ASPECTS OF A MULTI-PLAYER GAMING APPLICATION

Represented by

Selçuk Onur Sümer and Alper Karaçelik

Supervised by

Asst. Prof. Bedir Tekinerdoğan

OUTLINE

- Introduction
- Problem
- Requirements Analysis
  - Use-Case Diagram
  - User Interface Prototypes
- Crosscutting Concerns
  - Request Synchronization
  - Distribution
  - Rendering
- Aspect Oriented Program
  - Production Aspects
  - Sequence Diagrams
  - Demo
- Conclusion

INTRODUCTION

INTRODUCTION

- Project: A client-server based multiplayer game application
- Client-server networking: Distributed application architecture that partitions tasks between service providers (servers) and service requesters (clients).
- One server waiting for incoming connections and multiple clients connecting to the server
- Multiplayer game concept: Provides all players (clients) to share the same gaming platform.
- A multiplayer game can be either turn-based or real-time

INTRODUCTION - PROJECT

- Object-Oriented and Apect-Oriented programming concepts are adopted
- Application is developed with Java and AspectJ
- Model-View-Controller, Abstract Factory and Singleton design patterns are used.
- We have implemented aspects for Request Synchronization, Distribution and Rendering concerns which have crosscutting behavior

PROBLEM

OUR CASE-STUDY

- Clients interact with the application by a graphical user interface (GUI)
- Clients connect to the server, and login with their usernames and passwords
- Clients can select a game to play, and then select an existing room or create a new one for joining the selected game
- Clients can chat with other clients in a game room
- Server application waits for incoming connections and incoming requests from clients

SERVER-SIDE PROBLEMS

- Synchronization of multiple client requests
- Distribution of state changes of the model to the clients
- Authorization of client actions
- Storage of user related information
- Secure transmission of the network messages

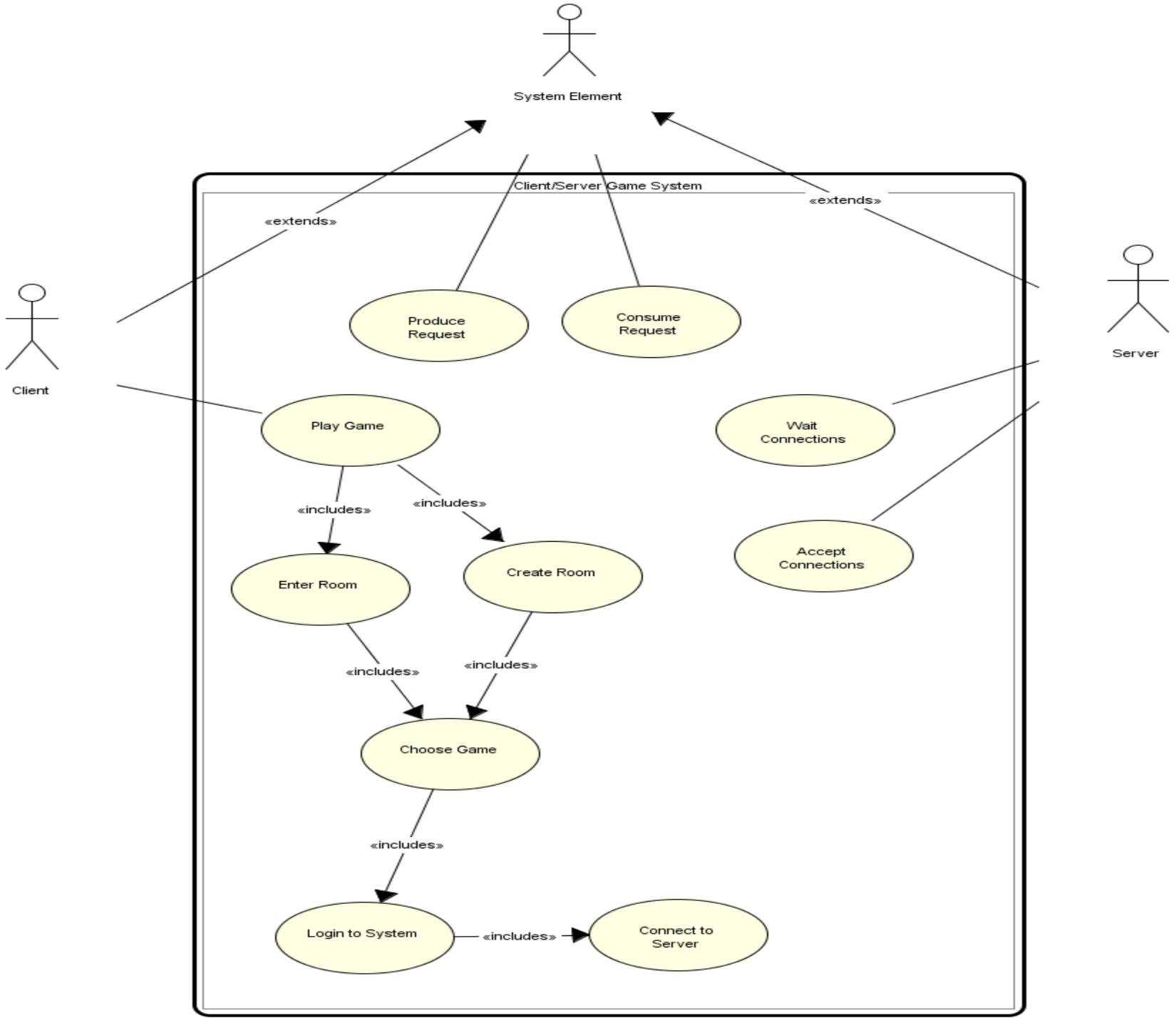
CLIENT-SIDE PROBLEMS

- Synchronization of requests coming from server and the GUI
- Rendering of the state changes in the client model
- Secure transmission of the network messages

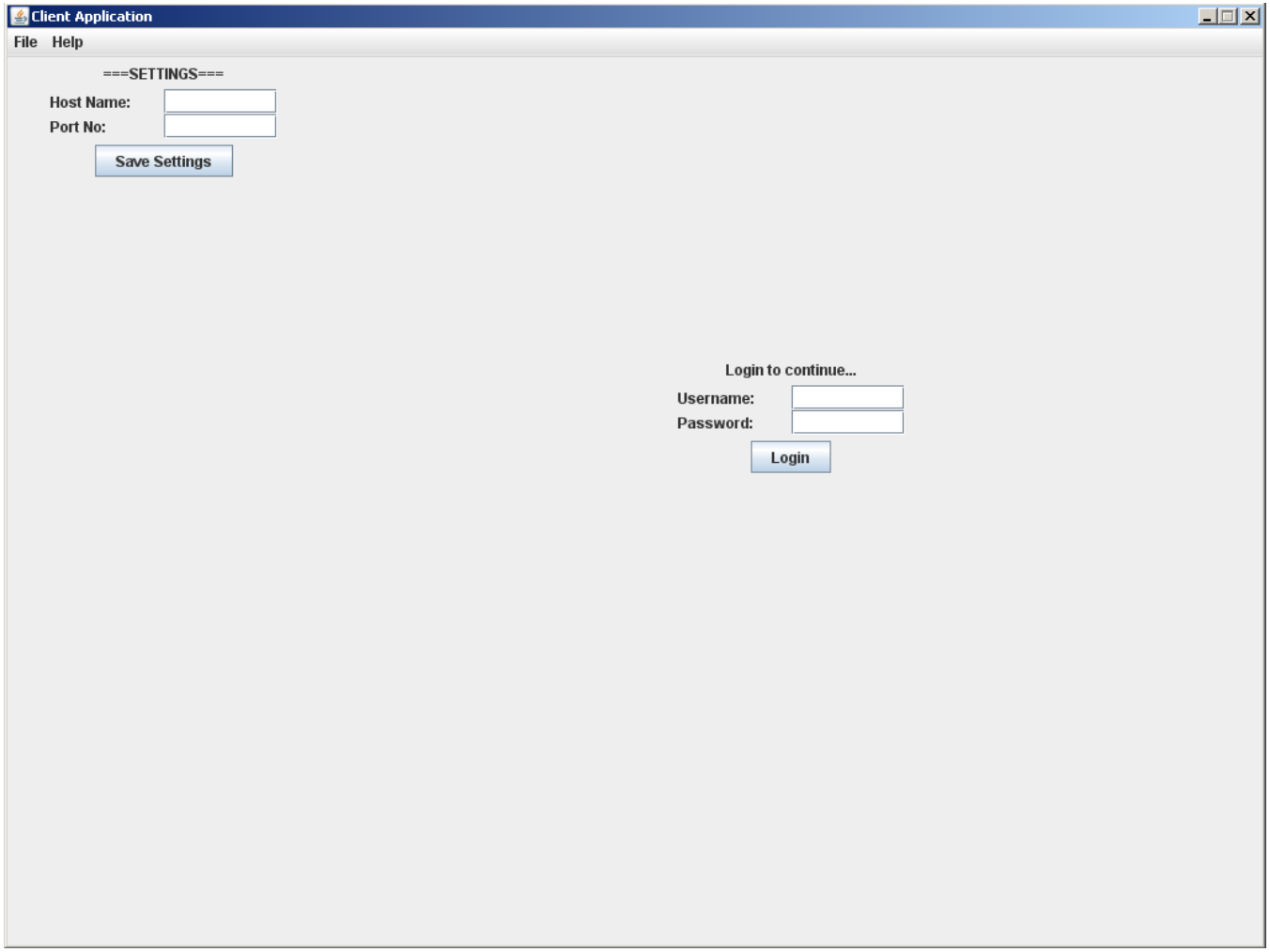
REQUIREMENT ANALYSIS

USE-CASE DIAGRAM

- Our system has two major actors: **client**s and **server**s
- We can think that, both of these actors extend more general actor called **system element**
- All system elements produce and consume requests
- Server waits connection and accepts connection
- Client can connect to server. After connecting, it can login to system.
- Client can choose a game to play. After choosing, it can create a room or join an existing room to start playing.

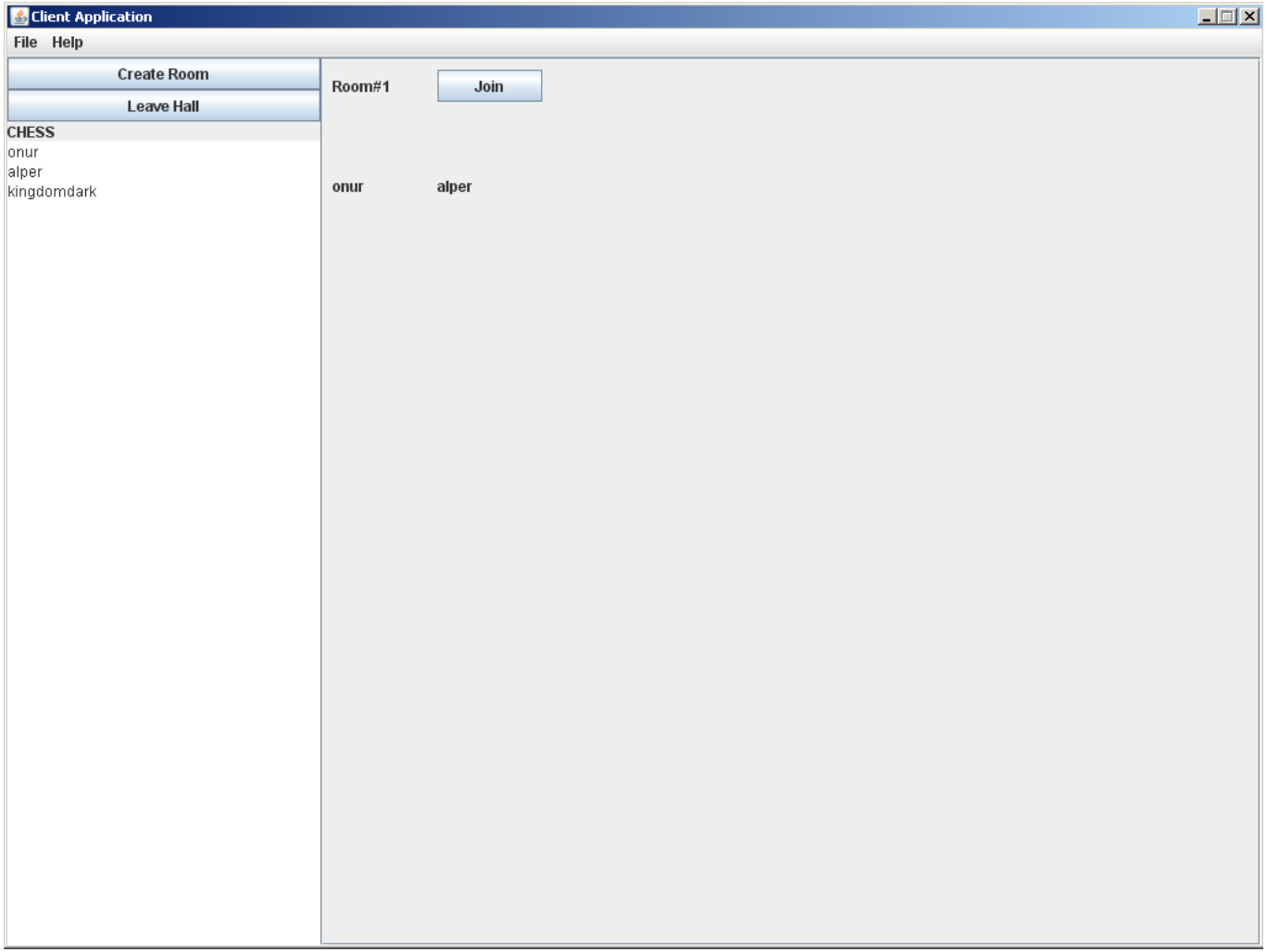

System Element
Client/Server Game System
«extends»
«extends»
Client
Server
Produce Request
Consume Request
Play Game
Wait Connections
«includes»
«includes»
Enter Room
Create Room
Accept Connections
«includes»
«includes»
Choose Game
«includes»
Login to System
«includes»
Connect to Server

USER INTERFACE PROTOTYPES



USER INTERFACE PROTOTYPES

USER INTERFACE PROTOTYPES

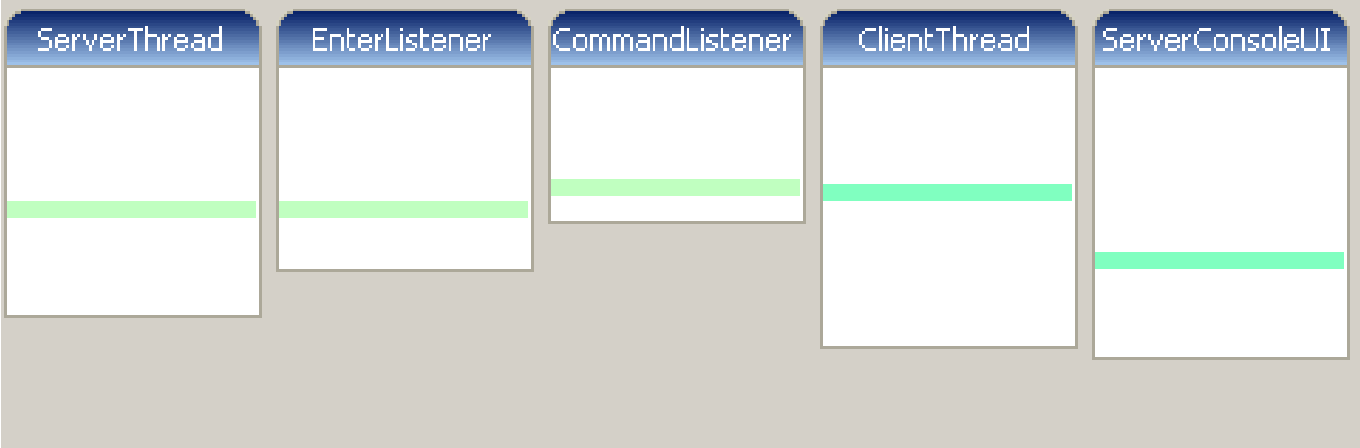


CROSSCUTTING CONCERNS

REQUEST SYNCHRONIZATION

- The server simultaneously accepts incoming requests from the clients
- A client simultaneously receives requests from both the server and the client GUI
- These simultaneous requests should be handled by the in order to develop consistency among the shared resources of the system
- There are more than one source which can produce a request. Therefore, producing and handling of a request is scattered among the request producing methods.

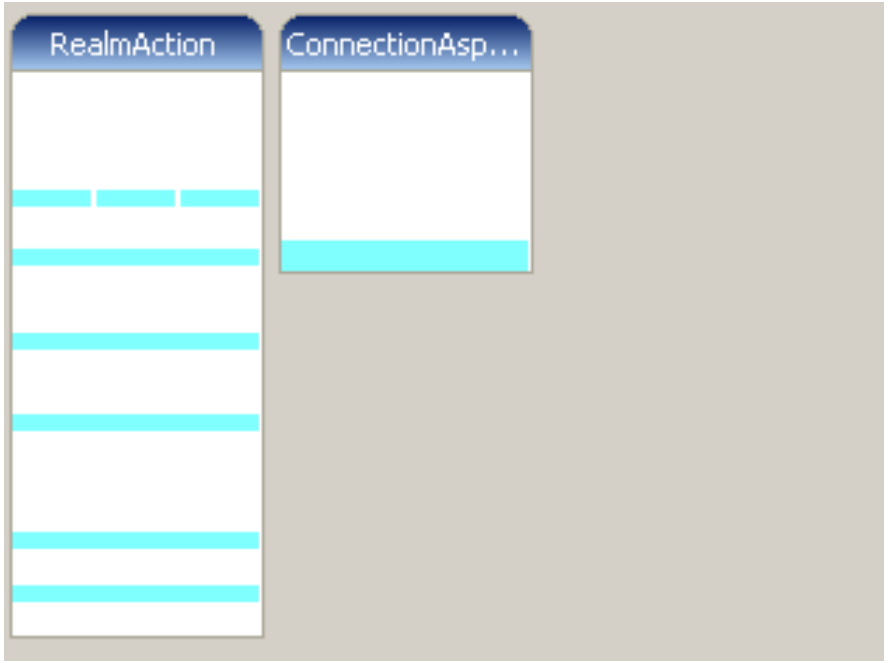
REQUEST SYNCHRONIZATION



DISTRIBUTION

- Clients should be notified by the server after a state change in the server model.
- There are many actions (requests) which can change the state of an object on the server.
- Each action (request) is handled by a specific action method of a specific action class.
- Therefore, distribution concern is scattered among these action classes and methods.

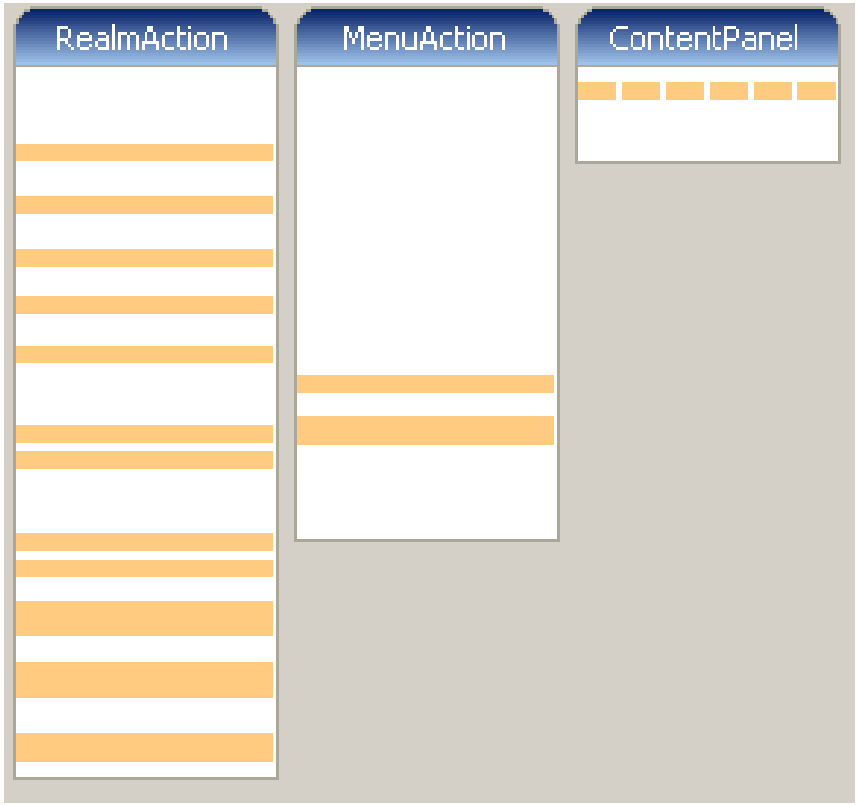
DISTRIBUTION



RENDERING

- A client should see the most up-to-date graphical representation of the state of the client context.
- After each update of the model of the client, it is also required to update the view of this model accordingly.
- There are many actions which can change the state of the client context, and each action is handled by a specific action method.
- Therefore, rendering concern is scattered among these action classes and methods.

RENDERING



ASPECT ORIENTED PROGRAM

PRODUCTION ASPECTS

- Producer-Consumer aspect provides solution to request synchronization concern.
- Requests are accepted in parallel but they are first buffered in a synchronized command list.
- After buffering, requests are processed sequentially in order to prevent inconsistency among shared resources.

PRODUCTION ASPECTS

```
pointcut production() :
   call(@Producer * *(..));

after () returning (Command command) :
   production()
{
   synchronized (commandList)
   {
      commandList.add(command);
      commandList.notifyAll();
   }
}
```

PRODUCTION ASPECTS

- Distribution aspect provides solution to the distribution concern
- Every client is not notified about every state change. Instead, only related clients about a specific change is notified
- Several pointcuts are defined for the actions that are needed to be distributed
- These pointcuts are advised according to the action type

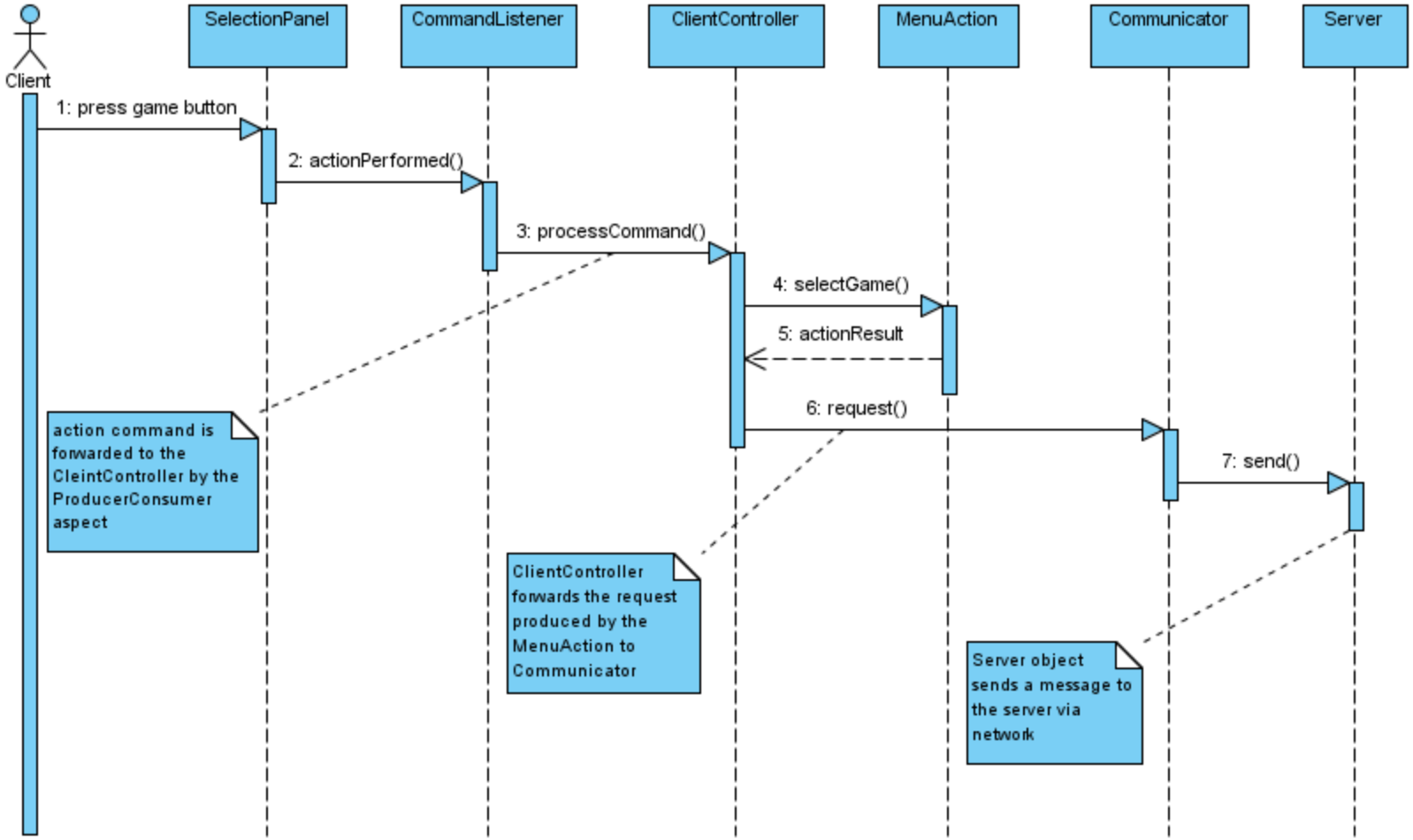
PRODUCTION ASPECTS

```
pointcut enteringHall(GameHall hall, Client client) :
   call(@Distributed * add*(..)) && target(hall) && args(client);

before (GameHall hall, Client client) : enteringHall(hall, client)
{
   // send list of all clients in the hall to the entering client
   controller.getDistributor().sendClientList(client, hall.getClientList());
}

after (GameHall hall, Client client) : enteringHall(hall, client)
{
   // send list of all rooms in the hall to the entered client
   controller.getDistributor().sendRoomList(client, hall.getRoomList());
   // for all clients in the game hall send new entered user
   controller.getDistributor().sendClient(hall.getClientList(), client);
}
```

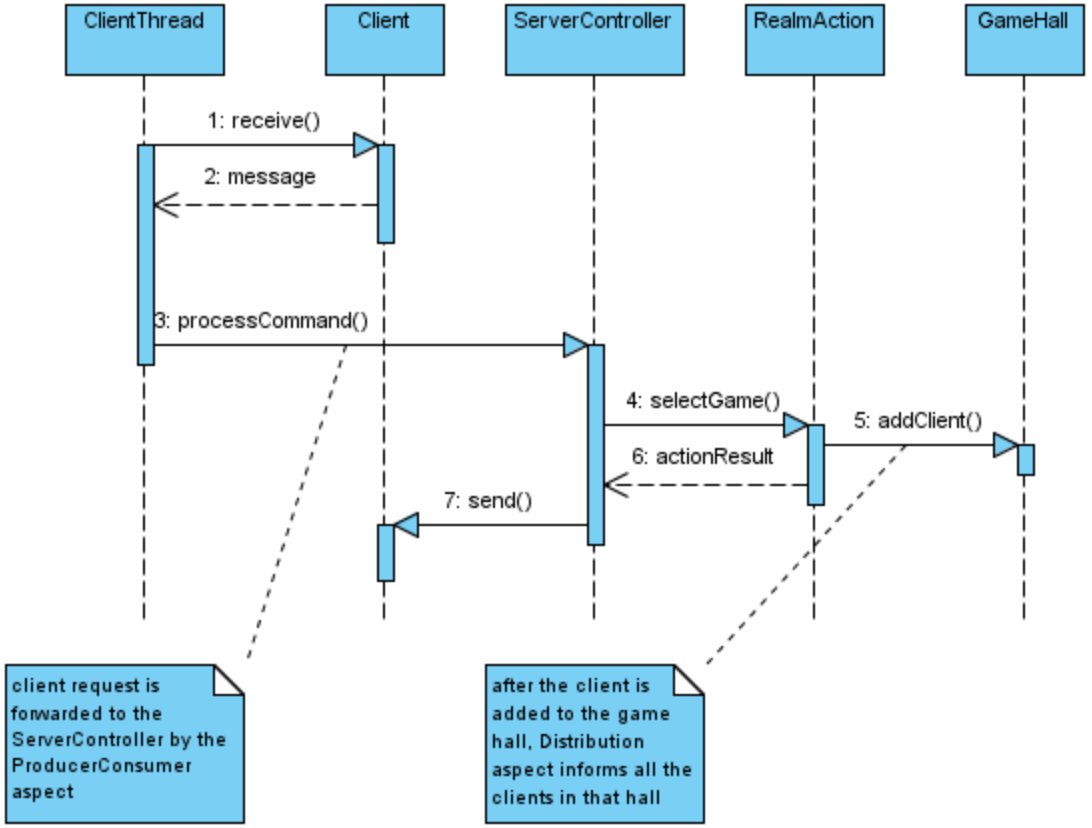
PRODUCTION ASPECTS

- Rendering aspect provides solution to the rendering concern
- When a client is notified about a change by the server, required change is applied to the client model by the corresponding action method.
- Several pointcuts are defined for the actions that changes the state of the client context.
- These pointcuts are advised according to the action type in order to update the related view.

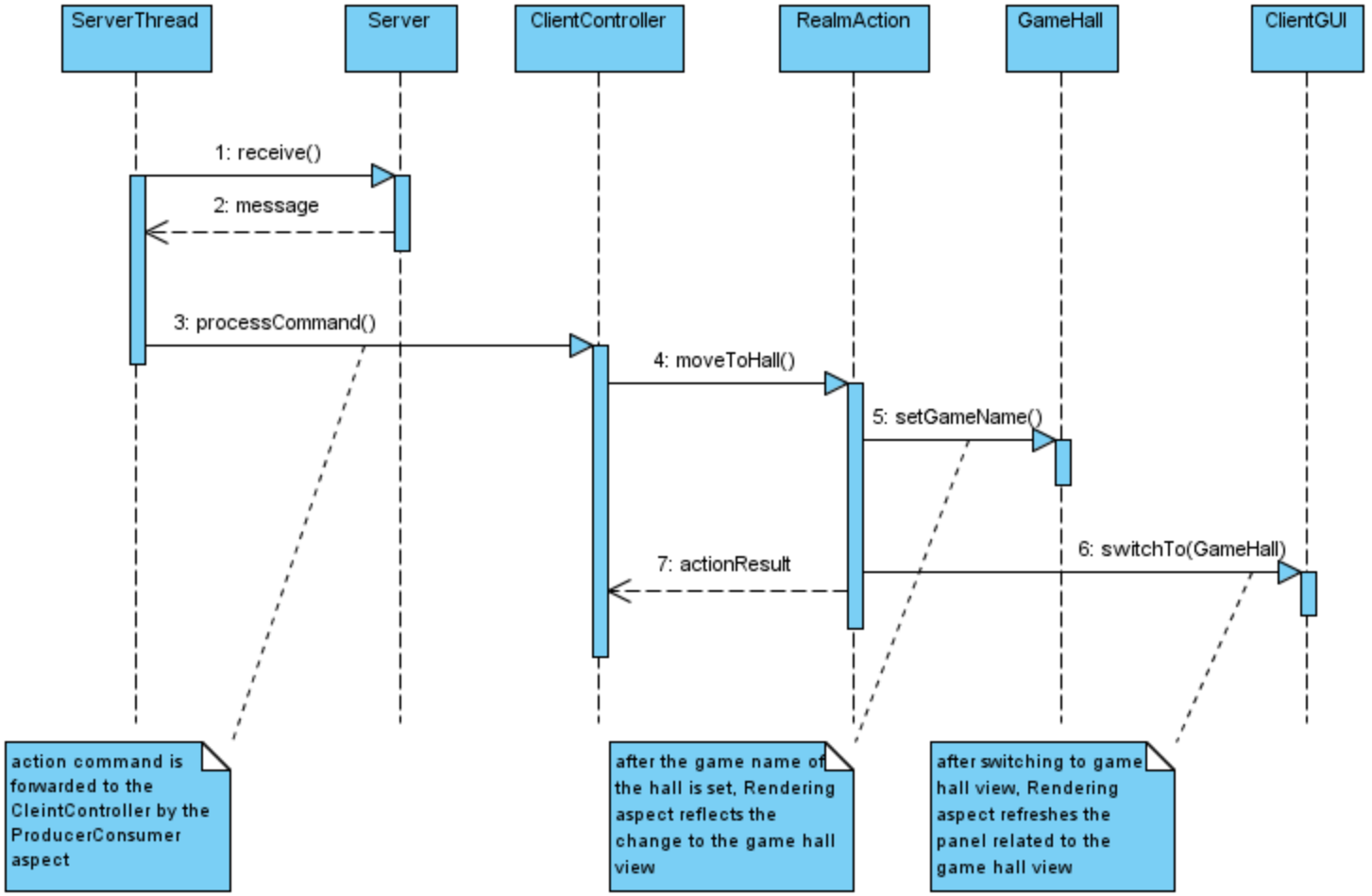
SEQUENCE DIAGRAMS



SEQUENCE DIAGRAMS



SEQUENCE DIAGRAMS



CONCLUSION

CONCLUSION

- Both producing and handling of a request were caused scattering codes among the request producing methods. So, request synchronization concern is seperated.
- Changes in states of objects have to be distributed to relevant units. And states can be changed by any of the action methods which are scattered. So, distribution concern is seperated
- Since the view (user interface) is affected by most of the changes in the model, rendering concern is seperated from the model and scattered codes are cleared.

FUTURE WORK

- There are many concerns left to be identified and implemented by using AOP
- Authorization, security, persistency, and latency are some of these concerns needed to be implemented for a fully functional system.
- Add support for encrypted and variable-sized messages to the communication protocol between the server and clients
- We would like to implement a real-time game and test its playability

QUESTIONS