Danmarks Tekniske Universitet

02291 System Integration

Exercise 2:

Design of MUD game

Mikkel Riber Bojsen
s093255

Johan van Beusekom
s093251

Andreas Foldager
s093285

Kasper Aaquist Johansen
s112461

Martin Kasban Tange
s093280

March 19th 2013

Contents

1 Requirements

This chapter will outline the requirement analysis of the Mud Game. We will provide an analysis of domain with both glossary and domain model. We will then provide use case diagrams outlining the functional requirements of the system and give 3 detailed use cases. Finally we will describe the non-functional requirements.

1.1 Domain Analysis

In this section we provide a glossary of important terms in the domain along with descriptions of the individual terms. We provide a class diagram showing a model of the domain and finally we provide acceptance tests for the system.

1.1.1 Glossary

Player The person interacting with the server and the game. *(Optionally: user)*

Character The player of the game is controlling a character through the levels.

Server The server communicates with the players of the game and is responsible for authenticating players before they enter a game. The game objects are also kept on the server, such that different players can be sure to get the same representation of the game. Trading and communication between players also runs through the server.

Game The MUD game consists of a number of levels. All levels have to be completed in order to win the game.

Level A level consists of a set of rooms that are connected. The level is completed when the player reaches the special room.

Room Each level consists of a number of rooms. Each room can contain an unrestricted number of objects and has at least one connection to another room on the same level.

- **Start** At the beginning of each level the player is placed in the start room.
- **Special** Upon reaching the special room of a level, the level is completed and the player advances to the next level or wins the game.

Inventory Each player has an inventory in which up to five objects can be kept.

Object Objects can be found throughout the rooms of each level. A player can pick up objects and place them in his inventory.

Trading The process of two players exchanging objects that they have in their inventory.

1.1.2 Class Diagram

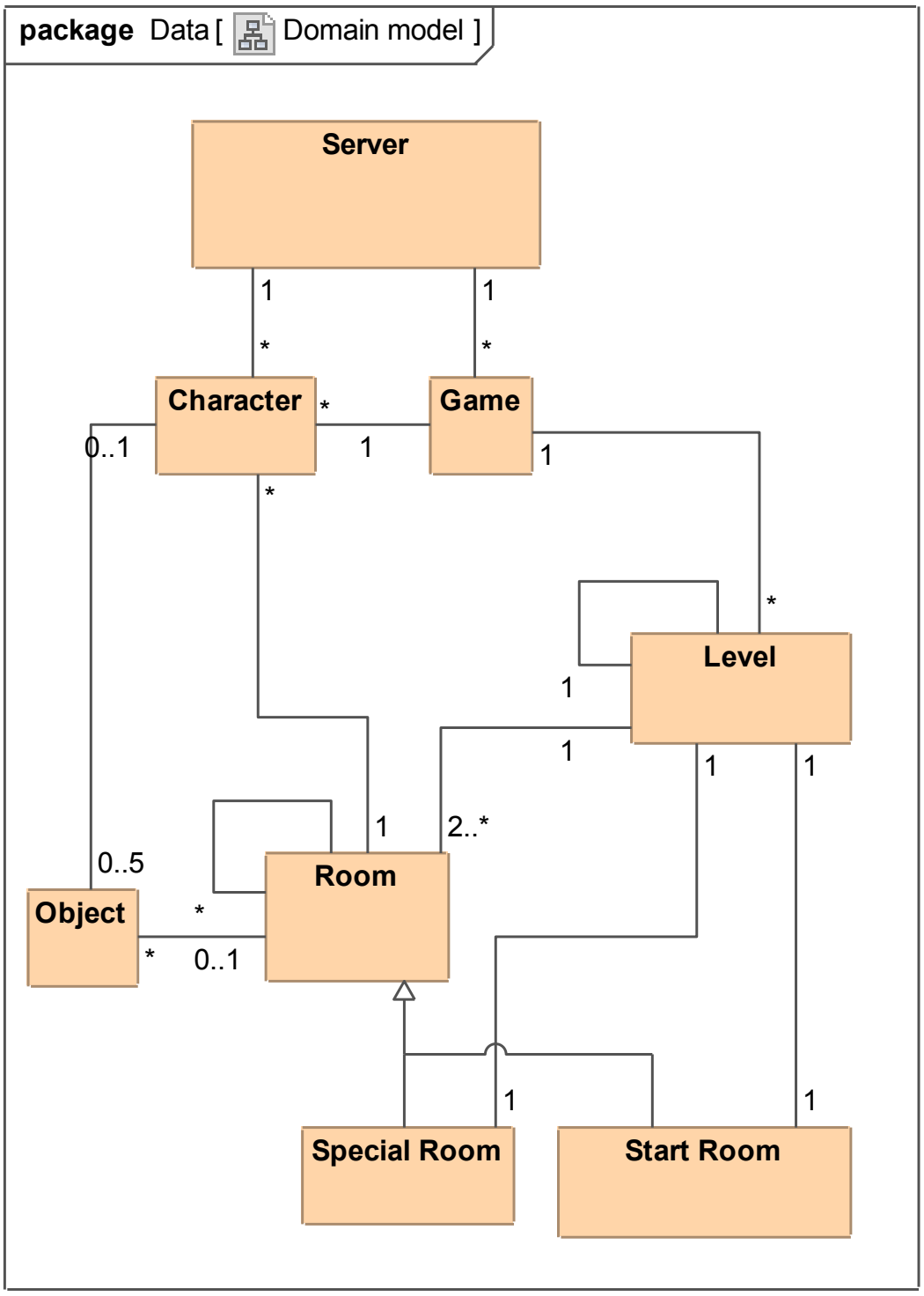

Figure 1.1: Class diagram - showing the domain model.

1.2 Functional Requirements

In this section, we provide an overview of the functional requirements for the system. We first show a series of use case diagrams where use cases are grouped with similar use cases. Afterwards we provide three detailed use cases.

1.2.1 Use Case Diagrams

Inventory Management

Fig. 1.2 depicts the Use Cases that the player can do with his inventory/item management. He will at all times be able to pickup any item that might be in the room, or look at his inventory. However in order to drop or trade any items, he must first see his inventory.

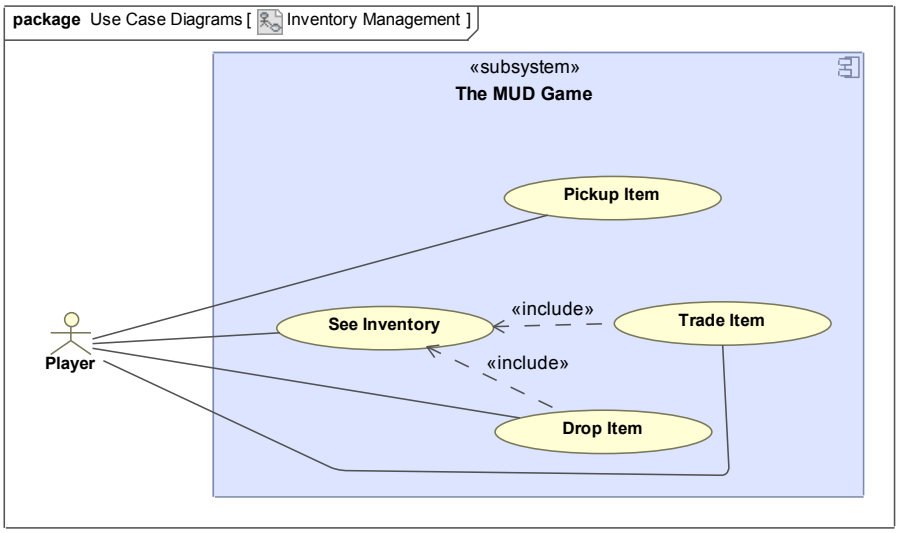

Figure 1.2: Inventory Management

Room Management

The player has three basic Use Cases that he might execute while being in a room, i.e. Move to Room next to the current one, Look at Room to see what items are located in the room or Speak to whoever might be located in the room. We have one special Use Case as seen in 1.3, namely Complete Level, which will be executed by the game itself, whenever the player enters the last room in the level.

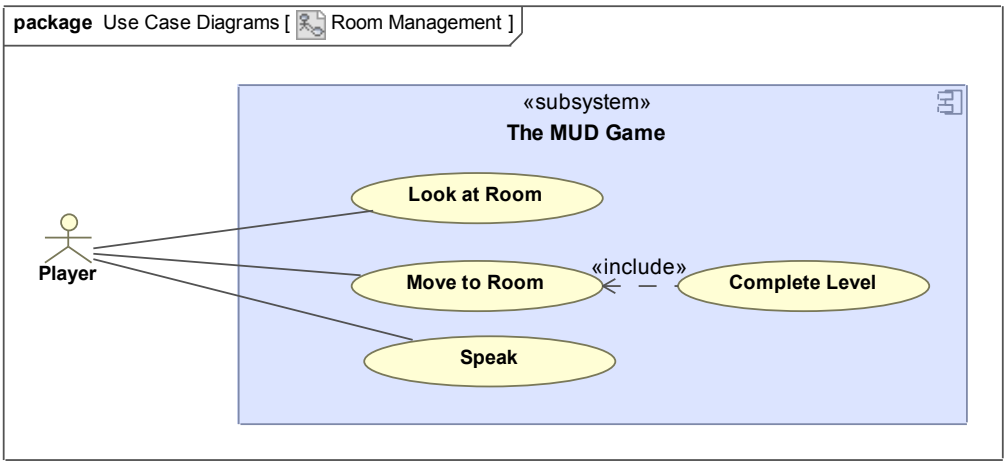

Figure 1.3: Room Management

Trade Management

Whenever the player decides to Trade Item, he can make an offer to another Player by using the Offer Trade option, which enables him to trade any items he has achieve during the game. Whenever a Trade has been offered, the other Player has the option to either Accept, Decline or make a counter offer.

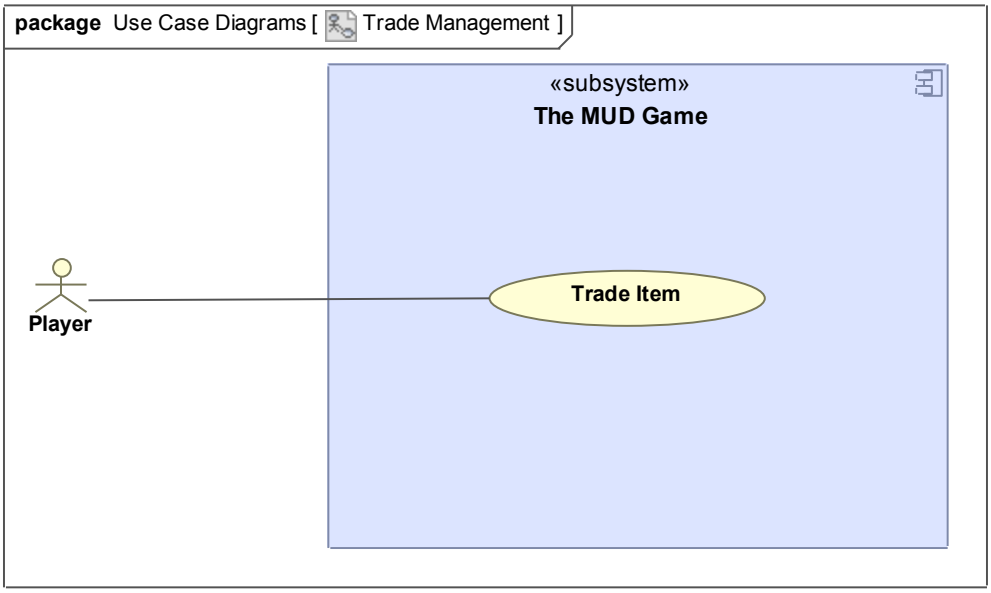

Figure 1.4: Trade Management

Character Management

With respect to Character Management, as seen in fig. 1.5, we have given the Player the possibility of performing typical CRUD operations on a characters, where Read, Update and Delete all depends on the Player having previously created a character.

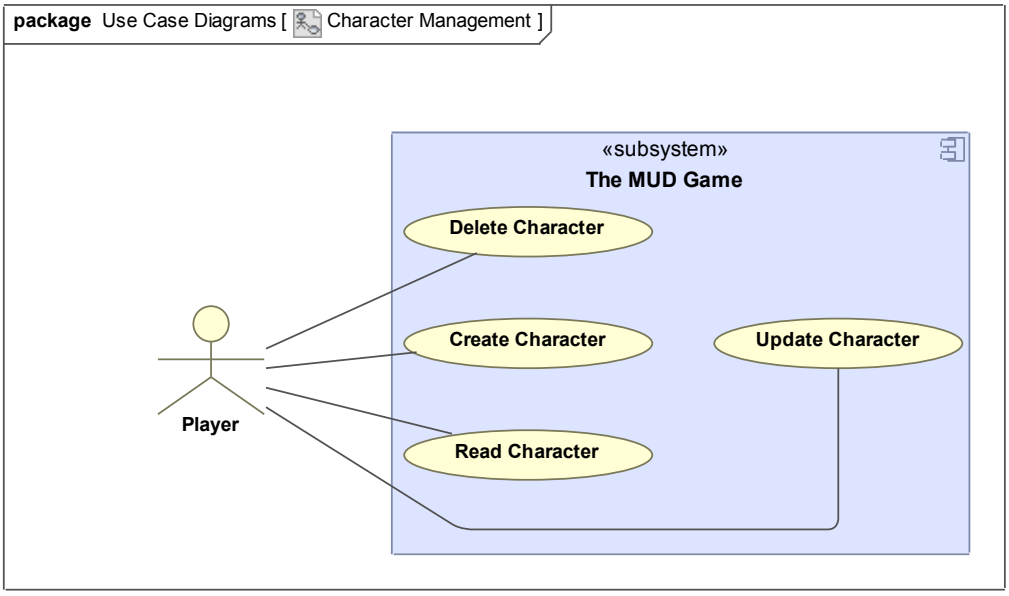

Figure 1.5: Character Management

Access Management

For Access Management, we have that all Use Cases rely on the Player gaining access to the Server, by using a Logon Server, which in turn will make the server Authenticate the Player. When logged on, it is possible for the Player to Join Game, which will join an ongoing game, or Logoff Server. The Leave Game is only enabled when the Player has joined an ongoing game, and decides to leave. The Player can also leave a game, by logging off the server at any time.

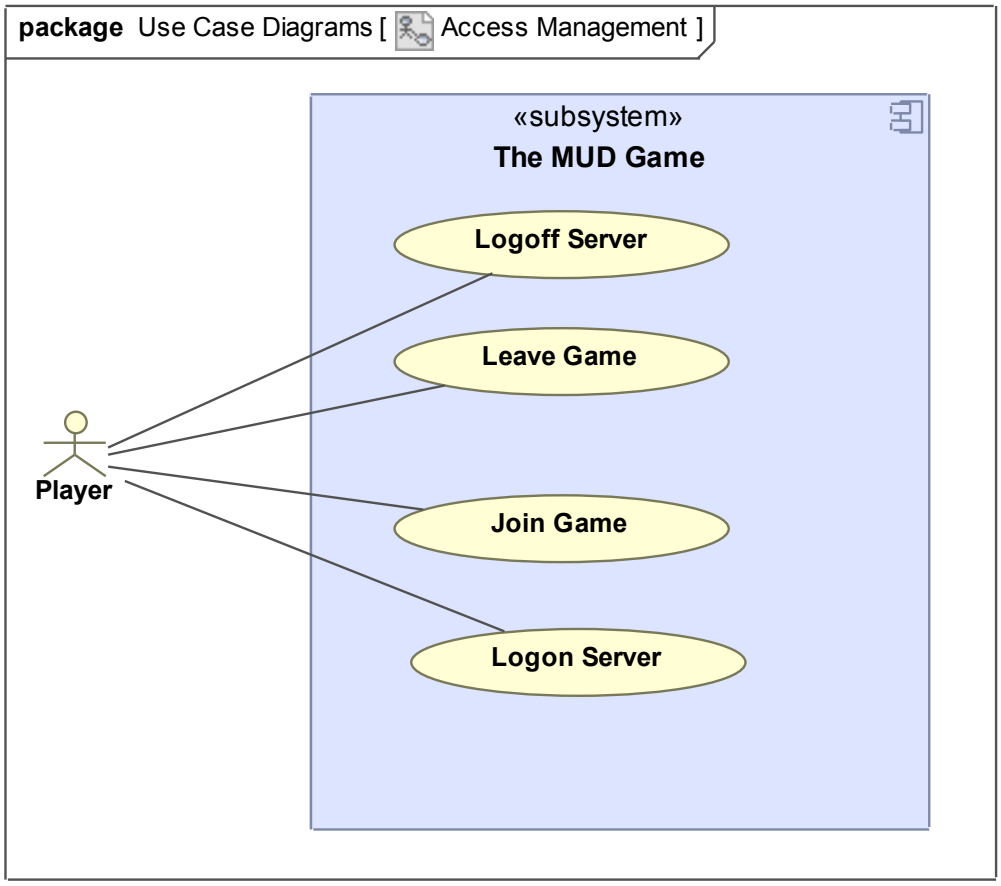

Figure 1.6: Access Management

1.2.2 Detailed Use Cases

The following subsection will give a detailed description of the three Use Cases: **Create Character**, **Delete Character** and **Offer Trade**.

Use Case 1	**Create Character**
Description:	The player creates a new character on the server
Actor:	Player
Preconditions:	

Main Scenario:

1. The player provides the server with information about what character name, e-mail address and password he wishes to use on the server.
2. The player presses the "Create Character" button.

Alternative Scenario:

2.a Character name already in use:

1. System shows a failure message, explaining the issue
2. The player retries step 1 with a new character name

2.b Invalid e-mail address:

1. System shows failure message, explaining the issue
2. Player returns to step 2 and corrects the e-mail address

2.c Password doesn't comply with the rules set up on the system

1. System shows a failure message, explaining the issue and what the password has to follow
2. The player retries step 1 with a new password, following the rules

Postconditions:	The player has created a character on the server.

Use Case 2	**Delete Character**
Description:	The player wants to delete one of his existing characters
Actor:	Player
Preconditions:	Player has authenticated his account on the server. Player has to have one or more characters created on the server

Main Scenario:

1. The player provides the server with information about which character he wants to delete.
2. The player confirms the deletion providing the password for the character.

Alternative Scenario:

1.a Invalid character name:

1. System shows failure message, explaining the issue.
2. Player returns to step 1 and corrects the errors

2.a Wrong password provided:

1. System shows failure message, explaining the issue.
2. The player retries step 2.

2.a Wrong password provided - three times:

1. System shows failure message, explaining the issue.
2. The player gets returned to main menu.
3. The player that has registered the character get a notification regarding the deletion request.

Postconditions:	The player has deleted a character he had a password for.

Use Case 3	**Offer Trade**
Description:	A player wants to offer a trade from his character to another character.
Actor:	Player1, Player2
Preconditions:	Player1 and Player2 have to have joined the same game, and both of their characters have to be in the same room.

Main Scenario:

1. Player1 offers a trade with another character in the room, by choosing a desired item to trade with from his own characters inventory.
2. Player2 who gets the offer confirms the trade and chooses an item from his characters inventory to trade with.
3. Player1 confirms the offer.
4. The trade is done.

Alternative Scenario:

2.a Player2 declines the offer:

1. Player1 is shown the decline message from Player2.
2. Player1 returns to step 1 and can choose to offer a different trade.

3.a Player1 declines the counter offer:

1. System offers a text field to Player1 where he can type a message to Player2 to let him know the offers has been declined.
2. Player1 retry step 1.

Postconditions:	The players have successfully completed an item trade.
Notes:	

1.2.3 Acceptance test Fit tables

Create Character

Use case "Create Character" success scenario

ActionFixture		
start	Player	
enter	character name	Marnizzle
enter	password	Fisk1234
enter	email	mktange@gmail.com
press	Create Character	
check	error message	

Alternative scenario 2a: the character name is already in use

ActionFixture		
start	Player	
enter	character name	Marnizzle
enter	email	mktange@gmail.com
enter	password	Fisk1234
press	Create Character	
check	error message	Character name already in use

Alternative scenario 2b: Invalid e-mail address

ActionFixture		
start	Player	
enter	character name	Marnizzle
enter	email	mktange@gmail,com
enter	password	Fisk1234
press	Create Character	
check	error message	Invalid e-mail address

Alternative scenario 2c: Password does not follow the set rules

ActionFixture		
start	Player	
enter	character name	Marnizzle
enter	email	mktange@gmail.com
enter	password	Fiskefars
press	Create Character	
check	error message	Password must contain numbers

Delete Character

Use case "Delete Character" success scenario

ActionFixture		
start	Player	
enter	character name	Marnizzle
enter	password	Fisk1234
press	confirm	
check	error message	

Alternative scenario 1a: Invalid character name

ActionFixture		
start	Player	
enter	character name	Marnizzlee
enter	password	Fisk1234
press	confirm	
check	error message	Character name does not exist

Alternative scenario 2a: Wrong password provided

ActionFixture		
start	Player	
enter	character name	Marnizzle
enter	password	Fisk123
press	confirm	
check	error message	Wrong password provided for the character

Alternative scenario 2a: Wrong password provided - three times

ActionFixture		
start	Player	
enter	character name	Marnizzle
enter	password	Fisk123
press	confirm	
check	error message	Wrong password provided for character. This action has been tried three times

Offer Trade

Use case "Offer Trade" success scenario

ActionFixture		
start	Player	
check	Inventory	Bottle
enter	offer trade	Bottle
check	get offer back	true
enter	get offer	Torch
enter	confirm trade	true
check	Inventory	Torch
check	error message	

Alternative scenario 2a: Second player declines the offer

ActionFixture		
start	Player	
check	Inventory	Bottle
enter	offer trade	Bottle
check	get offer back	false
check	error message	offer declined

Alternative scenario 3a: First player declines the counter offer

ActionFixture		
start	Player	
check	Inventory	Bottle
enter	offer trade	Bottle
check	get offer back	true
enter	get offer	Torch
enter	confirm trade	false
check	Inventory	Bottle
check	error message	

1.3 Non-functional Requirements

This section provides a list of non-functional requirements we have derived based on the description of the game combined with some assumptions about the business case for creating the game.

Availability

Working days (Monday - Thursday) The system should be available 100.0% of the time between 08:00 AM to 04:00. The system should be available 95% of the time between 04:00 AM and 08:00 AM in order for updates to be rolled onto the server.

Weekends (Friday - Sunday) The system should be available 100.0% of the time at any time during the day.

Restartability

- The system should be able to restart from a crash in at most 2 minutes, and an average of at most 1 minute.

Portability

- The server side software must be able to run on Linux and Windows servers.
- The client side must be able to run on any smartphone OS.
- It must be possible to port the game to 95 % of new OS's within two weeks of work.

Maintainability

- A programmer with 1 year of experience must be able to fix 90 % of confirmed bugs in 5 hours.

Privacy

- Only the player himself as well as administrators are able to see the player profile info after being authenticated in the system.

Usability

- An experienced player should be able to perform any feature on a average time of 30 seconds.
- A player with no prior experience must be able to play the game with all features (chatting, trading, object manipulation and movement) within 15 minutes.

Security

- All player input is validated before processing, in order to avoid e.g. SQL injection.

- A player has to authenticate himself before being able to join a game.

Installability

- Initial installation for the end-user should at most take 15 minutes.
- Updates for the game is notified to the end-user through notification when the app is started, and the user will be required to download any updates before he can use the game.
- It must be possible to set-up the game server for a person with 1 years of experience within 1 hour of work.
- A person with 1 year of experience must be able to set up a game with all necessary parameters within 15 minutes.

2 Design

This chapter will outline the design of the Mud Game. We will provide a rough system design using CRC cards to describe the detailed use cases mentioned in the above section about *Detailed Use Cases 1.2.2*. We will also provide a component design alongside state machines outlining the protocols used to communicate between the components. Finally we will provide a detailed Class diagram and behaviour diagrams of the flow in the program.

2.1 Rough System Design

In this section we use CRC cards to give a rough design of the system. The CRC cards are focusing on the three detailed Use Cases mentioned in the *Detailed Use Cases 1.2.2* section. Each CRC card show the responsibilities for the Classes we intended to have in our *Detaile Class Design 2.3*. The CRC card are also showing the collaborating classes.

Class: Server	
Responsibility:	Collaboration:
Authenticate Player Create Player Registered Games[] Create New Game Give List Of Active Games	Game

Class: Player	
Responsibility:	Collaboration:
Join Server Create Player/Profile Get Active Games[] Create Game Join Game Move Character Make Trade With Character Check Character Inventory	Server Game Character

Class: Character	
Responsibility:	Collaboration:
Create Character Get Info On Room Get Inventory Info Add To Inventory Remove From Inventory Goto Room Leave Room	Inventory Room

Class: Game	
Responsibility:	Collaboration:
Create Game Create Character IsActive Make Moves etc. With Character	Character

Class: Room	
Responsibility:	Collaboration:
Knows Characters Knows Items Knows Rooms Add/Remove Items Add/Remove Character	Item/Object Character

Class: Inventory	
Responsibility:	Collaboration:
Provide List Of Items Knows The Items Check If Full Add/Remove Item	Room

2.2 Component Design

In this section, we provide an overview of the Component Diagram for the system. We first show the component design and then we provide five state machine diagrams showing the port protocol used between the components.

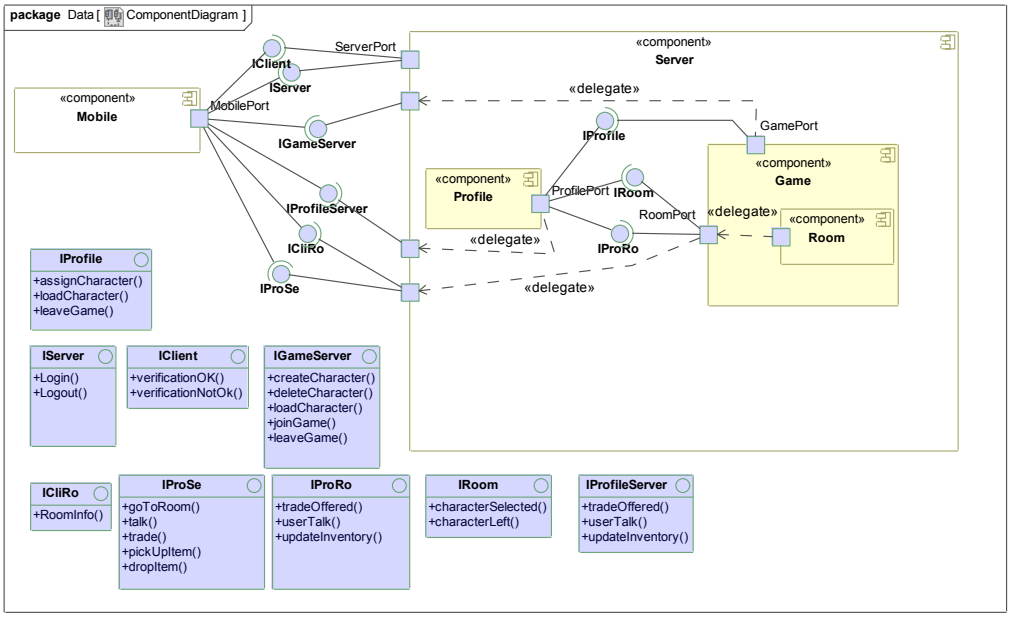

Figure 2.1: Component Diagram of the MUD game

The above diagram shows the Component diagram and the interfaces for the MUD game. All the ports a described using a state machine. These diagrams are shown below.

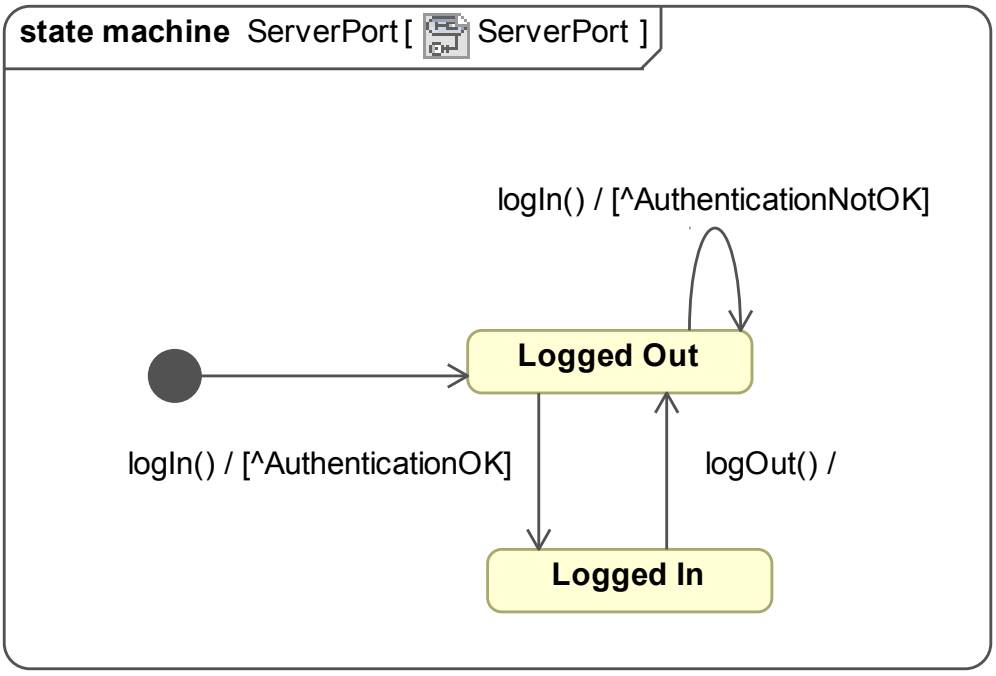

Figure 2.2: State machine for the Server port

The Server port shows the messages send when a Player tries to logon to the Server. If the player is not authenticated, meaning wrong password is provided or username is incorrect, the player will stay in the *Logged Out* sate. If authentication is ok the player will move to state *Logged*

In.

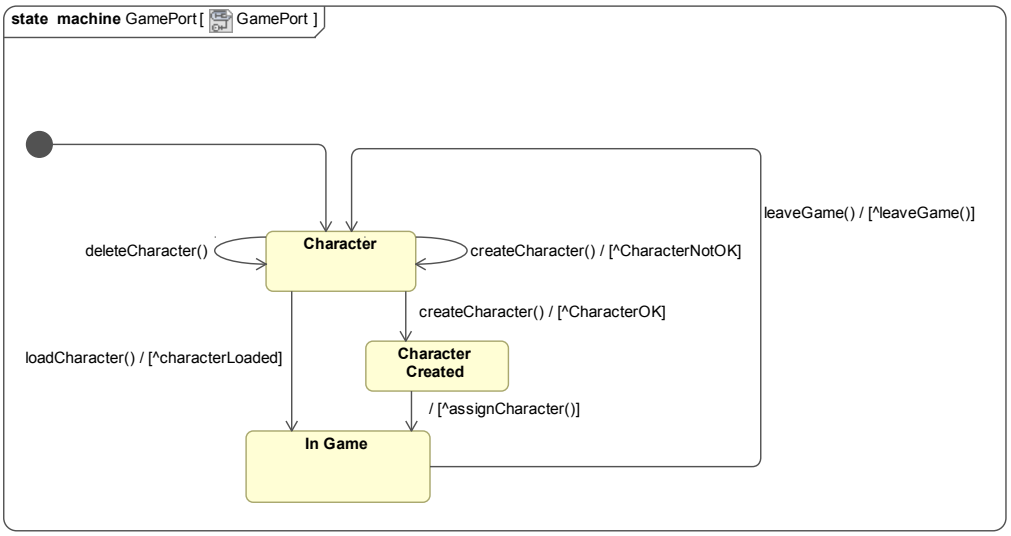

Figure 2.3: State machine for the Game port

When the player has logged in he will be in the *Character* state choosing either *deleteCharacter*, *loadCharacter* or *createCharacter* letting him move to the next state. If *createCharacter* is chosen and it succeed the next state is *Character Created* and from here he is moved to the *In Game* state. If he choses to *loadCharacter* and it succeed the next state is *In Game* too.

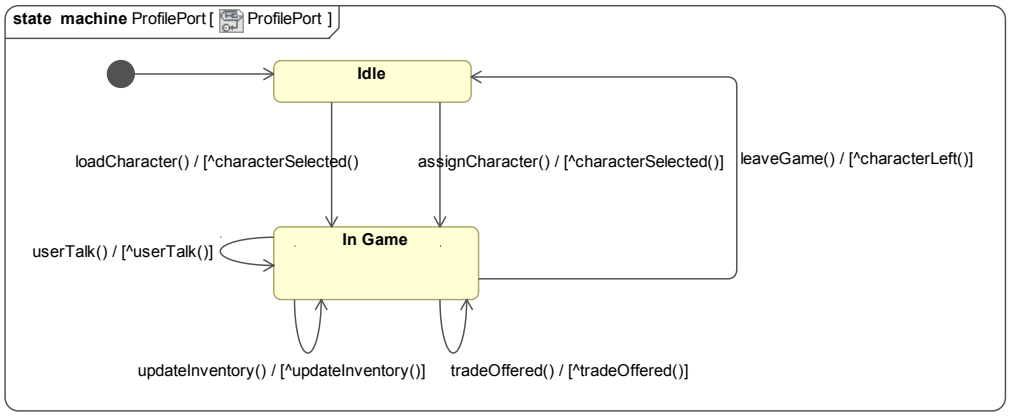

Figure 2.4: State machine for the Profile port

When the player has loaded a character the *Profile* for the character is loaded. The profile enables the character to use specific actions. The *updateInventory* action make sure that the inventory is updated whenever a trade has been done etc.

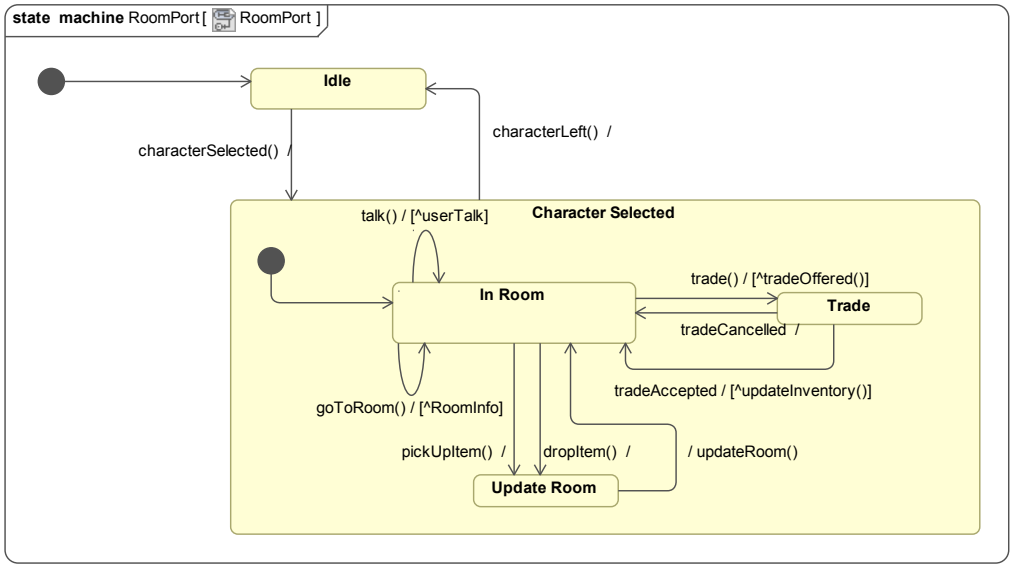

Figure 2.5: State machine for the Room port

When the player has chosen a character and it has been loaded successfully, then the player will have entered the *In Game* state. In the MUD game each character will move from room to room and when the character is in the room a set of actions will be available depending on the other characters, if any, there is in the room. And also if the are Items that can be collected from the room etc.

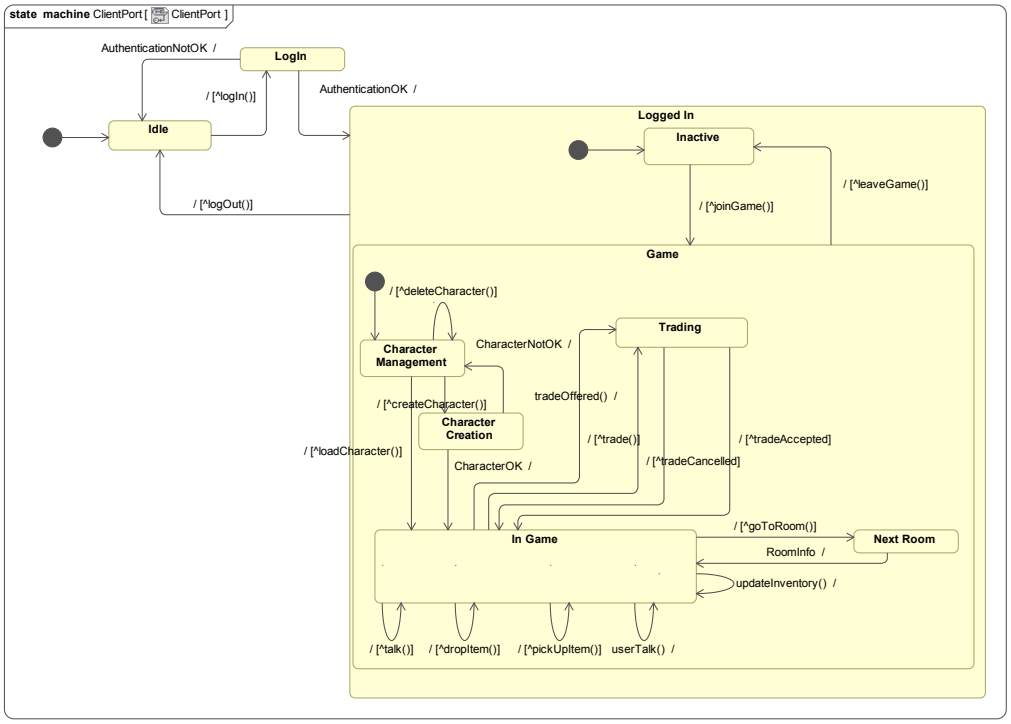

Figure 2.6: State machine for the Client port/Mobile port

The Client Port/ Mobile port shows the stats a player can be in when either using the MUD

game or taken a break from it. The above diagrams has shown in more details the stats the player can be in when he has successfully logged in to the server so here it is left out of the description.

2.3 Detailed Class Design

In this section, we provide an overview of the Class Diagram for the MUD Game.

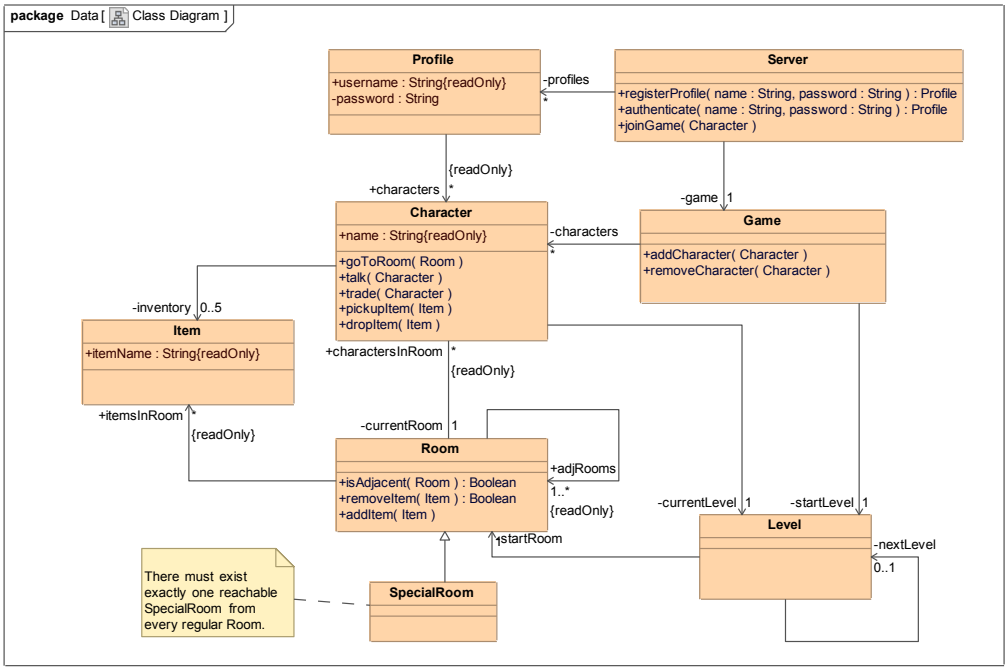

Figure 2.7: Class Diagram of the MUD game

This diagram shows the interconnection between the classes and the most important methods and attributes for each class. The next section *Behaviour Design 2.4* will describe the behaviour of the classes in more details.

2.4 Behaviour Design

In this section, we provide an overview of the design by showing the behaviour of each class using a state machine. Some classes do not have any methods and can therefore not change their state. These classes are not shown in this section.

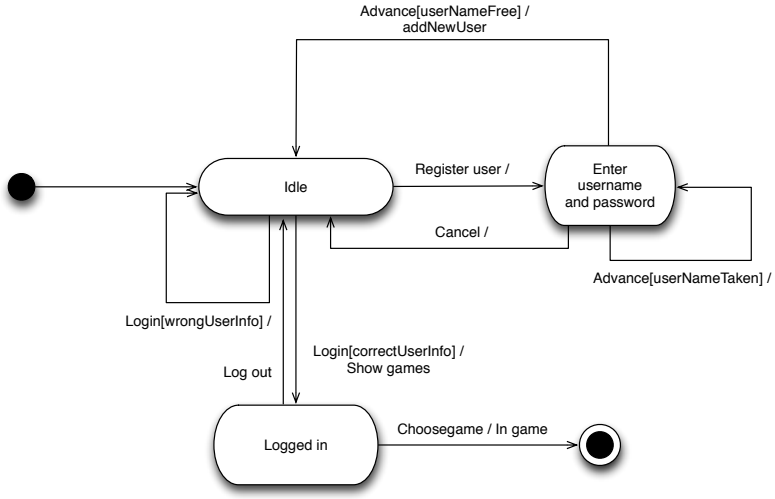

Figure 2.8: State machine for the server class

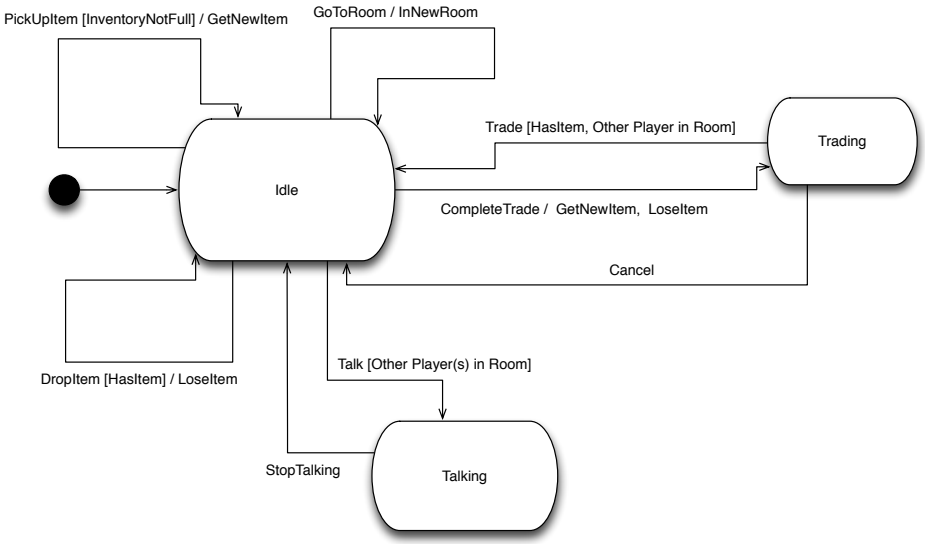

Figure 2.9: State machine for the character class

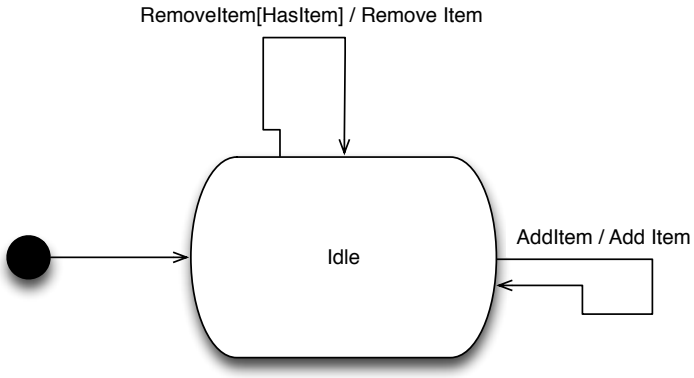

Figure 2.10: State machine for the room class

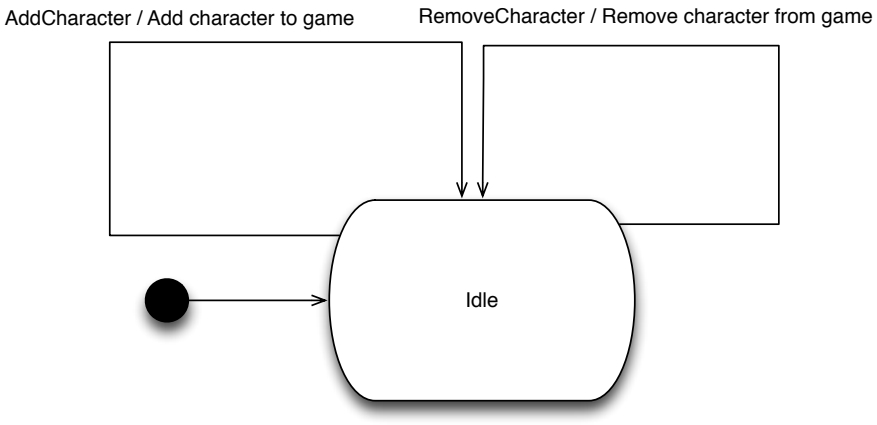

Figure 2.11: State machine for the game class