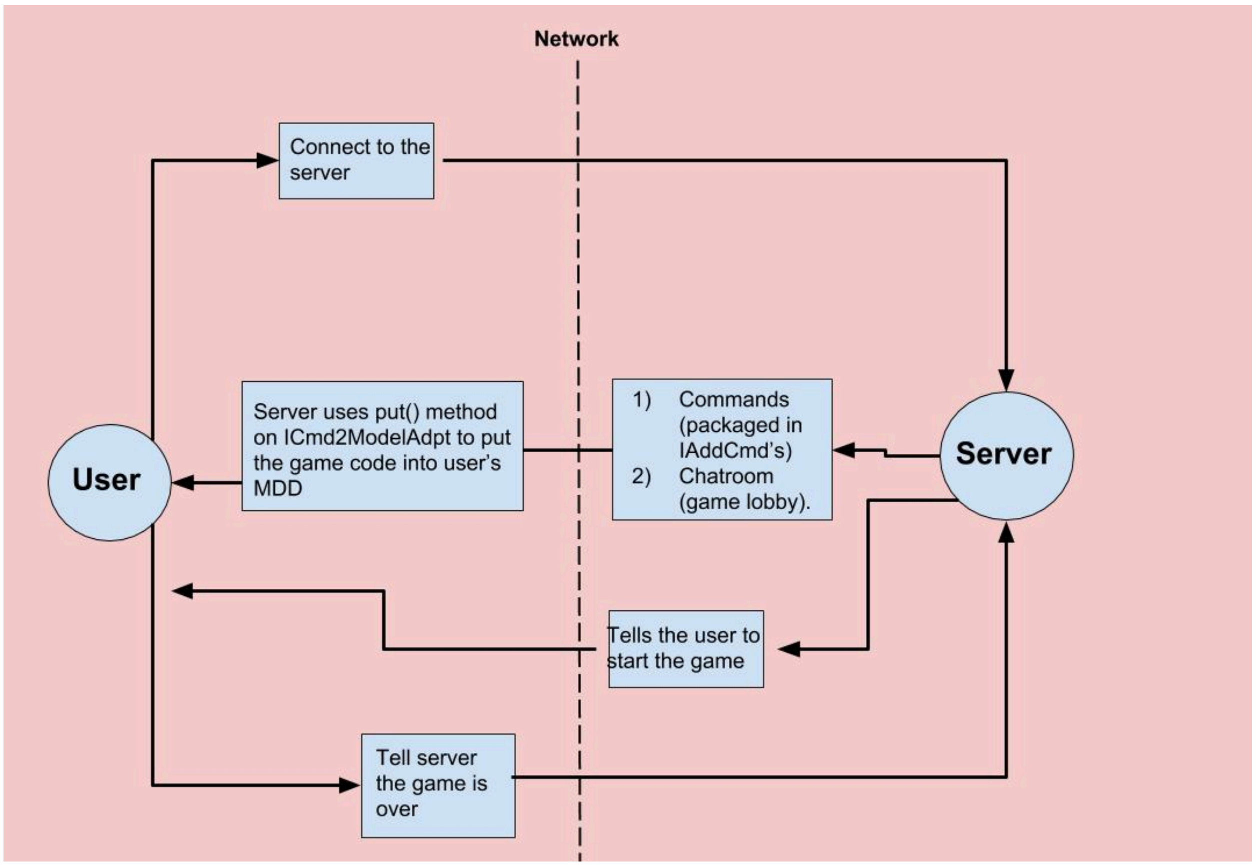
Final Project API Design Documentation – Group C


Network
Connect to the server
Server uses put() method on ICmd2ModelAdpt to put the game code into user's MDD
User
1) Commands (packaged in IAddCmd's)
2) Chatroom (game lobby).
Server
Tells the user to start the game
Tell server the game is over

Note: The server in the use case diagram could be a server, but it could also just be another user, depending on how the game is implemented.

Note: The actual game is sent to other users to be stored in local Mixed Data Dictionaries. Others won't know how to handle this game, so it will be handled as an unknown datatype that the sending user will provide a command for.

Process Descriptions

Initial connection including auto-connect-back

- Same process as chatapp. New User2 connects to the IP of the User1. The new User will send IUser of its own and in return User2 will return a IUser of User2.

Joining/leaving a team

- In each team chatroom, there will be IMsgreceiver of each user along with the Lobby or server's IMsgreceiver. Doing so, IJoin or IQuit message will be sent to everyone in the team chatroom and to the server. If it is required, a server can then process the message and also send the same message to every other team chatroom using the IMsgreceiver it has in other chatrooms. The IJoin and IQuit will happen in a same format as it happened for chatapp.

Team membership synchronization

- This is handled the same way chatroom synchronization was handled in ChatApp. When a player (as an IMsgReceiver) joins a team, that IMsgReceiver sends an IAdd command to all the other players in on the team. The teammates that receive this message can then add the new teammate to their local collection of existing teammates.

Starting a game

- Starting a game is not API dependent. We want the creator of the game to determine how starting a game is handled and thus do not want to limit all the users to starting every game a certain way. The reason this cannot be a well-know message type is because each game will start in a different format, so making it a well-known message type does not make sense. Our API do not want to restrict how the starts.

Ending a game

- This is up to the game implementation : it could be send a certain message to all the users, but maker of the game will determine how this message is handled. The reason this cannot be a well-know message type is because each game will end in a different format, so making it a well-known message type does not make sense. Our API do not want to restrict how the ends.

Winning a game

- Similar to ending a game, the creator of the game will determine what happens when someone wins a game. The game creator will simply install commands in all other user's MDD to handle the message that is sent when someone has won a game.

Intra-team/inter-team message passing (by the local system, by foreign commands)

- Because each game has a "lobby" represented as a chatroom, this allows for game-level message passing (inter-team). Game implementations can then decide how teams are created (probably assigning a chatroom for each team). Within a chatroom, messages (non-well known) can be passed and handled how the game designers want them to be handled.

Unknown message handling (synchronous/asynchronous, caching/non-caching)

- Unknown message is handled in the same way as it was in ChatApp. However, when installing commands for a game in another user's visitor, you can assume that they don't know how to handle your commands and can simply wrap the commands needed for your game in an IAdd datapacket so that the command is installed upon receival.

Adapter for foreign commands (GUI component management, text display, message sending capability, local user information, connect-level access)

- This is handled with the cmd2ModelAdapter. You can use the get() and put() methods in the cmd2ModelAdpater to access another user's MDD and install/get commands through the MDD.

GUI component management (scrolling and/or non-scrolling, labeling added components)

- This is also handled with the cmd2ModelAdapter in the same way that it was handled in ChatApp with the methods of buildComponent and appendString.

Inter-command communications

- Basically the inter-command communication is handled by the MixedDataDictionary. The ICmd2ModelAdapter has a new method "put" and "get" that has access to the MixedDataDictionary. This allows Inter-command communications that was not available in the previous chatapp. For better understanding go to this website : https://www.clear.rice.edu/comp310/s18/lectures/lec40/