

MET CS682 ASSIGNMENT 5

The purpose of this exercise is to give you practice performing system design and using UML.



February 15, 2024

Mahim Choudhury

Boston University

1/26/2024

Table of Contents

[5.1 Class Model 3](#_Toc158892562)

[Figure 1: Class Model Updated with design goals in mind 3](#_Toc158892563)

[A5.1 (ChatGPT regarding Class Model) 4](#_Toc158892564)

[5.2 Packages 5](#_Toc158892565)

[5.3 Logical Data Flow Diagram 6](#_Toc158892566)

[Figure 2: Logical Data Flow diagram 6](#_Toc158892567)

[5.4 Physical Data Flow Diagram - Microservices 6](#_Toc158892568)

[Figure 3: Physical Data Flow diagram 7](#_Toc158892569)

[Appendix 7](#_Toc158892570)

[References 10](#_Toc158892571)

[Evaluation 11](#_Toc158892572)

[Hints 12](#_Toc158892573)

[Overall Assignment Notes 12](#_Toc158892574)

[Design Goals 12](#_Toc158892575)

[Class Model 12](#_Toc158892576)

[Logical and Physical Data Flow Diagrams 12](#_Toc158892577)

MET CS682 ASSIGNMENT 5

TERM PROJECT PART 3: Design Goals and Data Flow

*Mahim Choudhury*

The purpose of this exercise is to give you practice performing system design and using UML.

1. OBSERVE LIMITS; 2. **USE AI GENERATION** **but only in Explaining your class model in question 5.1. You will need to explain this in your own words as well.** 3. OUTSIDE RESEARCH; 4. NAME YOUR FILE; 5. EVALUATION; 6. HINTS: as before

You will continue developing *EmbeddedAI* by specifying UML design for it, making the same assumptions listed in project parts 1 and 2 (assignments 3 and 4). For clarity, include notes about the scope (what aspects of the whole) your solution addresses and supply the information requested in the Appendix.

The last section contains numerous hints.

# Class Model

A screenshot of a computer

Description automatically generated

### Figure 1: Class Model Updated with design goals in mind

The original class diagram illustrated the pet monitoring aspect of the system, where EmbeddedAI enhances functionality. It begins with a non-entity stereotype class named Design, conforming to the UML User Guide's definition of stereotypes as labels for similar classes within a specific domain, essentially creating a metatype [2]. The model showcases aggregation, composition, and inheritance relationships among its classes. The Detection class serves as the integration point for EmbeddedAI to analyze data. The detail description of relationships between classes can be found in the Appendix section. In this updated class diagram, the focus is streamlined towards incorporating some design goals. Notice the classes are organized by packages in this updated diagram. The new packages are, Monitoring, Iot, Interaction, and safety. The goal is to keep the similar classes together to increase cohesion which in return lessened coupling. Compared to the original diagram, this diagram is much steadier and clearer regarding better design goals. The Monitoring packages contains four classes who are related in terms of usage, “PetMonitoring” class handles the AI processing of monitoring and is related through aggregation with “Pet” class which holds pets information, User class is inherited by “EmergencyContact” class. The Interaction package has two classes. Here, we have classes responsible for managing interactions between users and the system, as well as the Dispenser class, which handles dispensing pet items within the system. Safety encompasses classes related to safety and security measures within the system. The Notification and Detection classes handle notifications and detection of anomalies or events, contributing to the overall safety and security of the environment. The IoT package encapsulates classes related to IoT functionality, which is crucial for integrating sensors and other IoT devices into the system. Further illustrations of the rationales behind the new packages are described in the part 2 section.

## A5.1 (ChatGPT regarding Class Model)

Prompt: This week's assignment's part 1 is to introduce design goals into the class diagram. The goal is to have more cohesion and less coupling. Meaning likewise classes should be packaged and relationships between the classes should be lessened. Also keep in mind the reliability, robustness, flexibility, sufficiency, etc key attributes of design goals. Could you give me some ideas?

Response:  
Certainly! Here are some design goals and considerations to enhance cohesion and reduce coupling in your class diagram:

Sufficiency: Ensure that the classes and their relationships in your diagram adequately cover all the functional and non-functional requirements of the system. Each class should contribute towards fulfilling specific system requirements without unnecessary redundancy.

Encapsulation: Ensure that each class encapsulates its data and functionality, exposing only necessary interfaces to other classes.

High Cohesion: Aim for classes to have a single responsibility or purpose, ensuring that each class performs a well-defined set of related tasks.

Low Coupling: Reduce dependencies between classes by minimizing direct interactions and using abstraction layers where necessary.

Modularity: Organize classes into coherent modules or packages based on their functionality, promoting easier maintenance and scalability.

Abstraction: Utilize abstract classes and interfaces to define common behaviors and establish clear contracts between components.

Flexibility: Design classes to be flexible and adaptable to changes in requirements, allowing for easy modification and extension without affecting other parts of the system.

Reuse: Identify opportunities for code reuse by extracting common functionalities into reusable components or base classes.

Separation of Concerns: Ensure that each class addresses a distinct aspect of the system's functionality, avoiding overlap or duplication of responsibilities.

Information Hiding: Hide implementation details and internal state within classes, exposing only essential information through well-defined interfaces.

Dependency Inversion: Favor dependency injection and inversion of control patterns to decouple components and promote testability and flexibility.

By incorporating these design goals into your class diagram, you can achieve a more modular, flexible, and maintainable system architecture with improved cohesion and reduced coupling.

# Packages

In at most one and a half single-space pages (using 12-point type) not including appendices apply design goals to identify appropriate packages, showing which classes belong to which package. Use the headings below to explain how your packages fulfill applicable design goals. Your design goals should focus on Sufficiency, Flexibility, and Reusability, explaining cohesion and coupling in your design and their relationship with design goals. Packaging choices require trade-offs. Explain these as appropriate.

For each package:

1. **Package**: Monitoring

**Design goals for this package**: This package aims to provide comprehensive monitoring functionalities for sufficiency, reusability, and flexibility.

**Design tradeoffs for this package**: The package contains some of the important classes that handles the original requirements. PetMonitoring class has all the methods related to AI analysis which is one of the main requirements. It also holds the User and Pet class. Notice there is room for flexibility as one can add more monitoring method in the petmonitoring class or add guist user unde User class. The primary trade-off in this package is balancing cohesion and flexibility. While a cohesive package ensures that related functionalities are grouped together for clarity and maintainability, it may lead to higher flexibility in my experience. Martin (2017) emphasizes that too much flexibility can lead to complexity and reduced maintainability [1]. Thus, there are less breakdown of “petmonitoring” class to balance between flexibility and cohesion. For example, if I broke it down to another package named “Activity” where it handles the activity portion, there would be less cohesion more flexibility.

2. **Package**: Interaction

**Design goals for this package**: The Interaction package facilitates user interaction with the system and focuses on sufficiency, reusability, and flexibility.

**Design tradeoffs for this package**: The package contains two main classes which can contribute towards the reusability design goal of the system. It also incorporates the sufficiency using AI in the “Interaction” class. Flexibility is also present as the interaction can include another interaction related class in future i.e. “scheduling”. The trade-off involves balancing reusability with coupling. Ensuring that interaction-related functionalities are grouped together promotes clarity and ease of maintenance. Less coupling between classes and packages indicates it is easier to reuse for other functionalities. Also, the coupling involve aggregation and not composition which reduces the dependency on other components of the system.

3. **Package**: Safety

**Design goals for this package**: The Safety package focuses on ensuring the security and well-being of pets for sufficiency, reusability, and robustness goals.

**Design tradeoffs for this package**: The Safety package involves some crucial sufficiency related classes in it. It handles the safety aspect of the original requirement. It also ensures robustness of the system by adding more composition between classes. This in terms sacrifices the flexibility and reusability of the package. More coupling within the package also has impact on reusability goal. Gamma et al. (1994) discuss the trade-off between coupling and reusability, highlighting that excessive coupling can impede the reusability of safety functionalities [4]. Therefore, to accomplish Robustness and sufficiency, more composition based coupling is used within the package.

4. **Package**: IoT

**Design goals for this package**: The IoT package handles the activation, deactivation, and data processing of IoT sensors for flexibility, reliability, and reusability.

**Design tradeoffs for this package**: The Sensors package has very good flexibility and resualbility using inheritance. One can easily add more instances of sensors later to introduce new IoT devices. Relaibilty is achieved through redundancy in my opinion. Multiple instances of sensors can act as redundant to each other and provide reliability to the system. This package, coupling was reduced by incorporating reusability of various classes through inheritance. Fowler (Website) discusses the trade-off between coupling and reusability, emphasizing the need to minimize coupling while maximizing the reusability of sensor components [3].

# Logical Data Flow Diagram

Draw a logical data flow diagram for your design. Explain the parts of the diagram that are not obvious. Your diagram should have roughly 8-10 logical processing elements. Explain the parts of your logical data flow diagram that are not obvious.

### Figure 2: Logical Data Flow diagram

Your logical data flow diagram and notes replaces this

# Physical Data Flow Diagram - Microservices

Draw a physical data flow diagram for your design focusing on incorporating Microservices within your design. Explain the parts of the diagram that are not obvious. Your diagram should have roughly 3-5 physical processing elements. You may want to extend the scope of the design to include additional physical processing elements. Consider including elements in your design such as interfaces or Internet of Things as additional physical processing elements. Explain the parts of your physical data flow diagram that are not obvious.

### Figure 3: Physical Data Flow diagram

Your physical data flow diagram and replaces this.

# Appendix

Since Term Project Part 3 builds on Term Project Part 1 & 2, provide the mission statement here. For clarification, you may also provide requirements and design you have outlined last week in Term Project Parts 1 and 2—parts you feel are helpful in supporting the updated UML design. This section is not graded.

Scope:

Scenario:

Imagine a Smart Home Automation System designed to manage various aspects of home living, such as lighting, temperature control, security, and appliance automation. This system utilizes IoT devices and sensors to enable remote monitoring and control via mobile apps or voice commands.

Scope: The scope entails integrating EmbeddedAI into an existing Smart Home Automation System to enhance user experience, efficiency, and security. This integration will leverage generative AI like ChatGPT and IoT elements to provide personalized assistance, intelligent automation, and proactive home management capabilities. Key focus areas include personalized home assistance, natural language interaction, predictive home automation, and enhanced security and safety measures.

The project will involve analyzing user behavior, preferences, and environmental data collected by IoT sensors to offer tailored recommendations and automate routine tasks. It will enable natural language interactions with EmbeddedAI, allowing users to communicate with their smart home system using voice commands or text input. Predictive capabilities will anticipate user needs and potential issues, while enhanced security measures will detect anomalies and trigger automated responses to ensure the safety of the home environment.

Initial Use Case:

|  |  |  |
| --- | --- | --- |
| **Use case Name** | Pet activity monitoring | |
| **Actor:** | Pet owner | |
| **Description:** | The integrated EmbeddedAI system can aid the smart home system to empower the pet owners with the ability to leave their pets home without worry. AI-enabled monitoring system enhances pet safety and allows pet owners to stay connected with their pets, even when they are not physically present at home. | |
| **Pre-condition:** | The system is online and connected to IoT devices. | |
| **Step #** | **Actor** | **System** |
| **1** | Pet Owner activates pet monitoring mode | System initiates AI-enabled pet monitoring functionalities, including smart cameras, motion sensors, and environmental sensors, to track pet movements and behaviors. |
| **2** | Pet Owner receives activity alert | System utilizes AI algorithms to detect pet movements, sounds, or irregular behaviors, and sends real-time alerts and notifications to the pet owner's mobile device. |
| **3** | Pet Owner views live monitoring feed | System provides access to live video streams and environmental data through the mobile app, allowing the pet owner to observe their pets' activities and surroundings remotely. |
| **4** | Pet Owner interacts with pets remotely | System enables the pet owner to engage with their pets through interactive features, such as remotely operated toys, treat dispensers, or two-way audio communication channels. |
| **5** | Pet Owner ensures pet safety | System allows the pet owner to monitor environmental conditions, such as temperature, humidity, and air quality, and take necessary actions to ensure the safety and comfort of their pets. |
| **Alternate Courses:** | [System Alt 6] If the system detects unusual behavior or potential hazards, it alerts the pet owner and provides guidance on mitigating risks or contacting emergency services.  [System Alt 7] If the pet owner is unavailable to respond to alerts, the system can activate pre-configured safety protocols, such as contacting designated emergency contacts or veterinary services. |  |

4.4 Documenting Classes and Relationships

Using the format below, select three most important nontrivial business classes and one non-business design class from the class model and the format below, explain its importance for your design, and its relationships with other classes. Select classes which are complex enough to require explanation.

4.4.1 First Business Class Selected: User

Importance for the Design: The “User” class defined in the class model diagram is one of most important business classes in the system. Although the main beam of light in this system is focused around the implementation of Iot devices and AI analysis, ultimately the destination is the User. User class defines what the system should know about the human users of it. User has their own id, username. It also allows them to create new users and update their information. The class has some relationships with other important classes which will be described later. It is the root of regular user, emergency contact, interactions with pet, and selecting what dispenser to use for interaction. Although the AI can help in this area, I still believe the pets need their human friend to interact with them instead of a cloud.

Relationship with other classes: The “User” class in the class diagram shows that it is inherited by two other sub classes named “EmergencyContact” and “Interaction”. From my previous experience, I could tell the inheritance of classes means they are the same class with additional features. Meaning the EmergencyContact and interaction classes already have the attributes of “User” class and then they have their own unique attributes and behavior. In this case, the interaction class have it’s own ID and the EMergencyContact has two more attributes, Email and Cell and their behaviors. For example, each object of Interaction class can have it’s own specific userID, username, and type id. Similarly, EmergencyContact can have it’s userID, username plus the email and cell per instances of the class itself. Interaction class has it’s own relationship with “Dispenser” class, which is also a branch coming out of the root “User” class.

4.4.2 Second Business Class Selected: Sensors.

Importance for the Design: The “Sensors” class is another important business class for. For this system, it works as the hub class for all the IoT devices to provide the data to the AI for analysis. The class is designed to hold information about the IoT devices such as Id, location, and behaviors like activate(), deactivate(). There could be more attributes and behaviors, however, we are only focusing on some. It is clearly visible in the class diagram that this class has many relationships with some other important classes in the system. This emphasizes the importance of this class in this system. Although it can connect to many other IoT devices, the diagram is only showing the thermostat, and camera as the main two sensors. The relationships are described in the next section.

Relationship with other classes: The “Sensors” class has aggregation relationship with Thermostat and Camera classes. From my understating from past learning, aggregation relationship means that on class is part of or has another class. For the Thermostat, it can control one and only one thermostat and a thermostat can only be controlled by that one sensor class. I chose only one thermostat because in most houses, there is only one thermostat to control the temperature. For the Camera class however, the Sensor class can control 1 to many cameras since a house can have many of them. Cameras can only be controlled by that one Sensor class though. The Aggregation relationship also tell us that the Thermostat and Camera classes can exist without the Sensors class. It has the same relationship with “PetMonitoring” class. However, the “detection” class cannot exist without the Sensor class and therefore, it has a composition relationship with it.

4.4.3 Third Business Class Selected: PetMonitoring.

Importance for the Design: The “PetMonitoring” class is another most important business class for this system. This class works as the main brain of the pet monitoring feature of the smart home system. It connects to various important classes and works as the manger. It holds information of the pet(s), monitors their activity, detects anomalies, checks the system health, etc. Without the “PetMonitoring”, the main purpose of this pet monitoring feature of smart home system will not function as required. It is required to communicate with other classes to gather data and work with it. The many relationships it has with other classes are described in the section below.

Relationship with other classes: The “PetMonitoring” class has aggregation relationship with “Sensors”, “Safety”, “User”, “Pet”. This way, it can get information from them and their related classes. It can have 1 to many Sensors, Users, and Pets. Those classes however, can only be associated with only one “PetMonitoring” class. The “PetMonitoring” class can be associated with 0 to 1 Safety class. The reason I chose 0 to 1 is because I think even without the information from safety class, it can still function with the activity detected from the sensors and keep recording it. On the other hand, the “Safety” class should only be associated with the on and only “PetMonitroing” class.

4.4.4 Non-Business Class Selected: <<Design>>.

Importance for the Design: As depicted by “<< >>”, the Design class is the non-busineess class for this system. It is pictured as the main home screen from where the user can interact with the system to enable pet monitoring mode. It can also display valuable information such as graphs of previous usage, pets weekly sleep time, etc from previous monitoring. Although it is a non-business class, it is important to the user from user interface of the pet monitoring system’s standpoint. It determines how user-friendly and easy to use pet monitoring feature would be to the users. It has a dependency with “PetMonitoring” class which is described in the section below.

Relationship with other classes: The dependency described in the class model is pretty self-explanatory. It displays the “PetMonitoring” class. As mentioned earlier, it need the data from the “PetMonitoring” class in order to display the information in the GUI. “PetMonitoring” class gather the data from various other classes across the design but it all needs be presented to the user through he non business class <<Design>>.

# References

[1] Martin, R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall.

[2] Booch G, Rumbaugh, J, Jacobson I (2014). The Unified Modeling Language User Guide. Upper Saddle River, NJ: Addison-Wesley.

[3] Fowler, M. (n.d.). Martin Fowler's Blog. Retrieved from https://martinfowler.com/tags/design.html

# Evaluation

A picture containing text, screenshot, number, parallel

Description automatically generated

**Please do not include Hints section in your solution.**

# Hints

## Overall Assignment Notes

* You may use Draw.io, Visio, LucidChart, or another design tool of your choice (please check with your facilitator in advance if not using Draw.io, Visio or LucidChart).
* The module notes are a primary source for explanations and examples; we also encourage you to do outside reading and research to gain additional perspective.
* **Thoroughness and Coverage:** Accompany each requested section with a brief description that explains your thinking and the choices that you made.
* **Consistency and Clarity:** Make sure that you have a consistent design throughout the solution. You will want to iteratively review all four sections to make sure that together they are consistent as a single solution.
* **Clarity:** Are diagrams clear to read? (e.g., avoid overlapping lines, non-polished designs).
* **Relevance**:
  + Is the scope of your design relevant to the solution? For example, choosing the data flow scope, classes, and methods that are secondary (for example, authentication might be important – but with limited scope of your assignment it may not be the focus).
  + This is the usual editing process. It is tempting to repeat important points but this often weakens your work and can appear to the reader like padding.

## Design Goals

* Your focus should be the beginning sections of the Module 5 Notes “The Goals of System Architectures and Designs”
* **Technical Soundness:**
  + Solution shows understanding in context of goals of system design (i.e., a thorough discussion of how sufficiency is satisfied in the design, with links to the solution’s class model and links back to the requirements).
  + Do you show distinction and understanding between cohesion and coupling of classes and packages within your design to support the selected design goals?
* **Thoroughness and Coverage:**
  + Make sure to explain how the selection of your packages specifically supports the goals of system design for this scenario, avoid being generic.
  + Are tradeoffs between goals of design covered?
  + Is cohesion/coupling covered within the context of goals of system design?
* **Clarity:** Are tradeoffs between cohesion and coupling clearly explained? For example, mentioning high cohesion is not enough - a clear link between the design needs to be made, and explained (e.g., classes and relationships in the class model show cohesion in package).

## Class Model

* **Clarity**: Did you add a new class or relationship – is it aggregation or composition? Look to explain your updated design and show changes in a different color.
* Other hints and notes on approaching class modeling are covered in Assignment 4.

## Logical and Physical Data Flow Diagrams

* Physical data flow diagrams are described in the Module 5 Notes within the “Data Flow Diagrams” section.
* **Technical Soundness:** 
  + Make sure to understand the difference between logical and physical data flow diagrams
  + Is the focus on the physical processing elements and not on the logical processing elements in the physical data flow?
  + Are all the data flows correct (e.g., external actor not sending data flow directly to data store, but through a processing element, data flows appropriately labeled with data and not a process (action).
  + No “dead-ends” in processing elements. Each processing element should have an input and an output.
* **Clarity:** Explain the scope of your data flow diagrams.