國立臺灣大學電機資訊學院資訊網路與多媒體研究所
碩士論文

Graduate Institute of Networking & Multimedia
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

從需求建構類別圖:應用Transformer爲基礎的機器學習方法

Constructing Class Diagram from Requirements with a Transformer-Based Machine Learning Approach

張馨尹
HSIN-YIN CHANG

指導教授： 李允中 博士
Advisor: Jonathan Lee, Ph.D.

中華民國 112 年 7 月
July, 2023

doi:10.6342/NTU202303318

國立臺灣大學碩士學位論文
口試委員會審定書

從需求建構類別圖：應用 Transformer 為基礎的機器學習方法

Constructing Class Diagrams from Requirements with a Transformer-Based Machine Learning Approach

本論文係張馨尹君（學號 R10944053）在國立臺灣大學資訊網路與多媒體研究所完成之碩士學位論文，於民國一百一十二年七月二十八日承下列考試委員審查通過及口試及格，特此證明。

口試委員：李宏毅（簽名）

（指導教授）

郭忠義　劉建宏

[illegible]　李信杰

所　長：鄭卜壬

誌謝

首先，我要感謝我的指導教授李允中教授，這兩年以來的教導我許多知識以及做研究的方法，從一開始毫無方向，到可以理解老師解決問題的SOP，更重要的是研究的態度，謝謝老師願意給我犯錯的機會。再來，我要感謝臺灣大學軟體工程實驗室的成員，陳力聖、林怡伶、林辰臻、劉仁軒、許恆、梁峻瑞、錢怡君和學弟妹們，在一些共同的專案中一同努力、討論與合作。最後，我想感謝臺北科技大學郭忠義教授和教授的學生，提供給我硬體資源去執行模型。由於許多人的幫助，讓我得以完成此篇論文。

doi:10.6342/NTU202303318

摘要

過往將軟體工程需求自動構建類別圖的方法，都是使用建立規則和自然語言處理技術來抓取類別、類別的屬性和方法和類別之間的關係。然而，此類方法會因爲每個人書寫需求方式的不同，而遺漏屬性或是方法等等，並且也沒有辦法完整的找到所有類別間的關係。 因此，本研究提出了一個自動化的流程，能夠從軟體 需求完整地辨別出類別、類別的屬性和方法和類別之間的關係，其包含以下四個 步驟:將自然語言書寫的軟體需求改寫爲EARS(Easy Approach to Requirements Syntax)格式;利用Transformer-based的模型以及自然語言處理技術分析需求; 生成描述類別圖的UML文件;使用開源軟體工具PlantUML產出類別圖。

關鍵詞 — 類別圖、機器學習、Transfomer模型

doi:10.6342/NTU202303318

Abstracts

The previous methods of automatically converting software engineering requirements into class diagrams involved using rule-based approaches and natural language processing techniques to extract the classes, attributes, methods, and relationships between classes. However, such methods may miss attributes or methods and cannot fully find all relationships between classes due to different writing styles of individuals. Therefore, in this research work, we propose an automated process that can identify classes, attributes, methods, and relationships between classes from software requirements comprehensively with the following four steps: 1. rewrite the software requirements written in natural language into EARS (Easy Approach to Requirements Syntax) format, 2. Use Transformer-based models and natural language processing techniques to analyze requirements, 3. Generate UML documents to describe class diagrams, 4. Use the open-source tool PlantUML to produce class diagrams.

Index terms — Class Diagram, Machine Learning, Transformer Model

doi:10.6342/NTU202303318

Contents

doi:10.6342/NTU202303318

doi:10.6342/NTU202303318

doi:10.6342/NTU202303318

List of Figures

doi:10.6342/NTU202303318

doi:10.6342/NTU202303318

List of Tables

doi:10.6342/NTU202303318

Chapter 1

Introduction

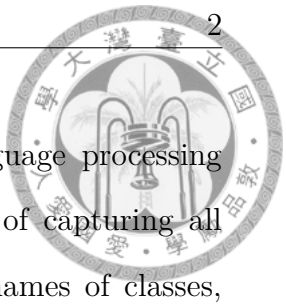
Text is the most common way to express requirements, however, natural language usually has the following characteristics:

- Ambiguity: Less precise, making it difficult to determine exact intentions and functional requirements.
- Subjectivity: Different roles may use different natural languages to describe requirements, which often influenced by individual writing styles, leading to less objectivity in the content of the requirements.

The UML Class diagram is a graphical notation used in object-oriented languages to express structure of the system. It helps us clarify the attributes and methods owned by each class, as well as the relationships between classes. It is highly valuable in system implementation.

Previous research has constructed class diagrams from natural language require-

doi:10.6342/NTU202303318



ments, usually by establishing rules and employing natural language processing techniques. However, currently, these methods are not capable of capturing all the information comprehensively. They often only identify the names of classes, their attributes, and certain relationships, or they rely on interactions with users to complete missing information in the class diagrams.

Therefore, in this paper, we have proposed an automated system to solve the problems of ambiguity and subjectivity mentioned above by first rewriting natural language requirements into the ERAS format. Subsequently, we apply Transformer-based machine learning approaches and NLP (Natural Language Processing) technology to tackle the challenges posed by text diversity.

This thesis is organized as follows: Chapter 2 introduces the related work; Chapter 3 describes the process of constructing a class diagram from requirements and related design; Chapter 4 shows the result of our approach; Chapter 5 summarizes the contributions and the future work this research.

doi:10.6342/NTU202303318

Chapter 2

Related Work

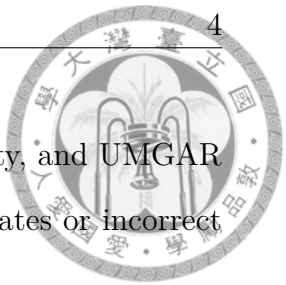
2.1 Related Work

In this section, we will survey the works that use NLP techniques to analyze NLP requirements and construct a class diagram.

Mohd Ibrahim and Rodina Ahmad proposed RACE (Requirements Analysis and Class Diagram Extraction) in 2010. They used heuristic rules and ontology concepts to construct class diagrams, but the RACE system has no way to capture attributes and multiplicity.

Priyanka More and Rashmi Phalnikar[4] proposed their system in 2011, called UMGAR (UML Model Generator from analysis of Requirements), they use NLP techniques, such as POS-Tagger, Word Net, Ontology concepts and heuristic rules to construct category diagrams. They treat nouns as categories, verbs as methods, and adjectives as attributes. Sometimes, this approach may capture incorrect in-

doi:10.6342/NTU202303318



formation. In addition, UMGAR has no way to capture multiplicity, and UMGAR is semi-automatic. For example, when facing issues such as duplicates or incorrect categories, users need to engage in interactive verification.

2.2 Background Work

2.2.1 Easy Approach to Requirements Syntax

Easy Approach to Requirements Syntax[1] abbreviated as EARS, was proposed by Alistair Mavin in 2009. It is a set of rules used in software engineering to describe system requirements. Using a fixed template can reduce the ambiguity arising from the writings of different people. The EARS template can be divided into the following five types (see Figure 2.1):

1. Ubiquitous Requirement
2. Event-Driven Requirement
3. Unwanted Behavior Requirement
4. State-Driven Requirement
5. Optional Feature Requirement

doi:10.6342/NTU202303318

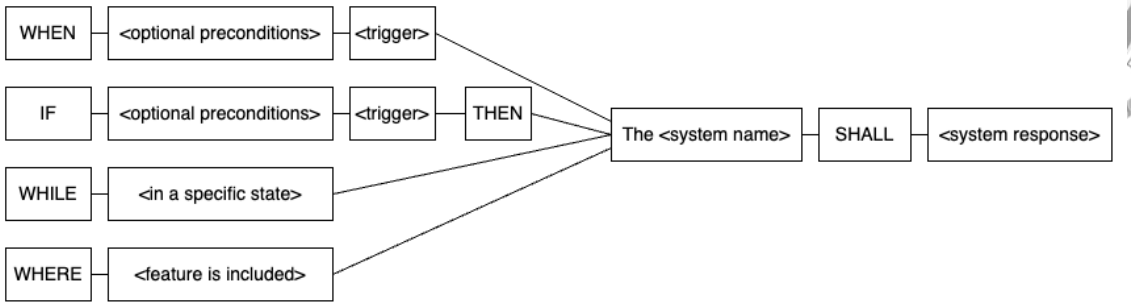


Figure 2.1: The EARS Template[1]

2.2.2 NLP Tools

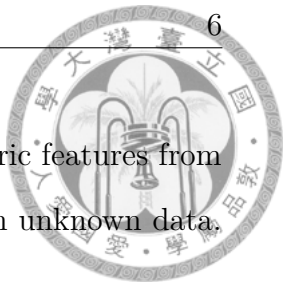
In this study, we will use three open-source python libraries: NLTK, spaCy, Flair. These libraries offer numerous tools related to natural language processing.

First, we will use NLTK to implement text tokenization and part-of-speech tagging. Next, we will utilize spaCy's named entity recognition database to help us identify significant entities in the text. Finally, we will employ Flair for chunking. A detailed process will be explained in Chapter 3.

2.2.3 Few-shot Learning

Few-shot learning is a machine learning approach that enables models to perform well even when hardware resources and training data samples are limited. In traditional machine learning, models usually require a large amount of training data and massive computational resources to achieve good performance. However, real-world scenarios may not always fully meet these conditions. Through Few-shot learning,

doi:10.6342/NTU202303318



we can enable the model to learn and extract meaningful and generic features from a very small number of samples, allowing it to make predictions on unknown data. The detailed training method will be described in Section 2.3.

2.2.4 PlantUML

PlantUML[7] is an open-source software written in Java. It allows users to create diagrams from a plain text language. It supports various types of diagrams, including class diagrams, sequence diagrams, use case diagrams, and more. There are three ways to use PlantUML.

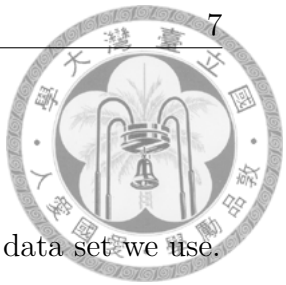
- Direct access to online server.
- Deploy the web server on own machine.
- Use Jar file.

In our research, we will use PlantUML to generate class diagrams.

2.3 Transformer Model

In this section, we will explain the reasons for choosing GPT-NeoX as our implementation model, as well as the parameters and methods used in the model implementation.

doi:10.6342/NTU202303318



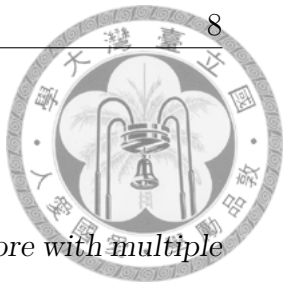
2.3.1 Test Requirements

The following text is the requirement for the experimental test data set we use.

An Order Handling System

- *Design the order handling functionality for a different type of an online shopping site that transmits orders to different order fulfilling companies based on the type of the goods ordered.*
- *Suppose that the group of order handling companies can be classified into three categories based on the format of the order information they expect to receive.*
- *These formats include comma-separated value (CSV), XML and a custom object. When the order information is transformed into one of these formats, appropriate header and footer information that is specific to a format needs to be added to the order data. The series of steps required for the creation of an Order object can be summarized as follows:*
  - *Create the header specific to the format*
  - *Add the order data*
  - *Create the footer specific to the format*

doi:10.6342/NTU202303318



A Sales Reporting Application

- *Build a sales reporting application for the management of a store with multiple departments.*
- *Users should be able to select a specific department they are interested in.*
- *Upon selecting a department, two types of reports are to be displayed:*
  - *Monthly report - A list of all transactions for the current month for the selected department.*
  - *YTD sales chart - A chart showing the year-to-date sales for the selected department by month.*

2.3.2 Fine-tuning model

Before choosing to use the Transformer model, we conducted a series of experiments aimed at comparing the performance difference between fine-tuning a small model and few-shot learning Transformer model. To determine which method performs better, we evaluated these approaches using the same test data. In model selection, we chose three different pre-trained models for fine-tuning: GPT-2, T5-small, and BART-Large. The parameter settings for each model are as follows: (see Figure 2.1)

doi:10.6342/NTU202303318

Table 2.1: Hyperparameter Of Fine-tuning Model

Model Name	Learning Rate	Train Batch Size	Eval Batch Size	Epochs	Optimzer	Environment	
						System RAM	GPU RAM
GPT-2	2e-05	4	4	20	AdamW	25.5GB	15.0GB
T5-small		16	16				
BART-large		16	16				

2.3.3 Few-shot Learning Transformer Model

In the selection of the Transformer model, we chose to use GPT-NeoX-20B[2][6], it was developed by Eleuther AI in 2022 and is currently the largest known open-source pretrained Transformer model. We used a 10-shot approach, which are as follows:

1. Ubiquitous requirements: 2 shots
2. State-driven requirements: 2 shots
3. Event-driven requirements: 2 shots
4. Optional features requirements: 2 shots
5. Unwanted behavior requirements: 2 shots

doi:10.6342/NTU202303318

2.3.4 Experiment Results

We used 268 requirements in non-EARS format as the fine-tuning training data set. After labeling, these requirements are as follows:

1. Ubiquitous requirements: 155
2. State-driven requirements: 16
3. Event-driven requirements: 45
4. Optional features requirements: 21
5. Unwanted behavior requirements: 21

The purpose of this experiment is to rewrite non-EARS formatted requirements into EARS format requirements and compare the performance of these models. The experimental results are shown in Figure 2.2.

doi:10.6342/NTU202303318

Original Requirements (Refer from OOC slide p.95)	Rewritten Requirements	☑, ▲, ✕			
	GPT-NeoX	GPT-2	T	B	GPT-NeoX
Build a sales reporting application for the management of a store with multiple departments.	The system shall build a sales reporting application for the management of a store with multiple departments.	☑	☑	☑	☑
Users should be able to select a specific department they are interested in.	The system shall allow users to select a specific department they are interested in.	▲	☑	☑	☑
Upon selecting a department, two types of reports are to be displayed: Monthly report - A list of all transactions for the current month for the selected department. YTD sales chart - A chart showing the year-to-date sales for the selected department by month.	When the user selects a department, the system shall display two types of reports: Monthly report - A list of all transactions for the current month for the selected department. YTD sales chart - A chart showing the year-to-date sales for the selected department by month.	▲	▲	▲	☑
Whenever a different department is selected, both of the reports should be refreshed with the data for the currently selected department.	Whenever a different department is selected, the system shall refresh both of the reports with the data for the currently selected department.	▲	▲	☑	▲
Design the order handling functionality for a different type of an online shopping site that transmits orders to different order fulfilling companies based on the type of the goods ordered.	The system shall allow the user to select the type of the goods ordered and the system shall allow the user to select the order fulfilling company.	✕	☑	☑	☑
Suppose that the group of order handling companies can be classified into three categories based on the format of the order information they expect to receive.	The system shall allow the group of order handling companies to be classified into three categories based on the format of the order information they expect to receive.	☑	☑	☑	☑
These formats include comma-separated value (CSV), XML and a custom object. When the order information is transformed into one of these formats, appropriate header and footer information that is specific to a format needs to be added to the order data. The series of steps required for the creation of an Order object can be summarized as follows: - Create the header specific to the format - Add the order data - Create the footer specific to the format	The system shall allow the user to create an order object in the format of his choice.	✕	▲	▲	✕

Figure 2.2: Experiment Result

The tick icon indicates a complete match, meaning it fully complies both semantically and in terms of sentence structure. The triangular icon represents a partial match, indicating that there might be a semantic mismatch, the sentence structure is not in EARS format, or there is missing information. The cross icon indicates a complete mismatch.

After this experiment, we have decided to use a few-shot learning transformer model for the subsequent models.

doi:10.6342/NTU202303318

Chapter 3

Constructing Class Diagram

3.1 System Architecture

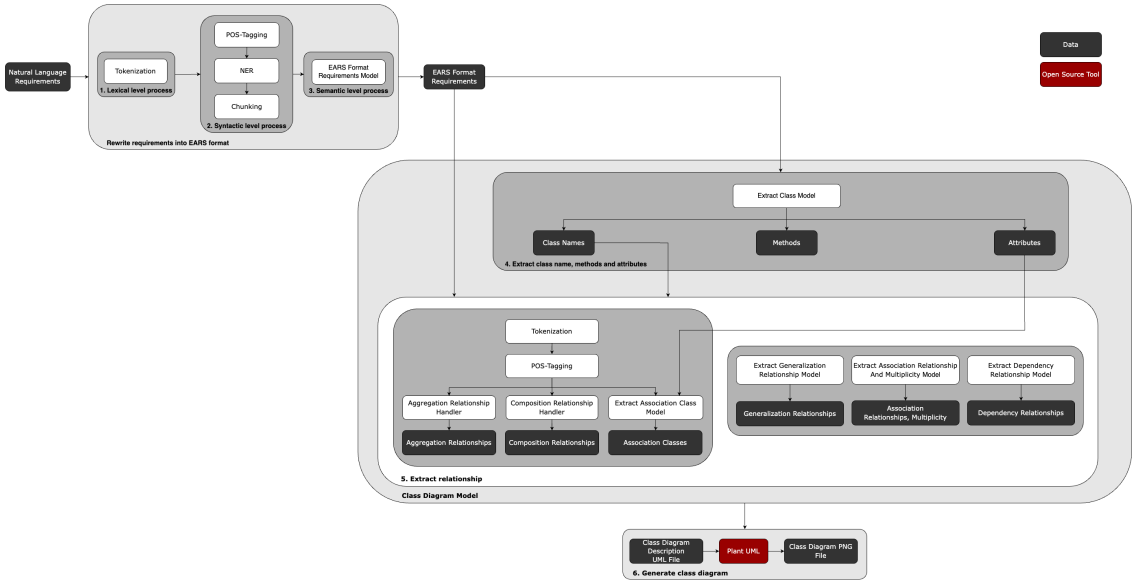
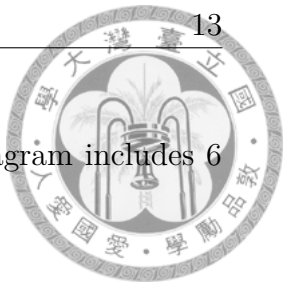


Figure 3.1: Class Diagram Generation System Architecture

doi:10.6342/NTU202303318



As shown in Figure 3.1, the process of constructing a class diagram includes 6 phases:

1. Lexical level process
2. Syntactic level process
3. Semantic level process
4. Extract class name, methods, and attributes
5. Extract relationship
6. Generate class diagram

White blocks represent steps in the process; black blocks represent data; red blocks represent the open-source software used.

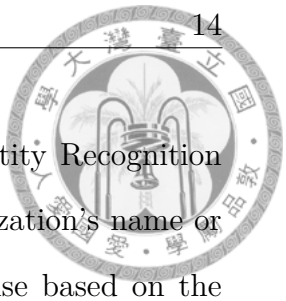
3.2 Lexical level process

The Lexical level process refers to the language processing process at the word level. In this stage, we used the word_tokenize function of NLTK to break down the text into a list of words for use in later syntactic and semantic level processes.

3.3 Syntactic level process

The primary goals of this stage is to identify the system name and system response in the EARS format. First, we tag all the words generated by the lexical

doi:10.6342/NTU202303318



level process with their parts of speech, and then use Named Entity Recognition to find the system name as a proper noun (for example, an organization's name or an English abbreviation). Then, we construct the system response based on the part-of-speech tags and confirm the system name.

3.4 Semantic level process

As shown in the previous experimental results (see Section 2.3.4), the GPT-NeoX-20b model performs the best. Therefore, we use it as the method for rewriting. We used 10 shots, the prompt is as follows:

prompt

We informed the model of the structure and elements of the EARS template, and instructed it that its task is to use the system name and system response obtained from the Syntactic level process, as well as the above template, to rewrite natural language requirements into EARS requirements.

doi:10.6342/NTU202303318

```
org_prompt="""
Separate paragraphs with five square brackets.
Templates:
- The <System Name> shall <System Response>.
- While in a specific state, the <System Name> shall <System Response>.
- When optional preconditions and trigger, the <System Name> shall <System Response>.
- Where feature is included, the <System Name> shall <System Response>.
- If optional preconditions and trigger then <System Name> shall <System Response>.
Your task is to perform the following actions: \
Follow the Templates to rewrite the <Requirement> as an <EARS Requirement> using the <System Name>.

Use the following format:
<Requirements>: the requirements
<System Name>: the name of system
<System Response>: the response of system
<EARS Requirement>: the EARS requirements
```

Figure 3.2: Prompt To Rewrite NLP Requirements

3.5 Extract class name, methods, and attributes

The goal of this stage is to find the class name, method and attribute, the prompt is as follows:

prompt

We break down the problem into 3 steps:

1. Identify nouns and noun phrases, and judge determine if they are classes and attributes.
2. Identify verbs and verb phrases, and determine if they are methods.
3. If the found class has been repeated, the model should not include it again.

The input is all requirements, considering that the input length of the model is limited to 2048 characters, if the requirement exceeds 2048 characters, the text will

doi:10.6342/NTU202303318

be input in blocks; the output is the class name, its attributes and methods.

```
org_prompt="""
Separate paragraphs with five square brackets.
Your task is to find all the class and its attributes.
To solve the problem do the following:
1 - Identify nouns and noun phrases in the requirements in order to find all the <Class> and its <Attribute>
2 - Identify any verbs or verb phrases in the requirement in order to find all the <Method> of the <Class>. \
Verbs often indicate method associated with the classes.
3 - If there is already a found class in the class list, \
there is no need to add it or add the found class to the class list.
4 - If you find all the class from requirements, you should <End>.

Use the following format:
<Requirements>: the requirements

<Class Diagram>:

<Class>: the class you found and class only appears once
<Attribute>: the attribute of the class you found
<Method>: the method of the class you found
```

Figure 3.3: Prompt To Extract Class Name, Methods, and Attributes

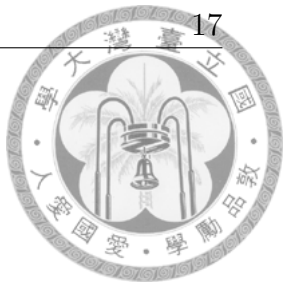
3.6 Extract relationship

The goal of this stage is to find all relationships, including association relationship, dependency relationship, generalization relationship, composition relationship, aggregation relationships, and association class. Among them, the first three types of relationships are captured using machine learning methods, and the latter three are captured using heuristic rules and natural language processing techniques.

3.6.1 Association and Multiplicity

We used 4 shots, the prompt is as follows:

doi:10.6342/NTU202303318



prompt

We break down the problem into 4 steps:

1. Determine whether there is an association relationship based on the given requirements and classes.
2. If the model believes there is an association relationship, it is necessary to write out the reason.
3. Based on the reasons it gives, find the classes that have an association relationship between them.
4. Find the multiplicity within the association relationship.

```
prompt="""
Separate paragraphs with five square brackets.
Typically, Association relationships do not have names.
1 - Determine whether there Has Association relationship according to the <Requirements> and <Class>.
2 - If <Has Association> is Yes then you should give a Reason.
3 - Accroding to the Reason to select the First Class and Second Class from <Class>.
4 - Find the Multiplicity of the <Association> relationship.
Note - Each group of First Class and Second Class cannot be repeated.
```

Figure 3.4: Prompt To Extract Association Relationship and Multiplicity

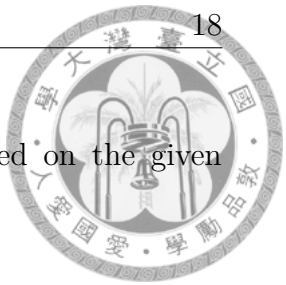
3.6.2 Dependency

We used 4 shots, the prompt is as follows:

prompt

We break down the problem into 3 steps:

doi:10.6342/NTU202303318



1. Determine whether there is a dependency relationship based on the given requirements and classes.

2. If the model believes there is a dependency relationship, it is necessary to write out the reason.

3. Based on the reasons it gives, find the classes that have a dependency relationship between them.

```
prompt="""
Summarize the text delimited by triple backticks into a single sentence.
Separate paragraphs with five square brackets.
In UML, a dependency relationship is a relationship in which one element, \
the client, uses or depends on another element, the supplier. \
You can use dependency relationships in class diagrams, component diagrams, \
deployment diagrams, and use-case diagrams to \
indicate that a change to the supplier might require a change to the client.
Typically, dependency relationships do not have names.
1 - Determine whether there Has Dependency relationship according to the <Requirements> and <Class>.
2 - If <Has Dependency> is Yes then you should give a Reason.
3 - Accroding to the Reason to select the First Class and Second Class from <Class>.
Note - Each group of First Class and Second Class cannot be repeated.
```

Figure 3.5: Prompt To Extract Dependency Relationship

3.6.3 Generalization

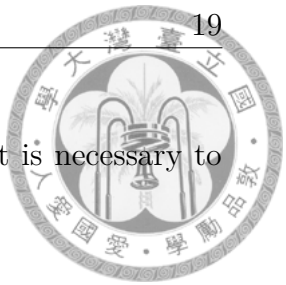
We used 5 shots, the prompt is as follows:

prompt

We break down the problem into 3 steps:

1. Determine whether there is a generalization relationship based on the given requirements and classes.

doi:10.6342/NTU202303318



2. If the model believes there is a generalization relationship, it is necessary to write out the reason.

3. Find the parent class and all its child classes based on the reasons it gives.

```
prompt="""
Summarize the text delimited by triple backticks into a single sentence.
Separate paragraphs with five square brackets.
Your task is to perform the following actions:
1 - Determine whether there Has Generalization relationship accroding to the <Requirements> and <Class>.
2 - If <Has Generalization> is Yes then you should give a Reason. Reason, Parent Class, Child Class won't be None. \
Accroding to the Reason to find the Parent Class and Child Class.
Note - Parent Class and Child Class are Nouns.
```

Figure 3.6: Prompt To Extract Generalization Relationship

3.6.4 Composition, Aggregation, Association Class

The following are the rules for capturing composition relationship, aggregation relationship, and association class:

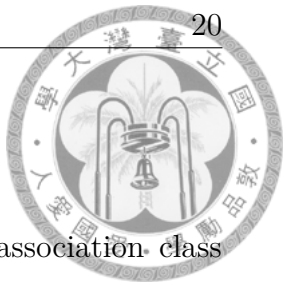
- composition relationship:

  1. There is a association relationship between two classes.
  2. Its syntax structure satisfies: class - verb - class, and the verb is ”part of”, ”consist of”, ”composition of”, ”divided to”... etc.

- aggregation relationship:

  1. There is a association relationship between two classes.
  2. Its syntax structure satisfies: class - verb - class, and the verb is ”an aggregation of”, ”own”, ”contain”, ”include”... etc.

doi:10.6342/NTU202303318



- association class:

  1. Each participating object contains a reference to the association class object.
  2. The association class object contains references to each of the related objects.

3.6.5 Generate Class Diagram

In this section, we will use the text files generated in the previous stage, generate a class diagram through PlantUML as mentioned in Section Chapter 2.2.4.

doi:10.6342/NTU202303318

Chapter 4

Result

In this chapter, we will present the results of our model and compare them with other Transformer models. Our input requirements are from the Pontis SRS and TACHOnet SRS in the SRS dataset [5].

4.1 Pontis SRS

The following figure shows the class diagram of the Pontis SRS.

doi:10.6342/NTU202303318

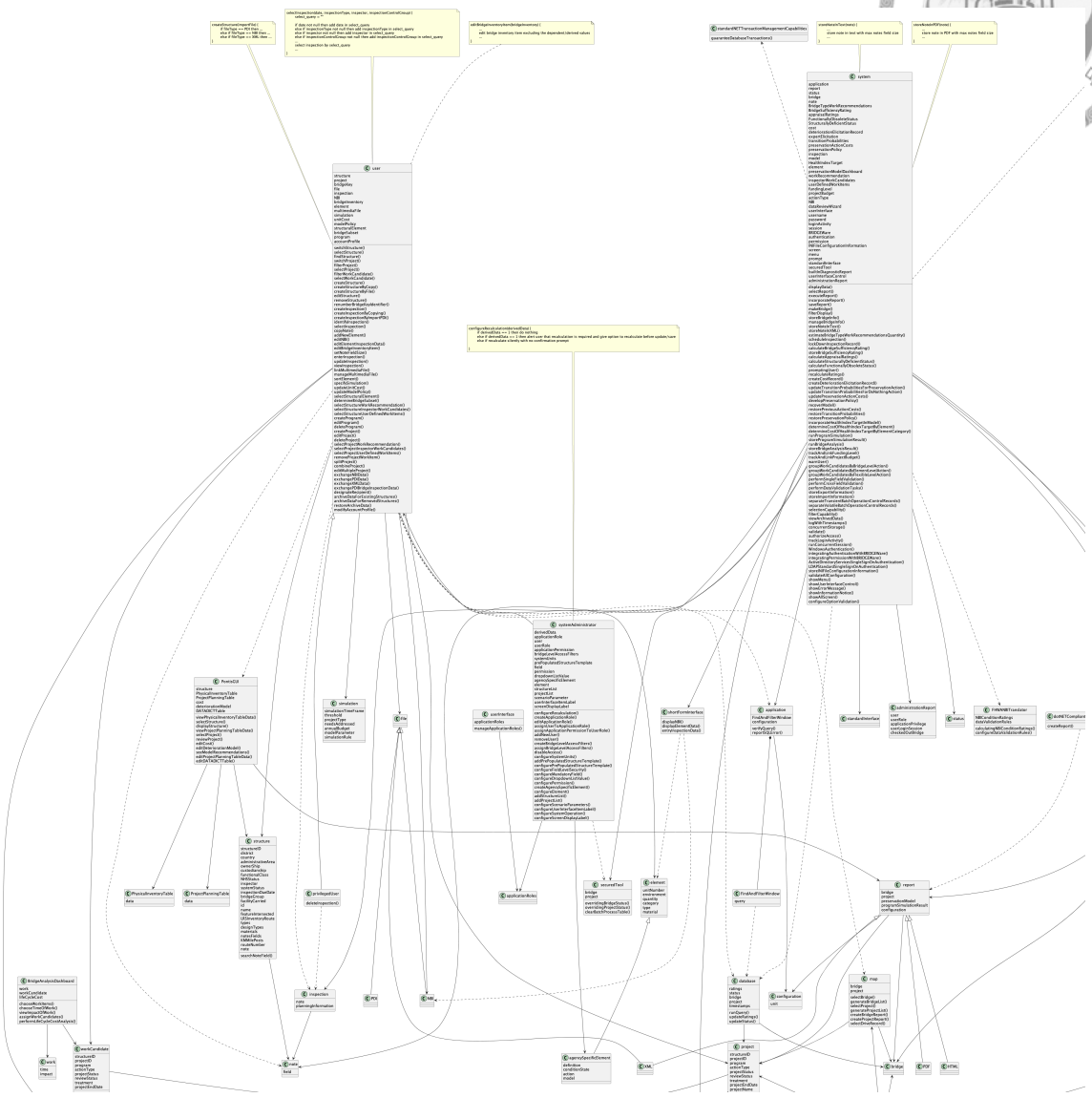


Figure 4.1: The Class Diagram Of Pontis SRS

4.1.1 Our Model

The figure below shows the Pontis class diagram we constructed.

doi:10.6342/NTU202303318

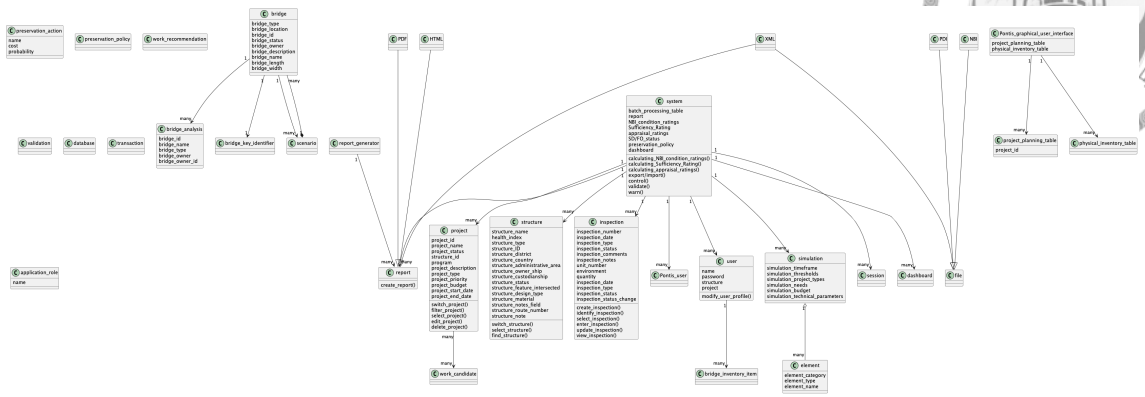


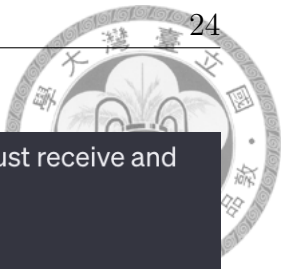
Figure 4.2: The Pontis Class Diagram Constructed By Our Model

4.1.2 GPT-3

The autoregressive language model, GPT-3, proposed by OpenAI[3] in 2020, uses 175 billion parameters in its neural network, making it the second-largest neural model currently available (only surpassed by GPT-4). We utilize a zero-shot learning approach to prompt the GPT-3 model. The following is our prompt for zero-shot learning:

- Extract class name, methods and attributes

doi:10.6342/NTU202303318



```
Do not answer yet. This is just another part of the text I want to send you. Just receive and
acknowledge as "Part 1/2 received" and wait for the next part.
[START PART 1/2]
Separate paragraphs with five square brackets.
Your task is to find all the class and its attributes.
To solve the problem do the following:
1 - Identify nouns and noun phrases in the requirements in order to find all the <Class> and
its <Attribute>
2 - Identify any verbs or verb phrases in the requirement in order to find all the <Method> of
the <Class>. \
Verbs often indicate method associated with the classes.
3 - If there is already a found class in the class list, \
there is no need to add it or add the found class to the class list.
4 - If you find all the class from requirements, you should <End>.

Use the following format:
<Requirements>: the requirements

<Class Diagram>:

<Class>: the class you found and class only appears once
<Attribute>: the attribute of the class you found
<Method>: the method of the class you found
#####
```

Figure 4.3: Prompt To Extract Class Name, Methods, Attributes (GPT-3)

- Extract relationship
  - Association and Multiplicity

doi:10.6342/NTU202303318

Do not answer yet. This is just another part of the text I want to send you. Just receive and acknowledge as "Part 1/2 received" and wait for the next part.
[START PART 1/2]
Separate paragraphs with five square brackets.
Your task is to perform the following actions:
1 - Determine whether there Has Association relationship according to the <Requirements> and <Class>.
2 - If <Has Association> is Yes then you should give a Reason.
3 - According to the Reason to select the First Class and Second Class from <Class>.
Note - Each group of First Class and Second Class cannot be repeated.

Use the following format:
<Requirements>: the requirements

<Class>: the classes

<Has Association>: Yes or No

<Association>
Reason: Why do you think there is a association relationship
First Class: The class in the association relationship
Second Class: Another class in the association relationship

Figure 4.4: Prompt To Extract Association Relationship and Multiplicity (GPT-3)

– Dependency

doi:10.6342/NTU202303318

```
Do not answer yet. This is just another part of the text I want to send you. Just receive and
acknowledge as "Part 1/2 received" and wait for the next part.
[START PART 1/2]
Separate paragraphs with five square brackets.
In UML, a dependency relationship is a relationship in which one element, \
the client, uses or depends on another element, the supplier. \
You can use dependency relationships in class diagrams, component diagrams, \
deployment diagrams, and use-case diagrams to \
indicate that a change to the supplier might require a change to the client.
Typically, dependency relationships do not have names.
Your task is to perform the following actions:
1 - Determine whether there Has Dependency relationship according to the <Requirements>
and <Class>.
2 - If <Has Dependency> is Yes then you should give a Reason.
3 - Accroding to the Reason to select the First Class and Second Class from <Class>.
Note - Each group of First Class and Second Class cannot be repeated.

Use the following format:
<Requirements>:  the requirements

<Class>: the classes

<Has Dependency>: Yes or No

<Dependency>
Reason: Why do you think there is a dependency relationship
First Class: The class in the dependency relationship
Second Class: Another class in the dependency relationship
```

Figure 4.5: Prompt To Extract Dependency Relationship (GPT-3)

– Generalization

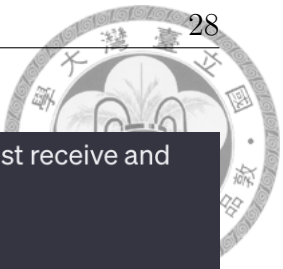
doi:10.6342/NTU202303318

```
Do not answer yet. This is just another part of the text I want to send you. Just receive and
acknowledge as "Part 1/2 received" and wait for the next part.
[START PART 1/2]
Separate paragraphs with five square brackets.
Summarize the text delimited by triple backticks into a single sentence.
Separate paragraphs with five square brackets.
Your task is to perform the following actions:
1 - Determine whether there Has Generalization relationship according to the
<Requirements> and <Class>.
2 - If <Has Generalization> is Yes then you should give a Reason. Reason, Parent Class, Child
Class won't be None. \
According to the Reason to find the Parent Class and Child Class.
Note - Parent Class and Child Class are Nouns.

Use the following format:
<Requirements>:  the requirements

<Class>: the classes

<Has Generalization>: Yes or No

<Generalization>
Reason: Why do you think there is a generalization relationship
Parent Class: The parent class in generalization relationship
Child Class: The child class in generalization relationship
```

Figure 4.6: Prompt To Extract Generalization Relationship (GPT-3)

– Composition

doi:10.6342/NTU202303318



```
Do not answer yet. This is just another part of the text I want to send you. Just receive and
acknowledge as "Part 1/2 received" and wait for the next part.
[START PART 1/2]
Separate paragraphs with five square brackets.
Your task is to perform the following actions:
1 - Determine whether there Has Composition relationship according to the <Requirements>
and <Class>.
2 - If <Has Composition > is Yes then you should give a Reason.
3 - According to the Reason to select the First Class and Second Class from <Class>.
Note - Each group of First Class and Second Class cannot be repeated.

Use the following format:
<Requirements>:  the requirements

<Class>: the classes

<Has Composition >: Yes or No

<Composition>
Reason: Why do you think there is a composition relationship
First Class: The class in the composition relationship
Second Class: Another class in the composition relationship
```

Figure 4.7: Prompt To Extract Composition Relationship (GPT-3)

– Aggregation

doi:10.6342/NTU202303318

```
Do not answer yet. This is just another part of the text I want to send you. Just receive and
acknowledge as "Part 1/2 received" and wait for the next part.
[START PART 1/2]
Separate paragraphs with five square brackets.
Your task is to perform the following actions:
1 - Determine whether there Has Aggregation relationship according to the <Requirements>
and <Class>.
2 - If <Has Aggregation > is Yes then you should give a Reason.
3 - According to the Reason to select the First Class and Second Class from <Class>.
Note - Each group of First Class and Second Class cannot be repeated.

Use the following format:
<Requirements>:  the requirements

<Class>: the classes

<Has Aggregation >: Yes or No

< Aggregation >
Reason: Why do you think there is a aggregation relationship
First Class: The class in the aggregation relationship
Second Class: Another class in the aggregation relationship
```

Figure 4.8: Prompt To Extract Aggregation Relationship (GPT-3)

The figure below shows the Pontis class diagram constructed by GPT-3.

doi:10.6342/NTU202303318

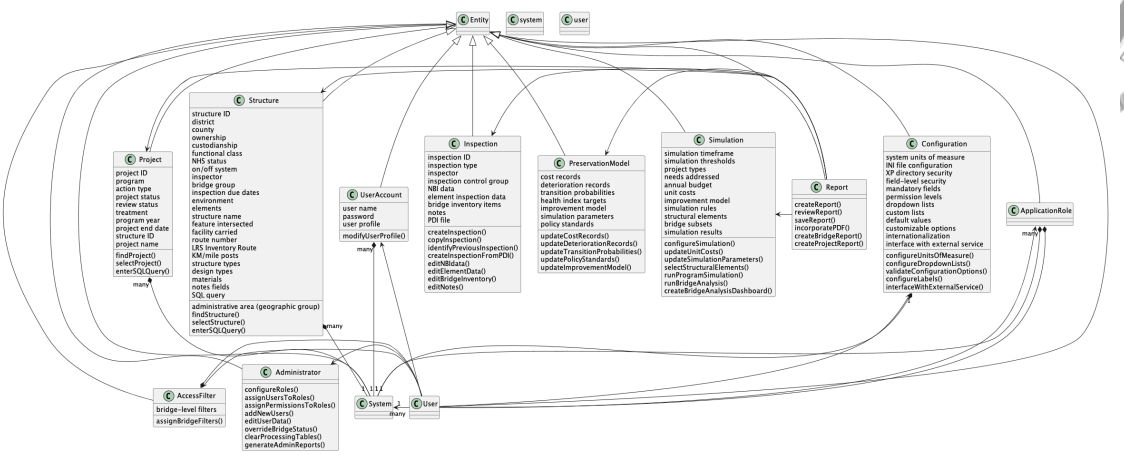


Figure 4.9: The Pontis Class Diagram Constructed By GPT-3

4.1.3 BARD

The generative artificial intelligence chatbot proposed by Google in 2023, based on the LaMDA model[8]. We utilize a zero-shot learning approach to prompt BARD. The following is our prompt for zero-shot learning:

- Extract class name, methods and attributes

doi:10.6342/NTU202303318

```
Separate paragraphs with five square brackets.
Your task is to find all the class and its attributes.
To solve the problem do the following:
1 - Identify nouns and noun phrases in the requirements in order to find all the <Class> and its <Attribute>
2 - Identify any verbs or verb phrases in the requirement in order to find all the <Method> of the <Class>. \
Verbs often indicate method associated with the classes.
3 - If there is already a found class in the class list, \
there is no need to add it or add the found class to the class list.
4 - If you find all the class from requirements, you should <End>.

Use the following format:
<Requirements>: the requirements

<Class Diagram>:

<Class>: the class you found and class only appears once
<Attribute>: the attribute of the class you found
<Method>: the method of the class you found
```

Figure 4.10: Prompt To Extract Class Name, Methods, Attributes (BARD)

- Extract relationship
  - Association and Multiplicity

doi:10.6342/NTU202303318

```
Separate paragraphs with five square brackets.
Your task is to perform the following actions:
1 - Determine whether there Has Association relationship according to the <Requirements> and <Class>.
2 - If <Has Association > is Yes then you should give a Reason. Reason, First Class, Second Class won't be None. \
According to the Reason to find the First Class and Second Class.
3. 3. Find the multiplicity of the First class and Second class
Note - Each group of First Class and Second Class cannot be repeated.

Use the following format:
<Requirements>:  the requirements

<Class>: the classes

<Has Association >: Yes or No

<Association>
Reason: Why do you think there is a association relationship
First Class: The class in the association relationship
Second Class: Another class in the association relationship
```

Figure 4.11: Prompt To Extract Association Relationship and Multiplicity (BARD)

– Dependency

doi:10.6342/NTU202303318

```
Separate paragraphs with five square brackets.
In UML, a dependency relationship is a relationship in which one element, \
the client, uses or depends on another element, the supplier. \
You can use dependency relationships in class diagrams, component diagrams, \
deployment diagrams, and use-case diagrams to \
indicate that a change to the supplier might require a change to the client.
Typically, dependency relationships do not have names.
Your task is to perform the following actions:
1 - Determine whether there Has Dependency relationship according to the <Requirements> and <Class>.
2 - If <Has Dependency> is Yes then you should give a Reason.
3 - Accroding to the Reason to select the First Class and Second Class from <Class>.
Note - Each group of First Class and Second Class cannot be repeated.

Use the following format:
<Requirements>:  the requirements

<Class>: the classes

<Has Dependency>: Yes or No

<Dependency>
Reason: Why do you think there is a dependency relationship
First Class: The class in the dependency relationship
Second Class: Another class in the dependency relationship
```

Figure 4.12: Prompt To Extract Dependency Relationship (BARD)

– Generalization

doi:10.6342/NTU202303318

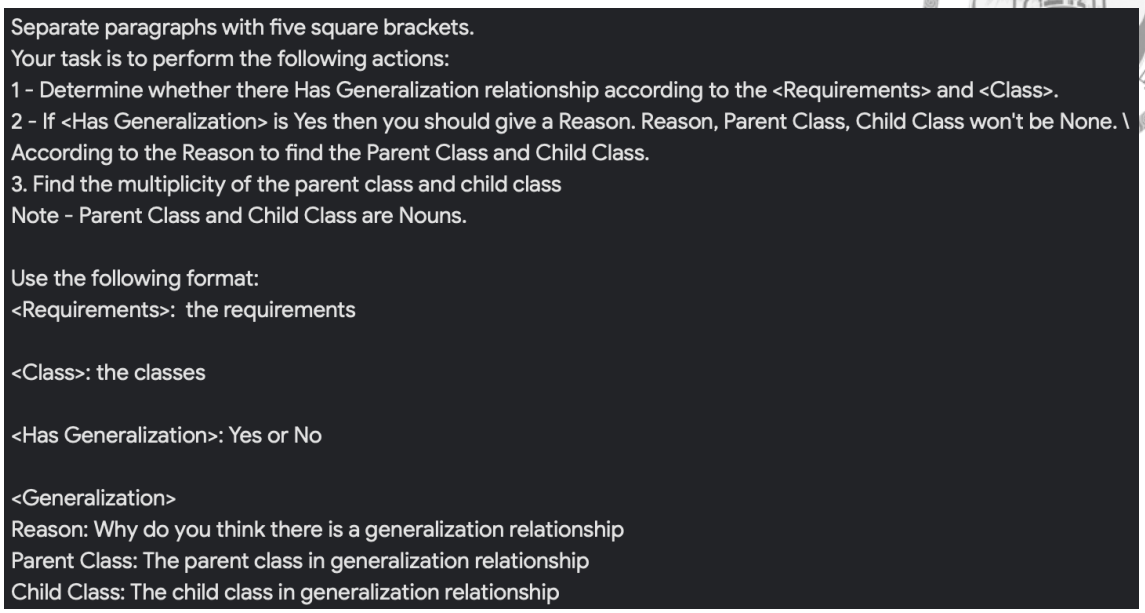


Figure 4.13: Prompt To Extract Generalization Relationship (BARD)

– Composition

doi:10.6342/NTU202303318

```
Separate paragraphs with five square brackets.
Your task is to perform the following actions:
1 - Determine whether there Has Composition relationship according to the <Requirements> and <Class>.
2 - If <Has Composition > is Yes then you should give a Reason. Reason, First Class, Second Class won't be None. \
According to the Reason to find the First Class and Second Class.
3. 3. Find the multiplicity of the First class and Second class
Note - Each group of First Class and Second Class cannot be repeated.

Use the following format:
<Requirements>:  the requirements

<Class>: the classes

<Has Composition >: Yes or No

< Composition >
Reason: Why do you think there is a composition relationship
First Class: The class in the composition relationship
Second Class: Another class in the composition relationship
```

Figure 4.14: Prompt To Extract Composition Relationship (BARD)

– Aggregation

doi:10.6342/NTU202303318

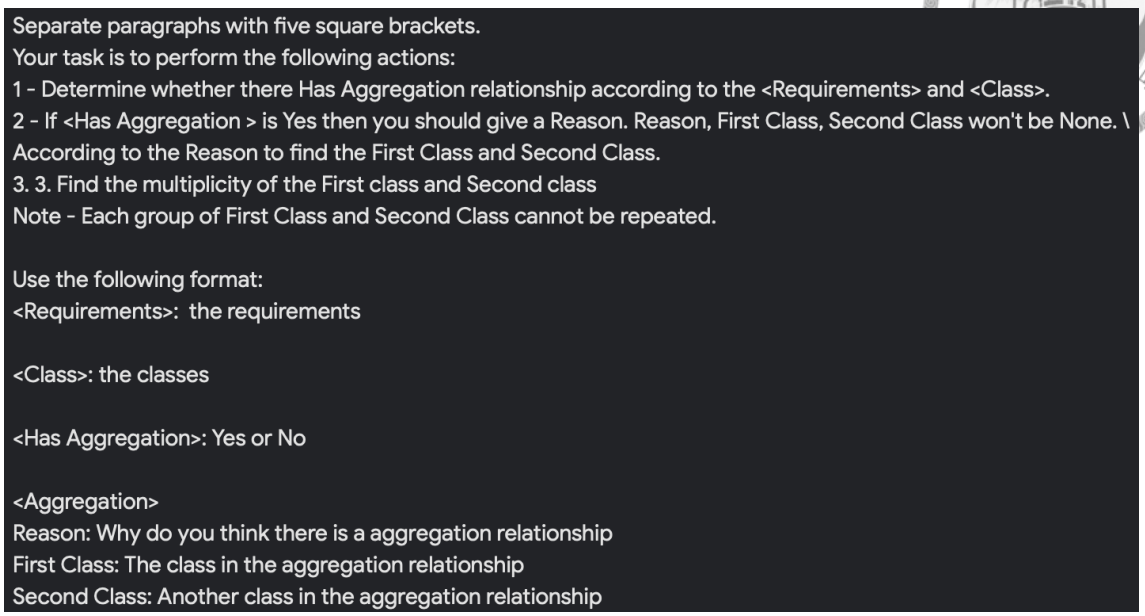


Figure 4.15: Prompt To Extract Aggregation Relationship (BARD)

The figure below shows the Pontis class diagram constructed by BARD.

doi:10.6342/NTU202303318

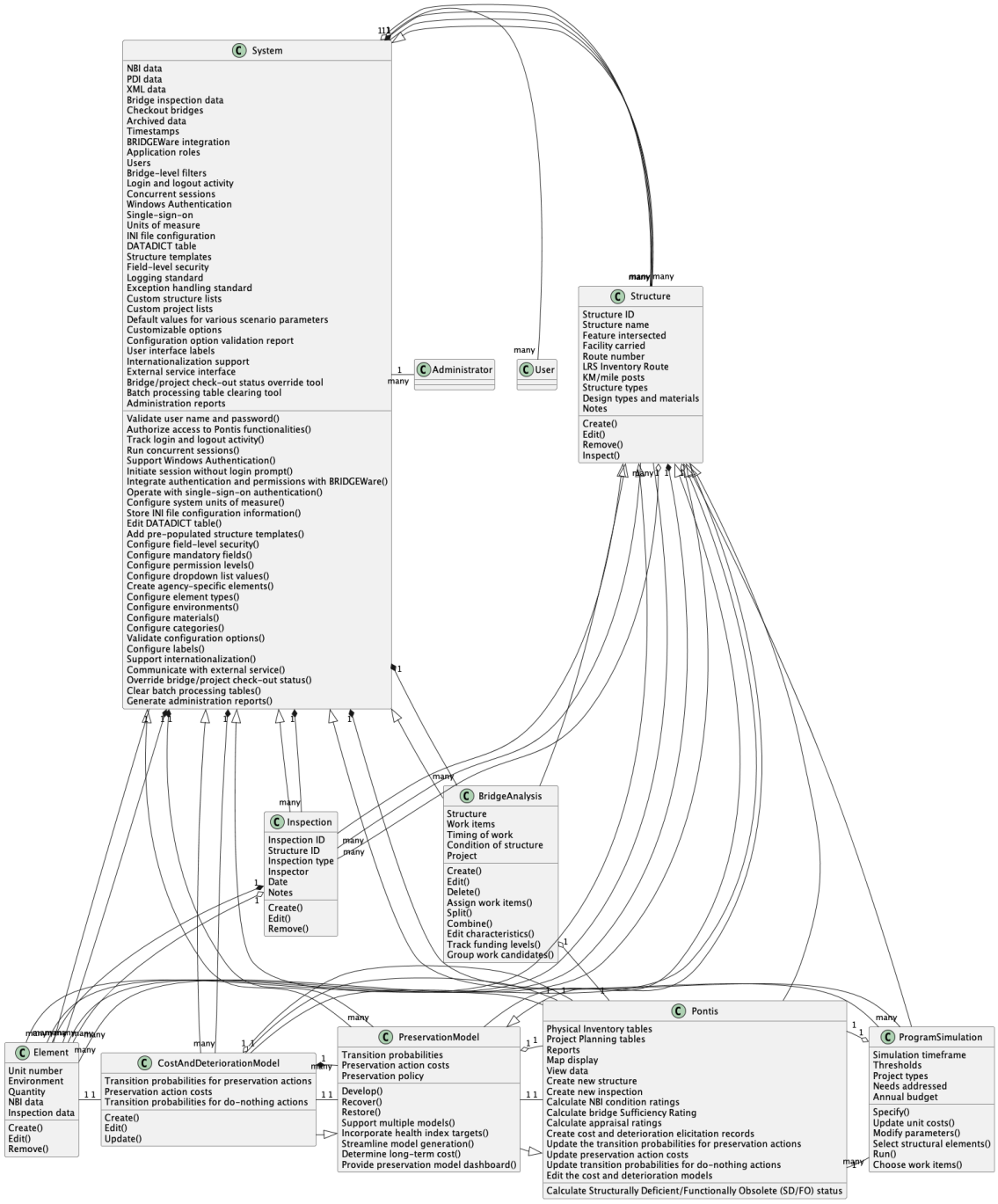


Figure 4.16: The Pontis Class Diagram Constructed By BARD

doi:10.6342/NTU202303318

4.2 TACHOnet SRS

The following figure shows the class diagram of the TACHOnet SRS.

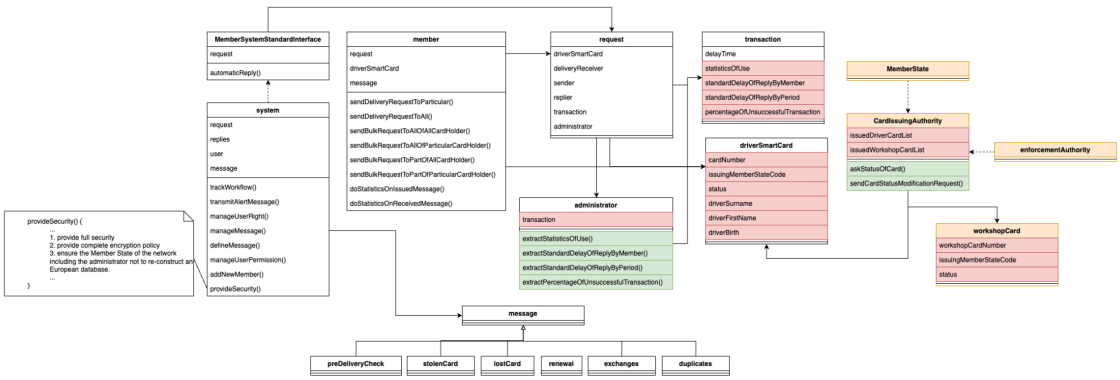


Figure 4.17: The Class Diagram Of TACHOnet SRS

4.2.1 Our Model

The figure below shows the TACHOnet class diagram we constructed.

doi:10.6342/NTU202303318

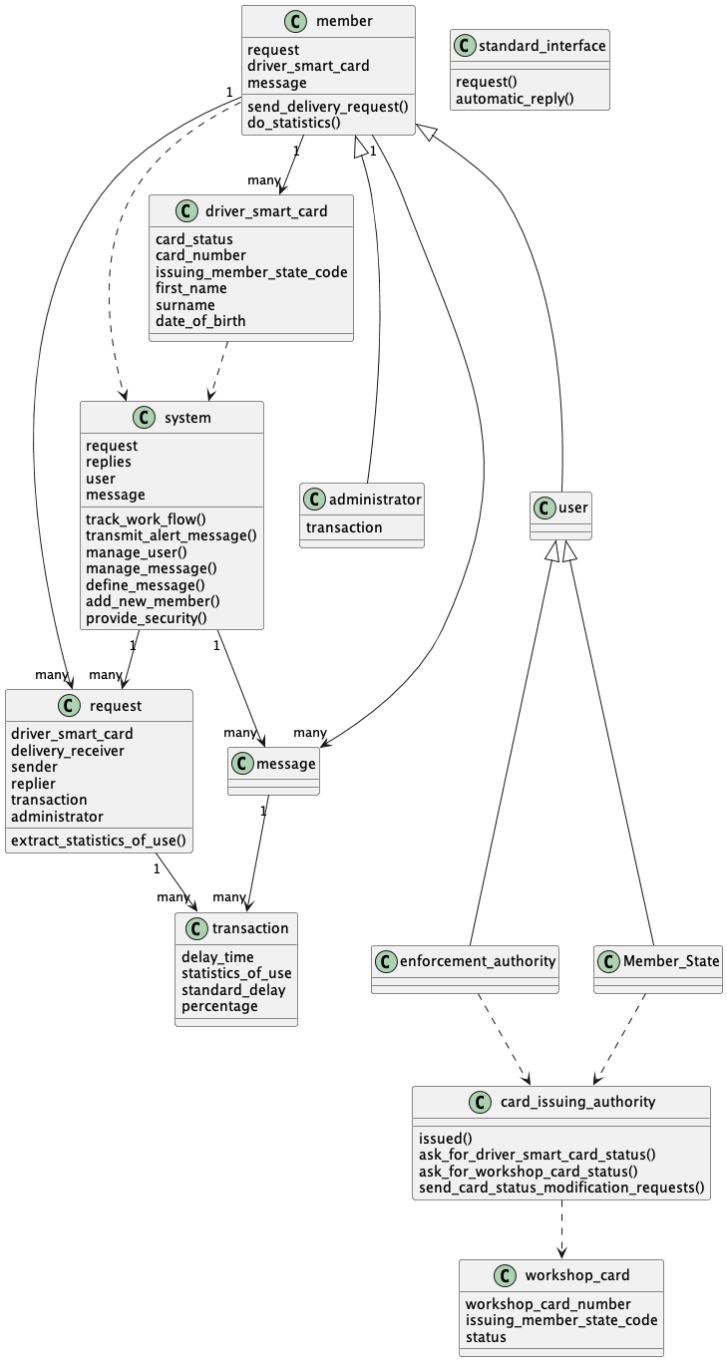


Figure 4.18: The TACHOnet Class Diagram Constructed By Our Model

doi:10.6342/NTU202303318

4.2.2 GPT-3

The figure below shows the TACHOnet class diagram constructed by GPT-3.

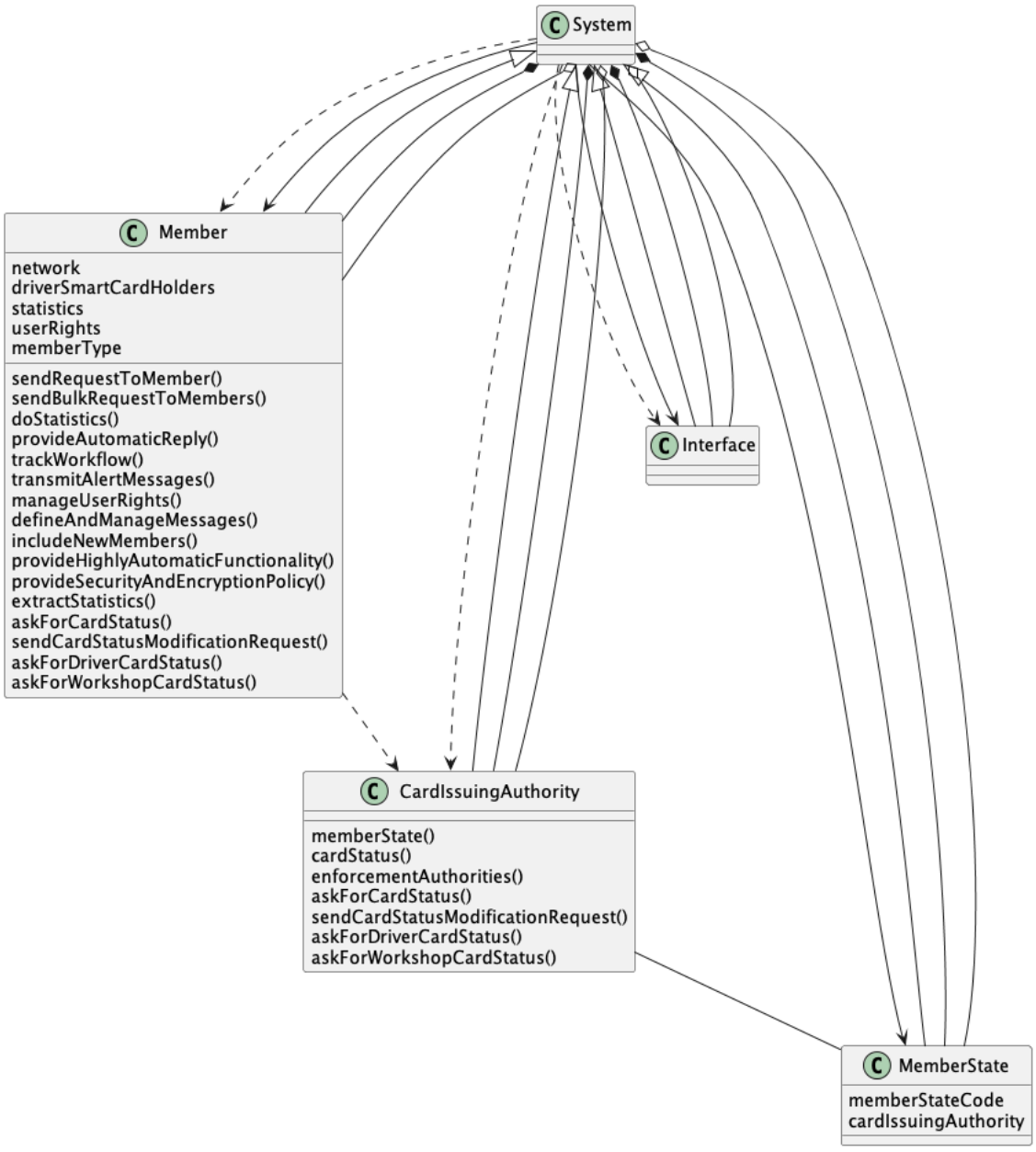


Figure 4.19: The TACHOnet Class Diagram Constructed By GPT-3

doi:10.6342/NTU202303318

4.2.3 BARD

The figure below shows the TACHOnet class diagram constructed by BARD.

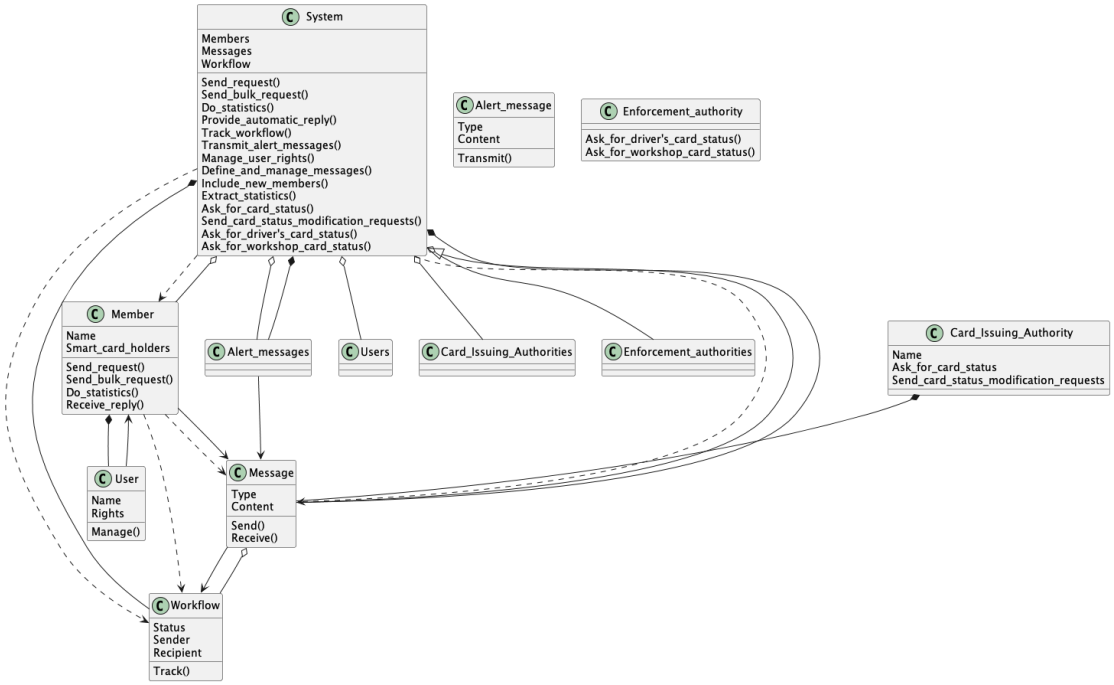


Figure 4.20: The TACHOnet Class Diagram Constructed By BARD

doi:10.6342/NTU202303318

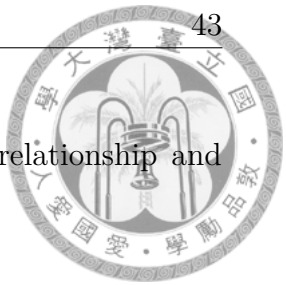
Chapter 5

Conclusion

5.1 Summary

From the experimental results (see Chapter 4), we can observe that the performance of the model deteriorates with the increase in the length of the requirement text. We speculate this is because the GPT-NeoX model does not have the capability to remember, and when we encounter requirement lengths that exceed maximum input length of the model, our current approach is to take segmented input, which leads to the model's inability to extract class, attributes, or methods without referencing the context. Additionally, when producing class diagrams from short requirements descriptions (e.g., TACHOnet SRS, which is much shorter than Pontis SRS), both GPT-3 and BARD perform well, but our model is more precise in the judgment of relationships, we speculate that this is due to the utilization of few-shot learning. Unlike GPT-3 and BARD do not have shots to refer to, hence they are

doi:10.6342/NTU202303318



unable to distinguish differences between things like association relationship and dependency relationship.

5.2 Future work

Although we can construct class diagrams from requirements using methods based on the Transformer model, however, there are still some pieces of information that have not been captured yet. There are two directions for future research summarized below:

- System Function
  - Our concurrent research does not support programming logic, association qualifier, parameter of a method, and visibility of attribute.
  - We may be able to use the class diagram constructed by the model to debug software requirements in reverse.
- Transformer Model
  - The current time to construct a class diagram is several times that of GPT-3 and BARD.
  - There isn't a standardized and automated method to evaluate the quality of the class diagram generated by various models.

doi:10.6342/NTU202303318

Bibliography

[1] A. H. a. M. N. Alistair Mavin, Philip Wilkinson. Easy approach to requirements syntax (EARS). In *2009 17th IEEE International Requirements Engineering Conference*, pages 317–322. IEEE, 2009.

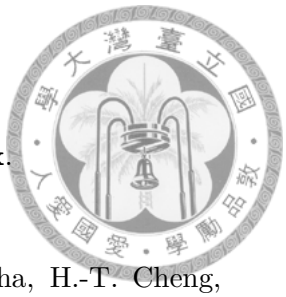
[2] S. Black, S. Biderman, E. Hallahan, Q. Anthony, L. Gao, L. Golding, H. He, C. Leahy, K. McDonell, J. Phang, M. Pieler, U. Sai Prashanth, S. Purohit, L. Reynolds, J. Tow, B. Wang, and S. Weinbach. GPT-NeoX-20B: An Open-Source Autoregressive Language Model. *arXiv e-prints*, Apr. 2022.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. *arXiv e-prints*, page arXiv:2005.14165, May 2020.

[4] M. A. B. Deva Kumar Deeptimahanti. Semi-automatic generation of uml models from natural language requirements. In *Proceedings of the 4th India Software Engineering Conference*, pages 165–174, 2011.

[5] A. Ferrari, G. O. Spagnolo, and S. Gnesi. Pure: A dataset of public requirements documents. pages 502–505, 2017.

doi:10.6342/NTU202303318



[6] GPT-NeoX-20B. https://www.github.com/eleutherai/gpt-neox.

[7] PlantUML. https://github.com/plantuml/plantuml.

[8] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, Y. Li, H. Lee, H. S. Zheng, A. Ghafouri, M. Menegali, Y. Huang, M. Krikun, D. Lepikhin, J. Qin, D. Chen, Y. Xu, Z. Chen, A. Roberts, M. Bosma, V. Zhao, Y. Zhou, C.-C. Chang, I. Krivokon, W. Rusch, M. Pickett, P. Srinivasan, L. Man, K. Meier-Hellstern, M. Ringel Morris, T. Doshi, R. Delos Santos, T. Duke, J. Soraker, B. Zevenbergen, V. Prabhakaran, M. Diaz, B. Hutchinson, K. Olson, A. Molina, E. Hoffman-John, J. Lee, L. Aroyo, R. Rajakumar, A. Butryna, M. Lamm, V. Kuzmina, J. Fenton, A. Cohen, R. Bernstein, R. Kurzweil, B. Aguera-Arcas, C. Cui, M. Croak, E. Chi, and Q. Le. LaMDA: Language Models for Dialog Applications. *arXiv e-prints*, page arXiv:2201.08239, Jan. 2022.

doi:10.6342/NTU202303318