between two mentions. The calculated degree is called distance. If the distance is smaller than a predefined threshold, then we link the two mentions. We want to define our threshold in a way that it is not too strict to miss partly identical coreferent mentions and not too lenient to let partly identical but uncoreferent mentions through. In spite of that, there are mentions that bear little resemblance yet still corefer to each other. For example, *Teastore*, the name of an application, corefers to *the application*. Another example is *KIT* and *Karlsruher Institut für Technologie*. For such mentions, it would be more preferable to use resolvers that take the context of the input text into consideration or resolve the abbreviation. To that end, we can break down the coreference resolution into more than two specific resolution. We do that by creating more interfaces that *CoreferenceResolver* have to implement. The *CombinableCoreferenceResolver* is then responsible to delegate the resolution of mentions to the correct resolvers.

When the concrete pronoun and non-pronoun resolvers are at our disposal, we can create new coreference resolution libraries out of them. Classes extending *CombinableCoreferenceResolver* have to specify a concrete class that implements *PronounResolver* and another that implements *NonPronounResolver*. Concrete classes that directly extend *CoreferenceResolver*, e.g., *NeuralCoref*, can serve both as pronoun resolvers and non-pronoun resolvers. Therefore, they can also be used for combination. With one pronoun resolver, two non-pronoun resolvers, two coreference resolvers (Stanford CoreNLP has also 3 different modes) in our implementation, we can create multiple combination.

5. Evaluation

The goal of this thesis is to resolve coreferences in software architecture documentation. To achieve this goal, a number of approaches from Chapter 4 has been proposed. For the evaluation of the work of this thesis, we have to look to how the approaches perform based on three different sets of coreferences. The first set is all coreferences in the documentation. This will show us how the approaches perform for coreference resolution for software architecture documentation. Then, to see the influence of coreference resolution for specific activities that process the documentation, we use traceability link recovery (TLR) as an example. The second set is the coreferences, that must be resolved so that TLR can create tracelinks between related mentions and model elements in formal documentation. We can call these coreferences relevant coreferences. The last set is the coreferences resulted from pronouns. One of such examples is the first example in Section 2.3. These pronoun coreferences is one of the biggest factors that contribute to missing tracelinks. To summarize we have to answer the following three questions for coreference resolution for software architecture documentation:

1. How do the approaches perform for all coreferences ?
2. How do the approaches perform for relevant coreferences ?
3. How do the approaches perform for pronoun coreferences ?

First, Section 5.1 describes the process of creating gold standards. These gold standards are created from case studies and contain resources necessary for the evaluation. Next, Section 5.2 describes the metrics used to evaluate the results. These metrics help answering the above-mentioned questions in a measurable way. Then, Section 5.3 presents the evaluation results based on the gold standards and metrics that we have created and defined in two previous sections. The results will show how the approaches perform in regard to the questions. Last but not least, Section 5.4 lists the threats to the validity of the work of this thesis.

5.1. Creation Of Gold Standards

Gold standards are created from a number of case studies. These case studies are Mediastore [8], TEAMMATES [9], Teastore [10], BigBlueButton [11], CoronaWarnApp [12] and Docker [13]. The first three case studies are from the SoftWare Architecture Text Tracelink Recovery project [1]. The case studies are chosen because they are informal software architecture documentation of software projects. For example, case study Docker is an informal documentation describing the software Docker. It is mainly used to containerize software in packages. To create a gold standard, we begin with creating a key for each case study. A key is a set of all coreferences that we manually resolve. On the contrary, a response is a set of all coreferences that an approach resolved. Responses and keys can be used together with the metrics in Section 5.2 to calculate the evaluation results.

The coreference resolution results from the responses are in the forms of chains. Each chain in turn represent a cluster and related links. Mentions, links, clusters and chains are described in more details in Section 4.3. Links and clusters are the central figures in the evaluation models. These models will be explained in Section 5.2. Therefore, we should represent each key as a set of chains. To do this, we can form each key as JSON data format. A key will be an array of chains, a chain will be an array of mentions and a mention will be an array of exact two integers. The following is an example of a key.

```
[
    ...
    [ [6,7], [215, 216] ],
    [ [9, 13], [16, 18], [50, 52], [113, 115] ],
    ...
]
```

Regarding the manual coreference resolution, the above key may be suitable for the evaluation Java library, but not for humans. It is difficult to know which noun phrase mention *[215, 215]* represents without having to look to the tokenized case study. If we intend to continuously work on the keys, it is best to have another version of the keys, that are more understandable to us. An automatic process can create a simplified version of a key by providing for each mention its noun phrase, and the sentence to which it belongs. It is worth noting, that this simplified key is for continuously improving the result of manual coreference resolution only. To serve as input for the evaluation, the key in JSON is more preferable. Below is an excerpt of a simplified key.

```
[
    Media Store, 1, [6,7]
    Media Store, 13, [215, 216]
    ----
    a server-side web front end, 1, [9, 13]
    the Facade component, 1, [16, 18]
    the Facade component, 3, [50, 52]
    the Facade component, 6, [113, 115]
]
```

We also have to create relevant keys and pronoun keys. The chains in relevant keys are the chains in original keys. Nonetheless, those chains must represent mentions that TLR should link to the model elements in formal documentation. In other words, the chains of a relevant key are a subset of the chains of the original key. For the pronoun keys we must make some adjustments. Each mention will have another integer element in its JSON form to indicate whether it is a pronoun or not. If it is a pronoun, the integer will be 1, otherwise -1. Below is an example of a pronoun key.

Case Studies	Words	Links			Clusters	
		All	Relevant	Pronoun	All	Relevant
Mediastore	572	70	33	19	22	10
TEAMMATES	**2513**	**315**	**70**	**56**	**113**	12
Teastore	661	65	31	32	19	7
BigBlueButton	1190	136	62	24	41	**17**
CoronaWarnApp	1309	126	27	16	34	5
Docker	915	99	45	30	26	7

Table 5.1.: Case studies and the numbers of links and clusters in them

```
[
    ...
    [ [17, 17, -1], [137, 137, 1], [156, 156, -1]],
    [ [19, 20, -1], [143, 143, 1], [188, 190, -1]],
    ...
]
```

Table 5.1 shows the case studies used in the evaluation. The smallest is Mediastore with 572 words and the biggest is TEAMMATES with 2513 words. Moreover, the numbers of different types of links and clusters are also presented. For the all, relevant and pronoun links, TEAMMATES has the most. It also has the most all clusters. However, BigBlueButton has the most relevant clusters. The numbers of pronoun clusters are not shown because they are not used for the evaluation. The reason for this will be explained after the evaluation models have been introduced.

5.2. Metrics

The work of this thesis are evaluated by recall, precision and F1-score. This section will describe these metrics and how we can calculate them based on a key and a response. Moreover, the metrics depend on specific evaluation models, e.g., MUC and BCUBED. This section will also describe those evaluation models and explain why we need them.

The idea behind recall, precision and F1-score is based on the relevance of the elements that are retrieved and of those that are not retrieved. Table 5.2 assigns trueness and positivity to elements based on their relevance and whether they are retrieved or not . The elements in this thesis depend on which evaluation models we use. At the moment, we deem them as abstract. On one hand, retrieved elements are elements in the response. On the other hand, relevant elements are elements in the key. Correct elements are elements from both response and key. Therefore, we have recall and precision as follows.

Recall is a metric that describes the ratio between the number of correct elements and the number of all relevant elements. Its value is between 0 and 1.

$$\text{Recall} = \frac{\text{number of correct elements}}{\text{number of relevant elements}} = \frac{\text{true positives}}{\text{true positives + false negatives}}$$

Element	**Retrieved**	**Unretrieved**
Relevant	true positive	false negative
Irrelevant	false positive	true negative

Table 5.2.: Trueness and positivity of elements based on their relevance and whether they are retrieved or not

Precision is a metric that describes the ratio between the number of correct elements and the number of all retrieved elements. Its value is between 0 and 1.

$$\text{Precision} = \frac{\text{number of correct elements}}{\text{number of retrieved elements}} = \frac{\text{true positives}}{\text{true positives + false positives}}$$

To combine recall and precision, we use the harmonic mean of both of them. This metric is called F1-score.

$$\text{F1-score} = 2\frac{\text{recall . precision}}{\text{recall + precision}}$$

There are many existing models to evaluate coreference resolution, but the most used are MUC and BCUBED [37]. Therefore, we evaluate the results of the approaches with MUC and BCUBED. Each of them has its advantages and disadvantages and will be described as follows.

MUC stands for Message Understanding Conference [45]. It considers links as elements in the evaluation of coreference resolution. Links are provided by keys and responses. Retrieved links are links in the response and relevant links are links in the key. A correct link is both in a key and a response. Therefore, recall and precision can be calculated like the above-mentioned formulas.

This metric is simple to calculate, but unsuitable for singleton mentions. Singleton mentions are mentions that are mentioned only once [18]. MUC cannot evaluate the result of an input that has only singleton mentions, since there are no links created [35]. Another problem of MUC is that it considers every error to be the same [47]. That means every missing link affects recall in the same way. The same thing goes for every wrong link to precision. A link can wrongly connect two large chains and still affect precision exactly like a link wrongly connecting two small chains. The following example illustrates the mentioned shortcomings of MUC.

Figure 5.1 illustrates a correct answer of coreference resolution. We have chains that represent entities. Note that entities are different from mentions. A mention refers to one entity. One entity can be referred by many mentions. Mentions are denoted with 1 to 6, x to z, and A to F to refer to three different entities. In this answer we have 12 links.

Figure 5.2 illustrates a possible response of coreference resolution. We have 13 links, 12 links correct and 1 link wrong. First, we have recall of 100% because, all relevant links are retrieved. Then, we have precision of 92.3%, because 12 links over 13 retrieved links are correct. Finally, F1-score is 96% resulted from the calculated recall and precision.

Figure 5.3 illustrates another response of coreference resolution libraries. This response is clearly worse than the first response, because it wrongly connects two large chains of coreferences. However, it has the same recall, precision and F1-score, because it has the same number of wrong and correct links as the previous response.

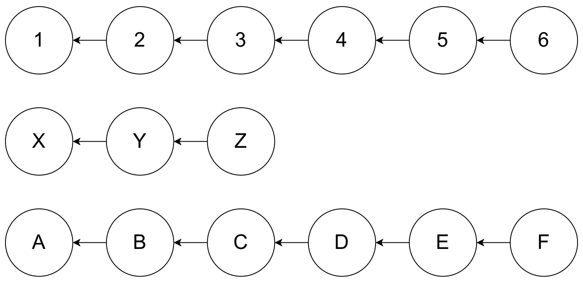

Figure 5.1.: The correct answer of coreference resolution

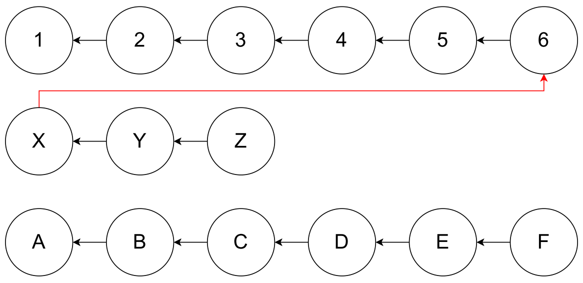

Figure 5.2.: A possible response from coreference resolution libraries

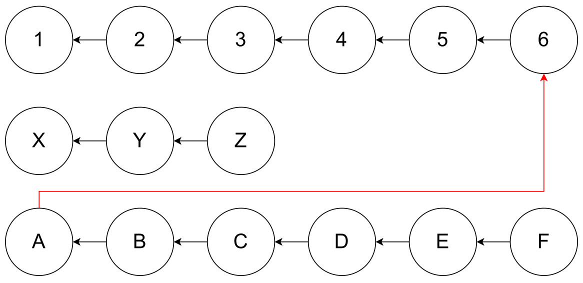

Figure 5.3.: Another possible response from coreference resolution libraries

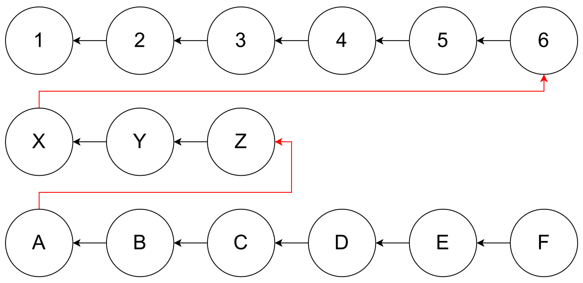

Figure 5.4.: A response of coreference resolution where all mentions are in a cluster

BCUBED is proposed to overcome the shortcomings of MUC. There are two things that BCUBED are different from MUC. First, it does not evaluate the overall result but rather the result based on each mention and then sum them up. That means, for each mention it will calculate the recall and precision separately and add each of them to the overall recall and precision respectively. Second, it does not consider links but instead mentions. For the calculation of recall and precision of each mention, mentions that corefer with this mention are the elements. The set of mentions from the key is called key cluster, and the set of mentions from the response is called response cluster. The correct mentions are in the intersection between the key and response cluster. Recall and precision are calculated by dividing the number of correct mentions to the number of mentions in key cluster and response cluster respectively. The following shows how recall and precision of a mention can be calculated.

$$\text{Recall} = \frac{\text{number of correct mentions}}{\text{number of mentions in key cluster}} = \frac{\text{true positives}}{\text{true positives + false negatives}}$$

$$\text{Precision} = \frac{\text{number of correct mentions}}{\text{number of mentions in response cluster}} = \frac{\text{true positives}}{\text{true positives + false positives}}$$

Though improved compared to MUC, BCUBED can be counter-intuitive when the response has only one cluster and all mentions are in it. This results in 100% recall even though the response cluster is not a subset of the correct answer. Figure 5.4 illustrates one of such cases.

For each mention of entity 1 we have recall of 100% (all six elements are in the cluster) and precision of 40% (6 out 15). With the same rules of calculation, recall and precision for each mention of entity 2 are 100% and 20%, for entity 3 100% and 40%. Overall we have recall of 100%. This is counter-intuitive because we have three different entities in the key. However, in the response we have only one. One may argue that we have all mentions in the cluster so recall of 100% is reasonable. Nonetheless, the evaluation of the overall coreference resolution does not consider mentions as central elements but rather the evaluation of each mention does. The overall recall or precision is calculated by summing up recall or precision of each individual mention.

For the evaluation of the approaches as pronoun resolution models, we can constraint the links between mentions to pronoun links. Each of these links has at least one pronoun. Thus we can use MUC to evaluate this aspect of the approaches. However, BCUBED is not intended for pronoun resolution evaluation. Thus, the pronoun clusters are not needed as stated in the previous section.

Case Studies	Definition
CS1	Mediastore
CS2	TEAMMATES
CS3	Teastore
CS4	BigBlueButton
CS5	Corona Warn App
CS6	Docker

Table 5.3.: Case studies used in evaluation

Approaches	Definition
A1	StanfordCoreNLP_Neural
A2	StanfordCoreNLP_Statistical
A3	StanfordCoreNLP_Deterministic
A4	NeuralCoref
A5	Hobbs+Naive
A6	Hobbs+Levenshtein
A7	Hobbs+StanfordCoreNLP
A8	Hobbs+NeuralCoref
A9	StanfordCoreNLP+NeuralCoref
A10	NeuralCoref+StanfordCoreNLP

Table 5.4.: Approachess used in evaluation

In Section 3.1, it was mentioned that beside MUC and BCUBED, $CEAF_{\phi}4$ evaluation model is also used to calculate the average metrics. By BCUBED, repeated key mentions in the response can lead to recall larger than one [36]. CEAF or $CEAF_{\phi}4$ are proposed to tackle this problem. However, in the implementation of the approaches we prevented key mentions from repeating in the response. Hence, $CEAF_{\phi}4$ is not needed.

5.3. Evaluation Results

Before we dive into the evaluation results, Table 5.3 and Table 5.4 will present the case studies and approaches that are used in the evaluation. The first three approaches are different modes of coreference resolution from Stanford CoreNLP. They are followed by NeuralCoref, an approach from Neural Coref 4.0. Then six different approaches from different combination between pronoun resolution, non-pronoun resolution and coreference resolution will be presented. The two operands between the plus symbol represent the two resolution. The left operand is for pronoun resolution, while the second is for non-pronoun resolution.

First, Table 5.5 presents the result of coreference resolution for software architecture documentation. Each of the results is evaluated based on MUC and BCUBED models. However, the table shows only the average values of metrics from the two models. The full results can be found in the appendix. Generally speaking, the values of the metrics from MUC are lower than those of BCUBED. That is to be expected, because BCUBED is more lenient than MUC in a sense that a mention only has to be in a coreference chain with other mentions to be considered coreferencing with them. Whereas MUC demands strict order of appearance, i.e, a mention in a chain can only corefer with mentions directly behind or after it. Hobbs+Naive is the approach with the highest average precision (0.72) and F1-score (0.63). Besides, StanforeCoreNLP_Deterministic has the second highest F1-score (0.59) but the highest recall (0.61). Hobbs+Levenshtein has the same F1-score as StanforeCoreNLP_Deterministic, higher precision (0.63 to 0.57) but lower recall (0.57 to 0.61). It is surprising that StanforeCoreNLP_Deterministic outperformed the other two modes, even though the authors claim that neural and statistical mode are more performant than the deterministic one. Furthermore, NeuralCoref has the lowest average F1-score (0.42). If a case study has more than 50 lines of text, the approach performs poorly. For example, NeuralCoref only has an average F1-score of 0.33 with case study TEAMMATES (198 lines). Therefore, combinable approaches that have NeuralCoref in their combination have average F1-score less or equal than 0.50. For coreference resolution for software architecture documentation, Hobbs+Naive performs best among the approaches.

Next, we look at how the approaches perform when they only concern mentions that are important to traceability link recovery (TLR). These mentions are mentioned in the formal software architecture documentation of the case studies. Table 5.6 displays the results of the evaluation. The metrics are the average results of MUC and BCUBED metrics. The full results can be found in the appendix. StanforeCoreNLP_Deterministic surpasses other approaches with the highest average F1-score (0.63). Other modes of Stanford CoreNLP along with Hobbs+Naive all have the second highest average F1-score (0.59). Most notably, Hobbs+Naive has the highest precision (0.82). However, its recall is quite low (0.49). This can be due to the fact that it uses a naive approach to resolve non-pronouns. Therefore only identical non-pronoun mentions can be resolved, hence the high precision and low recall. Similar to previous evaluation, NeuralCoref and approaches that use it have poor results with long case studies. Thus, to resolve coreferences for TLR, StanforeCoreNLP_Deterministic is most preferable among the approaches.

Lastly, there are the results of pronoun resolution from the approaches. The metrics are based on MUC evaluation models. In this evaluation, combinable approaches using Hobbs to resolve pronouns all have the highest average F1-score (0.74). They have the same recall (0.74) and precision (0.74). This is understandable since they all use Hobbs' algorithm to resolve pronouns. Moreover, neural and statistical modes all beat deterministic mode in this evaluation (average F1-scores of 0.71 and 0.68 to 0.47). NeuralCoref and the approaches using it to resolve pronouns have the lowest F1-scores (0.20 for A4 and A10). Among the approaches, approaches using Hobbs' algorithm are most suitable when it comes to pronoun resolution.

5.4. Threats To Validity

There are some threats to the validity of the evaluation of the approaches. First, there is reliability. The manual coreference resolution was done only by the writer of this thesis

Case Studies	CS1			CS2			CS3			CS4			CS5			CS6			A		
Libraries	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F
A1	.57	.50	.53	.47	.58	.52	.49	.59	.54	.58	.61	.59	.56	.56	.56	.48	.51	.49	.53	.56	.54
A2	.57	.54	.55	.46	.60	.52	.50	.61	.55	.57	.61	.59	.54	.58	.56	.46	.52	.49	.52	.58	.55
A3	**.74**	.69	.71	.61	.55	.58	.54	.58	.56	**.63**	.59	.61	**.61**	.53	**.57**	**.52**	.47	.49	**.61**	.57	.59
A4	.47	.53	.49	.28	.41	.33	.44	.52	.48	.35	.47	.40	.40	.51	.45	.30	.34	.32	.38	.47	.42
A5	.68	**.81**	**.74**	**.63**	.69	**.65**	**.59**	**.79**	**.67**	.56	**.75**	**.64**	.48	.63	.54	.48	.63	**.54**	.57	**.72**	**.63**
A6	.68	.69	.68	**.63**	.62	.62	.57	.69	.62	.56	.64	.59	.49	.54	.51	.48	.55	.50	.57	.63	.59
A7	.49	.62	.55	.42	.64	.51	.49	.68	.57	.50	.69	.58	.46	.63	.53	.41	.58	.48	.47	.64	.54
A8	.42	.62	.50	.25	.67	.35	.42	.69	.52	.32	.74	.43	.35	.71	.47	.28	**.70**	.40	.34	.69	.45
A9	.43	.62	.50	.26	**.70**	.36	.37	.62	.46	.32	.73	.44	.36	**.75**	.48	.26	.65	.36	.34	.68	.44
A10	.47	.61	.53	.38	.64	.48	.44	.67	.53	.45	.70	.55	.43	.63	.51	.32	.54	.40	.42	.64	.50

Table 5.5.: Results of coreference resolution for software architecture documentation from different approaches

Case Studies	CS1			CS2			CS3			CS4			CS5			CS6			A		
Libraries	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F
A1	.69	.59	.63	.51	.64	.57	.48	.61	.53	**.58**	.62	.60	**.70**	.74	**.72**	**.40**	.57	**.47**	.56	.63	.59
A2	.69	.59	.63	.49	.65	.56	.48	.64	.55	**.58**	.62	.60	.67	.76	.71	.38	.56	.45	.55	.64	.59
A3	**.80**	**.94**	**.86**	**.67**	.72	**.70**	.44	.69	.54	.54	.55	.55	.63	.73	.68	.35	.50	.41	**.58**	.69	**.63**
A4	.56	.49	.52	.13	.52	.21	.46	.68	.55	.23	.68	.33	.51	.78	.62	.10	.44	.16	.34	.60	.40
A5	.68	.93	.78	.45	**.78**	.56	**.52**	**.84**	**.64**	.52	**.85**	**.64**	.41	.80	.54	.35	.70	.44	.49	**.82**	.59
A6	.68	.93	.78	.43	.71	.52	.51	.77	.61	.48	.75	.58	.34	.67	.45	.35	.64	.43	.47	.75	.57
A7	.58	.72	.64	.39	.70	.50	.50	.76	.60	.49	.73	.58	.52	.80	.63	.28	.56	.36	.46	.72	.56
A8	.50	.65	.57	.17	.69	.27	.43	.69	.53	.26	.79	.38	.43	**.89**	.58	.15	.73	.24	.33	.74	.43
A9	.51	.64	.57	.19	.75	.30	.34	.57	.43	.25	.76	.37	.43	**.89**	.58	.15	**.81**	.24	.32	.74	.42
A10	.56	.69	.62	.37	.66	.48	.49	.77	.60	.46	.73	.57	.52	.80	.63	.24	.54	.32	.44	.70	.54

Table 5.6.: Coreference resolution results for relevant mentions from different approaches

Case Studies	CS1			CS2			CS3			CS4			CS5			CS6			A		
Libraries	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F
L1	.60	.60	.60	**.67**	**.80**	**.73**	.67	.67	.67	.56	**.71**	**.63**	**1.0**	**1.0**	**1.0**	.58	.64	.61	.68	.74	.71
L2	.60	.75	.67	.61	.79	.69	.67	.75	.71	.44	.67	.53	.75	**1.0**	.86	.58	.64	.61	.61	**.77**	.68
L3	**1.0**	**1.0**	**1.0**	.39	.50	.44	.33	.43	.38	.22	.40	.29	.25	.25	.25	.42	.45	.43	.44	.51	.47
L4	.40	.40	.40	.00	.00	.00	.67	.86	.75	.00	.00	.00	.00	.00	.00	.00	.00	.00	.18	.21	.20
L5	.60	.60	.60	**.67**	.63	.65	**1.0**	**1.0**	**1.0**	**.67**	.60	**.63**	.75	.75	.75	**.75**	**.82**	**.78**	**.74**	.74	**.74**
L6	.60	.60	.60	**.67**	.63	.65	**1.0**	**1.0**	**1.0**	**.67**	.60	**.63**	.75	.75	.75	**.75**	**.82**	**.78**	**.74**	.74	**.74**
L7	.60	.60	.60	**.67**	.63	.65	**1.0**	**1.0**	**1.0**	**.67**	.60	**.63**	.75	.75	.75	**.75**	**.82**	**.78**	**.74**	.74	**.74**
L8	.60	.60	.60	**.67**	.63	.65	**1.0**	**1.0**	**1.0**	**.67**	.60	**.63**	.75	.75	.75	**.75**	**.82**	**.78**	**.74**	.74	**.74**
L9	.60	.60	.60	**.67**	**.80**	**.73**	.67	.67	.67	.56	**.71**	**.63**	**1.0**	**1.0**	**1.0**	.58	.64	.61	.68	.74	.71
L10	.40	.40	.40	.00	.00	.00	.67	.86	.75	.00	.00	.00	.00	.00	.00	.00	.00	.00	.18	.21	.20

Table 5.7.: Coreference resolution results for pronoun mentions from different approaches

and the resolution can be subjective. Different researchers can come up with different resolution results. Take the following for example:

> To this end, the requested files are first reencoded. The re-encoded files are then digitally and individually watermarked by the TagWatermarking component.

One can link *The re-encoded files* to *the requested files* simply by taking the context into consideration. However, another may argue that the files due to re-encoding are completely different from each other.

Second, half of the case studies (3 out of 6) are the case studies used in SoftWare Architecture Text Tracelink Recovery (SWATTR). We want to see how coreference resolution contribute to activities, that process software architecture documentation. SWATTR is a project for traceability link recovery. This can contribute to selection bias, because other activities can process different kinds of case studies. Selection bias is a threat to internal validity. Thirdly, we only looked in the case of TLR but not other activities. Hence, a generalization of the findings of this study may not be entirely correct. For example, Klym Shumaiev and Manoj Bha propose an activity that uses natural language processing techniques to detect uncertainty cues in software architecture documentation [43]. The work mainly looks for words like *would*, *could*, *most likely* in an input text. It is not entirely sure how coreference resolution can contribute to the performance of this work. This is an ecological validity and a threat to external validity.

6. Conclusion And Future Work

In natural language software architecture documentation, coreferences can occur. These coreferences can be problematic for some activities that process the documentation. This thesis focused on resolving these coreferences. Therefore, it proposed a number of approaches for coreference resolution.

Beside conventional approaches, this thesis proposed a different type to resolve coreferences. Instead of solely using already implemented models or algorithms to resolve coreferences, we broke the problem of coreference resolution down to two specific resolutions, pronoun and non-pronoun resolution. In other words, we looked into the types of the mentions that are to be resolved. Then, we used different models or algorithms to resolve different types of mentions. This theoretically allows for higher level of maintainability, customization and optimization. This thesis also provided a software architecture to implement the approaches. The core idea behind was that we made abstraction of coreference resolution, and other specific types of resolutions. As a result, we can combine many types of resolutions to resolve coreferences.

The evaluation of the approaches was made in three different aspects. For coreference resolution for software architecture documentation, the standard Hobbs' algorithm combined with a naive approach to resolve non-pronouns outperformed other approaches. When it came to coreference resolution for a specific activity, the deterministic mode of Stanford CoreNLP surpassed others. For pronoun resolution, Hobbs' algorithm combined with any other non-pronoun resolution yielded high and consistent results.

With that said, there is still room for improvement. The coreference resolution for software architecture documentation results can still be improved by using better coreference resolution models. Moreover, the biggest issue in the proposed combination approaches was that they required all resolutions to have resolved coreferences beforehand. Consequently, each of the resolvers already have answers for all of its responsible mentions. When requested to resolve a mention, a resolver only has to look for the answers it has made. For instance, Hobbs' algorithm and naive non-pronoun resolver must process an input text in advance to provide answers for Hobbs+Naive. This can result into large calculation time and effort, when the number of types of resolvers increases. It was found out during inspecting the results of the approaches that nonsensical results considerably reduce the performance. For example. if we have two phrases of *In this situation* in a text, most approaches will link the two *this situation*, despite they do not corefer to each other. In this case, we might need a kind of "anti-resolution". This resolution will ignore mentions in idioms, common phrases or mentions that are deemed unnecessary for resolution by users. With the high level of extendibility of the architecture, it should be uncomplicated to integrate new resolution and also interesting to see how they play out.

The case studies used for evaluation can also be expanded to avoid selection bias. At the moment, half of the case studies are also used by the SoftWare Architecture Text Tracelink

Recovery project. These may not cover the case studies of other activities. Moreover, the manual coreference resolution result may need to be checked by the writers of the case studies for the sake of correctness. This will increase the reliability of the evaluation. Furthermore, we should look for other activities beside TLR, that also process software architecture documentation. This will provide a more general statement on the influence of coreference resolution for software architecture documentation.

Bibliography

[1] URL: https://github.com/ArDoCo/SWATTR.

[2] URL: https://nlp.stanford.edu/projects/coref.shtml.

[3] URL: https://opennlp.apache.org/.

[4] URL: http://www.alias-i.com/lingpipe/.

[5] URL: https://stanfordnlp.github.io/CoreNLP/.

[6] URL: https://spacy.io/.

[7] URL: https://github.com/huggingface/neuralcoref.

[8] URL: https://github.com/ArDoCo/SWATTR/blob/main/case_studies/mediastore/text/Palladio_MediaStore_Text.txt.

[9] URL: https://github.com/ArDoCo/SWATTR/blob/main/case_studies/teammates/text/TeammatesForEvalText.txt.

[10] URL: https://github.com/ArDoCo/SWATTR/blob/main/case_studies/teastore/text/EcsaText.txt.

[11] URL: https://github.com/ArDoCo/Benchmark/blob/main/bigbluebutton/bigbluebutton.txt.

[12] URL: https://github.com/corona-warn-app/cwa-server/blob/main/docs/ARCHITECTURE.md.

[13] URL: https://delftswa.github.io/chapters/docker/.

[14] A Arunthavanathan et al. "Support for traceability management of software artefacts using Natural Language Processing". In: *2016 Moratuwa Engineering Research Conference (MERCon)*. IEEE. 2016, pp. 18–23.

[15] Felix Bachmann et al. *Software architecture documentation in practice: Documenting architectural layers*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2000.

[16] Eric Bengtson and Dan Roth. "Understanding the value of features for coreference resolution". In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. 2008, pp. 294–303.

[17] Grady Booch. *The unified modeling language user guide*. Pearson Education India, 2005.

[18] Jie Cai and Michael Strube. "Evaluation metrics for end-to-end coreference resolution systems". In: *Proceedings of the SIGDIAL 2010 Conference*. 2010, pp. 28–36.

[19] Sylvain Chardigny and Abdelhak Seriai. “Software architecture recovery process based on object-oriented source code and documentation”. In: *European Conference on Software Architecture*. Springer. 2010, pp. 409–416.

[20] Paul Clements et al. “Documenting software architectures: views and beyond”. In: *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE. 2003, pp. 740–741.

[21] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[22] Vladimir Dobrovolskii. “Word-Level Coreference Resolution”. In: *arXiv preprint arXiv:2109.04127* (2021).

[23] David Garlan. “Software architecture”. In: (2008).

[24] Mandar Joshi et al. “BERT for coreference resolution: Baselines and analysis”. In: *arXiv preprint arXiv:1908.09091* (2019).

[25] Mandar Joshi et al. “Spanbert: Improving pre-training by representing and predicting spans”. In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 64–77.

[26] Daniel Jurafsky and James H Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.*

[27] Ben Kantor and Amir Globerson. “Coreference resolution with entity equalization”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.* 2019, pp. 673–677.

[28] Jan Keim and Anne Koziolek. “Towards consistency checking between software architecture and informal documentation”. In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C).* IEEE. 2019, pp. 250–253.

[29] Jan Keim et al. “Trace Link Recovery for Software Architecture Documentation”. In: *European Conference on Software Architecture.* Springer. 2021, pp. 101–116.

[30] Yuval Kirstain, Ori Ram, and Omer Levy. “Coreference Resolution without Span Representations”. In: *arXiv preprint arXiv:2101.00434* (2021).

[31] Jóakim von Kistowski et al. “TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research”. In: *Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems.* MASCOTS ’18. Milwaukee, WI, USA, Sept. 2018.

[32] Raúl Lapena. “Traceability Links Recovery in BPMN Models.” In: *CAiSE (Doctoral Consortium).* 2019, pp. 52–59.

[33] Shalom Lappin and Herbert J Leass. “An algorithm for pronominal anaphora resolution”. In: *Computational linguistics* 20.4 (1994), pp. 535–561.

[34] Heeyoung Lee et al. “Stanford’s multi-pass sieve coreference resolution system at the conll-2011 shared task”. In: *Proceedings of the fifteenth conference on computational natural language learning: Shared task.* 2011, pp. 28–34.

[35] Xiaoqiang Luo. “On coreference resolution performance metrics”. In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. 2005, pp. 25–32.

[36] Xiaoqiang Luo and Sameer Pradhan. “Evaluation metrics”. In: *Anaphora Resolution*. Springer, 2016, pp. 141–163.

[37] Lluís Màrquez, Marta Recasens, and Emili Sapena. “Coreference resolution: an empirical study based on SemEval-2010 shared Task 1”. In: *Language resources and evaluation* 47.3 (2013), pp. 661–694.

[38] Chris Mills. “Automating traceability link recovery through classification”. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 2017, pp. 1068–1070.

[39] Karthik Raghunathan et al. “A multi-pass sieve for coreference resolution”. In: *Proceedings of the 2010 conference on empirical methods in natural language processing*. 2010, pp. 492–501.

[40] Ralf H Reussner et al. *Modeling and simulating software architectures: The Palladio approach*. MIT Press, 2016.

[41] Aaron Schlutter and Andreas Vogelsang. “Improving Trace Link Recovery using Semantic Relation Graphs and Spreading Activation”. In: (2021).

[42] Sophie Schulz. *Linking So ware Architecture Documentation and Models*. 2020.

[43] Klym Shumaiev and Manoj Bhat. “Automatic uncertainty detection in software architecture documentation”. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE. 2017, pp. 216–219.

[44] Moon Ting Su. “Capturing exploration to improve software architecture documentation”. In: *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. 2010, pp. 17–21.

[45] Marc Vilain et al. “A model-theoretic coreference scoring scheme”. In: *Sixth Message Understanding Conference (MUC-6): Proceedings of a Conference Held in Columbia, Maryland, November 6-8, 1995*. 1995.

[46] Liyan Xu and Jinho D Choi. “Revealing the myth of higher-order inference in coreference resolution”. In: *arXiv preprint arXiv:2009.12013* (2020).

[47] Yimeng Zhang and Yangbo Zhu. “Machine Learning for Coreference Resolution: Recent Developments”. In: ().

[48] Yonggang Zhang et al. “Ontological approach for the semantic recovery of traceability links between software artefacts”. In: *IET software* 2.3 (2008), pp. 185–203.

A. Appendix

A.1. Tables

Case Studies		CS1			CS2			CS3			CS4			CS5			CS6			A		
Libraries	Models	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F
L1	M	.38	.39	.38	.27	.40	.32	.33	.43	.37	.40	.44	.42	.39	.40	.40	.30	.30	.30	.35	.40	.37
	B	.76	.61	.68	.67	.76	.71	.66	.74	.70	.76	.77	.76	.73	.72	.73	.65	.71	.68	.71	.72	.71
	A	.57	.50	.53	.47	.58	.52	.49	.59	.54	.58	.61	.59	.56	.56	.56	.48	.51	.49	.53	.56	.54
L2	M	.38	.42	.40	.26	.41	.32	.33	.45	.38	.39	.45	.42	.37	.41	.39	.29	.31	.30	.34	.41	.37
	B	.76	.66	.70	.65	.78	.71	.67	.76	.71	.74	.77	.76	.72	.74	.73	.63	.73	.68	.70	.74	.72
	A	.57	.54	.55	.46	.60	.52	.50	.61	.55	.57	.61	.59	.54	.58	.56	.46	.52	.49	.52	.58	.55
L3	M	.60	.54	.57	.41	.32	.36	.33	.35	.34	.43	.37	.40	.40	.30	.35	.27	.20	.23	.41	.35	.38
	B	.87	.84	.85	.82	.78	.80	.76	.82	.79	.83	.81	.82	.82	.76	.79	.76	.74	.75	.81	.80	.80
	A	.74	.69	.71	.61	.55	.58	.54	.58	.56	.63	.59	.61	.61	.53	.57	.52	.47	.49	.61	.57	.59
L4	M	.27	.39	.32	.00	.01	.01	.22	.29	.25	.08	.14	.11	.20	.28	.23	.00	.00	.00	.13	.19	.16
	B	.67	.66	.67	.56	.81	.66	.65	.76	.70	.62	.79	.70	.60	.75	.66	.60	.67	.63	.62	.74	.67
	A	.47	.53	.49	.28	.41	.33	.44	.52	.48	.35	.47	.40	.40	.51	.45	.30	.34	.32	.38	.47	.42
L5	M	.67	.71	.69	.58	.52	.55	.54	.66	.60	.52	.61	.56	.37	.41	.39	.44	.45	.44	.52	.56	.54
	B	.69	.91	.78	.68	.86	.76	.63	.91	.75	.61	.88	.72	.58	.84	.69	.52	.81	.63	.62	.87	.73
	A	.68	.81	.74	.63	.69	.65	.59	.79	.67	.56	.75	.64	.48	.63	.54	.48	.63	.54	.57	.72	.63
L6	M	.67	.54	.60	.57	.42	.49	.52	.52	.52	.49	.47	.48	.36	.30	.33	.42	.34	.38	.51	.44	.47
	B	.70	.84	.76	.69	.82	.75	.62	.85	.72	.62	.82	.71	.61	.78	.69	.54	.75	.63	.63	.81	.71
	A	.68	.69	.68	.63	.62	.62	.57	.69	.62	.56	.64	.59	.49	.54	.51	.48	.55	.50	.57	.63	.59
L7	M	.38	.44	.40	.27	.42	.33	.39	.51	.44	.41	.54	.47	.37	.45	.40	.33	.38	.35	.36	.46	.40
	B	.60	.81	.69	.57	.85	.68	.58	.84	.69	.58	.84	.69	.55	.81	.65	.50	.77	.61	.57	.83	.67
	A	.49	.62	.55	.42	.64	.51	.49	.68	.57	.50	.69	.58	.46	.63	.53	.41	.58	.48	.47	.64	.54
L8	M	.29	.44	.35	.06	.39	.11	.28	.52	.37	.15	.56	.23	.23	.54	.32	.12	.47	.20	.19	.49	.27
	B	.55	.80	.65	.44	.94	.60	.55	.87	.67	.48	.93	.64	.48	.89	.62	.44	.92	.60	.49	.90	.63
	A	.42	.62	.50	.25	.67	.35	.42	.69	.52	.32	.74	.43	.35	.71	.47	.28	.70	.40	.34	.69	.45
L9	M	.29	.45	.35	.06	.46	.11	.22	.42	.29	.14	.52	.22	.24	.59	.34	.10	.41	.16	.18	.48	.25
	B	.56	.79	.65	.45	.95	.61	.52	.82	.63	.50	.93	.65	.48	.90	.63	.42	.89	.57	.50	.88	.63
	A	.43	.62	.50	.26	.70	.36	.37	.62	.46	.32	.73	.44	.36	.75	.48	.26	.65	.36	.34	.68	.44
L10	M	.35	.43	.39	.21	.41	.28	.33	.50	.39	.35	.54	.42	.34	.45	.39	.21	.31	.25	.30	.44	.36
	B	.59	.80	.68	.55	.86	.67	.55	.84	.67	.56	.85	.68	.52	.81	.63	.44	.78	.56	.54	.83	.65
	A	.47	.61	.53	.38	.64	.48	.44	.67	.53	.45	.70	.55	.43	.63	.51	.32	.54	.40	.42	.64	.50

Table A.1.: Coreference resolution results from different approaches

Case Studies		CS1			CS2			CS3			CS4			CS5			CS6			A		
Libraries	Models	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F
L1	M	.57	.59	.58	.41	.57	.48	.38	.50	.43	.42	.46	.44	.64	.67	.65	.32	.35	.33	.46	.53	.49
	B	.81	.58	.68	.60	.71	.65	.58	.71	.64	.73	.77	.75	.77	.82	.79	.49	.78	.60	.67	.73	.69
	A	.69	.59	.63	.51	.64	.57	.48	.61	.53	.58	.62	.60	.70	.74	.72	.40	.57	.47	.56	.63	.59
L2	M	.57	.59	.58	.40	.57	.47	.38	.53	.44	.42	.46	.44	.59	.68	.63	.29	.34	.31	.45	.53	.48
	B	.81	.58	.68	.59	.73	.65	.59	.74	.66	.73	.77	.75	.75	.84	.79	.47	.79	.59	.66	.75	.69
	A	.69	.59	.63	.49	.65	.56	.48	.64	.55	.58	.62	.60	.67	.76	.71	.38	.56	.45	.55	.64	.59
L3	M	.74	.89	.81	.60	.60	.60	.33	.53	.41	.38	.36	.37	.50	.55	.52	.24	.26	.25	.47	.54	.50
	B	.86	.98	.91	.74	.85	.79	.55	.84	.66	.70	.74	.72	.77	.91	.83	.47	.74	.57	.69	.85	.75
	A	.80	.94	.86	.67	.72	.70	.44	.69	.54	.54	.55	.55	.63	.73	.68	.35	.50	.41	.58	.69	.63
L4	M	.39	.43	.41	.02	.13	.03	.33	.53	.41	.07	.43	.12	.45	.71	.56	.00	.00	.00	.21	.38	.26
	B	.74	.55	.63	.25	.91	.39	.58	.82	.68	.39	.94	.55	.56	.85	.68	.19	.89	.32	.46	.78	.55
	A	.56	.49	.52	.13	.52	.21	.46	.68	.55	.23	.68	.33	.51	.78	.62	.10	.44	.16	.34	.60	.40
L5	M	.70	.89	.78	.52	.68	.59	.50	.75	.60	.51	.77	.61	.41	.69	.51	.45	.57	.50	.52	.73	.60
	B	.66	.96	.79	.38	.87	.53	.55	.92	.69	.53	.93	.68	.41	.90	.56	.25	.83	.39	.47	.91	.61
	A	.68	.93	.78	.45	.78	.56	.52	.84	.64	.52	.85	.64	.41	.80	.54	.35	.70	.44	.49	.82	.59
L6	M	.70	.89	.78	.48	.58	.53	.50	.67	.57	.47	.62	.53	.32	.50	.39	.45	.50	.47	.49	.63	.55
	B	.66	.96	.79	.37	.83	.52	.52	.88	.65	.50	.87	.63	.37	.84	.51	.26	.79	.39	.45	.87	.59
	A	.68	.93	.78	.43	.71	.52	.51	.77	.61	.48	.75	.58	.34	.67	.45	.35	.64	.43	.47	.75	.57
L7	M	.57	.62	.59	.40	.57	.47	.50	.67	.57	.44	.61	.51	.59	.72	.65	.32	.39	.35	.47	.60	.53
	B	.60	.83	.70	.39	.82	.53	.51	.85	.64	.54	.85	.66	.45	.89	.60	.24	.74	.37	.46	.83	.59
	A	.58	.72	.64	.39	.70	.50	.50	.76	.60	.49	.73	.58	.52	.80	.63	.28	.56	.36	.46	.72	.56
L8	M	.43	.53	.48	.07	.44	.12	.38	.56	.45	.11	.63	.19	.45	.83	.59	.05	.50	.10	.25	.59	.33
	B	.57	.78	.66	.26	.94	.41	.49	.82	.61	.40	.95	.56	.42	.95	.58	.24	.97	.39	.40	.91	.54
	A	.50	.65	.57	.17	.69	.27	.43	.69	.53	.26	.79	.38	.43	.89	.58	.15	.73	.24	.33	.74	.43
L9	M	.43	.53	.48	.09	.56	.15	.25	.40	.31	.09	.57	.15	.45	.83	.59	.05	.67	.10	.23	.60	.30
	B	.58	.76	.66	.30	.94	.45	.43	.75	.55	.41	.95	.58	.42	.95	.58	.24	.96	.38	.40	.89	.54
	A	.51	.64	.57	.19	.75	.30	.34	.57	.43	.25	.76	.37	.43	.89	.58	.15	.81	.24	.32	.74	.42
L10	M	.52	.57	.55	.34	.53	.42	.46	.69	.55	.40	.60	.48	.59	.72	.65	.26	.34	.30	.43	.58	.50
	B	.59	.81	.69	.40	.80	.54	.51	.86	.64	.52	.86	.65	.45	.89	.60	.23	.74	.35	.45	.83	.58
	A	.56	.69	.62	.37	.66	.48	.49	.77	.60	.46	.73	.57	.52	.80	.63	.24	.54	.32	.44	.70	.54

Table A.2.: Coreference resolution results for relevant mentions from different approaches