Unified Modeling Language (UML) and Software Requirements Specification (SRS) Diagrams

Hayden Eubanks

September 8, 2023

Table of Contents

Section/Sub-section Title
1. Use Case Diagram
2. Activity Diagram
3. State Diagram

Introduction

Before software can be effectively developed, a shared vision of the relationships between components must be established through the use of diagrams such as UML Diagrams (Koç et al., 2021). Further, diagrams can allow a development team to effectively model system requirements regarding use cases, activities, software states, and even classes to be used in implementation (Ozkaya & Erata, 2020). Taking the time to accurately model system requirements in diagrams can then clear ambiguities in software design as well as enable the efficient development of components by providing a visual aid to structure that can be used throughout the design (Júnior, Farias, & Silva, 2021). From this, the immense value of diagrams can be seen highlighting the importance for a development team to ensure sufficient time is given to their design and completeness.

Throughout this assignment, example use case, activity, and state diagrams will be generated to model the diagram creation process as well as the relationship between diagrams. This will be accomplished by analyzing a system requirement standard (SRS) document for the function of withdrawing cash from an ATM and then modeling the requirements through the different diagrams. The result of this assignment will then provide a set of diagrams through which the requirements can easily be visualized within the context of an application assisting the development team in the actual implementation of the feature.

The first diagram category that will be generated for this use case is the use case diagram. A use case diagram models the relationship between entities that interact with the product such as consumers and stakeholders as well as the core functions that those entities interact with (Tsui, Karam, & Bernal, 2016). From this, a relationship can be drawn between each interacting entity and the core features of the system providing an understanding of which

entities must be kept in mind in a components design. This feature of use case diagrams can then be seen to allow the development team to fully consider all use cases and consumers of a system before development begins decreasing the likelihood that a use case is missed in development (Koç et al., 2021).

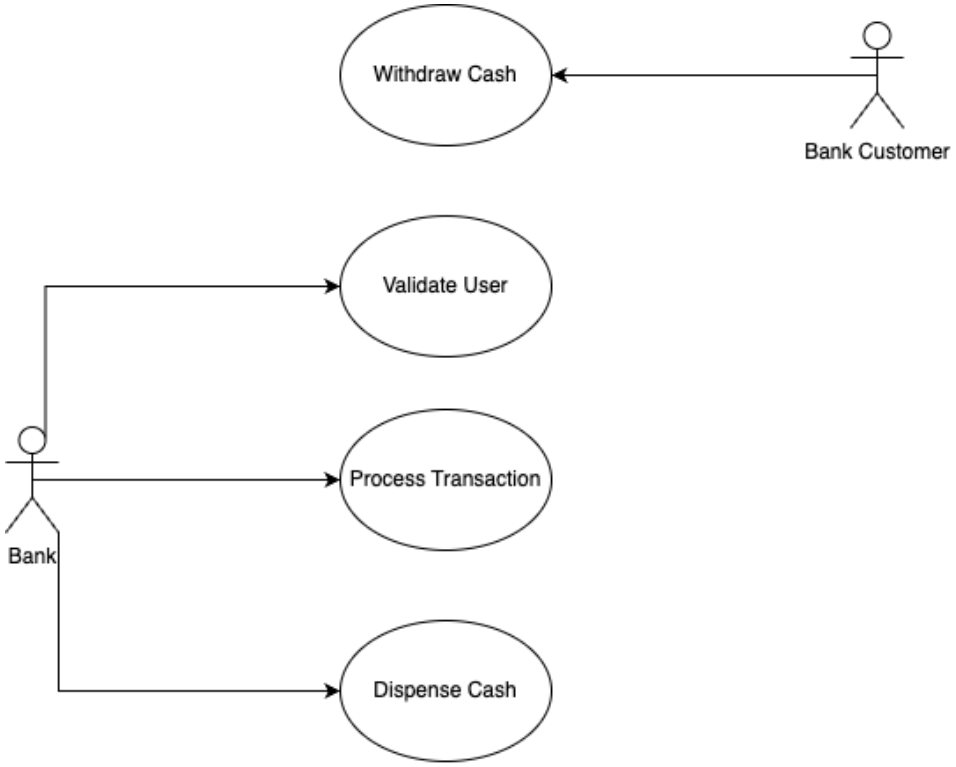
The next diagram format that will be generated and addressed is the activity diagram. The activity diagram can be used to model the flow of an activity which will allow the development team to easily visualize activity flow as well as decision points when implementing the design (Jaffari & Yoo, 2019). Further, activity diagrams can assist a developer in creating test cases that test specific aspects of an activity's flow (Jaffari & Yoo, 2019) further increasing the quality and resilience of the software product. Activity diagrams break down each of the specific use cases into activities and as such should follow the principles of algorithm design to avoid faults that could occur from the improper structuring of activity flows (Farrell, 2018). Activity diagrams can be used to model activity flows and as such can greatly assist a development team in designing software that effectively implements those flows.

The final diagram type that will be implemented in reference to the withdrawal feature will then be a state diagram. A state diagram has some resemblance to an activity flow diagram, but instead of modeling activity flows the state diagram models the state of the software throughout its usage (El Musleh, 2020). This allows for a drilled-down view of activities where the state of the system can be examined providing developers with the information needed to implement system state changes in design. State diagrams are essential to software design as the state of software must change to allow for functionality such as user

authentication and input validation highlighting the importance of modeling this data prior to design implementation.

The development of diagrams can allow for a detailed view of a system at multiple levels providing developers with the tools needed to develop a system. This can be accomplished through the use of tools such as use case, activity, and state diagrams highlighting the importance of developing diagram generation as a skill for software development. Developers can then use these diagrams to thoroughly and efficiently model and implement design features improving the effectiveness of the overall design lifecycle.

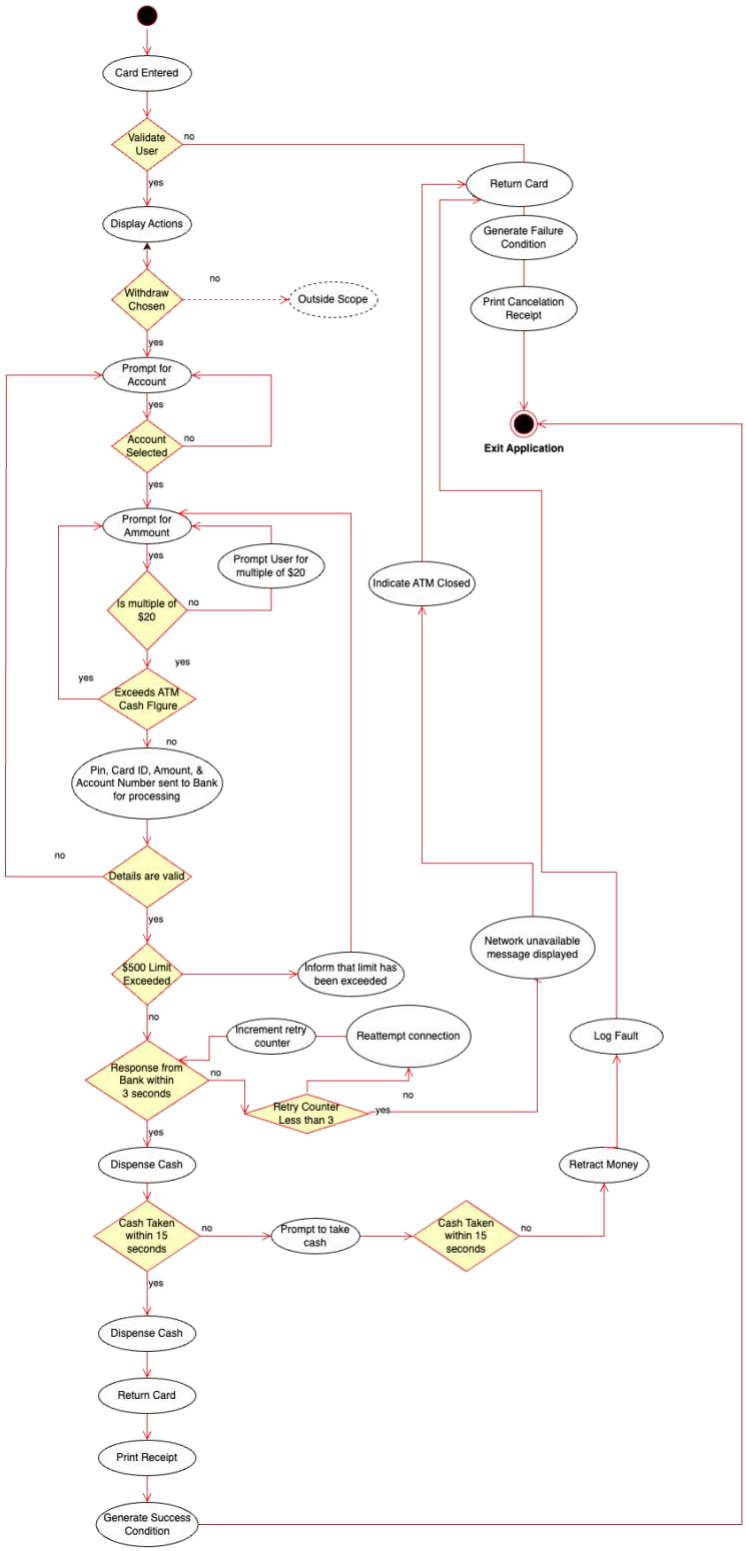
1. **Use Case Diagram: Withdraw Cash From ATM**



Within the use case diagram for the withdrawal of cash, two parties must be considered being the bank and bank customers who interact with the system. While the activity flow for this system will be more complex, the use case diagram can be succinctly created by generalizing the activities into their core responsibilities. For the customer, this can be represented as the use case of withdrawing cash as the ATM serves as an interface for the rest of the underlying actions. In a real-life ATM, a bank customer would most likely have more use cases, but for this specific example, it has been specified that the user will only ever seek to withdraw cash. The bank can then use the ATM to fulfill several activities that contribute to the withdrawal of cash including user validation, transaction processing, and the dispensing of cash. It is worth noting that this use case diagram could be displayed in an even more succinct manner by further

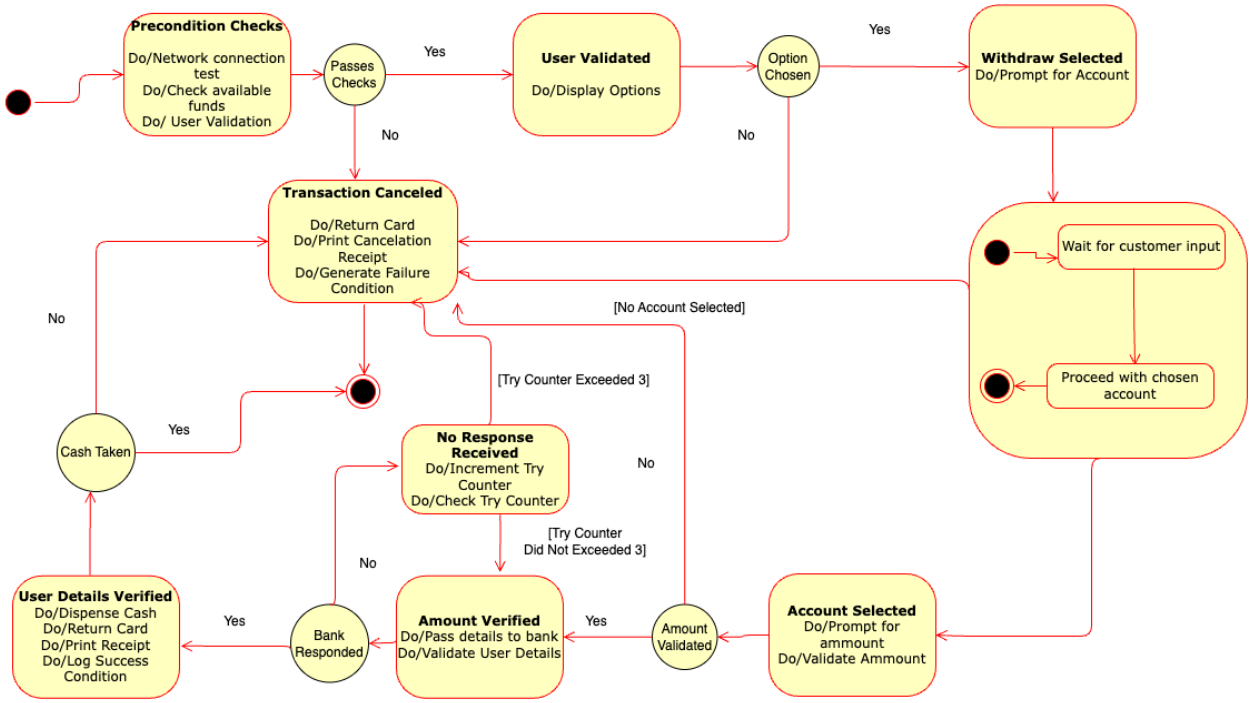
generalizing the use cases to “withdraw cash” and “process transaction”, but to demonstrate the functionality of use case diagrams a broader implementation was chosen.

2. Activity Diagram



This activity diagram demonstrates the flow of the withdrawal cash activity identifying key decision points where errors could be occurred and resolution steps to take at each decision point. Further, this activity diagram implements a structured design to ensure that the code is functional and efficient in its implementation. In the diagram above it can be seen that all elements of the withdrawal action are addressed allowing the development team to easily identify the action flow and implement the presented functionality in a product. Action diagrams provide a comprehensive overview of the actions a system performs and modeling these actions can set a development team up for efficient and successful development making them a valuable tool for development teams to use.

3. **State Diagram**



The state diagram shown above demonstrates the transition of states as the application moves through the action flow steps. This diagram highlights key moments in the system design when the state of the system may change and further dictates what actions are associated with that state. In this way, the state diagram can be used to track the functional working of software and model the system concerning use scenarios. In the example above, state changes can be observed whenever the system needs to validate details or accept user input. Each state is then associated with the input data being validated or not and the actions to be performed in each state are associated with their appropriate state boxes. Another aspect of this state diagram that is important to its design is the fact that each step of the process allows for process failure by which the termination state is entered. The existence of the termination state then allows for system or user errors to be caught and processed effectively (El Musleh, 2020). The state diagram can then be used for system testing by allowing the developers to track the system state regarding bugs the

system is encountering (El Musleh, 2020). State diagrams are an important tool for developers to use and through implementing state diagrams a developer can better track the state of a system in development and testing.

References

El Musleh, M. (November, 2020). Transformation of UML State Machine Diagram into Graph Database to Generate Test Cases. Upsala University. https://www.diva-portal.org/smash/get/diva2:1503406/FULLTEXT01.pdf

Farrell, J. (2018). *Programming logic and design, comprehensive* (9th ed.). Cengage Learning. https://ng.cengage.com/static/nb/ui/evo/index.html?deploymentId=5745322456083822711874447219&eISBN=9781337274609&id=1696581637&snapshotId=3303162&dockAppUid=16&nbId=3303162&

Jaffari, A., & Yoo, C. J. (2019). An experimental investigation into data flow annotated-activity diagram-based testing. *Journal of Computing Science and Engineering*, *13*(3), 107-123. http://jcse.kiise.org/files/V13N3-03.pdf

Júnior, E., Farias, K., & Silva, B. (September, 2021). A Survey on the Use of UML in the Brazilian Industry. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering*, 275-284. https://dl.acm.org/doi/abs/10.1145/3474624.3474632

Koç, H., Erdoğan, A. M., Barjakly, Y., & Peker, S. (March, 2021). UML diagrams in software engineering research: a systematic literature review. *Proceedings 74(1)*, 13. MDPI. https://www.mdpi.com/2504-3900/74/1/13

Tsui, F, Karam, O., Bernal, B. (2016). *Essentials of Software Engineering* (4th ed.). Jones & Bartlett Learning. https://libertyonline.vitalsource.com/books/9781284129755

Ozkaya, M., & Erata, F. (2020). A survey on the practical use of UML for different software architecture viewpoints. *Information and Software Technology*, *121*, 106275. https://www.sciencedirect.com/science/article/abs/pii/S0950584920300252