Smart Home Simulator - Phase 2

<u>Team Members</u>

Christa Abou Arraje - ID: 40226631

(**role**: Domain Model, technologies used)

Gabriel Rodriguez - ID: 40208888

(**role**: Context Diagram)

Jose Semaan - ID: 40244141

(**role**: Domain Model)

Ashpreet Singh - ID: 40172137

(**role**: Problem Definition)

Mark Tadros - ID: 40250850

(**role**: Context Diagram, technologies used, <u>team leader</u>)

SOEN 343 - Software Architecture and Design

Concordia University

March 3rd, 2024

Table of Contents

Phase 1

1. Problem definition

1.1 Problem Statement

The problem of	a lack of comprehensive tools for simulating, testing, and educating on smart home configurations in a virtual environment
Affects	developers, educators, and smart home enthusiasts who require deep insights beyond the capabilities of current smart home management systems like Google Home and Amazon Alexa
The impact of which is	restricted innovation in smart home technologies due to the inability to experiment with complex scenarios and system integrations without the cost or risk of real-world deployment
A successful solution would be	a platform that bridges this gap by offering advanced simulation capabilities, fostering innovation, and providing educational value beyond the operational control provided by existing market solutions

1.2 Product Position Statement

For	developers, researchers, and educators seeking an in-depth tool for smart home technology experimentation and learning
Who	need to explore and innovate within the smart home space without the limitations of physical device constraints
Beta	is a comprehensive simulation platform
That	offers unparalleled insights into smart home system behaviors, interactions, and potential innovations
Unlike	mainstream smart home solutions like Google Home and Amazon Alexa, which focus on device management and control

Our product	enables a deeper understanding and experimentation with smart home technologies, positioning SHS as a unique educational and developmental tool in the market

1.3 Product Overview

1.3.1 Product Perspective

Product	Similar Features	Differentiating Features	Competitive Advantage
SHS	- Virtual environment for smart home simulation - API interfaces for device behavior emulation	- In-depth educational and experimental platform - Customizable scenarios for advanced testing beyond real-time control	- SHS allows for extensive 'what-if' scenario testing, which is not typically offered by consumer-grade products
Google Home	- Voice-controlled home automation - Integration with various smart devices	- Primarily focused on real-time device management - Limited to Google's ecosystem	- SHS provides a broader educational focus, whereas Google Home is optimized for end-user convenience and control
Amazon Alexa	- User-friendly interface for smart device management - Wide range of compatible smart home products	- Closed system with limited scope for user customization - Focus on voice interaction	- SHS's simulation-based approach is unique and offers a deeper dive into smart home management compared to Alexa's more surface-level control
Apple HomeKit	- Secure and private system for managing smart home devices - Seamless integration with Apple products	- Requires Apple hardware and is limited to HomeKit-compatible devices - Lacks a virtual simulation environment	- SHS is platform-agnostic and does not require specific hardware, offering flexibility and a wider reach
Samsung SmartThings	- Integrates with a variety of smart devices - Offers some level of automation and control	- More hardware-centric, requiring a SmartThings Hub - Focused on device connectivity rather than simulation	- SHS stands out by providing a risk-free environment for testing and learning, which SmartThings does not directly address

OpenHAB	- Open-source platform for smart home integration - Highly customizable and flexible	- Steeper learning curve - Focuses on real-world integration over simulation	- SHS is specifically designed to be user-friendly and educational, potentially serving a different market segment than OpenHAB

1.3.2 Assumptions and Dependencies

Assumptions	**Dependencies**
Users are looking for a simulation platform to understand and innovate in smart home technology, not just a control interface like those offered by Google Home or Amazon Alexa.	The simulator's advanced features and usability must be clearly communicated to differentiate it from the convenience-oriented products in the market.
There is a market need for a tool that can simulate complex smart home scenarios for educational and developmental purposes, which is not currently met by existing consumer-grade products.	Ongoing updates and compatibility with various smart home protocols and devices to ensure SHS remains relevant against platforms like Apple HomeKit and Samsung SmartThings.
Educators and developers prefer a platform-agnostic tool that does not require specific hardware, unlike systems such as Apple HomeKit, which operates within the Apple ecosystem.	The success of SHS may depend on the availability of a robust online community or support system similar to that which supports open-source platforms like OpenHAB.
Potential users have the technical skill or willingness to engage with a more complex system that provides greater control and customization options than mainstream smart home systems.	Dependencies on external APIs and services must be managed to ensure SHS can simulate a range of devices and scenarios accurately.

2. Technology Used

2.1 Control version System

For the control version system, we will be using GitHub.
Here is the link to our repository: christa-ux/Beta (github.com) .
Since we're still in sprint 1, our GitHub is mainly empty.

2.2 Team Collaboration

Concerning team collaboration and communication, Discord is our platform of choice. It allows us to have different channels, which means that our conversations can be divided into types like "general", "sprint1", "documents", etc. This helps with the organization, and it allows for easy access. For example, in the "documents" channel, we only share documents related to our work. For example if someone finished their part, then can send it there, or if we're working on a specific sprint, the instructions would also be found there.

2.3 Monitoring and Verification

Starting Sprint 2, we'll be using commits on Github to track each person's finished tasks. We also have our main branch protected, so that any thing that wants to be merged will have to be reviewed by 2 people first. In addition, to stay on track and on the same page, we do regular meetings, mainly on Discord or Zoom to make sure everyone is okay with their part.
For testing the code, we will be conducting unit testing with hopefully at least 80% coverage, using JUnit.

2.4 Design and Modeling Work

The design of the Context Diagram and Domain Model were done using *draw.io*, a simple software that provides us with built-in tools to draw our models and diagrams. PowerPoint was also used since it provides us with shapes and some teammates are more familiar with it.

2.5 Development Framework

As a first decision, we opted for React as the framework for JavaScript for the front-end. Concerning the back-end, we'll be using Java, and we're considering Spring boot as it offers rich functionality for web-applications.

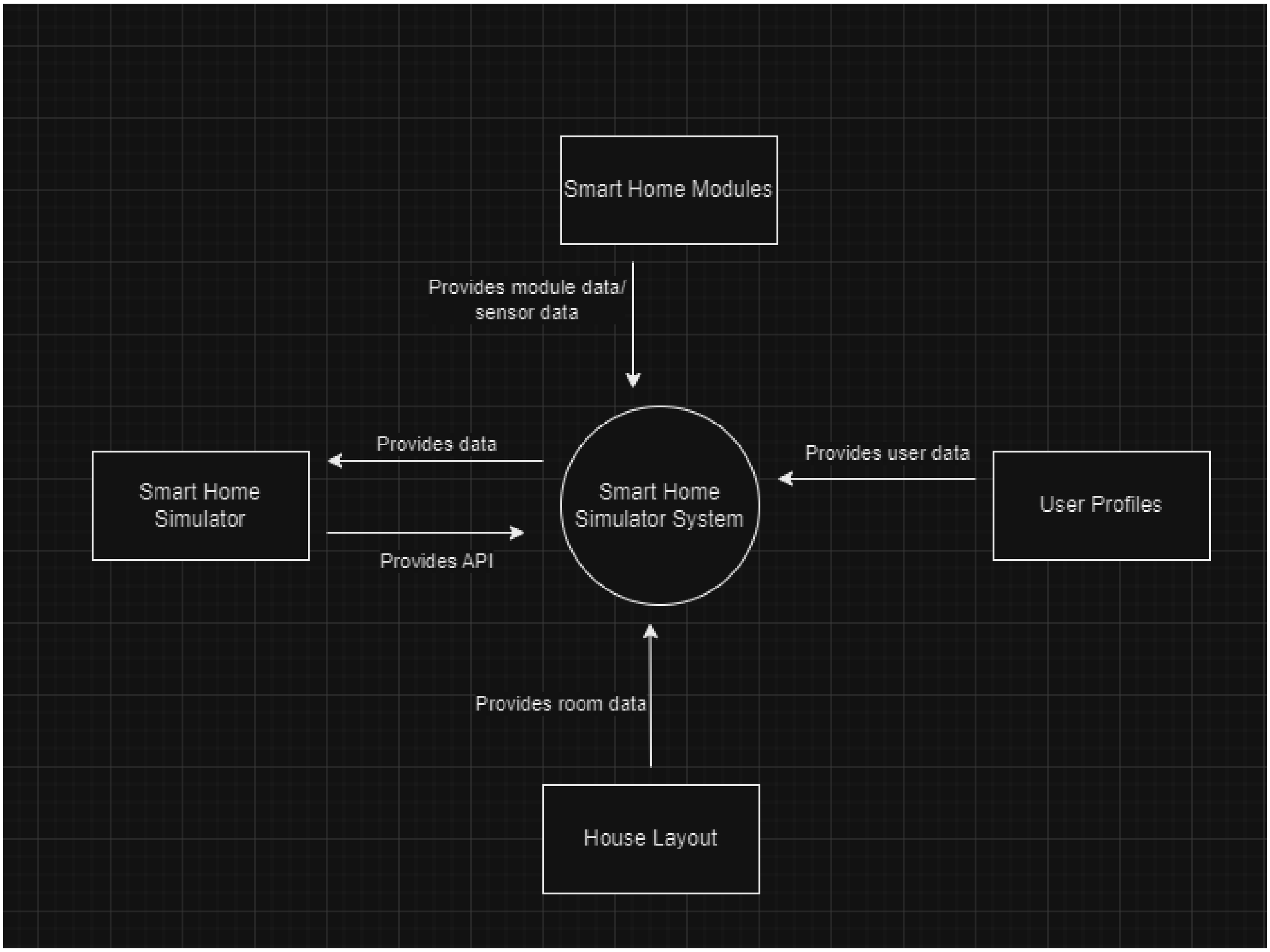
2.6 Coding

For the development of the “Smart Home” simulator, we discussed, as a team, that the most appropriate and direct programming languages to work with are: HTML, CSS, JavaScript and Java. The frontend, mainly the looks and the visual design of the simulator, will be implemented using the first 4 languages. As for the backend, which has to do with how the system functions, the storing of information and the interaction between entities, it will mainly be developed using Java which is an Object-oriented Programming language.

For simplicity, you can look at the following table to see which technology will be used for which activity:

Activities	Used Technology
Control Version System	GitHub and Moodle
Team Collaboration	Discord
Monitoring and Verification	GitHub, Discord and Zoom
Design and Modeling Work	Draw.io and PowerPoint
Development Framework	ReactJS/React Native, Spring boot
Coding	HTML, CSS, JavaScript and Java

3. Context Diagram



<u>Figure 1</u>: The context diagram displays how the external factors affect the system. The house layout is taken from a text file with the rooms and its configurations. User profiles are created to interact with the system as family members, guests or strangers. The modules are the elements the systems work with such as heating and security. Finally, there is the simulator with the dashboard and API that makes it all work together.

4. Domain Model

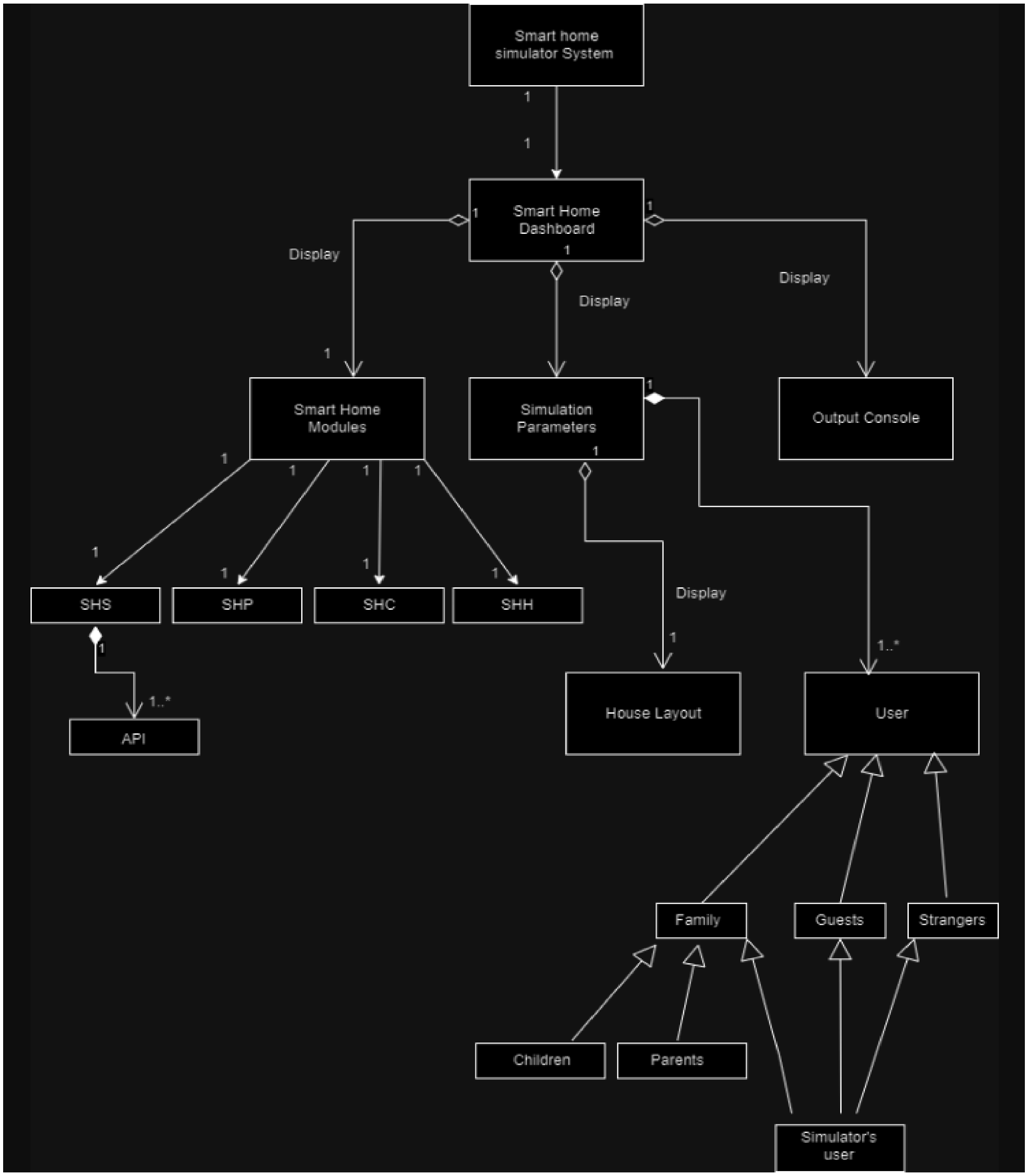


Figure 2: Our smart home system simulator is centered on the dashboard which is the main interface. This dashboard is divided into 4 sub parts: System Parameters, Output Console, House View, and Smart Home Modules. Moreover, users can be of different types, and the simulator user has access to every type, in addition to using the system simulator as a whole.

Phase 2

5. System Architecture

5.1 UI Layer

UI Layer is the section in which all the coding related to what the user sees, the visual characteristics and effects. It is the group of all the interactions between the user and the computer. For example, the organization of the simulation dashboard, the organization of the map in which the user sees the other users' location and the color and image of icons used to show the status of each functionality. Another example, when the user clicks on the button to turn on the lights, is there a vibration or a sound to confirm that the user pressed on the button and the system received the command.

5.2 Application

Application is the section in the system architecture that handles the requests of the users. It is part of the backend of coding. The user does not see the process in which the information retrieved from their command is read and causing a chain of reaction. For example, if the user wants to turn on the lights of a room, they just press the button corresponding to turning on the lights. Application everything that happens until the lights of the room turn on. So, the system retrieving that the user clicked the button in the section of turning on the lights, sends a message to the device that would close the circuit so that the electricity passes through, and the light bulb emits light.

5.3 Foundation

The foundation layer of the system architecture serves has a helper to the rest of the architecture layers. Its purpose is to make the implementation quicker and easier; this could be through the user of specific frameworks or built-in libraries. This layer is specifically for the developers who are working on creating the other layers.

6. Use Cases

ID:	1
Title:	Displaying the House Layout on the Dashboard
Description:	The house layout functionality includes everything needed to open and import a house layout file into the simulator, complete with mechanisms for handling exceptions and errors. The layout file, in text format, details the rooms present in a house, including their names and quantities of windows, lights, and doors. This layout is then visualized as a 2D drawing on the dashboard.
Primary Actor:	The client (researcher, student, or practitioner), main user of the smart home simulator, in order to monitor the house and each room.
Preconditions:	User logged in and layout house file provided for the information about the number and name of the rooms in the house.
Postconditions:	2D display of the house layout on the dashboard, with specific icons on each room representing any action happening.
Inputs:	Layout house file
Outputs:	2D house layout model
Main Success Scenario:	User logs in and uploads the layout house file containing all the necessary information. Afterwards, the 2D house layout will be displayed on the dashboard to monitor each action happening in the rooms.

ID:	2
Title:	Simulation Parameters
Description:	In this use case, the user is able to do the following things: -add, remove, edit a user profile -set the date and time -log into an existing profile and choose the layout of the house to be tested. -Add restrictions for each user type so the system grants/denies access to certain commands. -The system should save the profiles and their restrictions for later simulations.
Primary Actor:	The user is the primary actor. The user type is a parent, a child, a stranger, or a guest.
Preconditions:	The user should log in as one of the profiles that exist in the system.
Postconditions:	The commands chosen by the user should be executed or not, depending on the permissions that the user has.
Inputs:	user type, house layout.
Outputs:	The commands
Main Success Scenario:	The user chooses a profile (user type). The user can create a profile if it does not exist, and the user type should be chosen as well as the restrictions if any. Then the user chooses the house layout. After that the user starts the simulation by choosing commands to be executed. The system should execute these commands if the user type permits.

ID:	3
Title:	Manage Simulation Context
Description:	This use case allows the simulator user to control various aspects of the simulated environment, including time progression, simulation state, and environmental conditions, as well as the movement and location of individuals.
Primary Actor:	Simulator User
Preconditions:	The simulator is installed, operational, and the user has been authenticated with the necessary permissions.
Postconditions:	The simulator reflects all changes made to the context, including any modifications to time speed, simulation state, user and people placement, outside temperature, and the status of window obstructions.
Inputs:	User commands for changing time speed, starting/stopping the simulation, modifying date and time, user movements, placements of people, temperature adjustments, and window blocking objects.
Outputs:	Updated simulation context reflecting the new time speed, simulation state, date and time, user and people locations, outside temperature, and window status.
Main Success Scenario:	The user successfully changes the time speed, starts/stops the simulation, adjusts the date and time, relocates users and people, changes the outside temperature, and blocks/unblocks windows as desired.

ID:	4
Title:	Smart home core functionality (SHC) module
Description:	In this use case, the user is able to do the following things: • open/close doors • open/close windows • turn off/on lights • set Auto mode for when user enters a room
Primary Actor:	The system user is the primary actor as he/she has access to the inputs in the dashboard.
Preconditions:	User is logged in, User is in a house layout
Postconditions:	Depending on the user permissions, the user can execute the commands.
Inputs:	User type, house layout
Outputs:	The commands
Main Success Scenario:	With a user profile and house layout chosen, the system user can use the options on the dashboard to open/close windows and doors, open/close the lights and set the selected user auto mode where light open and close automatically

ID:	5
Title:	Dashboard (Table of actions that can be done to the house parameters)
Description:	The dashboard contains most of the actions a user can perform through clicking a button. Each user according to their restrictions will be able to perform a certain number of actions. This is the section that interacts with the user. Any time the user wishes to perform an action in the house or in its profile, the user clicks a button, and it should the action should be performed.
Primary Actor:	User signed in as a parent, child or guest
Preconditions:	1. If the user is a parent, they can turn on/off the lights, open/closed windows, and garage, and lock/unlock the doors at all times. 2. If the user is a child or a guest, they can only turn on/off the lights, open/closed windows in the room they are in. 3. If the user is a stranger, you have no permissions under any circumstance. User should be logged in as either parent/child/guest
Postconditions:	
Inputs:	assigned buttons to perform each allowed action
Outputs:	Icons should appear on the map in each room showing the following info: - Lights turn on/off - Door locked/unlocked - Window open/closed - Garage door opened/closed - User signed in/out

Main Success Scenario:	- When a user turns on/off the light of a room through the dashboard, the lights in the room should turn on/off. - When a user locks/unlocks the doors of a room through the dashboard, the doors in the room should lock/unlock. - When a user opens/closes the windows of a room through the dashboard, the windows in the room should open/close. - When a user opens/closes the garage door through the dashboard, the garage door should open/close. - The user should be able to see anyone that is in the parameters of the house. - If the user wants to sign out or change accounts, the system should successfully logout and sign in the current/new user.

7. Design Patterns

7.1 Factory Method

The factory method is used for the creation of users. It is the ideal design pattern when creating different types of users because the factory method is based on creation through an interface in a superclass and allows subclasses to modify the new object during its creation.

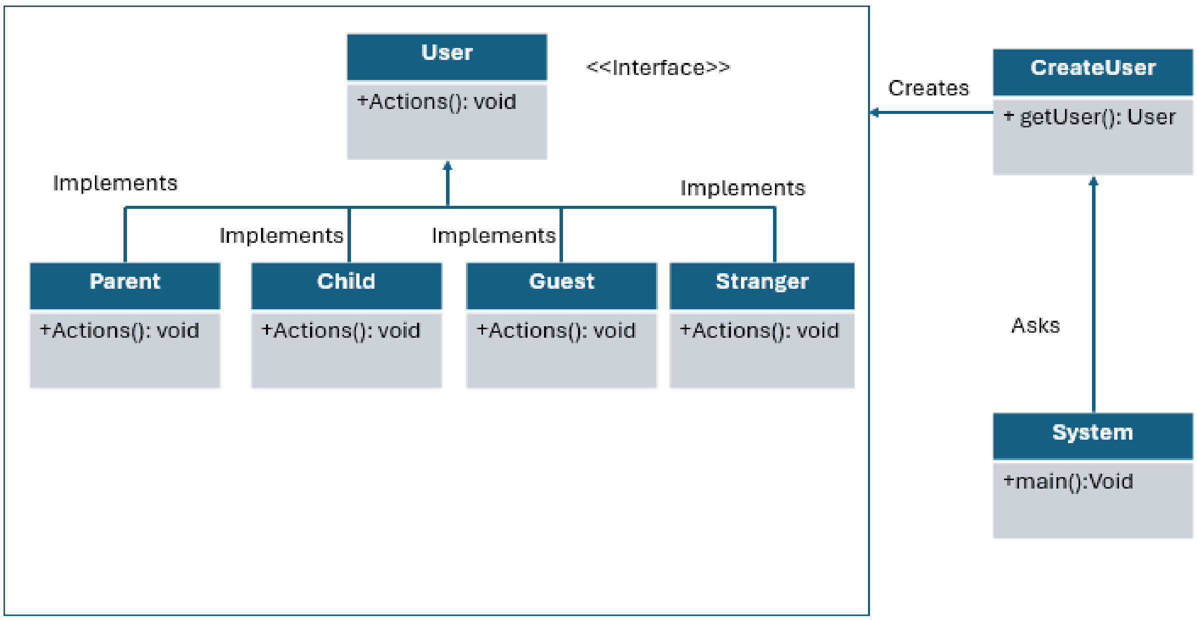


Figure 3: The Factory Method design pattern implemented through the handling of user creation.

7.2 Composite pattern

The composite design pattern is used for the implementation of the house layout since the intent of that design pattern is to structure objects into a hierarchy in a tree form. In the case of the house layout, it starts with 1 house, then it breaks down into a small number of rooms (e.i. bedrooms, kitchen garage, etc…) and in each room there will have a specific (according to the type of room) number of lights, doors, and windows. The number of objects increases the deeper down the tree. It started from one “House” object to a few dozen of light, door and window objects.

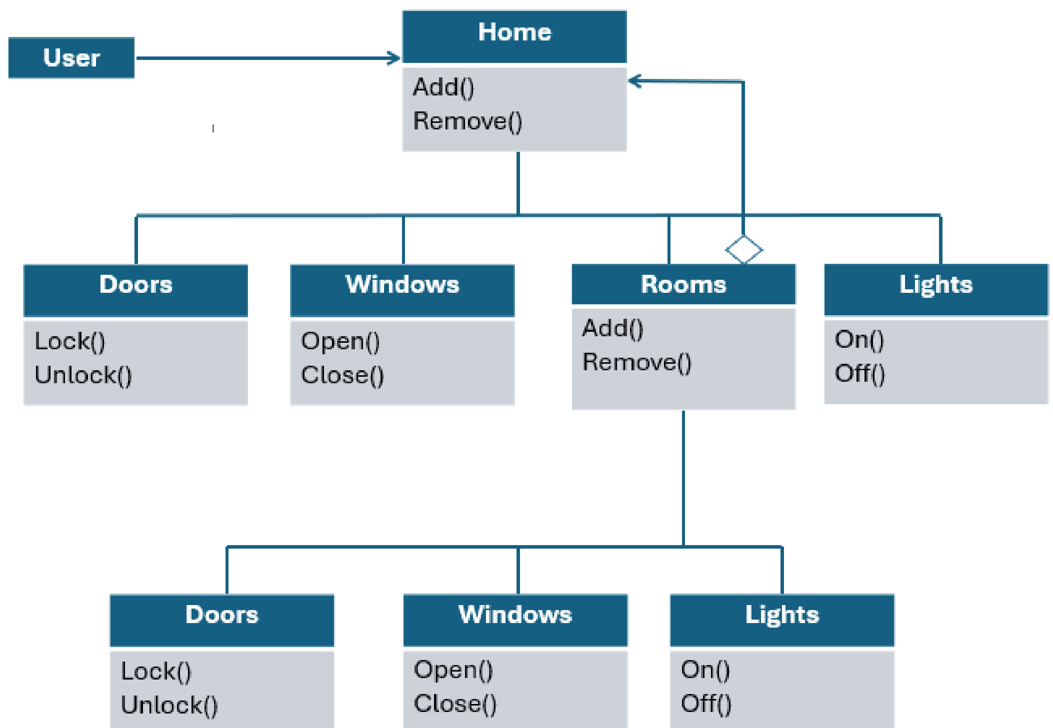


Figure 4: The Composite design pattern implemented through the handling house layout.

7.3 Singleton

The Singleton is the perfect design pattern for the implementation of the dashboard since it relies on a class having only one instance while giving a global access point to the instance. There should not be many dashboards, rather a single dashboard in the simulation that can be accessed by all existing users. The dashboard should include all the existing actions the user can perform, but depending on the type of user, some actions can be enabled or disabled in the dashboard, giving a level of restriction according to each user. Therefore, instead of creating a new instance of dashboard every time a new user appears, it would be easier for each user to access the dashboard.

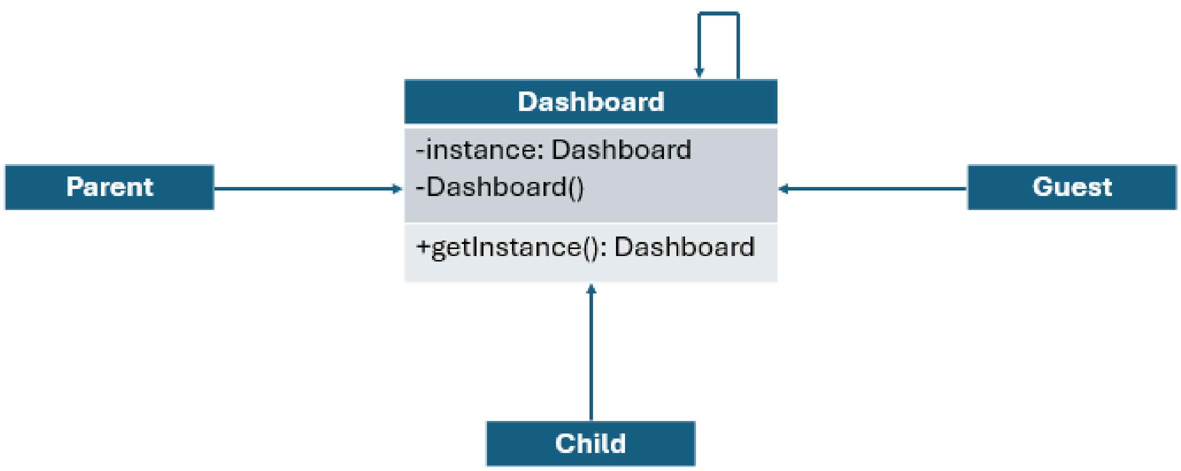


Figure 5: The Singleton design pattern implemented through the dashboard handling the restrictions of each user type.

8. Implementation of the Use Cases

Implementation of the Use Cases can be found on GitHub with the implemented code. The link to the repository can be found in section 2.1.

9. Reference

Context diagram. IST Project Management Office. (2023, October 5). https://uwaterloo.ca/ist-project-management-office/tools-and-templates/tools/context-diagram#:~:text=Context%20diagrams%20show%20the%20interactions,system%20will%20be%20part%20of.

Top 51 software development frameworks. Top Software Outsourcing Company in Vietnam - Orient Software. (n.d.). https://www.orientsoftware.com/blog/software-development-frameworks/

Johns, R. (n.d.). Want to level up in Java? Use these Java frameworks! *Hackr.io*. https://hackr.io/blog/java-frameworks