TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
Pulchowk Campus

Case Study On
SPACEXR: REALTIME 3D PRESENTATION IN AR/VR

Submitted by:
Anubhav Khanal (076BCT009)
Pramish Poudel (076BCT047)

Submitted to:
Department of Electronics and Computer Engineering
Pulchowk Campus

Date of submission:
28th February 2023

Functional and Non-functional Requirements

Functional requirements:

1. User authentication and authorization: Users must be able to create an account and log in to get access to creating a workspace and collaboration.

    1.1 Login

    - Input: username/email
    - Output: list of created projects and option to create new project is displayed

    1.2 SignUp

    - Input: username, email, password or signup with google
    - Output: login page is displayed to login
    - Processing: email verification

2. Real-time collaboration: Multiple users must be able to access the same 3D presentation in VR simultaneously and make changes to it in real-time.

    2.1 Share Workspace

    - Input: share button is selected on an active workspace
    - Output: a unique code of 4 digits is displayed

    2.2 Join Workspace

    - Input: 4 digit code
    - Output: takes into the workspace with multiple collaborators where their position and controller position is displayed in real time
    - Processing: verifies the code and connects to the right workspace

3. 3D Presentation creation and editing: Users must be able to drag and drop 3D models, place lighting, and other elements within the presentation.

    3.1 Search Models

    - Input: Model name
    - Output: list of related models
    - Processing: models marketplace api is hit and the list of related models are returned

    3.2 Upload Models

    - Input: Model file of gLTF format
    - Output: Upload success dialogue if success else display error
    - Processing: convert gLTF to suitable format for the web

    3.3 Select Model

- Input: point the model and press select
- Output: Displays highlighting bounding box of the model. The selected model by *collaborator1* is highlighted in a different color than the model selected by *collaborator2 and soon.*
- Processing: Get ray intersection of the controller and the scene and find the bounding box of the intersected model. The selection is broadcasted to all collaborators.

3.4 Transform Model

- Input: controller position and rotation
- Output: transformed model
- Processing: Apply translation and rotation to the model. The new position and rotation is broadcasted to all collaborators.

3.5 Scale Model

- Input: Relative distance between two controllers when both button is squeezed
- Output: Rescaled model
- Processing: Apply scaling directly proportional to the change in relative distance of two controllers. The new scale of the model is broadcasted to all collaborators.

3.6 Add Slide

- Input: Add slide is pressed
- Output: A new slide is created and the slide number is increased. The newly created slide becomes active.

4. Animation: Users can select different animation types like custom path follower, rotation on its axis for models. They can also select effects for slides which are triggered on slide change.

4.1 Select Animation Type

- Input: Select animation type from custom path follower, rotation on its axis, etc
- Output: Properties definition options according to the animation type

5. Presentation saving and sharing: Users must be able to save their presentations and share them with other users for further collaboration.

6. Compatibility with VR head-mounted displays: The application must be compatible with popular VR head-mounted displays, such as the Oculus and HTC Vive, Meta Quest Pro etc.

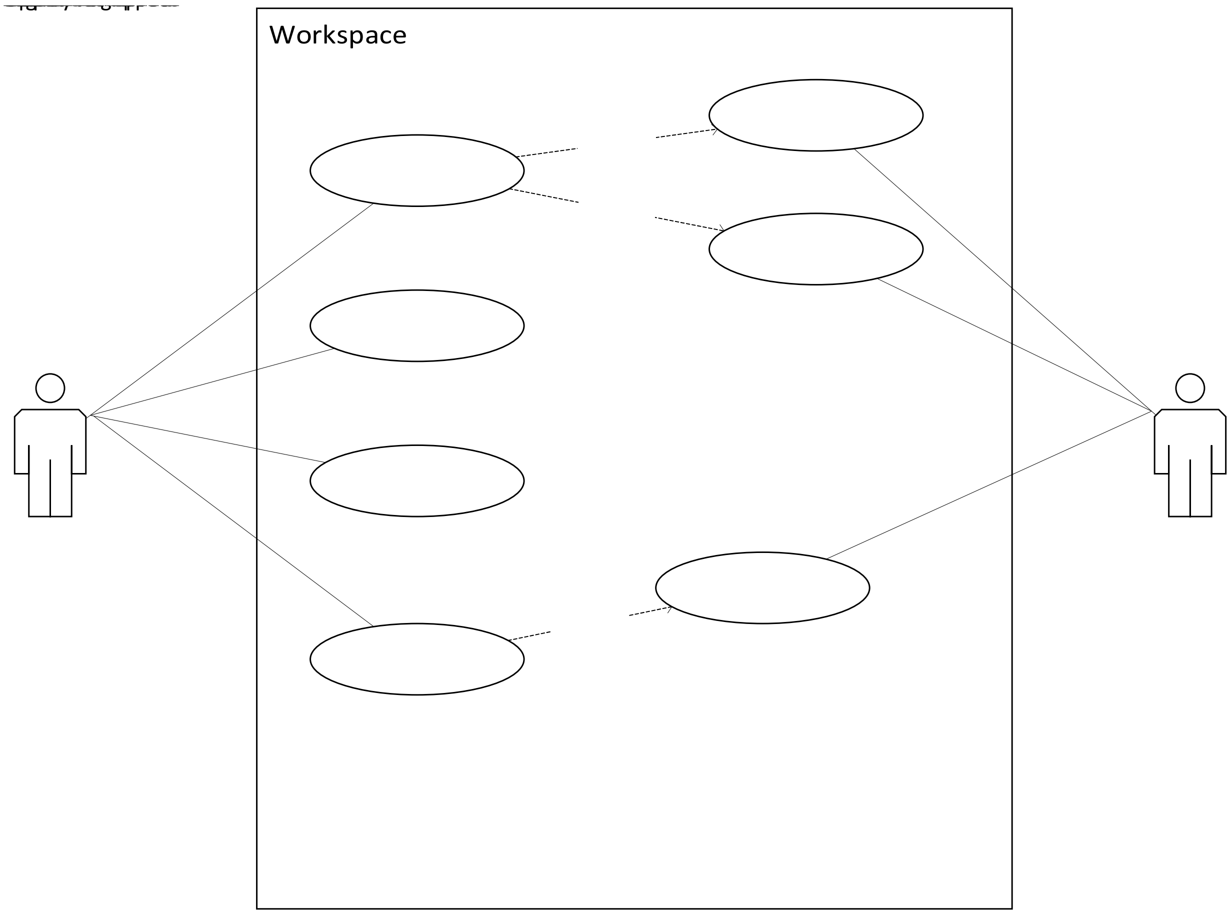
7. Platform support: The application should be supported on AR/VR headsets for presentation creation and web platforms for playback.
8. Compatibility with 3D file formats: The application should be able to import and export gLTF format.
9. Presentation playback: Users should be able to play back the presentation in AR/VR, web and navigate through different scenes or slides.
10. User interface: The application should have a user-friendly and intuitive interface for ease of use.

Non-functional requirements:

1. Performance: The application should be optimized for performance, with minimal latency and high frame rates, specially during presentation playback and collaboration.
2. Security: The application should have secure user authentication and data encryption to protect user data and prevent unauthorized access.
3. Scalability: The application architecture should be able to handle a large number of concurrent users.
4. User experience: The application should provide an immersive and interactive AR/VR experience for the users.
5. Usability: The application should be easy to learn and use, with minimal training required for users.

UML Use Case Diagram and Descriptions

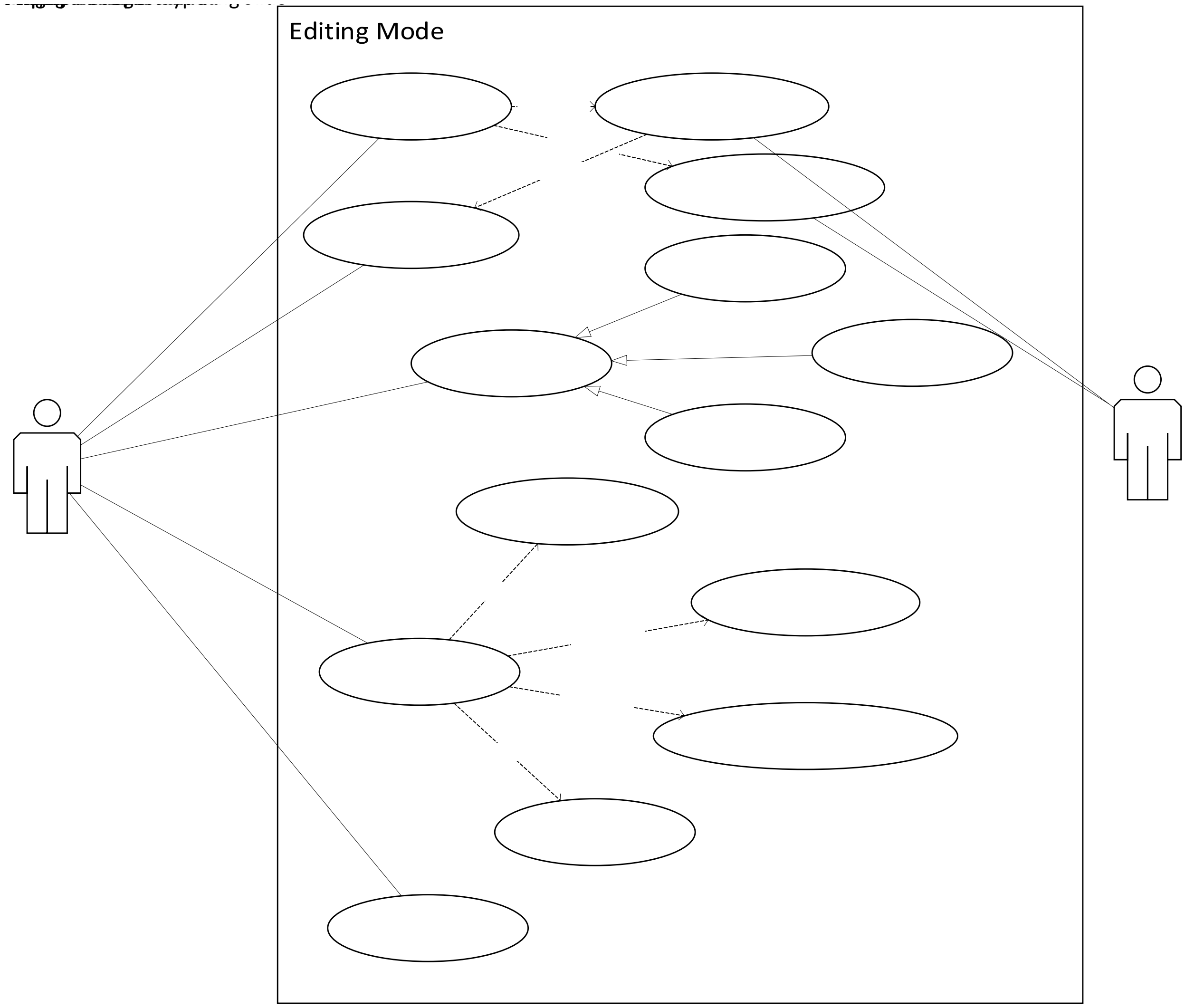
Enter Workspace



Actors	User, Server
Descriptions	A user can access their workspaces by logging in. Then credentials provided by the user are checked by the server. If the credentials provided by the user is accurate , the user has successfully logged in. In case the credentials provided by the user is inaccurate an error message will be displayed. If the user isn’t

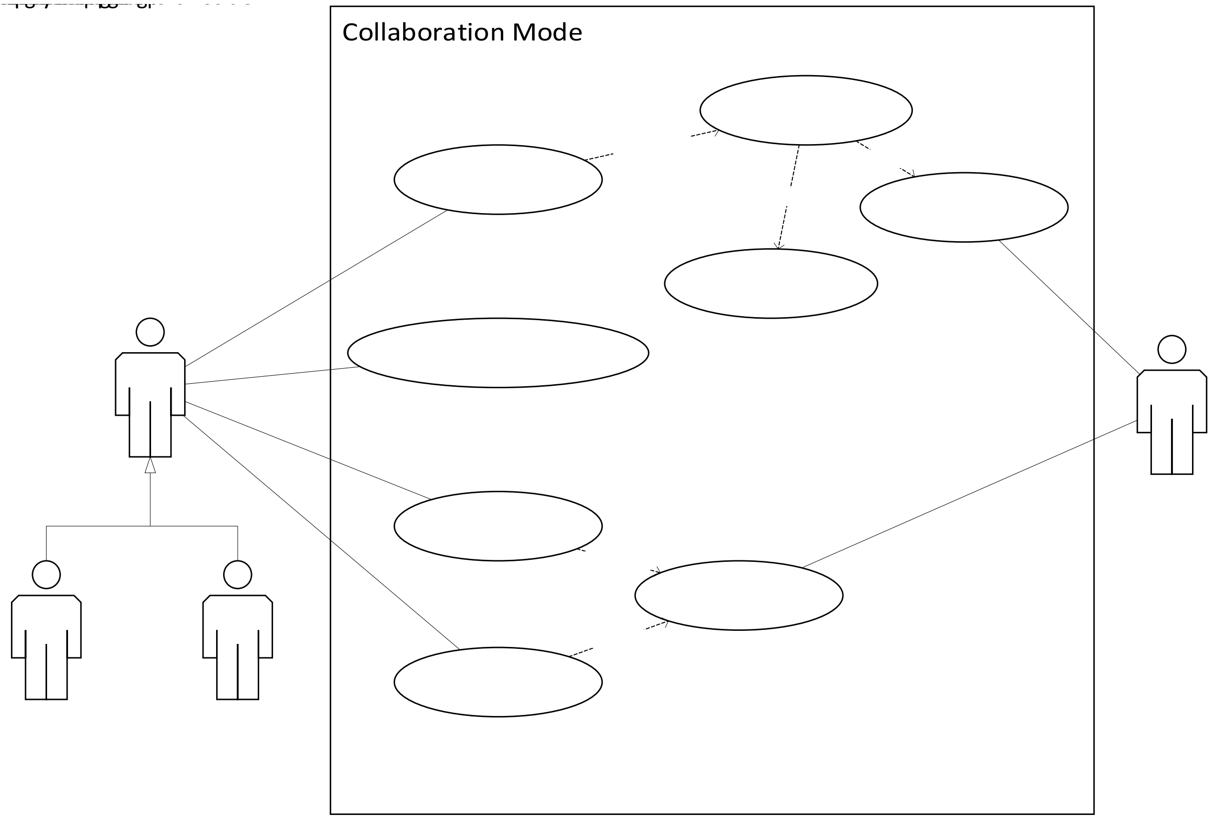
	registered, they can sign Up. Once the user has logged in, they can see the list of their workspace. The user can enter a workspace or create a new workspace. The user can also choose to share their workspace to other collaborators. A 4-digit pin will be created using which the collaborators can enter a workspace.
Pre-condition	Users need to be registered. If not, the user can register.

Editing Mode



Actors	User, Server
Descriptions	Once inside the workspace the user can search for models. If the models searched by the user are found then the user can drag and drop the model anywhere inside the workspace. The server displays a list of models related to the search result. If no models are found matching the user request then the server displays no result found .The user can translate, rotate and scale their models. These changes in properties are broadcasted in realtime to others. Once the user has completed designing their workspace, they can animate it as well. For animation the user can select the type of animation, time for animation as well.
Pre-condition	Users need to be logged in and have either selected a workspace or created a new one.

Collaboration Mode

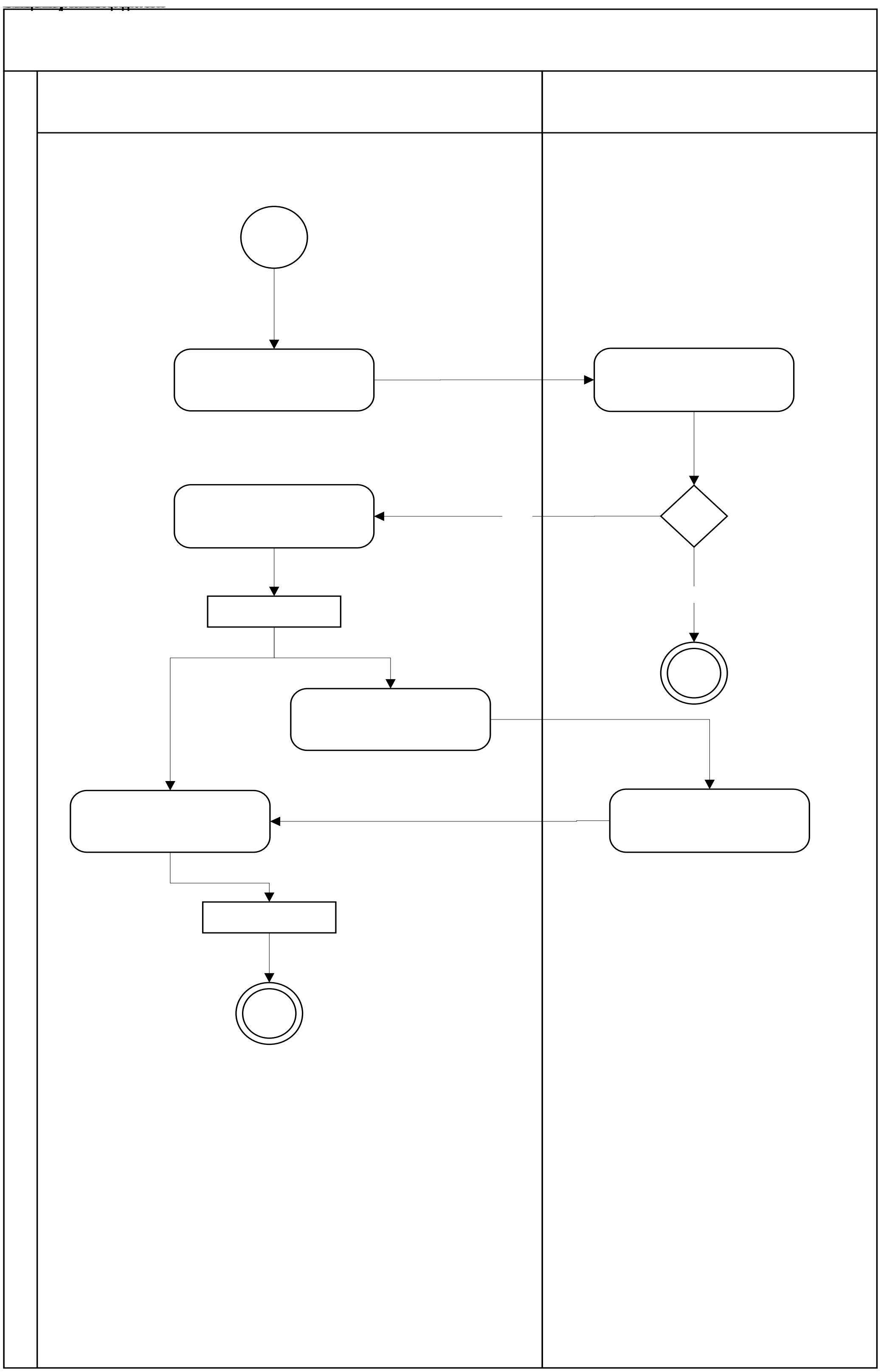


Actor	User, Server
Description	Inside the workspace the owner of the workspace and the collaborators can see each other and the contents of the workspace. They can make changes in real time. The server will broadcast the changes done by several users and the client merges their local copy with CRDT and displays the changes in the workspace. The collaborators or the user can also edit the workspace offline for a few minutes. The changes created by the user will be merged to the workspace when the user is back online.
Pre-condition	Collaborators need to have entered a workspace by using the pin for the workspace provided by the owner of that

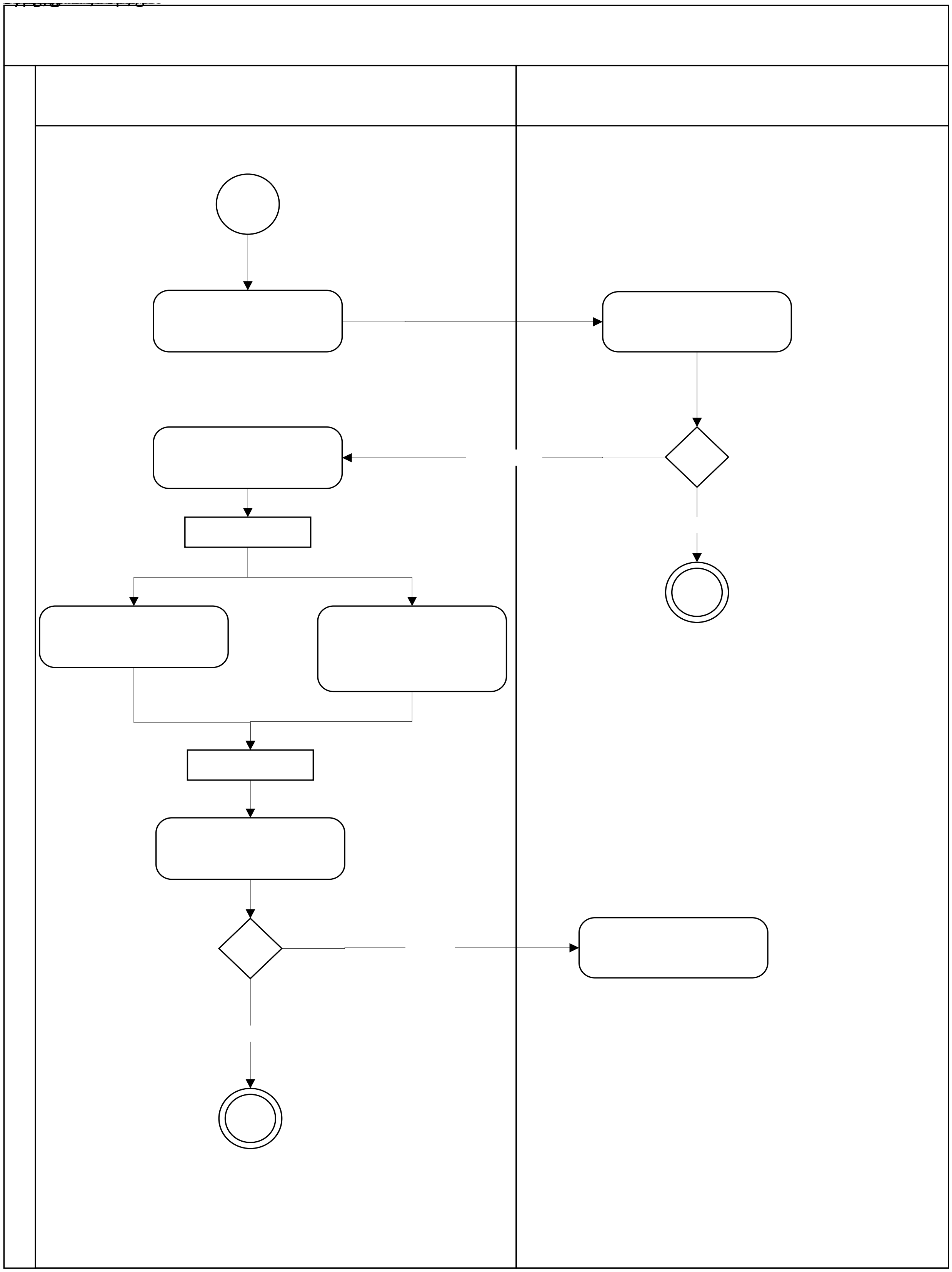
	workspace.

UML Activity Diagrams

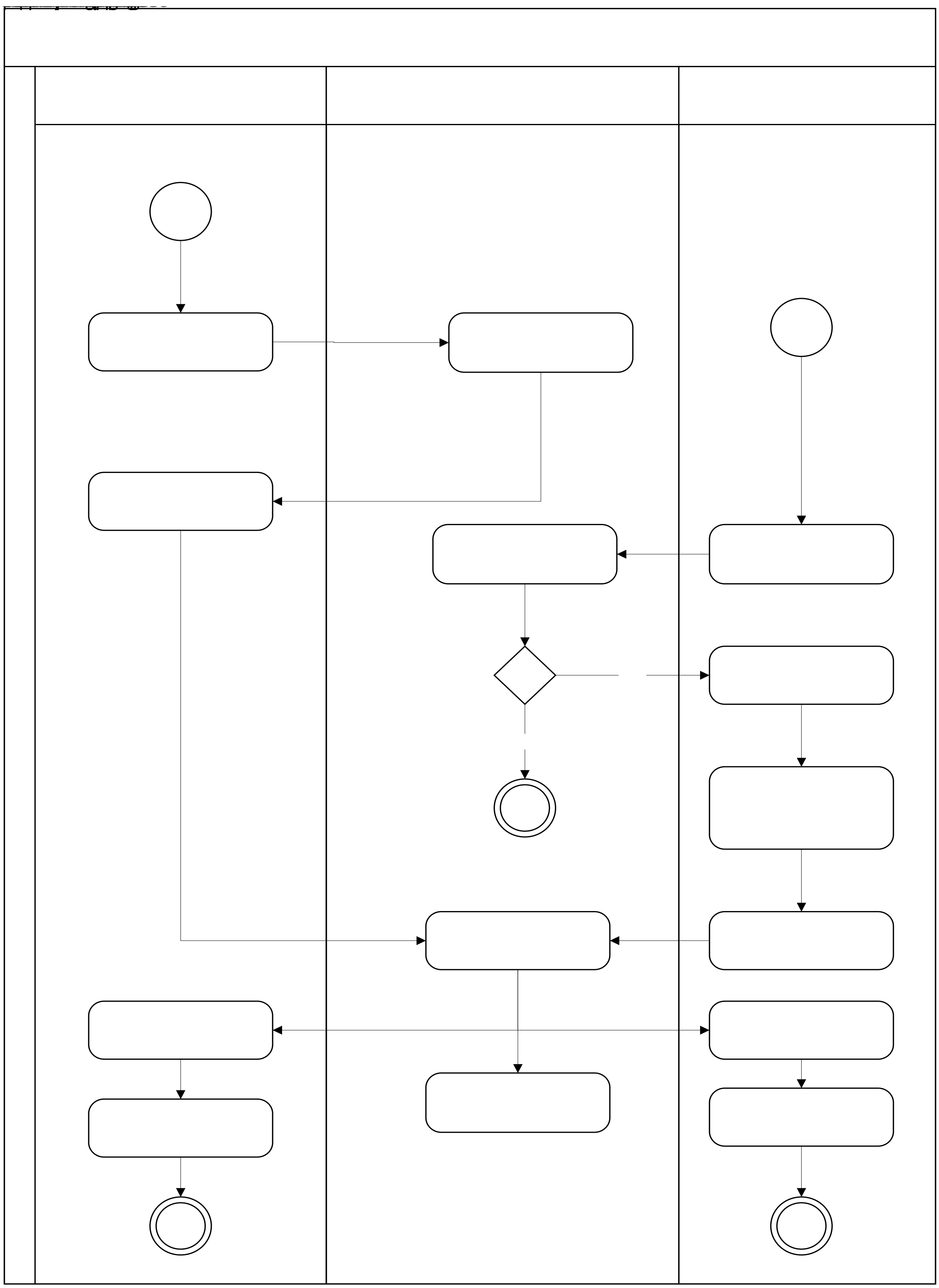
Workspcae



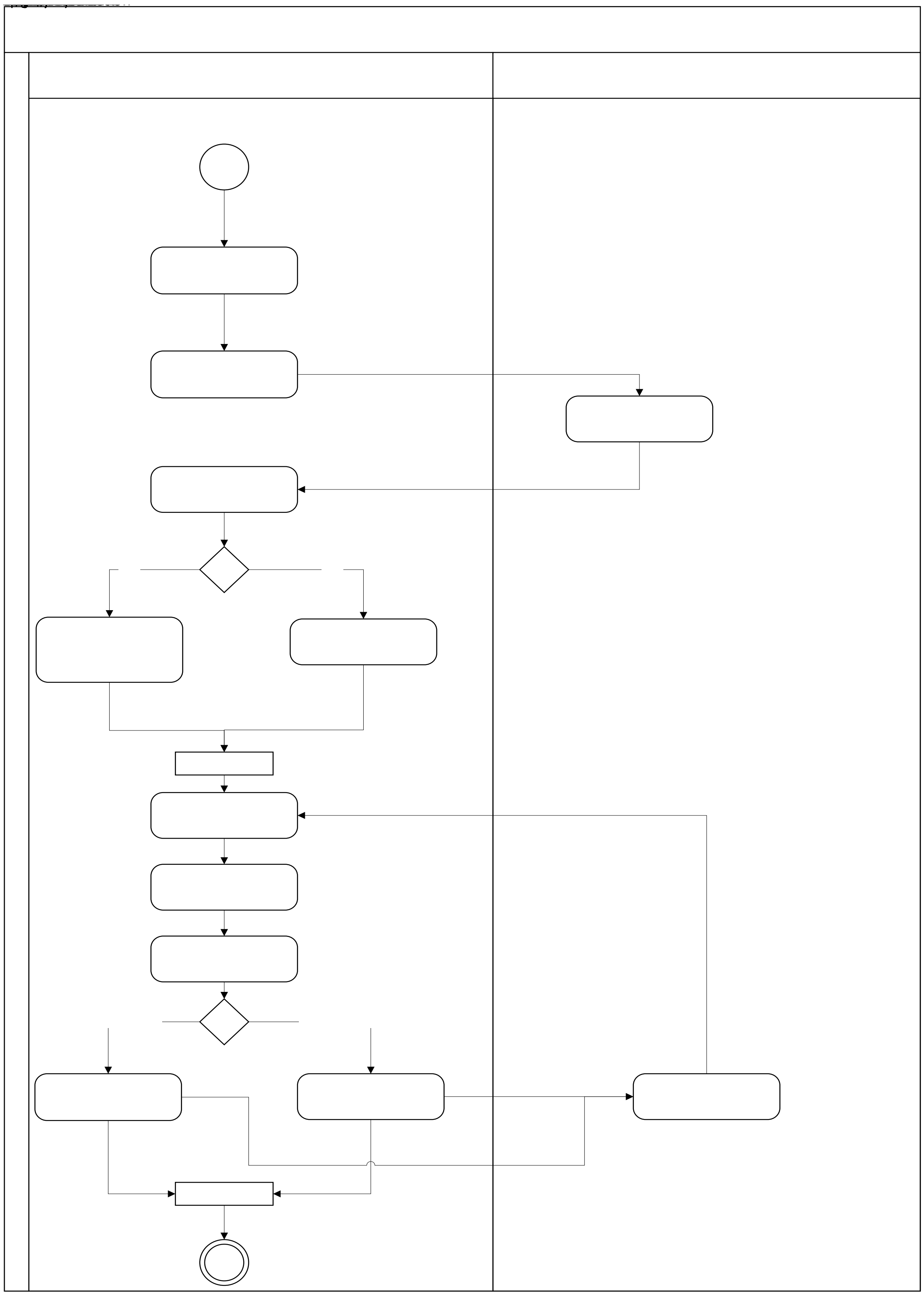
Editing Mode



Collaboration Mode

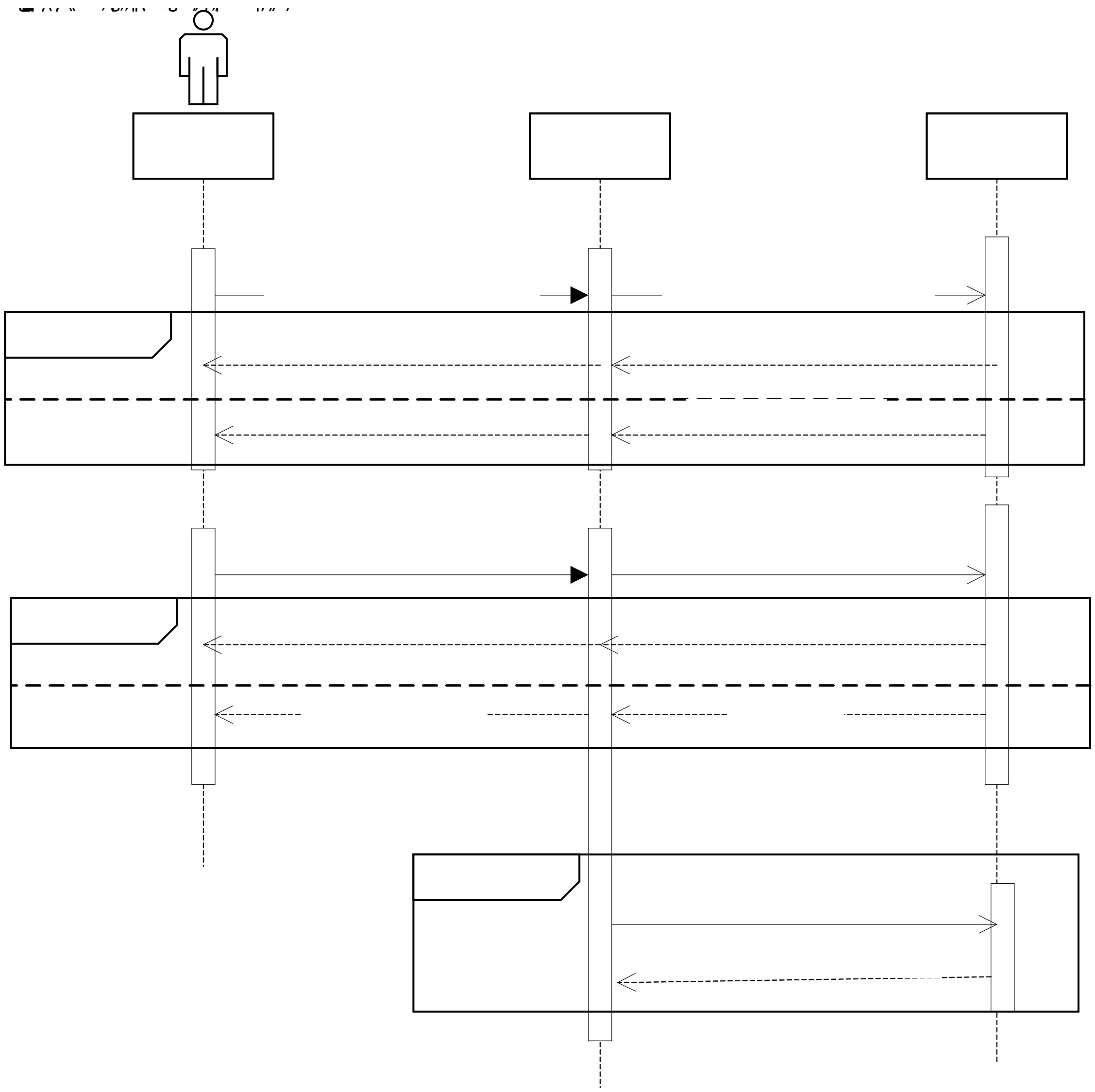


Preview

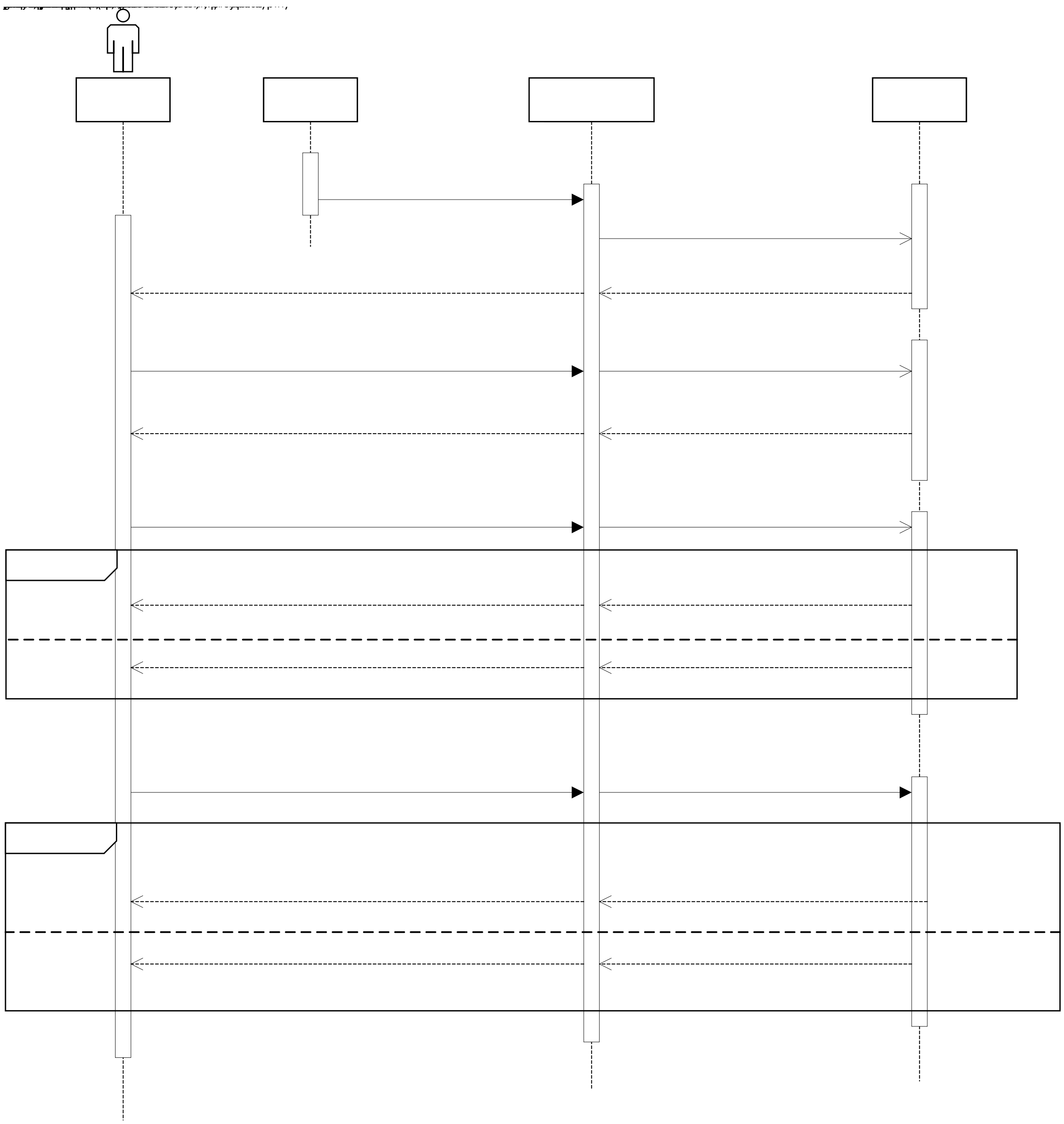


UML Sequence Diagram

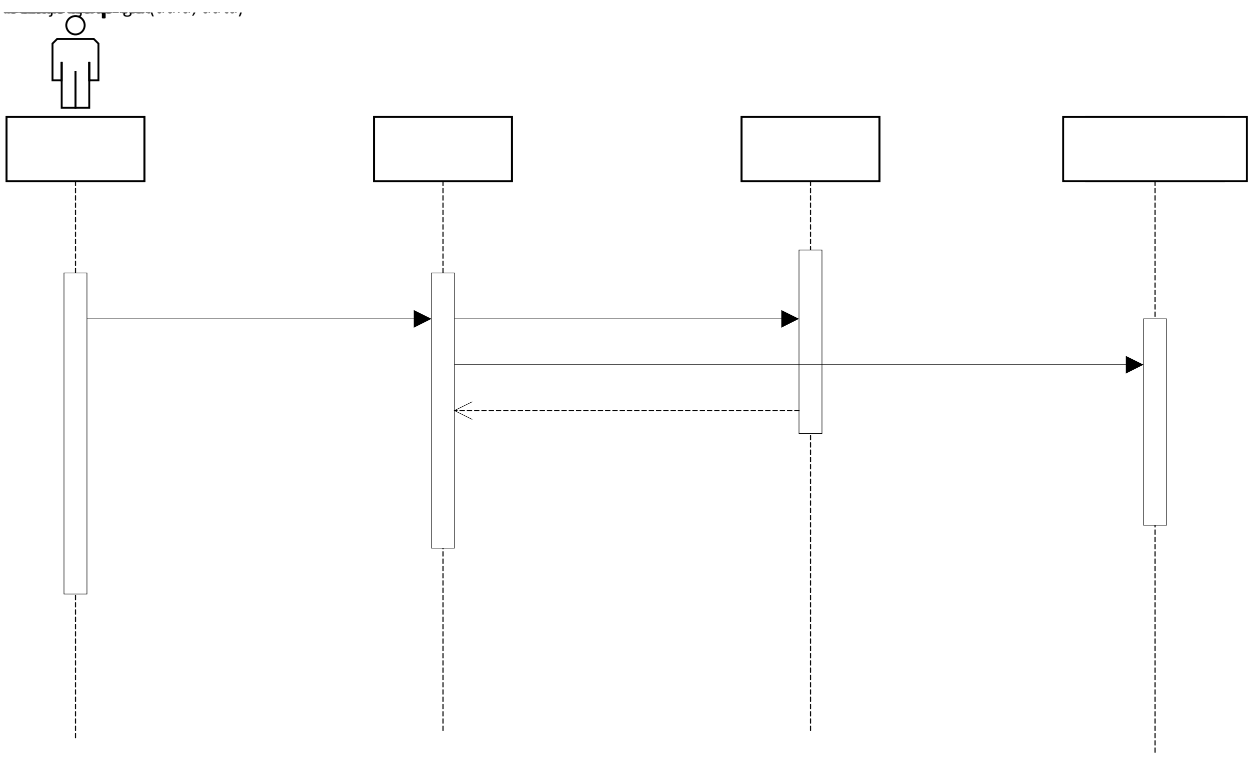
User Authentication Sequence Diagram



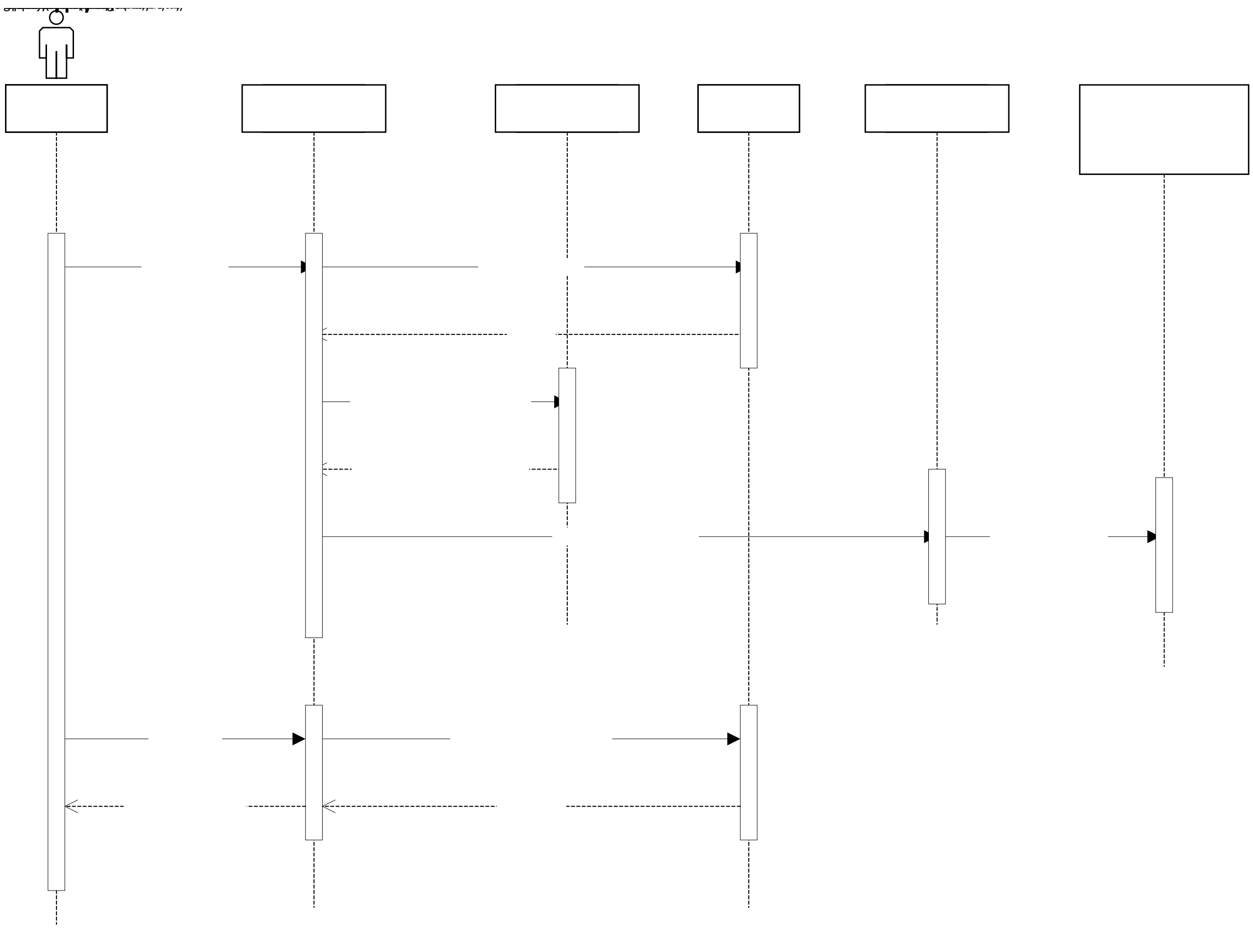
Dashboard Sequence Diagram



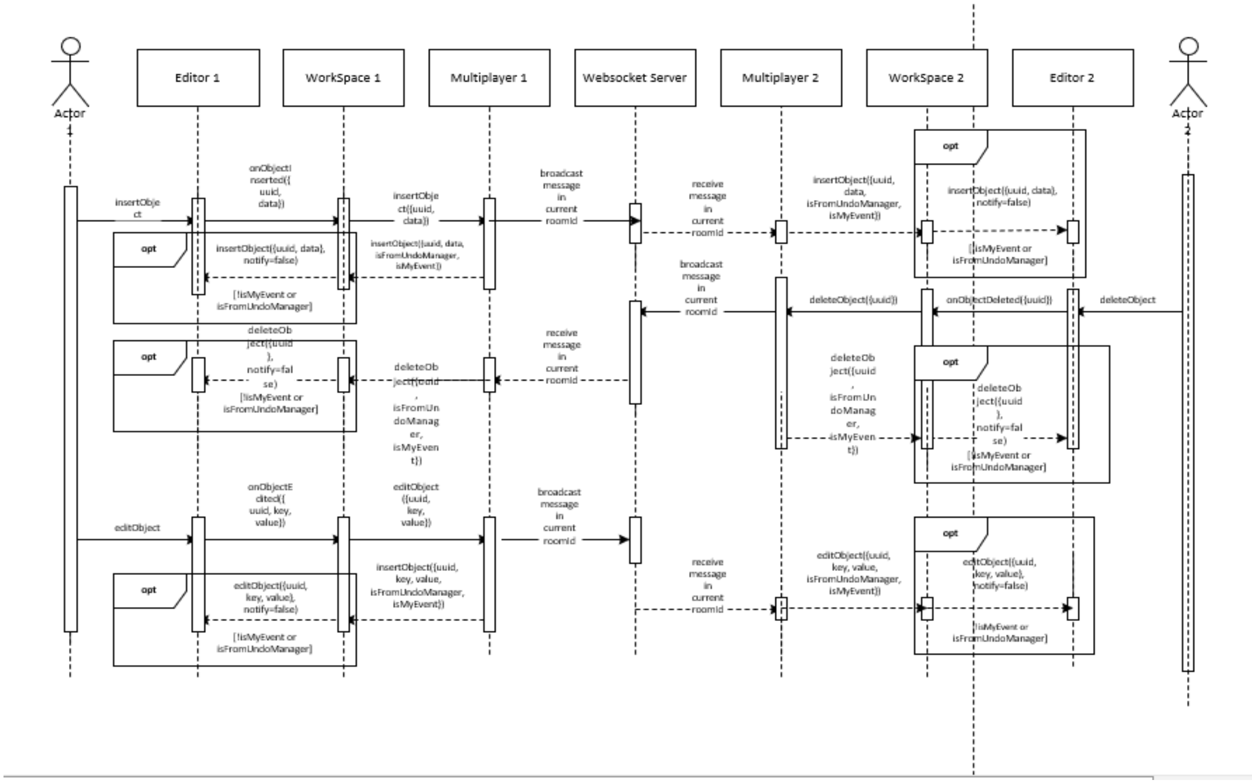
Editor Loader Sequence Diagram



Upload and Search Model Sequence

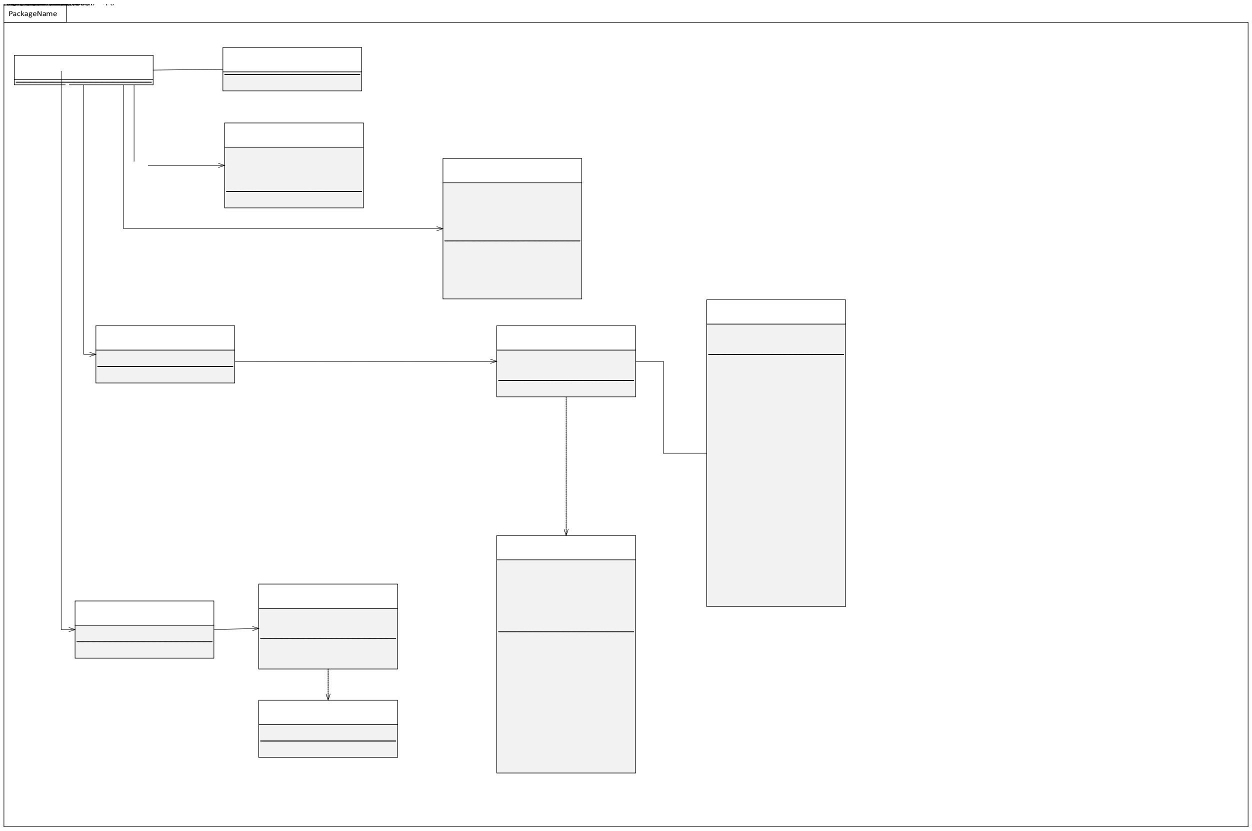


Mulitplayer Sequence Diagram

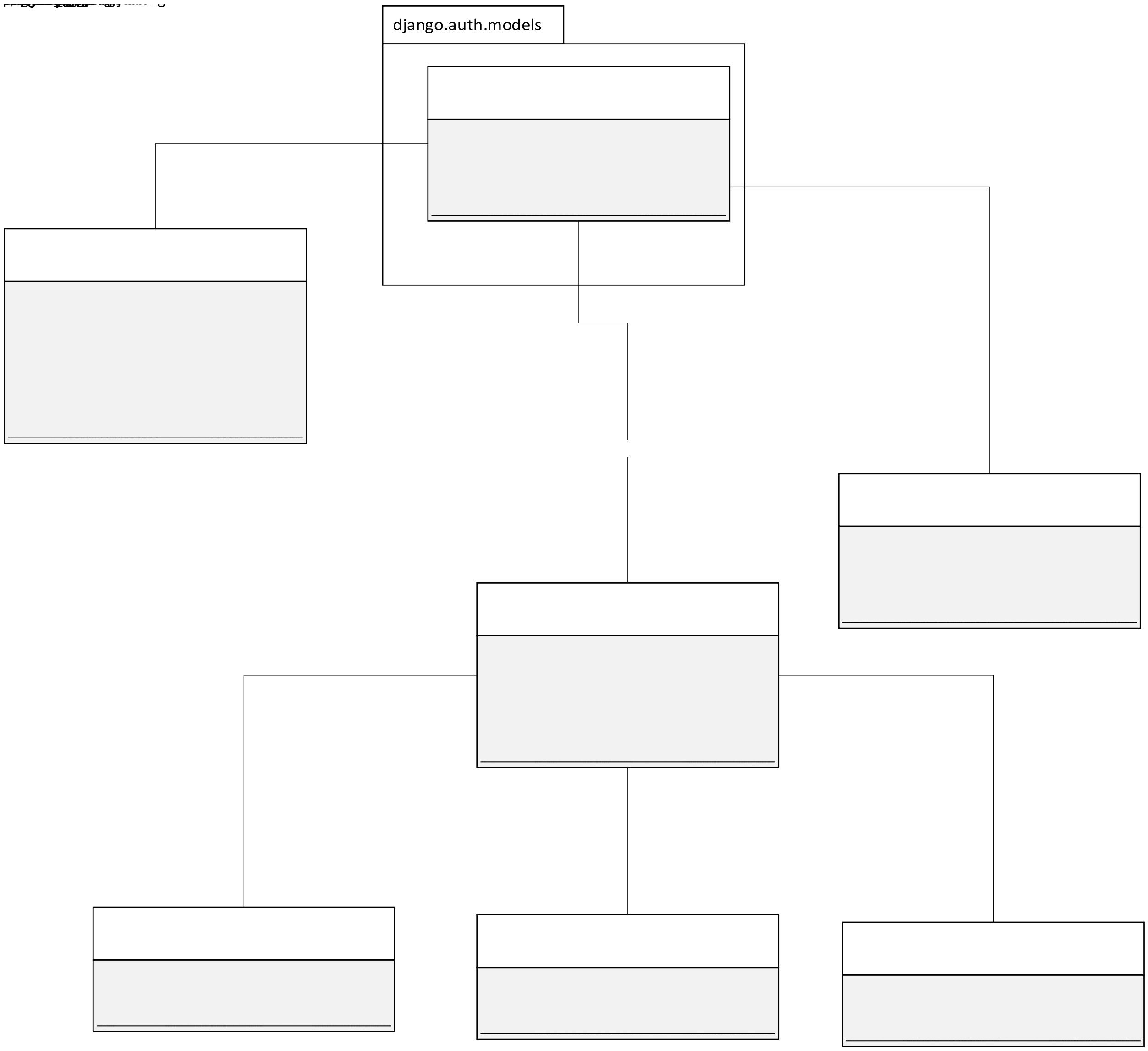

Actor 1
Editor 1
WorkSpace 1
Multiplayer 1
Websocket Server
Multiplayer 2
WorkSpace 2
Editor 2
Actor 2
insertObject
onObjectInserted({uuid, data})
insertObject({uuid, data})
broadcast message in current roomId
receive message in current roomId
insertObject({uuid, data, isFromUndoManager, isMyEvent})
opt
insertObject({uuid, data}, notify=false)
[!isMyEvent or isFromUndoManager]
opt
insertObject({uuid, data}, notify=false)
insertObject({uuid, data, isFromUndoManager, isMyEvent})
[!isMyEvent or isFromUndoManager]
deleteObject
onObjectDeleted({uuid})
deleteObject({uuid})
broadcast message in current roomId
opt
deleteObject({uuid , isFromUndoManager, isMyEvent})
deleteObject({uuid}), notify=false)
[!isMyEvent or isFromUndoManager]
receive message in current roomId
deleteObject({uuid , isFromUndoManager, isMyEvent})
opt
deleteObject({uuid}), notify=false)
[!isMyEvent or isFromUndoManager]
editObject
onObjectEdited({uuid, key, value})
editObject({uuid, key, value})
broadcast message in current roomId
receive message in current roomId
editObject({uuid, key, value, isFromUndoManager, isMyEvent})
opt
editObject({uuid, key, value}, notify=false)
!isMyEvent or isFromUndoManager]
insertObject({uuid, key, value, isFromUndoManager, isMyEvent})
opt
editObject({uuid, key, value}, notify=false)
[!isMyEvent or isFromUndoManager]

UML Class Diagram

Frontend Class Diagram



Backend Class Diagram



Conclusion

The SPACEXR case study has been completed effectively, identifying its functional and non-functional requirements, use case diagrams and descriptions, UML Activity Diagram, UML Sequence Diagram, and UML Class Diagram. These diagrams and descriptions offer a comprehensive understanding of the system, from an overall view to detailed information about its workings.

In essence, the diagrams and descriptions created for the system provide a clear understanding of the system's operations, from the perspective of both users and developers. They provide a visual representation of the system's behavior and structure, making it easier to comprehend and communicate. These insights will assist in the development and maintenance of the Bus Registration System, allowing it to function efficiently and effectively.