Usage of LLM for Generation of UML Class Diagrams from UML Use-Case Diagrams

Martina Basic[1]; Marko Vujasinovic[1]

[1]University of Applied Sciences Aspira
Department of Computer Engineering
Split

Publication Date: 2026/02/07

Abstract: LLMs have become increasingly popular tools for assisting software engineering tasks and specifically with automating the creation of UML design artefacts from requirements specified in a natural language. This work reports on the effectiveness of using LLM technologies to perform the translation of UML use-case diagrams into UML class Diagrams. Such task is traditionally performed by a human and requires both domain understanding and modeling expertise, however, it can be a tedious and labor-intensive manual task prone to errors during its process. Automated creation of UML class Diagrams from UML use-case diagrams requires LLMs to identify the domain entities, the responsibilities of each of those domain entities, and how domain entities relate to each other and therefore creating unique classes. The evaluation part of the work uses several use-case diagrams of varying levels of complexity to complete this task. The results showed that LLMs can effectively identify the crucial domain entities and provide class structures, and to alleviate the modelling efforts. However, there are still issues with the interpretation of ambiguous requirements and the maintenance of their associated semantics. Overall, LLM technologies are an effective aid to the early-stage development of UML class Diagrams; however, the final quality of the UML models is very dependent upon the domain experts that provide further guidance to the LLMs.

Keywords: *LLM; UML Class Diagram; UML Use-Case Diagram, Requirements Specification.*

How to Cite: Martina Basic; Marko Vujasinovic (2026) Usage of LLM for Generation of UML Class Diagrams from UML Use-Case Diagrams. *International Journal of Innovative Science and Research Technology*, 11(1), 3226-3245. https://doi.org/10.38124/ijisrt/26jan1576

I. INTRODUCTION

According to the Object Management Group (2017), Unified Modeling Language – UML is a standardized visual language used for modeling, visualizing, specifying, constructing, modifying and documenting complex software systems and business processes. UML is used in all phases of software engineering from specification through design and implementation to system maintenance. By using UML diagrams, it is easier to understand the functioning of the system, to define goals, to plan resources, to improve and optimize existing systems, and to facilitate the maintenance and evaluation of the system. Due to its standardization and ease of understanding, it enables and improves communication between all the participants in software design and implementation, from the client to designers, programmers and testers. In the early stages of software development, the UML use case diagrams help to define the requirements expected from the software (these requirements should be understandable to both the client and the developer), and then UML classes and relationships between classes are generated either from it and from the accompanying textual descriptions in natural language.

In software engineering, certain processes in software development are standardized, and manual modeling unnecessarily wastes time on models that are already idle, and reduces the time invested in analyzing more complex processes. Manual creation of UML diagrams has been described in the literature as a tedious and time-consuming process, particularly for large or complex systems (Rouabhia & Hadjadj, 2024). The need for automation of UML diagram creation arose, which was realized with the advent of large language models (LLM, in short). LLMs are advanced artificial intelligence systems designed to process, understand, and generate human-like text. LLMs are beginning to make it easier to create UML diagrams from user descriptions, saving time (Cámara et al., 2023; Al-Ahmad et al., 2025). Automated creation of UML Class Diagrams from UML use-case diagrams requires LLMs to identify the domain entities, the responsibilities of each of those domain entities, and how domain entities relate to each other and therefore creating the classes.

The goal of work described in this paper is to use LLM tooling such as ChatGPT (https://chatgpt.com/) to do automated creation of UML Class Diagrams from UML use-case diagrams and then to evaluate and discuss the

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

completeness and correctness of the generated UML class diagrams. The specific focus is to evaluate if the generated results match the UML class diagrams obtained by manually creating diagrams in the modeling tool by a human expert, then to evaluate if the generated UML class diagrams are functional for further use and how they can be improved by giving the additional instructions. The structure of paper is as follows. In the second chapter the related work is presented. Chapter 3 and 4 introduce theoretical background and methodology for the research done. Chapter 5 presents case studies and experimental work. Chapter 6 provides discussion of results and finally there is a conclusion.

II. RELATED WORK

Automated generation of UML diagrams has been one of main topics and challenges in software engineering. Traditional work on automated UML diagram generation focuses on natural language processing (NLP) methods that extract classes, relationships, and design elements from textual requirements, including also textual description of UML use-cases. NLP and machine-learning based frameworks have been proposed to classify sentences and construct UML class diagrams, leveraging classic text classification and feature extraction techniques (Zhao et al., 2021). These approaches demonstrate feasibility but are constrained by the quality of linguistic analysis and the need for extensive engineering of extraction rules. In contrary to that, Large language models (LLMs) has opened new possibilities for UML model generations and verifications. Several empirical studies have already demonstrated that LLMs such as GPT agents can generate UML models from natural language inputs with varying levels of correctness and completeness, producing UML textual notations such as PlantUML and also transforming the models from those textual notations into visual diagrams (Cámara et al., 2023). Recent research has also evaluated LLM-assisted UML modeling in educational settings, showing that GPT models can aid students in producing class, use-case, and sequence diagrams, though evaluations often focus on syntactic correctness rather than semantic alignment (Al-Ahmad et al., 2025). Further, there are more specialized studies about how LLM systems can specifically generate UML class diagrams from natural language requirements, introducing architectures that decompose modeling tasks into subtasks such as entity extraction and relationship classification (De Bari et al., 2024; Giannouris & Ananiadou, 2025; Babaalla et al., 2025). LLMs can enrich UML class diagrams with interactions diagrams such as UML sequence diagram, also from natural language requirements (Ferrari et al., 2024).

Despite these advances and related works, according to our best knowledge, evaluation of completeness and correctness of transformation from UML use-case diagrams to class diagrams using LLM tools has not be addressed. UML use-case diagrams are first diagrams to be created during early phase of software design and requirements specification, then usually followed by UML class diagrams. Our hypothesis, in contrary to other works, are that UML use-case diagrams, with the assistance of LLM tools, can be directly used for UML class diagram generation. Our study empirically evaluates how LLM capabilities can be used to generate UML class diagrams from UML use-case diagrams, with respect to desired completeness and correctness.

III. THEORETICAL BACKGROUND

An UML use case diagram belong to the group of the UML behavioral diagrams. UML use case diagrams are primarily related to the presentation of the system functionality from the perspective of end users and other systems (so called, actors) and their interaction with the system. This facilitates communication between technical and non-technical participants in a software development process, as it allows an abstract and visual presentation of user requirements without going into technical details. UML use case diagram has several basic elements, namely actors, system, use cases and links. The links can be between use cases and actors, between actors, and between use cases. An actor is any person or system that uses or communicates with a system. A use case is an action that an actor or system needs to perform. The most common link between a use case and an actor is an association link. Generalization serves to connect two or more actors or two or more use cases and take on characteristics from a more abstract artifact. For example, Person is a generalization of Student, Login Use case is a generalization of PIN Based Login. «include» and «extend» are links between use cases used to show how use cases are related and how behavior is reused or conditionally added.

An UML class diagram is a static and structural diagram used to visualize the structure of object-oriented systems. It shows classes, their attributes, operations, and relationships between classes. There are two basic types of relationships between classes, namely association and generalization.

Association shows a relationship between two classes, for example <hasEnroled> can be relationship between Student and Course classes. Associations have multiplicities. Multiplicity can be specified in several ways, by an exact number, a range of numbers using two dots between pairs of numbers, or an indefinite number using an asterisk.

Generalization represents inheritance, i.e. when class A and class B share the same attributes or the same methods. Further, special types of associations are aggregation and composition. An aggregation a special form of association that represents a whole–part relationship, in which the observed class belongs to a relationship within some other class, i.e. one class contains others (parts). A part has an independent lifecycle and may exist separately from the whole. Composition is similar to aggregation, but denotes a stronger relationship of whole-part relationship, in which a part cannot exist without the whole; if the whole is destroyed, all its parts are destroyed.

Conceptually and from an UML expert perspective, a first step in generating a UML class diagram from a use case diagram is a detailed analysis of the use case diagram. The use case diagram defines the actors who communicate with the system, i.e. the nouns from the use case diagram most

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

often represent UML classes. The functionalities of the system are represented by use cases, i.e. verb actions that refer to methods of the UML classes. In addition to the use case diagram, there may be additional text descriptions that represent attributes of the UML classes. Based on the scenarios from the use case diagram, relationships between UML classes are defined. Correctly identified and defined relationships are key to the accuracy and expressiveness of the UML class diagram.

Large Language Models (LLMs) are advanced artificial intelligence systems designed to process, understand, and generate human/natural language-like text. LLMs are based on deep learning techniques and trained on huge datasets that typically contain billions of words from various sources such as websites, books, and articles. The way LLM tools work is that the user interacts with LLM with natural language queries in the form of questions, tasks, or descriptions, and the LLM generates responses in the form of text or code based on the context and learned patterns. Some of the most important and well-known LLMs are: OpenAI GPT series, Anthropic Claude, Meta LLaMA (and Gemma), Google PaLM/Gemini, and BLOOM.

IV. METHODOLOGY

In this work, to generate UML class diagrams from UML use case diagrams, with assistance of an LLM, one of the most well-known and most widely used LLMs for generating UML diagrams was chosen - the OpenAI GPT with ChatGPT chatbot. More specifically, a free version of GPT-4o, an advanced multimodal version that generates text, understands images and audio, and responds to complex queries, was used.

As a sample use-case diagram used for experimentation and evaluation, the process of creating the final thesis at the Aspira University of Applied Science is used. The process in described in the beginning of next chapter.

The use-case diagram, as a base for UML class diagram creation, is created in three ways: (1) by ChatGPT that was given only an instruction to generate a use-case diagram for the process of creating the final thesis at the Aspira, without additional details, (2) by human expert in the UML modeling software tool and (3) by human expert in the form of text document.

More precisely, in the first case, ChatGPT was given the command to create a use case diagram, and then it generated a textual description of the diagram and the PlantUML (https://plantuml.com/) code to visualize diagram. ChatGPT is then instructed to create a class diagram from the previously described use case diagram, and it generates a textual description of the class diagram and the PlantUML code for visualizing the diagram. In addition to specifying a text command to create a class diagram, it is possible to load an image of a use case diagram obtained from the PlantUML code and then issue a command to create a class diagram.

In second case, ChatGPT was given an UML use case diagram image previously created by a human UML expert, with a command to ChatGPT to generate an UML class diagram for the loaded image.

In the third case, ChatGPT is provided with a text document describing a UML diagram of a use case previously created by a human UML expert, instructing ChatGPT to generate a UML class diagram for the loaded document.

The resulting UML class diagrams were compared with the loaded use case diagrams and with the UML class diagrams is generated manually based on the use case diagrams. Actor from the use case diagrams are compared with corresponding UML classes, use cases are compared with the generated methods within classes, and links from the use case diagrams are compared with the relationships between classes.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

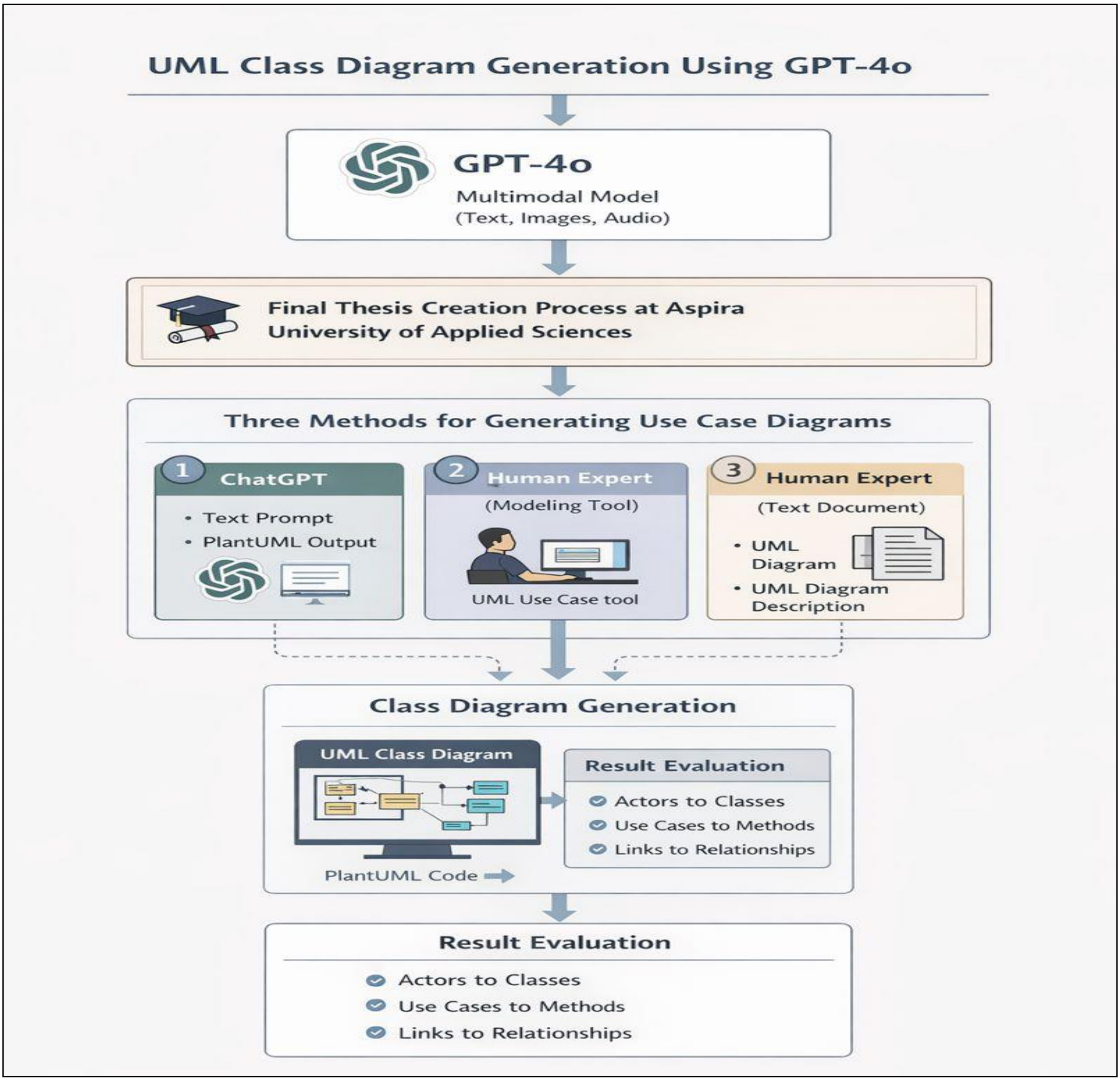


Fig 1 Methodology Overview Created in ChatGPT

V. CASE STUDY AND EXPERIMENT

At Aspira University of Applied Science, the process of creating the final thesis begins with a student selecting and submitting a thesis topic, usually in coordination with a proposed mentor. After the topic is approved by the academic and administrative office, the student develops the thesis under the mentor's supervision. Once completed, the thesis is submitted for evaluation, reviewed by a committee, and, if necessary, revised. The process concludes with the thesis defense and final approval, after which the thesis is officially archived. It is a medium-complexity system example and as such representative for the evaluation.

As stated previously, the use case diagram for creating a final thesis at Aspira Polytechnic was created by ChatGPT in three ways: (1) by giving the command to ChatGPT to create a use case diagram for creating a final thesis at Aspira Polytechnic, (2) by uploading an image of the diagram that was created in the Modelio modeling tool and (3) by uploading text document with textual description of creating a final thesis at Aspira Polytechnic.

In first case, after ChatGPT was given the command to create a use case diagram of the thesis production process at Aspira Polytechnic, it generated a description of the diagram and code that is implemented in the PlantUML diagram generation tool. This process is shown with a UML use case diagram, visible in Figure 2, that is created by ChatGPT.

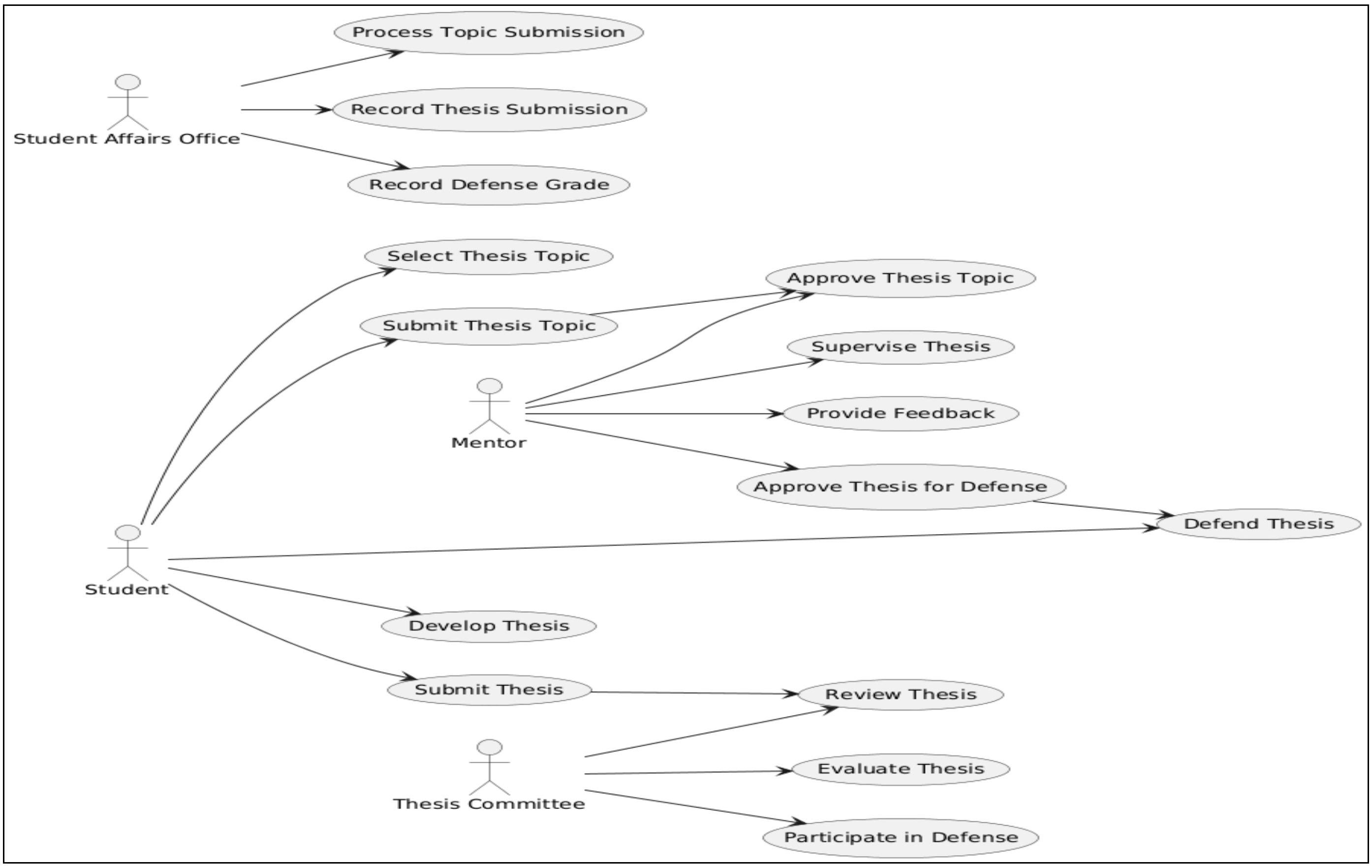


Fig 2 UML Use-Case Diagram Created by ChatGPT

Using ChatGPT, a class diagram was created from the use case diagram (command: create UML class diagram from the use case diagram). The class diagram is created in two ways by generating the code or directly generating the image in ChatGPT. Figure generated directly in ChatGPT is provided below.

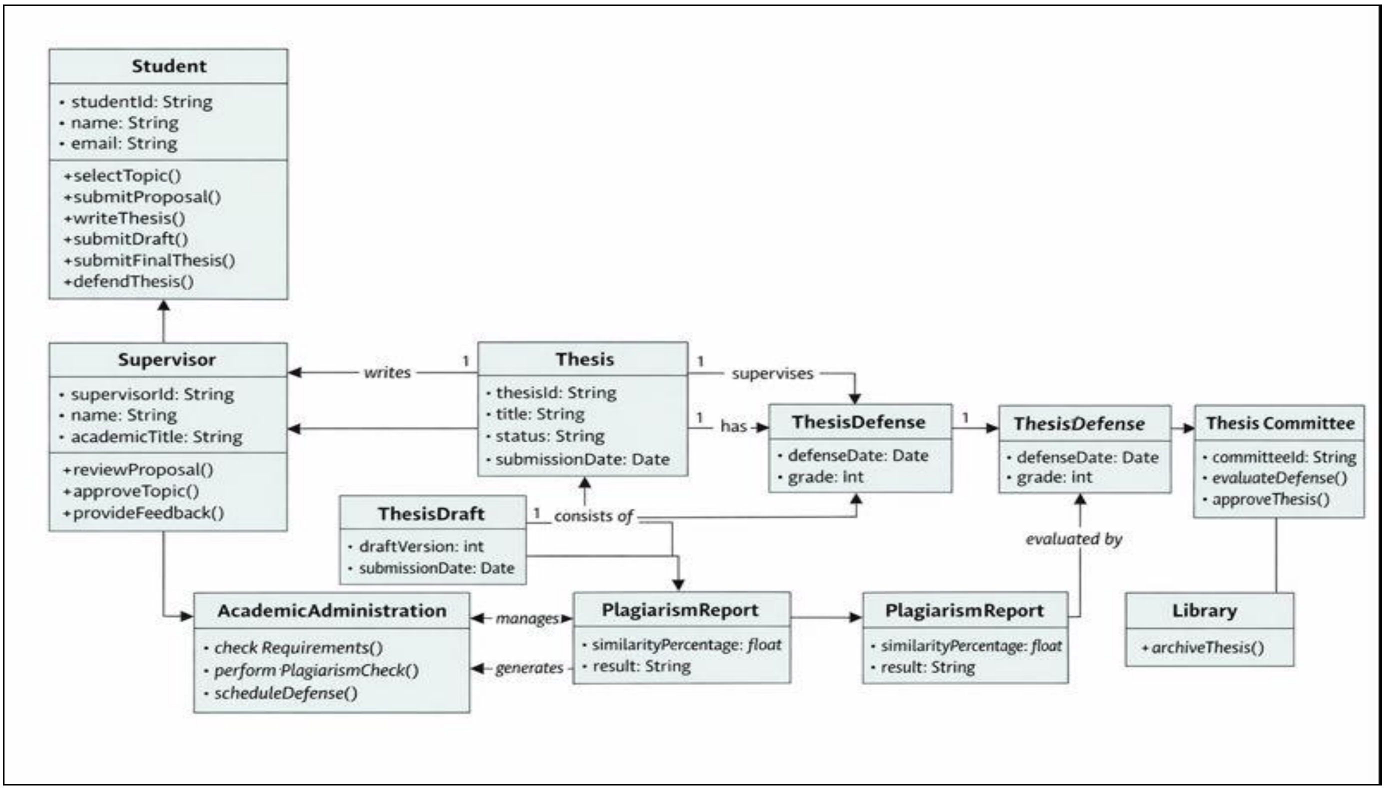


Fig 3 UML Class Diagram Generated from UML Use-Case Diagram

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

ChatGPT also generated PlantUML code, shown in Listing below.

```
@startuml

class Student {
  studentId
  firstName
  lastName
  email
  studyStatus
  selectTopic()
  submitTopic()
  developThesis()
  submitThesis()
  defendThesis()
}

class Supervisor {
  supervisorId
  firstName
  lastName
  title
  approveTopic()
  monitorThesis()
  giveGuidelines()
  approveThesisForDefense()
}

class Topic {
  topicId
  title
  description
  status
  approve()
  reject()
}

class Thesis {
  thesisId
  title
  description
  applicationDate
  submissionDate
  status
  changeStatus()
}

class Committee {
  committeeId
  name
  reviewThesis()
  evaluateThesis()
  participateInDefense()
}

class Defense {
  defenseId
  date
  location
  grade
  scheduleDefense()
}

class StudentOffice {
  officerId
  firstName
  lastName
  recordTopic()
  recordSubmission()
  recordGrade()
}

Student "1" -- "1" Topic
Student "1" -- "1" Thesis
Supervisor "1" -- "*" Thesis
Topic "1" -- "1" Supervisor
Thesis "1" -- "1" Defense
Committee "1" -- "*" Defense

@enduml
```

UML class diagram generated from code in the PlantUML tool that was provided by ChatGPT is given in Figure 4.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

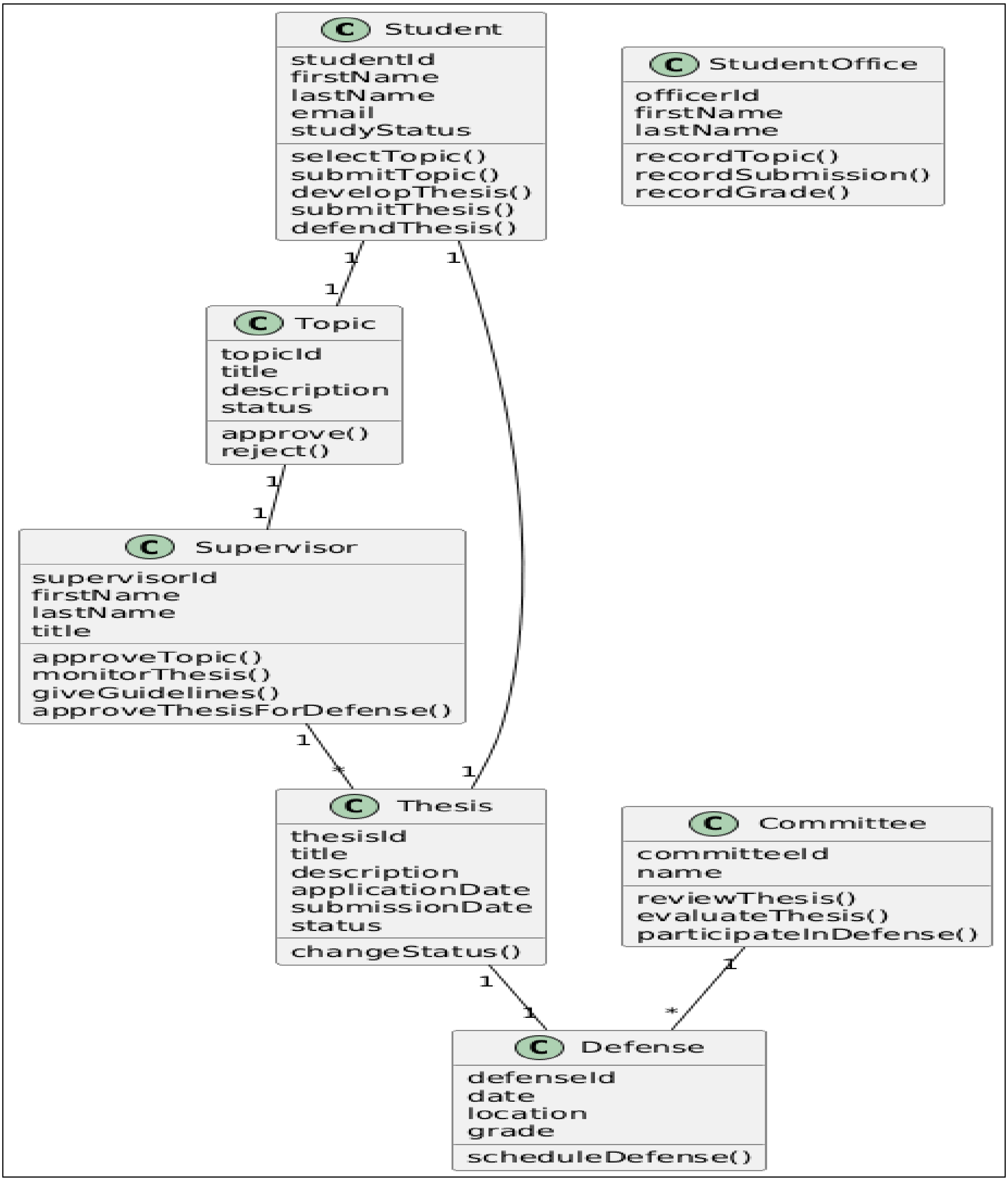


Fig 4 UML Class Diagram Generated from PlantUML of ChatGPT

From the Figure 4, it can be observed that the UML class diagram that is generated from the code has a StudentOffice class that is not connected to other diagram classes, although in the process of the final paper, the student department communicates with the student because the topic selection takes place through it. Apart from the unrelated StudentOffice class, it is evident that there is no connection between the mentor and the thesis committee, and that there is no connection between the topic and the thesis.

Further instruction was given to ChatGPT - that the mentor is a member of the committee, that the topic and final paper should be connected and the StudentOffice and topic should be connected. The new UML class diagram created is shown in Figure 5.

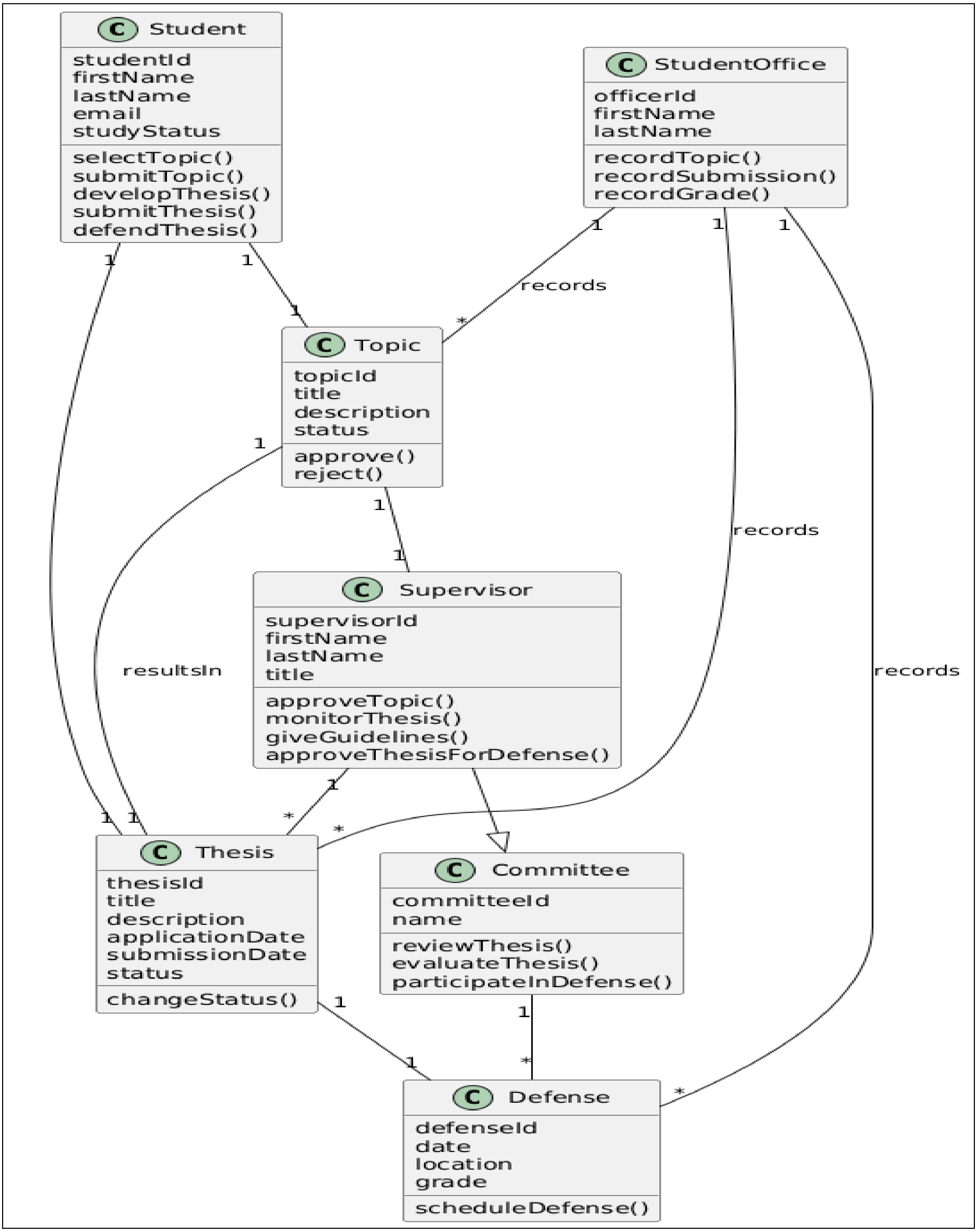


Fig 5 UML Class Diagram Refined~1 by ChatGPT

Then, the instruction was given to optimize the diagram and adapt it to the process of creating the final thesis at the Aspira. New diagram is shown in Figure 6.

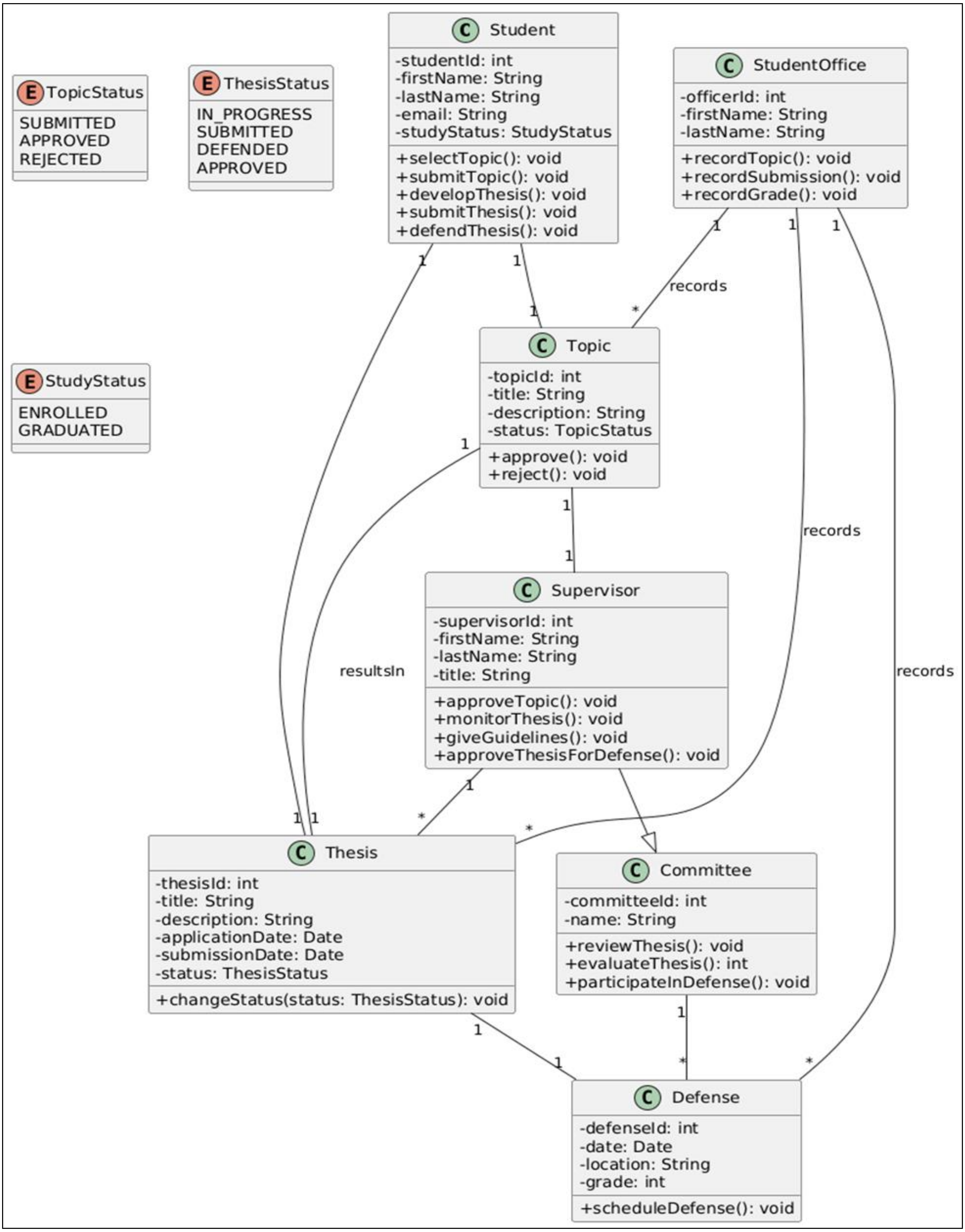


Fig 6 UML Class Diagram Refined~2 by ChatGPT

By continuing the conversation with ChatGPT and repeating the command to create a UML use case diagram for the process of creating a final thesis at Aspira, the following diagram was generated (Figure 7).

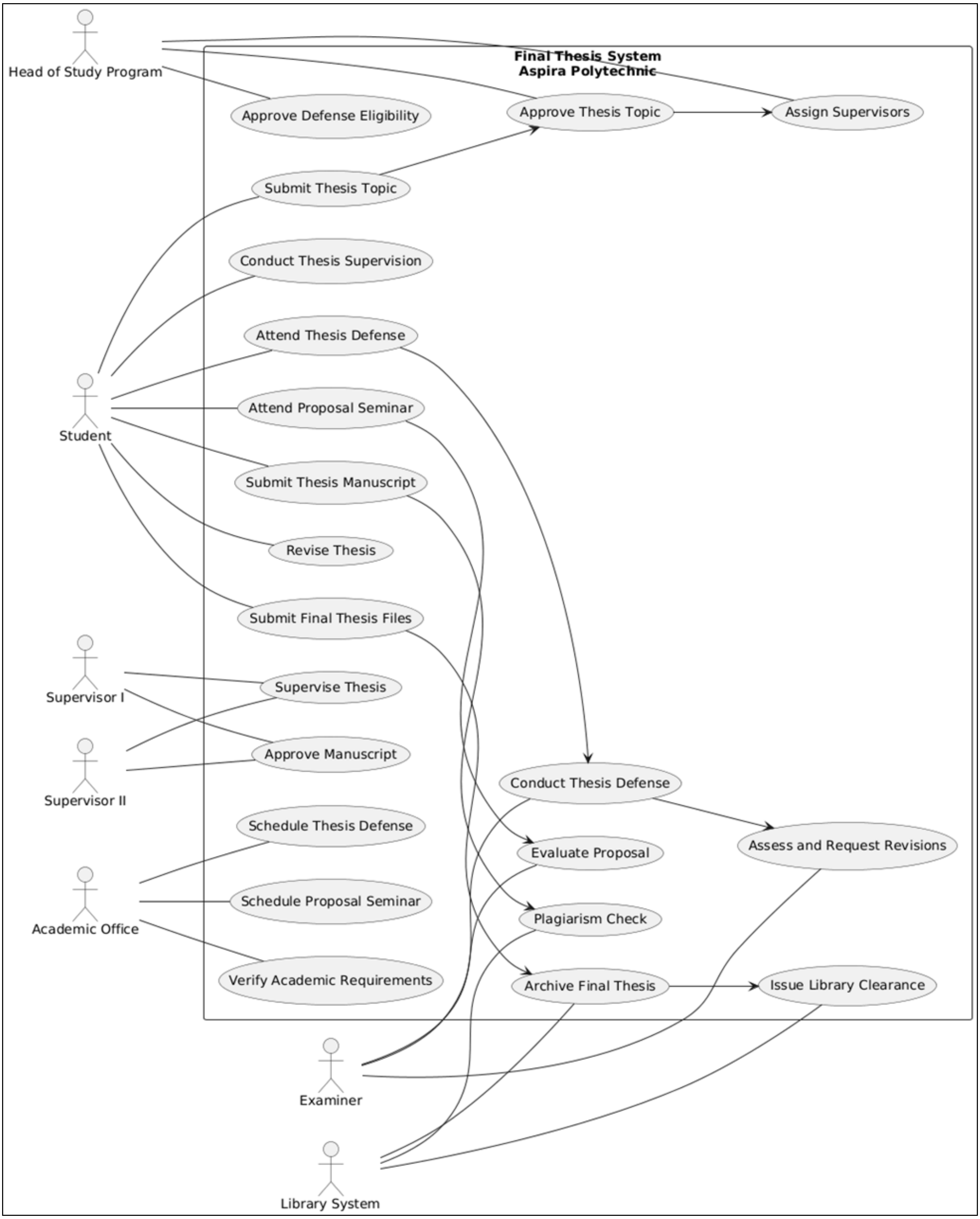


Fig 7 UML Use-Case Diagram by ChatGPT, Specific to Aspira

From UML use-case shown in Figure 7, by giving an command: create UML class diagram from the use case diagram, an UML class diagram was created by ChatGPT and it is shown in Figure 8.

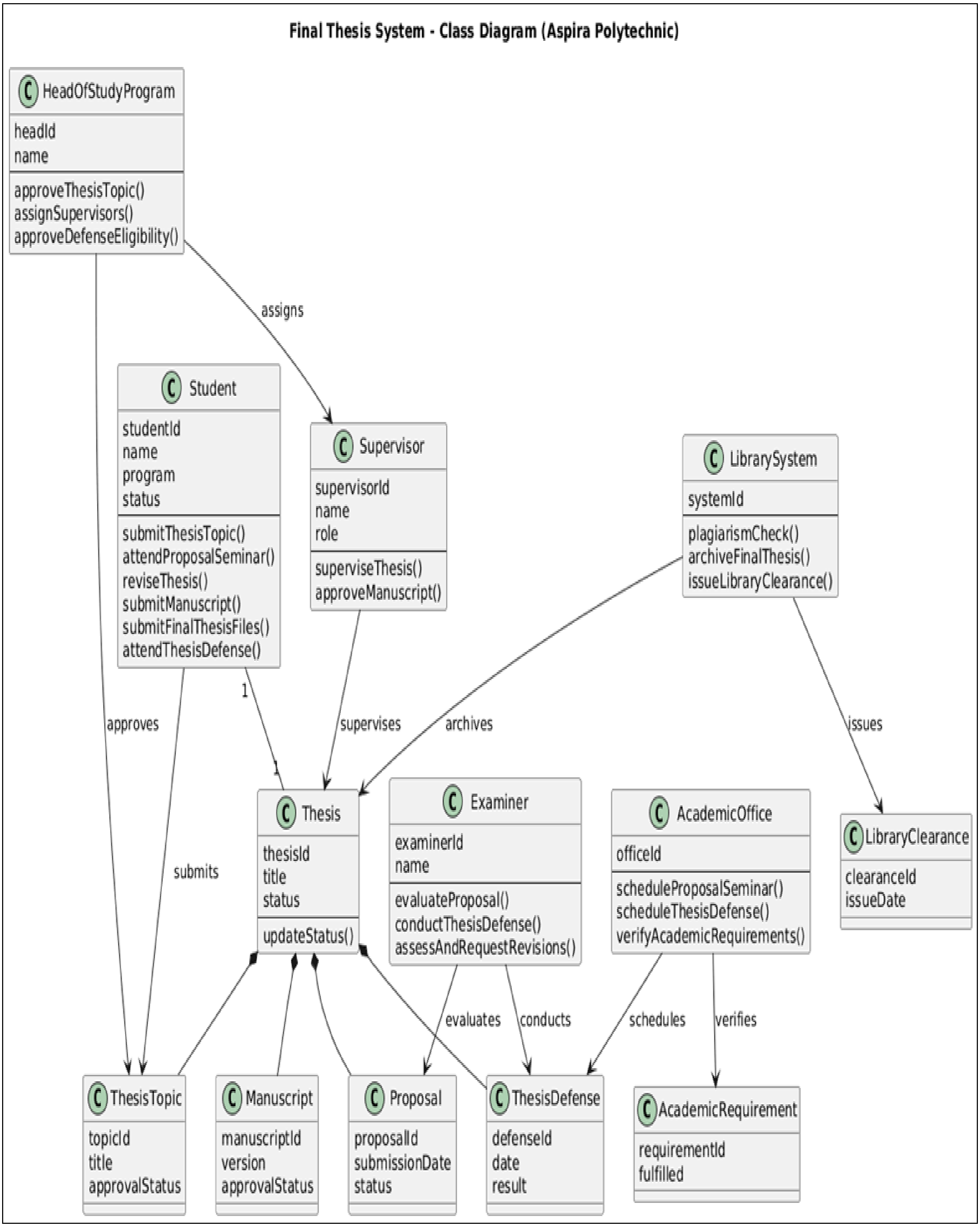


Fig 8 UML class diagram generated by ChatGPT, from use case in figure 7

Another experiment was to upload an image of the use case diagram into ChatGPT which was created in modeling tool Modelio by expert. This input is shown in Figure 9.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

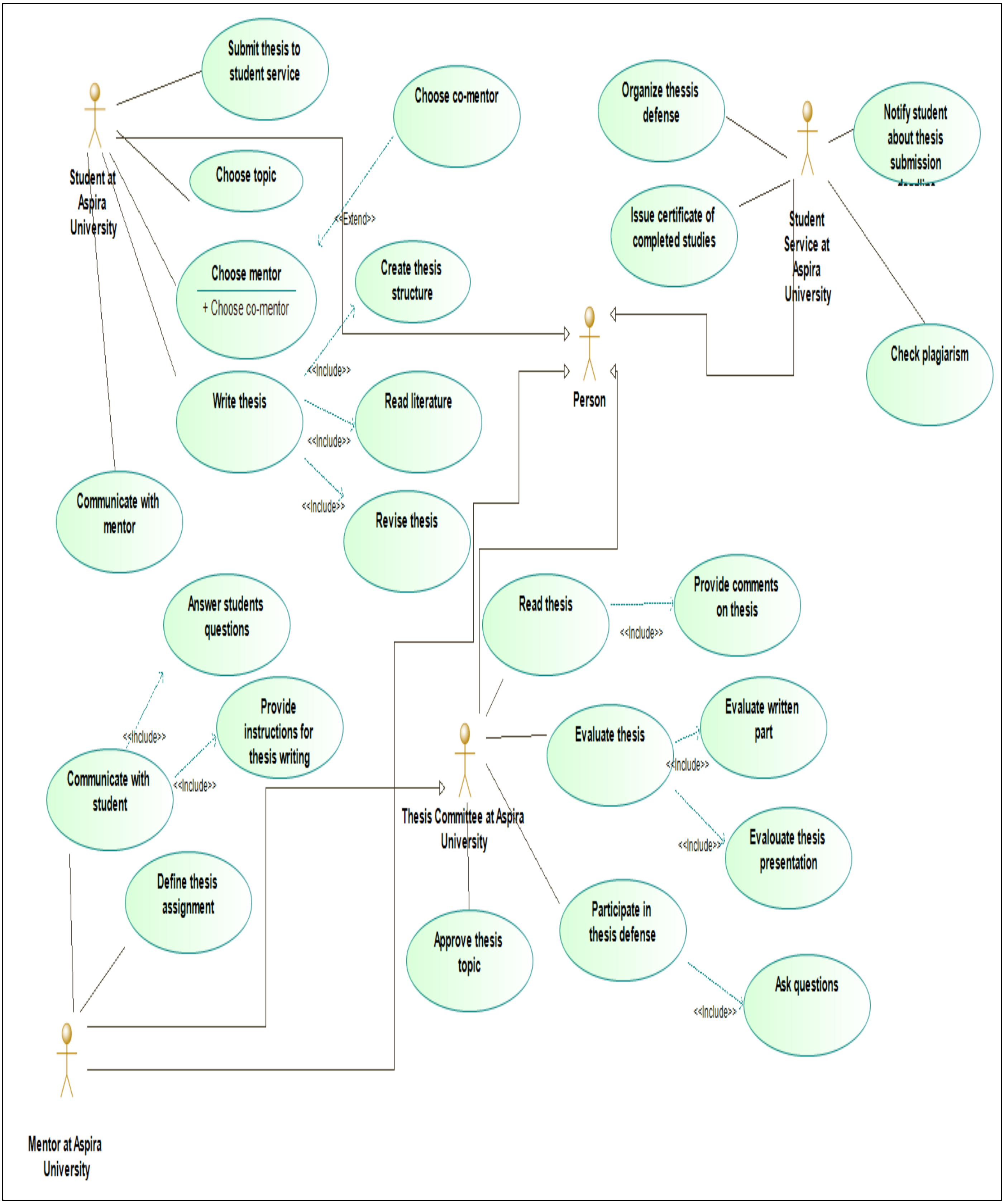


Fig 9 UML Use-Case Diagram Created by Expert

From the image of the use case diagram (Figure 9), a UML class diagram was generated in ChatGPT (Figure 10).

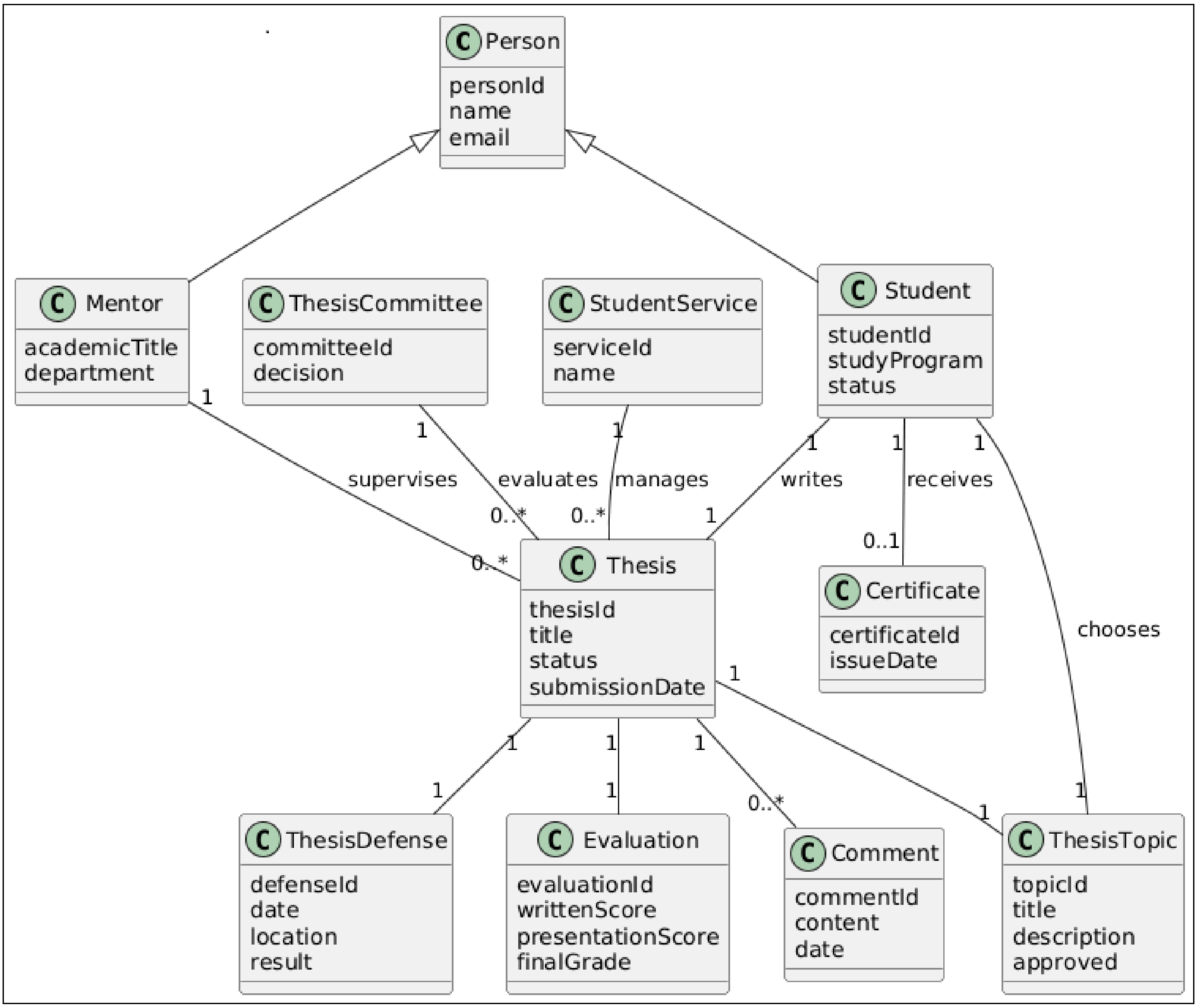


Fig 10 UML Class Diagram Created by ChatGPT from Figure 9

By providing instructions that the attribute visibility is missing, that the mentor is a member of the Thesis Committee and that the StudentService and ThesisCommitte are Person, ChatGPT generated PlantUML code, shown in Listing 2.

```
@startuml
skinparam classAttributeIconSize 0
class Person {
  -personId : int
  -name : String
  -email : String}

class Student {
  -studentId : int
  -studyProgram : String
  -status : String}
class Mentor {
  -academicTitle : String
  -department : String}
class StudentService {
  -serviceId : int
  -officeName : String}
class ThesisCommittee {
  -committeeId : int
  -decision : String}
class Thesis {
```

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

```
  -thesisId : int
  -title : String
  -status : String
  -submissionDate : Date}
class ThesisTopic {
  -topicId : int
  -title : String
  -description : String
  -approved : boolean}
class ThesisDefense {
  -defenseId : int
  -date : Date
  -location : String
  -result : String}
class Evaluation {
  -evaluationId : int
  -writtenScore : int
  -presentationScore : int
  -finalGrade : int}
class Comment {
  -commentId : int
  -content : String
  -date : Date}
class Certificate {
  -certificateId : int
  -issueDate : Date}
'================ INHERITANCE ================
Person <|-- Student
Person <|-- Mentor
Person <|-- StudentService
Person <|-- ThesisCommittee

'================ RELATIONSHIPS ================
Student "1" -- "1" Thesis : writes
Student "1" -- "1" ThesisTopic : chooses
Mentor "1" -- "0..*" Thesis : supervises
Mentor "1" -- "0..*" ThesisCommittee : member of
Thesis "1" -- "1" ThesisTopic
Thesis "1" -- "1" ThesisDefense
Thesis "1" -- "1" Evaluation
Thesis "1" -- "0..*" Comment
ThesisCommittee "1" -- "0..*" Thesis : evaluates
StudentService "1" -- "0..*" Thesis : manages
Student "1" -- "0..1" Certificate : receives
@enduml
```

UML class diagram generated from code in the PlantUML tool that was provided by ChatGPT is given in Figure 11.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

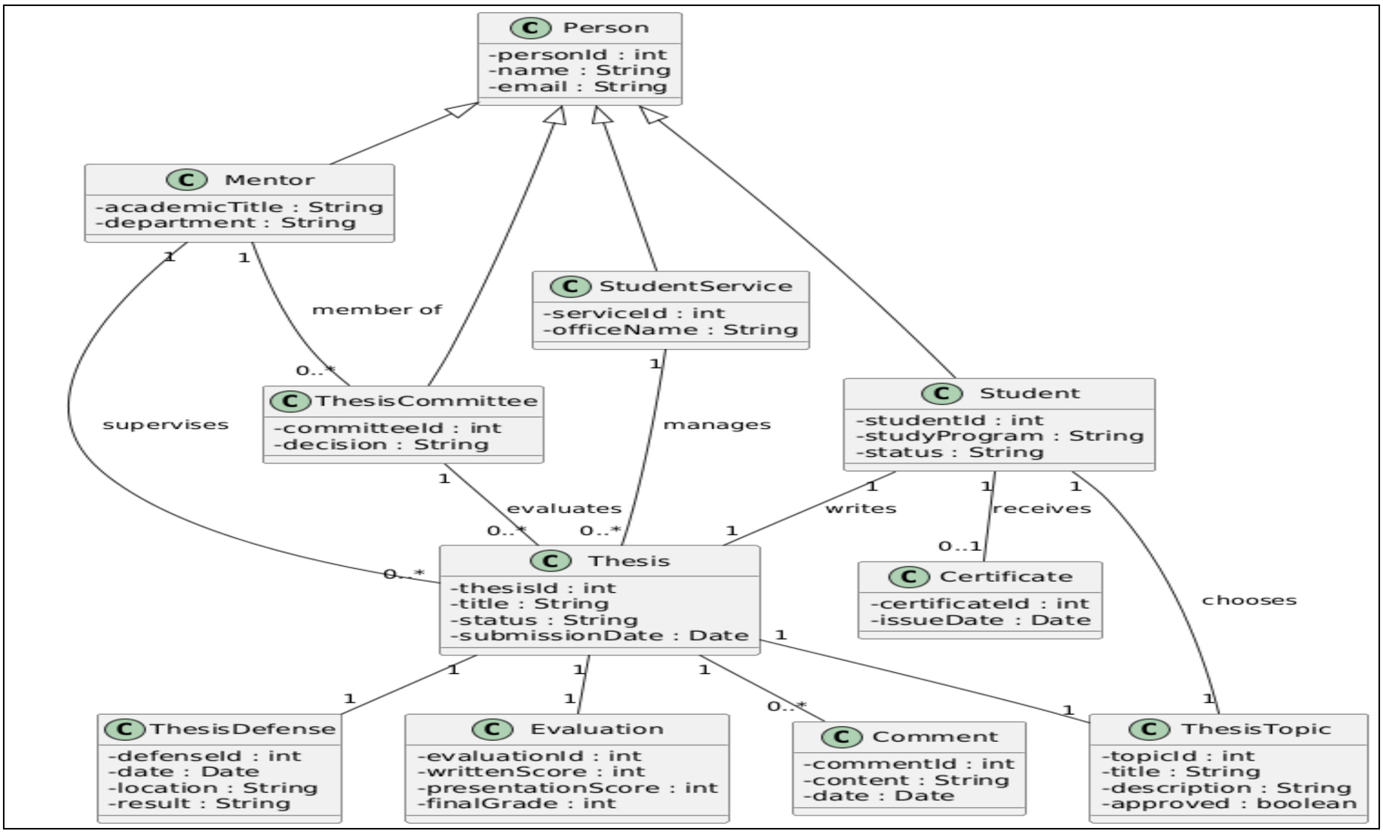


Fig 11 UML Class Diagram Created by ChatGPT

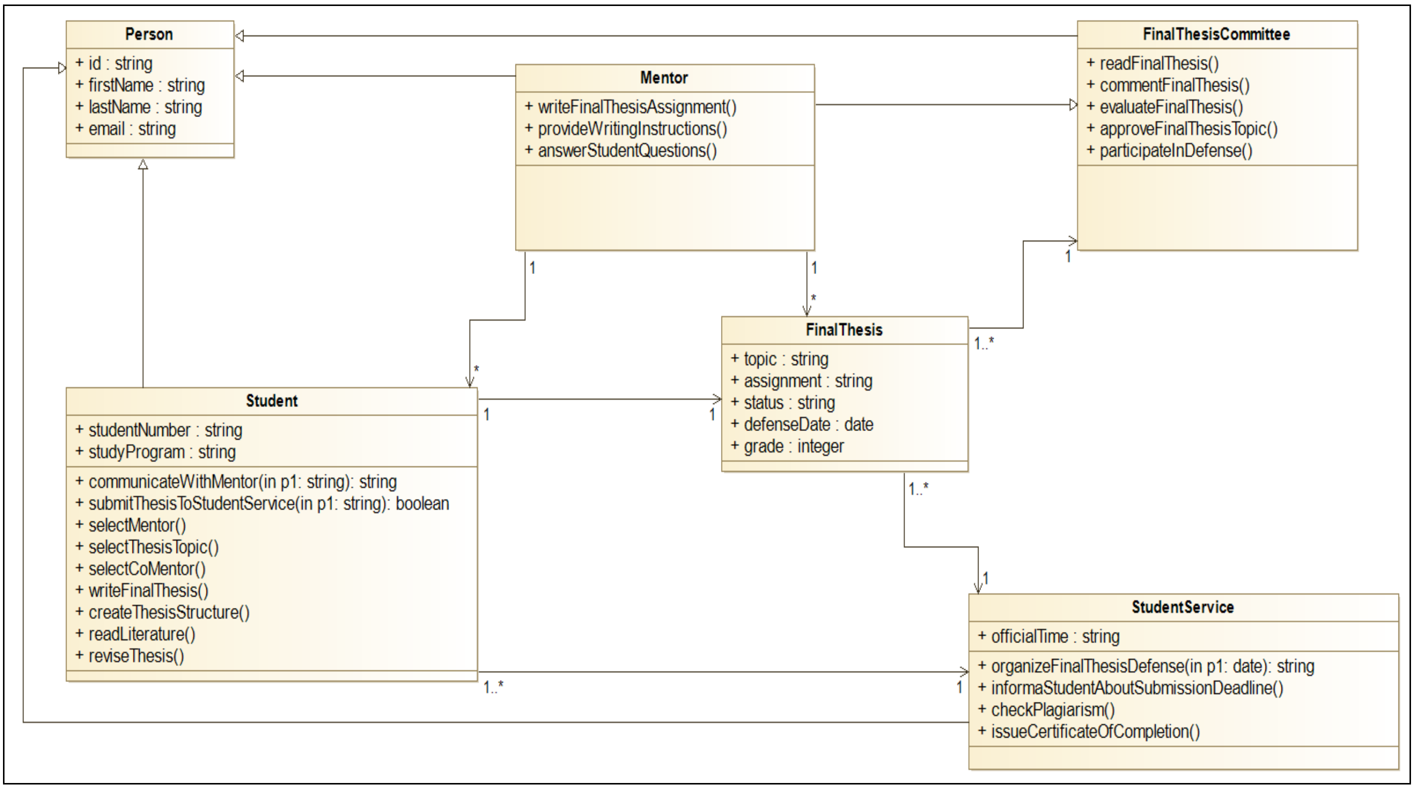
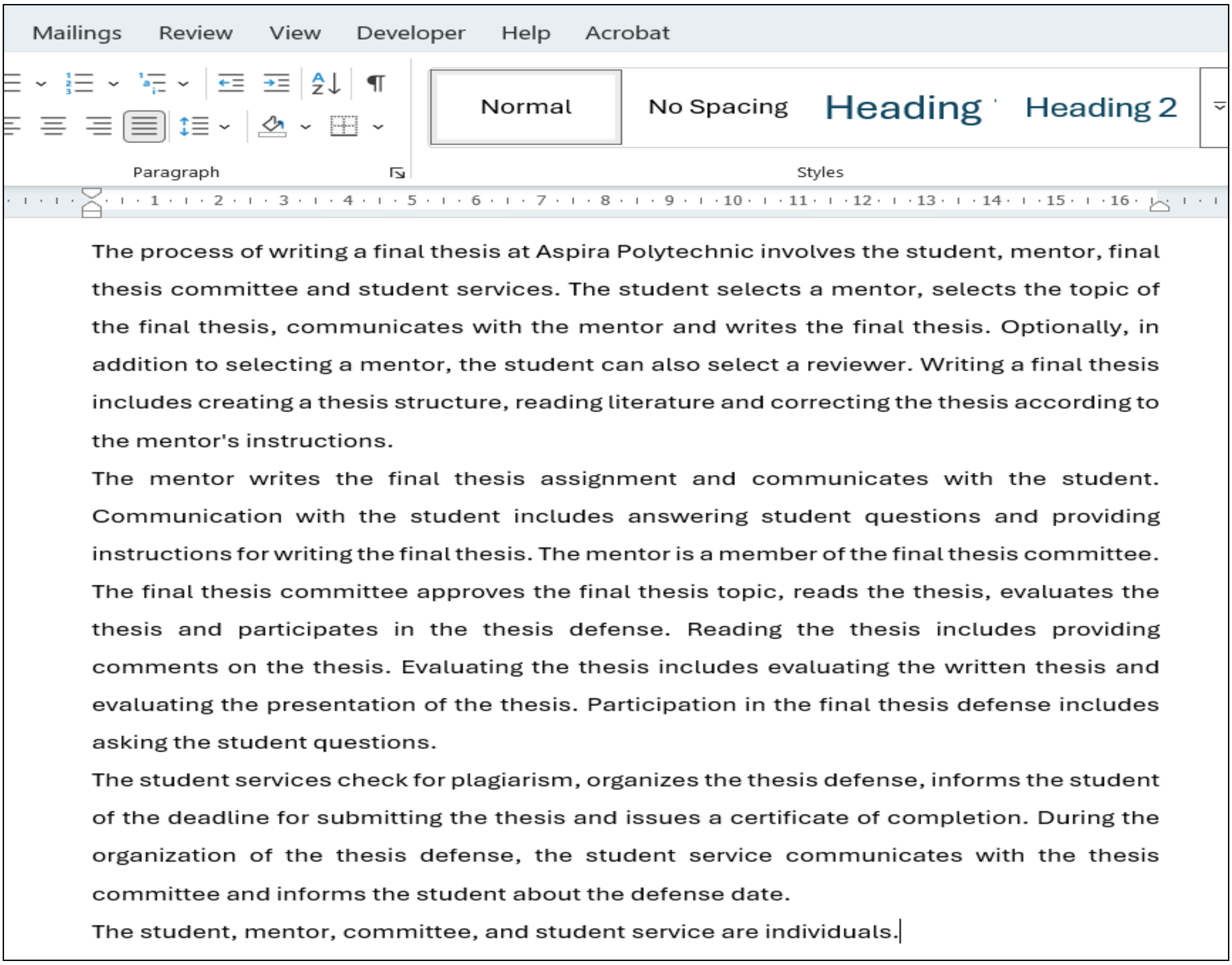


Fig 12 UML Class Diagram Created from UML Use Case in Figure 9 Created by Expert

The third experiment was to load a text document with a use case description into ChatGPT. This input is shown in Figure 13.

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576



The process of writing a final thesis at Aspira Polytechnic involves the student, mentor, final thesis committee and student services. The student selects a mentor, selects the topic of the final thesis, communicates with the mentor and writes the final thesis. Optionally, in addition to selecting a mentor, the student can also select a reviewer. Writing a final thesis includes creating a thesis structure, reading literature and correcting the thesis according to the mentor's instructions.

The mentor writes the final thesis assignment and communicates with the student. Communication with the student includes answering student questions and providing instructions for writing the final thesis. The mentor is a member of the final thesis committee. The final thesis committee approves the final thesis topic, reads the thesis, evaluates the thesis and participates in the thesis defense. Reading the thesis includes providing comments on the thesis. Evaluating the thesis includes evaluating the written thesis and evaluating the presentation of the thesis. Participation in the final thesis defense includes asking the student questions.

The student services check for plagiarism, organizes the thesis defense, informs the student of the deadline for submitting the thesis and issues a certificate of completion. During the organization of the thesis defense, the student service communicates with the thesis committee and informs the student about the defense date.

The student, mentor, committee, and student service are individuals.

Fig 13 Text Document with a Use Case Description Created by Expert

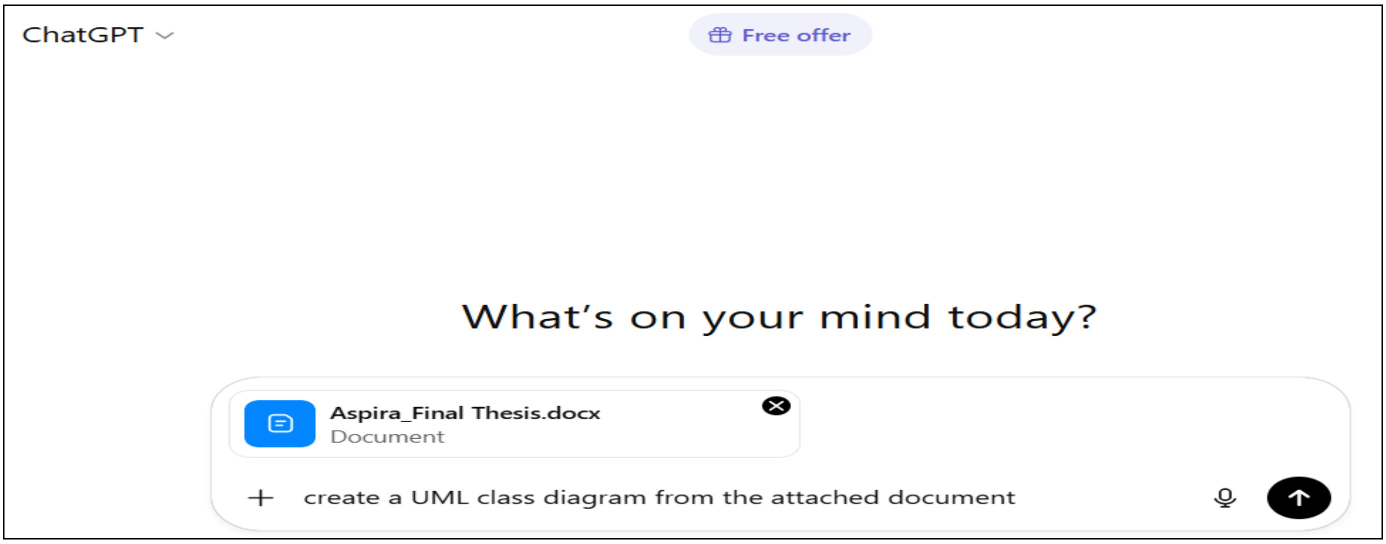


Fig 14 Uploading Document to ChatGPT and Command to Create UML Class Diagram

From the text document (Figure 13), ChatGPT generated use case diagram (Figure 15).

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

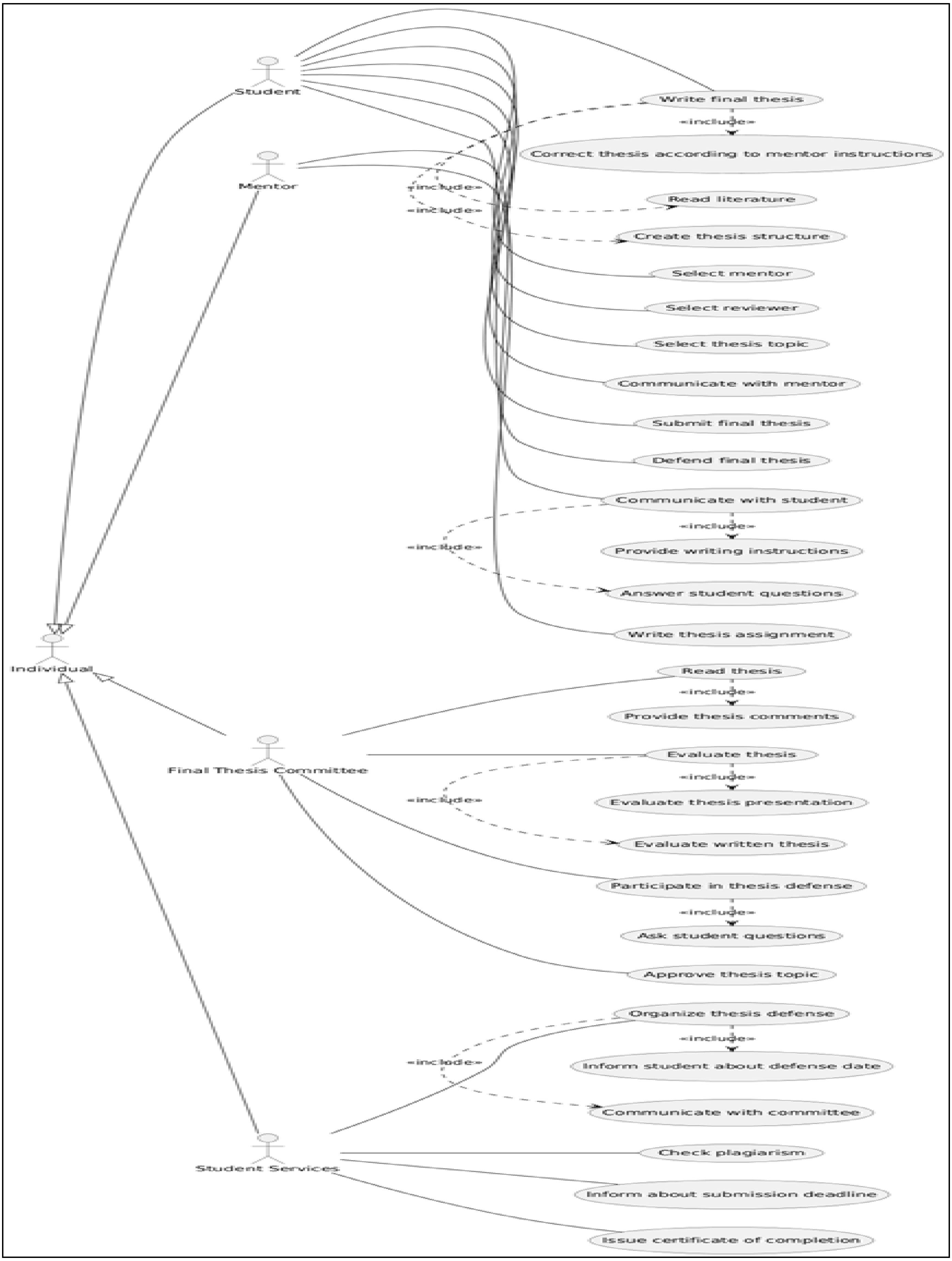


Fig 15 UML Use Case Diagram Generated by ChatGPT, from Text Document Figure 13

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

UML class diagram generated from code in the PlantUML tool that was provided by ChatGPT is given in Figure 15.

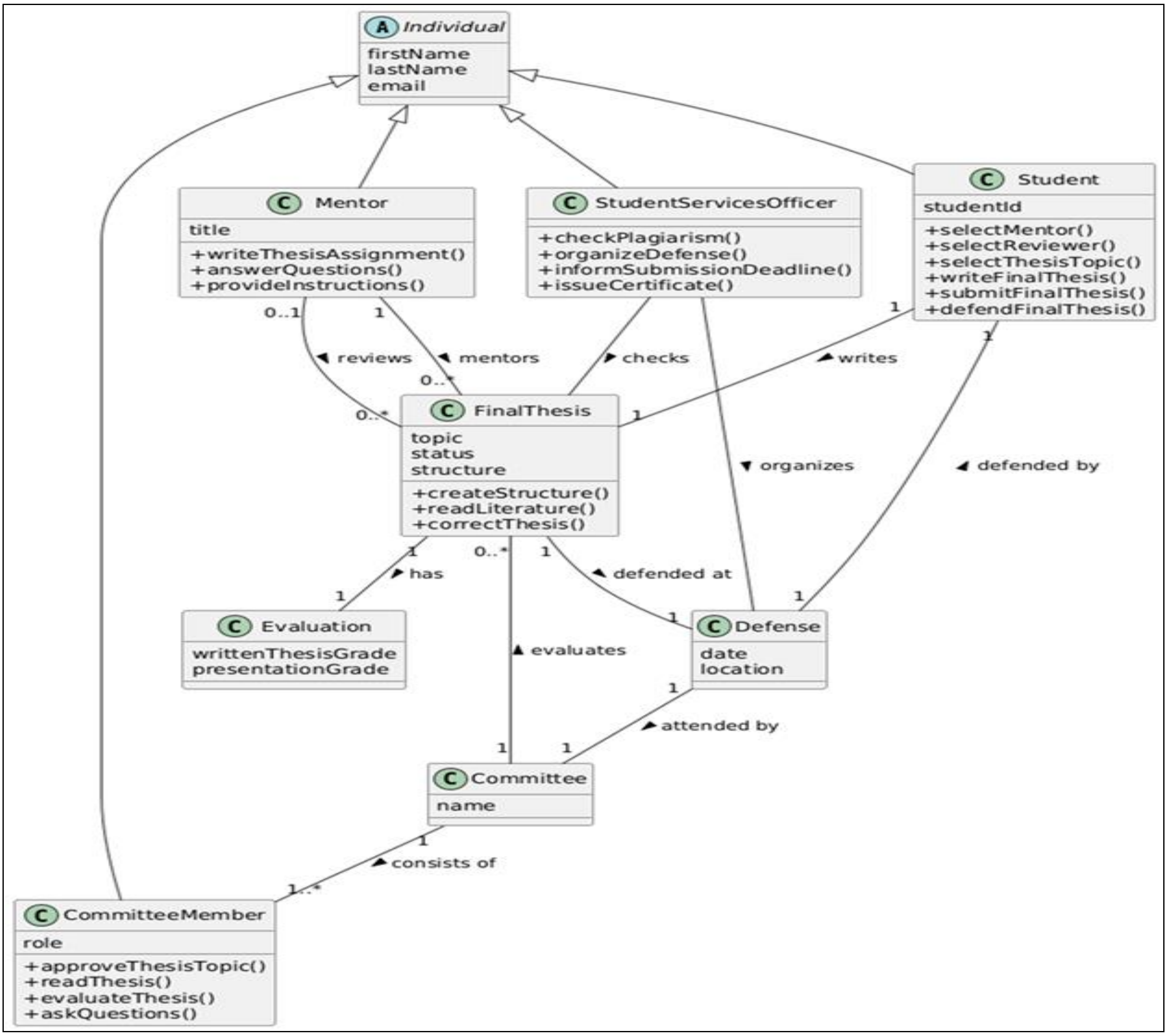


Fig 16 UML Class Diagram Created from PlantUML Code by ChatGPT

VI. RESULTS AND DISCUSSION

As described in the previous chapter, as a starting point for creating a use case diagram of the process of creating a final thesis at Aspira Polytechnic in ChatGPT, three methods were used: giving a command to ChatGPT to create a use case diagram, loading an image of the use case diagram created by a modeler in the modeling tool, and by loading a text document with a description of the participants and links in the use case diagram of the process of creating a final thesis at Aspira Polytechnic. ChatGPT described the obtained diagrams and generated PlantUML code for diagram visualization. After receiving the command to create the class diagram, ChatGPT generated a textual description and PlantUML code to visualize the diagram as described in the previous chapter.

ChatGPT was prompted to create a use case diagram to show the process of creating a final thesis at Aspira Polytechnic. The result obtained after that command is a universal use case diagram for creating the final thesis, it is not specific to Aspira Polytechnic.

From the use case diagram, a class diagram was generated as an image directly in ChatGPT and as code that needs to be implemented in the PlantUML program tool. The image generated by ChatGPT and the diagram obtained in PlantUML are different.

The generated image shows UML classes from the Plagiarism process that do not exist in the UML use case diagram. There is no connection between the Topic/Paper and the Student that should exist because one student

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

chooses one topic and only one final paper can be generated from one topic. There is a Library class that does not exist in the use case diagram. There is also a lack of relationship between the StudentService and the Student.

In the class diagram generated from the code, the StudentService class was left unconnected to other classes. ChatGPT explains this case as follows: "Student service does not have direct business responsibility over the objects, but it has a relationship because it performs administrative actions (records) over them. This is a frequent didactic mistake: Student service is an actor in the use case diagram, but in the class diagram it is often not a domain class, but: a system / service class or an administrative user."

After being instructed to include the student service class in the diagram, a new diagram was generated. Also, the diagram did not show that the mentor was a member of the committee, and the topic and thesis classes were not interconnected. By receiving these instructions, ChatGPT corrected the diagram and was then instructed to optimize the diagram and adapt it to the process of creating a thesis at Aspira Polytechnic. His response was that he would do so, but he generated the same diagram. The command was repeated several times and the result was the same. After noting that this was not a diagram specific to the processes at Aspira Polytechnic, ChatGPT wrote: "I understand — thanks for the clarification. You are right: the previous diagram uses generic names, not the actual terminology of Aspira Polytechnic. Below is a class diagram that uses the names most commonly used in Aspire practice and documentation (thesis, mentor, committee)." ChatGPT generated the same diagram again.

In the continuation of the conversation, the order to create a Use case diagram for the process of creating the final thesis at Aspira Polytechnic was repeated, and ChatGPT created a use case diagram that is different from the result obtained during the first order in the same conversation. From the generated use case diagram, a class diagram was created, which is also different from the first class diagram from the mentioned conversation, but in this diagram, the class StudentOffice (called AcademicOffice in the diagram) is connected to another class, and not left disconnected as in the first example. When repeating orders, it is evident that he often calls the same participants by different names, e.g. Aspira Polytechnic or Aspira University. The generated diagram does not accurately represent all the classes at the Aspira Polytechnic, and human intervention is required in order to generate a class diagram that corresponds to the processes for creating the final thesis in Aspira. By repeating commands within the same conversation, it is evident that it gives different results, ChatGPT learns and implements the acquired knowledge within the same conversation.

By loading an image of a use case diagram created by a expert into a modeling tool, ChatGPT accurately describes the image. The image name does not affect the description of the image content. The created class diagram lacks the visibility of attributes and the input and output parameters of operations that were supplemented after the command was given. In the class diagram that was created, the committee and the mentor are not connected, although the mentor is a member of the committee and are connected by a generalization link in the use case diagram. As an explanation, ChatGPT states: "generalization in the use case diagram is not automatically transferred as the same link to the class diagram." As in the previous example, the class diagram generated from the code is not identical to the class diagram generated as an image directly in ChatGPT.

By comparing the class diagram done by ChatGPT and a human for the same use case diagram, the difference in the way the classes are defined is visible. The class diagram generated by ChatGPT has more classes than the class diagram created manually by a human in a modeling tool. While the man showed all the use cases from the use case diagram as methods in the class diagram, ChatGPT did not generate methods in classes, but showed them in the form of classes, for example, the method commenting on the work, what the committee does, is shown as a class comment. In the handmade model, the generalization connection from the use case diagram is also visible in the class diagram, while in the case of the class diagram generated by ChatGPT it is not visible at all or is not visible among all entities as in the use case diagram.

In the third example, the data for creating the use case diagram in ChatGPT is loaded in the form of a text document written by a human. ChatGPT correctly interprets the content of the document, and the only problem in this example was the extension link. The name of the text document does not affect the interpretation of the content of the document. ChatGPT generates the code for the use case diagram from the description, and then after receiving the command it also generates the class diagram. He transferred the connection of generalization from the use case diagram to the classes, for example, student and mentor are persons. Although there are more classes than a manually created diagram, in this example it converts the use cases from the use case diagram into methods in the class diagram.

The tested examples, the results cannot meet the specific requirements of individual processes, i.e. they cannot show the specific process of creating a final thesis at the Aspira Polytechnic without human intervention.

The question arises whether the acquired knowledge is implemented within only the same conversation or applied to other conversations, whether it can generate the same result for the same command within the same or different conversations.

VII. CONCLUSION

This research examined the capability of Large Language Models to automate the transformation of UML use-case diagrams into UML class diagrams, with a particular focus on consistency, correctness, and domain specificity. The results demonstrate that LLMs can almost successfully identify core domain entities, infer basic class structures, and significantly reduce the manual effort

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/26jan1576

required during early-stage system modeling. As such, LLMs represent a valuable support tool for software engineers, especially in exploratory and preliminary design phases.

However, the findings also reveal important limitations. The generated class diagrams often exhibited inconsistencies, omitted or incorrect relationships, and the introduction of classes not present in the original use-case diagrams. Ambiguities in requirements were frequently misinterpreted, and semantic relationships—such as generalizations or domain-specific constraints—were not reliably preserved when transitioning from use-case to class diagrams. Furthermore, repeated prompts within the same or different conversations produced varying results, highlighting challenges related to determinism, reproducibility, and the persistence of learned context.

The case study centered on the thesis creation process at Aspira Polytechnic further emphasized that LLMs tend to generate generic, institution-agnostic models unless guided extensively by domain experts. Even with iterative prompting, the models struggled to fully adapt diagrams to specific organizational processes and terminology without human-expert correction. Differences between diagram representations generated as images and those produced via code (e.g., PlantUML) also underscore current limitations in coherence across output formats.

Overall, while LLM technologies show strong potential as an assistive mechanism for UML class diagram generation, they cannot yet replace human expertise in ensuring semantic accuracy, domain alignment, and modeling completeness. Human intervention remains essential to validate, refine, and contextualize the generated models. Future research should focus on improving consistency across interactions, enhancing domain adaptation, and exploring mechanisms for persistent learning across conversations to increase the reliability of LLM-assisted software modeling.

REFERENCES

[1]. Object Management Group, 2017. *Unified Modeling Language (UML), version 2.5.1.* https://www.omg.org/spec/UML/2.5.1/About-UML

[2]. Rouabhia, D., & Hadjadj, I., 2024. *Enhancing class diagram dynamics: A natural language approach with ChatGPT.* arXiv. https://arxiv.org/abs/2406.11002

[3]. Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K.J., Ajagbe, M.A., Chioasca, E.V. and Batista-Navarro, R.T., 2021. Natural language processing for requirements engineering: A systematic mapping study. ACM Computing Surveys (CSUR), 54(3), pp.1-41.

[4]. Cámara, J., Troya, J., Burgueño, L. and Vallecillo, A., 2023. On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. *Software and Systems Modeling*, *22*(3), pp.781-793.

[5]. Al-Ahmad, B., Alsobeh, A., Meqdadi, O. and Shaikh, N., 2025. A Student-Centric Evaluation Survey to Explore the Impact of LLMs on UML Modeling. *Information*, *16*(7), p.565.

[6]. De Bari, D., Garaccione, G., Coppola, R., Torchiano, M. and Ardito, L., 2024, October. Evaluating large language models in exercises of uml class diagram modeling. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 393-399).

[7]. Giannouris, P. and Ananiadou, S., 2025. NOMAD: A Multi-Agent LLM System for UML Class Diagram Generation from Natural Language Requirements. *arXiv preprint arXiv:2511.22409*.

[8]. Babaalla, Z., Jakimi, A. and Oualla, M., 2025. LLM-Driven MDA Pipeline for Generating UML Class Diagrams and Code. *IEEE Access*.

[9]. Ferrari, A., Abualhaija, S. and Arora, C., 2024, June. Model generation with LLMs: From requirements to UML sequence diagrams. In *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)* (pp. 291-300). IEEE.