MTAT.03.083: Systems Modelling Assignment 3

Group Members

Eldaniz Akbarzade (eldaniz.akbarzade@ut.ee)
Famil Babayev (famil.babayev@ut.ee)
Mushfug Soltanov (mushfug.soltanov@ut.ee)
Samir Musali (samir.musali@ut.ee)

2023
University of Tartu

Table of Contents

Assignment Description

1. **Task #1 (1 point)**: Please provide an updated description of the system you are modelling. Make sure that it matches the designed model as closely as possible and that all details (previously - classes, their attributes, relationships between classes, actors, use-cases, sequence of operations etc., now - states, operations etc.) are specified in both the model and the description. Expected length – ½ - 1 A4 page.
2. **Task #2 (4 points)**: Provide state chart for the system you described in Task #1. Make sure you use composite states (where relevant) at least once. Note that the model should consist of at least at least 4 states, 7 events, guards, at least three types of actions/operations, composite states (at least once, where relevant), history pseudo-states (at least once, where relevant).
3. **Task #3 (3 points)**: Provide application class model for the system you described in Task#1. Add at least one sentence describing how does it differ from the model you developed as part of assignment#1 Task#1.
4. **Task #4 (1.5 points)**: What is the environment, which you choose to design your statechart and application class model? What were the decisive factors to give the preference to this environment(s)? **(1 point out of 1.5):** Does it allow you to generate the code from the designed model? If yes, what are your observations regarding the generated code? Have you identified any weakness in it? Does the code match the designed model? If not, were the expectations-reality differences due to deficiencies in the original model that you were forced to correct?
5. **Task #5 (0.5 points)**: Provide evidence that the work was conducted in a team (e.g., in the form of a link to the VCS; the screenshot of the above etc.).
6. **Task #6 (optional)**: Explore and document ChatGPT or other LLM capabilities in modelling use-case model and sequence diagram model (in addition to Task#2 and #3, but not instead!). Was it possible to achieve the same result that you obtained in Task #2 and #3? If not, what were the differences? What are strengths and weaknesses of ChatGPT in addressing this task? Had the use of ChatGPT allowed you to improve either the description (Task#1) or the model (Task#2, #3)?

What to submit? Please submit the homework as a single file – .pdf or a package, if more than one files where created - zip file or .7z file or .tar.gz via Moodle.

Please organize and name the files in the package clearly, so that it is clear which file(s) corresponding to the first task and which ones correspond to the second task.

If you have completed the homework with another (teams of up to four), please write the name of your team-mate in the "Comments" field of the homework submission form. Only one submission per pair of students is needed.

Updated System Description

Grable's digital platform remains a sophisticated ecosystem, intricately woven to cater to the nuanced interactions between two primary user groups: restaurants and diners. Each faction establishes a secure account, marked by a unique identifier and essential contact details, encompassing phone numbers and email addresses. For restaurants, their identity is refined by including geolocation data, capturing their address, city, state, and postal code, and converging into a dynamic menu system. This system, a database at its core, houses many menu items, each distinguished by a unique ID and described through various attributes such as name, ingredients, dietary specifications, category, price, and availability status. The platform empowers restaurants with CRUD (Create, Read, Update, Delete) capabilities, enabling them to manage their menus in real-time.

Conversely, diners are represented through profiles that detail their dietary preferences and house a digital wallet. This wallet amalgamates various payment methods, each a complex data type that stores the payment modality and intricate transaction details. The diner interacts with the platform primarily through menu exploration and order placement. Orders within the system are distinct entities, each tagged with a unique identifier, linked to specific menu items, timestamped, assigned a table number, and equipped with status indicators for tracking the order and the payment process.

Central to this platform is the payment processing subsystem, a pivotal interface that liaises with diverse payment services to ensure transaction security. It meticulously records transaction specifics, including the amount, payment method, and status, and seamlessly facilitates the transfer of funds to the restaurant’s account after deducting Grable’s commission.

The architectural blueprint of the platform is tailored to emphasize security, integrating both authentication and authorization protocols. It supports real-time data processing, offers user customization, and incorporates a notification system that keeps users informed about the status of their orders and payments. The backend is scalable, designed to accommodate a growing base of users and transactions, and features comprehensive systems for logging and monitoring to detect performance issues and anomalies.

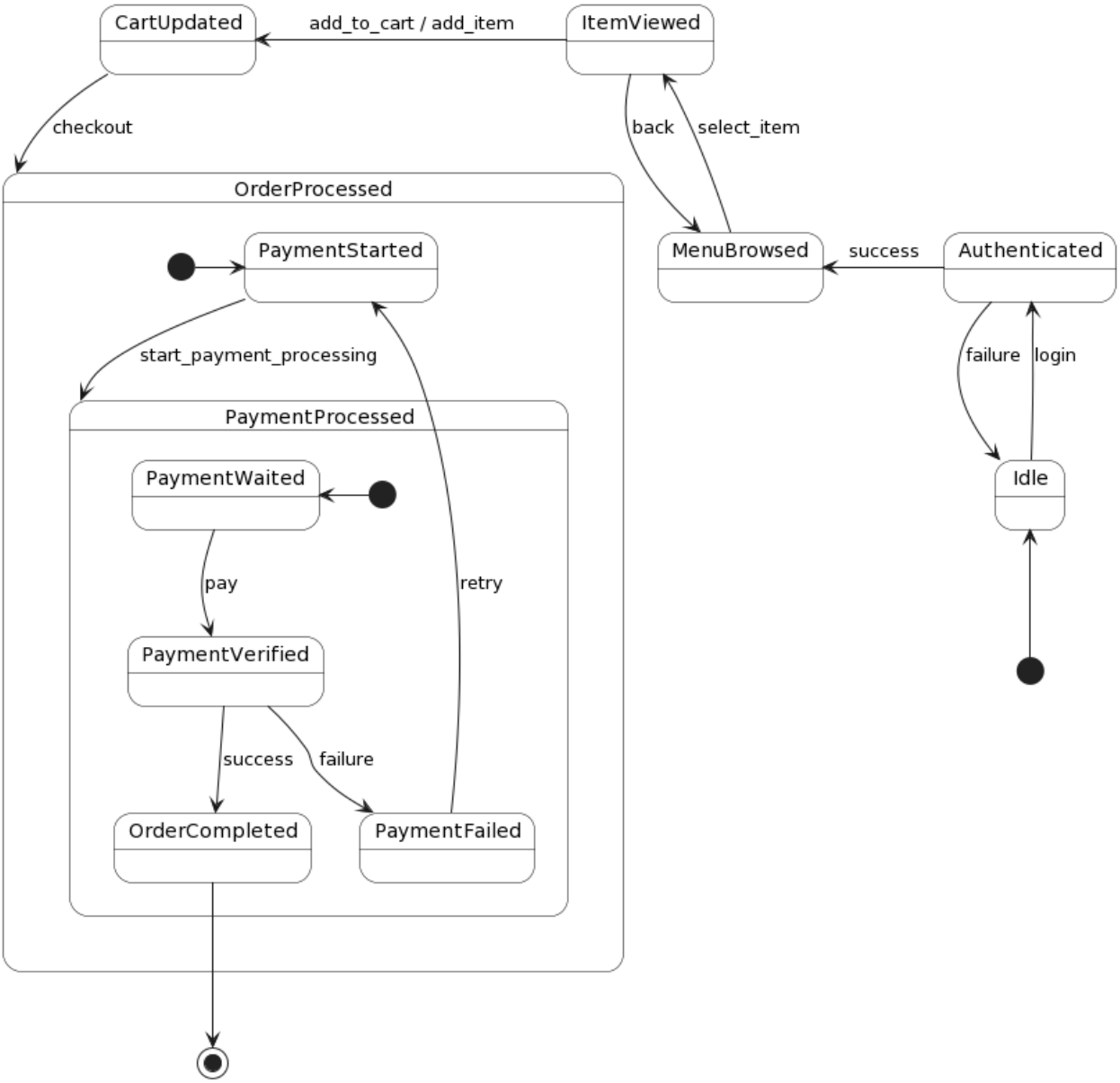
The technical workforce within the platform comprises system administrators, who ensure the platform’s integrity and user support, and developers, who are tasked with maintaining and enhancing system functionalities. External systems, such as payment gateways and notification services, play a crucial role in the platform’s operations.

The platform's design considers various constraints, including technological limitations and regulatory compliance, while presuming high user engagement and frequent transactions. Non-functional requirements like system responsiveness, data integrity, and operational uptime are prioritized to guarantee user satisfaction and the platform's reliability.

In essence, Grable's platform is multifaceted, fostering various user-system interactions. It is crafted to be secure, scalable, and responsive, with a functionality spectrum that spans user account management, real-time menu updates, order processing, and payment transactions, all within a framework that prioritizes user experience and operational efficiency. The system's evolution from its initial conception has led to a more intricate interplay of components and processes, reflecting a matured understanding of user needs and technological capabilities.

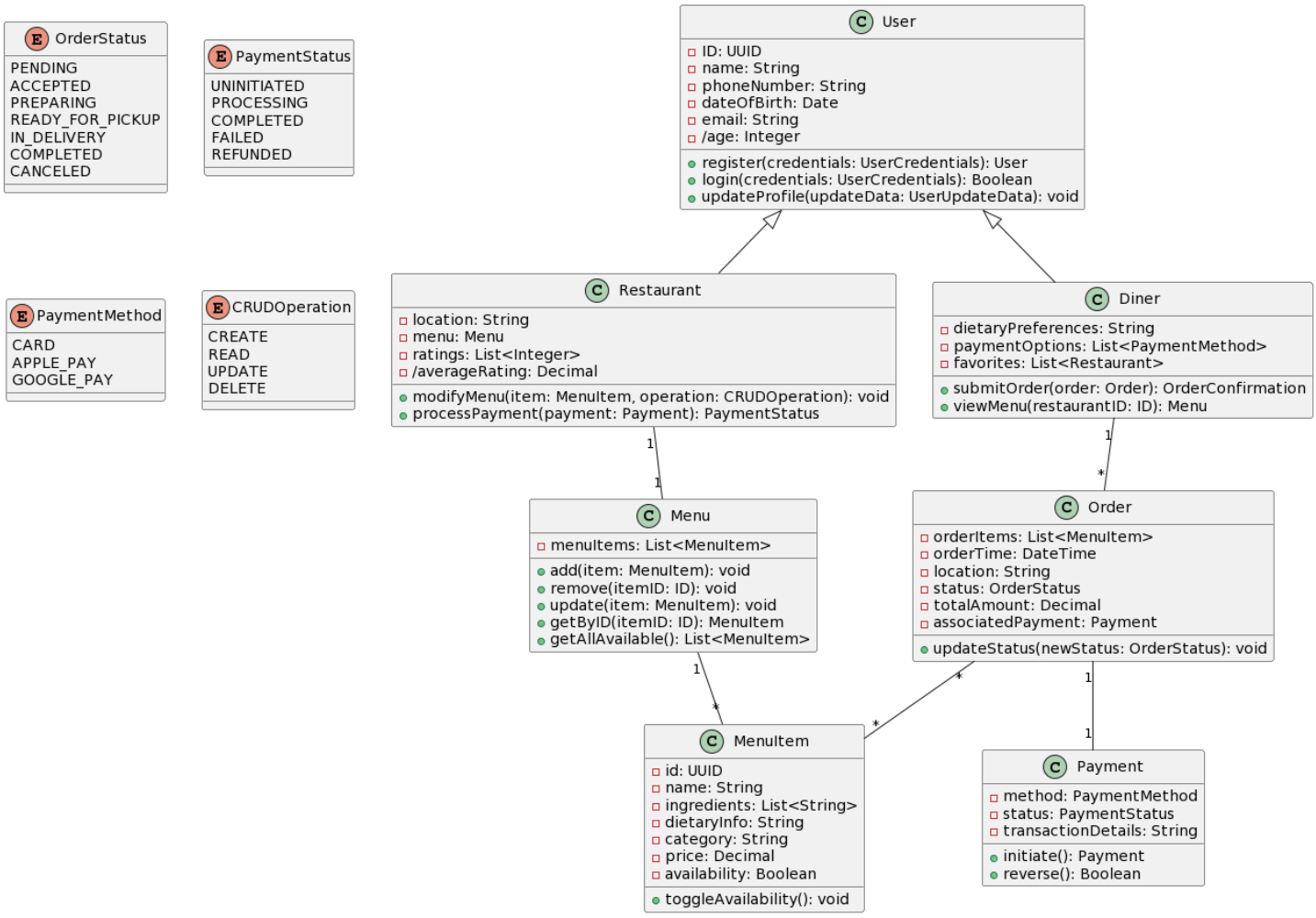
State Chart

For the state chart, based on the updated system description, we can visualize it in the following way:



Application Class Model

The only difference between the current and previous models we submitted in Assignment 01 is that due to the size of the newest System Description, we now have more classes, attributes, and enumeration options. Even though in the first assignment, only the Domain Class Model was asked to submit, we also added the types for the attributes and the methods, which should exist in the Application Class Model compared to the Domain Class Model:



Teamwork

While working on the previous assignment, we established the GitHub repository for our group project, a transparent platform to track and assess each team member's contributions. We continue using the same repository to share our findings. Here's how you can effectively monitor and evaluate our work:

1. **Commits:** The commit history in our GitHub repository is a valuable resource for tracking the evolution of our project. Each commit is associated with a specific team member, and the commit message concisely summarizes the changes made. This allows you to see who made the changes and when. By reviewing the commitment history, you can gain insight into the individual and collective efforts of the team.
2. **Folders and Files:** Project files and directories are organized structurally within the repository. We've adopted a naming convention where each folder is labelled with the group member's name responsible for creating it. This arrangement simplifies identifying which team member worked on a particular aspect of the project.

We aim to use this GitHub repository to ensure every team member actively contributes to the project. The commit history and the organization of folders and files provide transparency and accountability, allowing us to track progress, assign responsibilities, and maintain a collaborative and equitable project environment.

Besides the repository, we also have a shared Google Drive folder and have been having frequent online group meetings over Google Meet since; in that way, it is more accessible to screen share and demonstrate our findings.

Diagramming Environment

Choice of Environment: PlantUML

- **Comprehensive Features:** It offers a comprehensive set of features for modelling, including support for class relationships, attributes, methods, and annotations.
- **Text-Based Representation:** PlantUML uses a simple text-based syntax, which is easy to write and version control using plain text files.
- **Integration with Markdown:** Integration with Markdown allows seamless inclusion of state chart diagrams in the documentation.
- **Open Source:** Being an open-source tool, PlantUML is accessible and easily integrated into various environments.
- **Code Generation Capability:** PlantUML is not inherently designed for code generation but primarily focuses on visual representation.
- **Observations:** No code is directly generated from PlantUML Charts. Instead, these charts serve as a visual aid for understanding and communicating the system's behaviour
- **Weaknesses:** The lack of direct code generation is not a weakness but a design choice. The expectation is to use PlantUML Charts as a documentation and communication tool rather than a code generation tool.
- **Differences:** There are no expectations for code generation from PlantUML, so there are no differences in this context.
- **Intended Use:** PlantUML is intended for visual representation and communication, and any disparities are more likely related to misunderstandings or misinterpretations during the communication process rather than code generation issues.

Analysis of LLM Capabilities

Question: Could you achieve the same result in Task #2 and #3? If not, what were the differences?

Answer: Yes, achieving comparable results in Task #2 and #3 was possible using an external platform instead of ChatGPT. The differences between the outcomes stem from the nuanced capabilities of the chosen external tools compared to ChatGPT.

Precision and Detail:

1. Using an external platform instead of ChatGPT allowed us to carefully capture the complexities of states, events, and transitions in our state chart model.
2. ChatGPT's quickness allowed for a rapid sketch of the model, but the external tool was like having a collection of perfectly calibrated tools, ensuring that every part of our state chart was detailed and nuanced.
3. ChatGPT may provide a broad strokes perspective, but the external tool allowed us to zoom in and add intricate details, ensuring that no critical factor was overlooked in our state chart representation.

Adapting to Specifics:

1. The external platform offered a more straightforward approach to defining classes, attributes, relationships, and annotations, adapting the model precisely to our project's unique requirements.
2. Human expertise was pivotal in ensuring the model aligned perfectly with our needs. In contrast, ChatGPT may provide general ideas but could miss the specific details essential for our project.

General Differences:

1. Our team, being skilful in the field, leveraged their experience to make informed decisions and assumptions, enriching the models.
2. While ChatGPT is helpful, its understanding may not reach the same depth, potentially resulting in more generalized or less context-specific models.

Question: What are the strengths and weaknesses of ChatGPT in addressing this task?
System Description:

1. **Strengths:** ChatGPT is good at providing a quick and comprehensive system overview, assisting in the early stages of system description.
2. **Weaknesses:** A lack of expertise in the subject may lead to overlooking detailed aspects of complex systems.

State Chart Model:

1. **Strengths:** ChatGPT proved beneficial by swiftly generating a state chart, providing a foundational starting point for further development, especially when coupled with our previous assistance using PlantUML

2. **Weaknesses:** A lack of comprehensive domain expertise in complicated systems may result in less nuanced state chart models.

Application Class Model:

1. **Strengths:** ChatGPT may give a basic application class model, revealing class relationships and properties.
2. **Weaknesses:** Class models may be less precise and particular due to a lack of subject understanding.

Question: Had ChatGPT allowed you to improve the description (Task#1) or the model (Task#2, #3)?

Answer: In Task#1, ChatGPT provided early support in outlining the system, offering a swift overview and generating ideas in the initial stages. While its initial contribution set the foundation, human participation was crucial in enhancing and expanding the description. However, for more detailed tasks like creating in-depth state chart models (Task #2) and application class models (Task #3), using ChatGPT alone didn't significantly improve. The tool's limitations in understanding specific topics and being precise became clearer in these complex tasks. Human expertise and manual adjustments were crucial to make the models more accurate and detailed, ensuring they represented the system thoroughly.