Team organisation and project management (Individual)

Contribution:

As this project has started, I have attended every meeting and organised when each next one will be completed. During the first meeting it was decided that I would be volunteered by the group to be Team Leader. This included initially taking minutes each week and documenting what was discussed, (however this was later a shared responsibility), setting deadlines, plans for next week and taking attendance.
During each meeting, we decided that the chairperson would rotate on those members who were attending. This allowed each member to lead the conversation at least once.
During the first meeting, Ivan (19324937) had already set up the GitHub group repository the second he had sat down, before the group had been formed. As a group we had decided to keep that repository as he has the most experience with using Github.
As team leader it was my responsibility to organise the GitHub repository as well as managing, setting up, and writing the team report for the final submission.
Additionally, we all took turns to take minutes depending on who was chairing the meeting at that time.
The unavailability of the group flexibility was down to students living at home and having a difficult commute in, full time work getting in the way of personal time management and students disappearing for the entire project. Despite a few attempts at requesting a digital call to fix this issue it was never acted on, especially as some group members were still unable to attend due to other work commitments.
Finally, hard deadlines have had to be set to make sure that the final submission would be ready to submit the day before the deadline. This also gives all members more time to realise if there are any issues with their work.

Technical Contributions:

Over the project, tasks have been completed mostly on time. Most meetings would go into more theoretical conversations on how the system would be implemented and how each system would likely run. One student's work would mostly be missing from any contributions due to lack of attendance throughout the project.
Tasks sometimes were delayed due to other university courseworks taking priority meaning that the architectural review has been left to the week before submission due to waiting for the last few component diagrams to be completed. The missing group member had reappeared 24 hours before the final meeting with all of the work completed up to the component diagram and had joined for the final team review.
Our group has lacked in uploading the work to GitHub across the weeks partly due to delays but also due to the software required to use would have too many crashes and unrecoverable loss of data.
The work has been uploaded to my personal Github as the project went on, a few reuploads of work after Papyrus has corrupted the original files and recreations were necessary.
Despite the disruption to the work, it was still shown in most meetings; such as how I have done the diagram technically (explaining how nodes were done from the word documents on

moodle which also show how to create the diagrams). How I have envisioned my subsystem working and explaining why I have connected certain components together.
The biggest hurdle our entire group had faced was Papyrus and its technical issues. We had discovered that if a personal device was using the software, it would crash, and even corrupt file data and cause the user to restart entirely (I personally had reinstalled this software 4 times, even with lecturers watching to make sure it was done correctly). After discussing not only in my group but further groups it was discovered that this was a recurring issue across the board. We did discover that using a university device would prevent a lot of the issues. Overall the group itself has managed to complete all the work required, they have been good at communicating their work.

2 - Quality Requirements:

Brief description of requirements:

- **Approval of new society.** Once a request to create a society comes through. UO must look to see if it has the correct fields (name, theme, organisation team and contact address.). Post approval, the society should appear on the system and the unions website.
- **Approval of requests of changes in societies registration data.** Society leaders can request updates to the registration data (leaders, descriptions of themes etc) and once approved. Update and reflect the changes on the website.
- **Approval of request for termination of society.** Society leaders should be able to put in requests to terminate the society. Once it's terminated. Remove it from the website but keep some of the system data and the record of its termination.
- **Monitoring the operations of societies.** The system should let the union monitor the operations of the societies such as the events that are being run. Notifying the relevant members of the team when an event is created or updated.

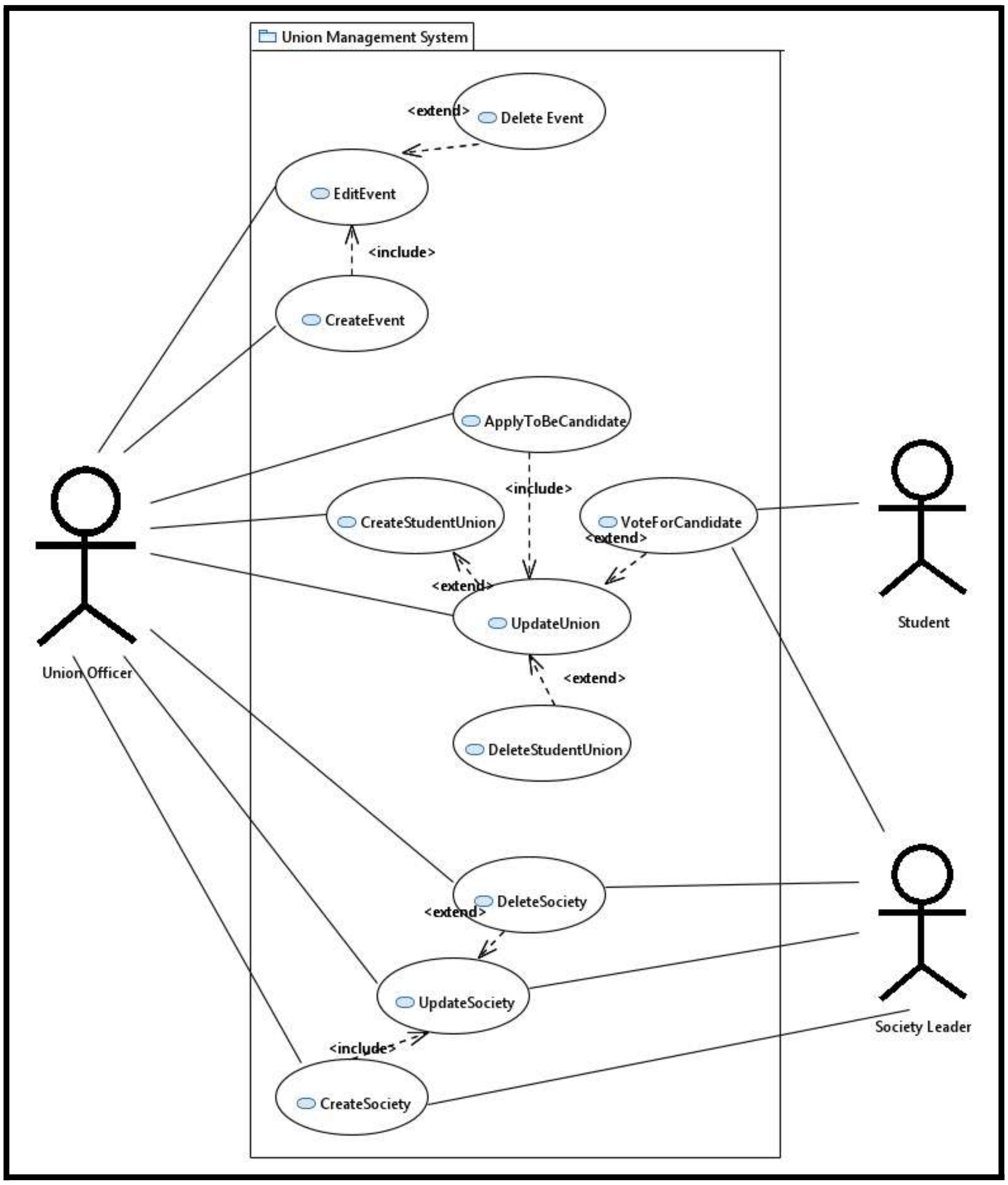
Quality Requirements for managing a society:

“A web-based application to run on desktop or tablet. For officers of Uni-Specific student unions to manage and operate the union.”

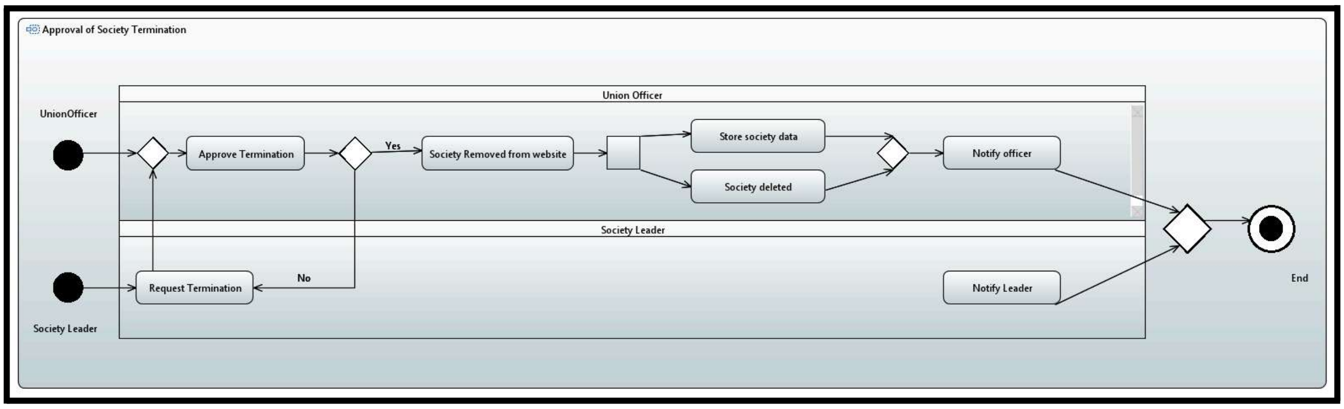
- Security and Privacy protection
    - When the society creation has been approved. Only notify the relevant personnel. (The requesting member and any UO).
    - Only the societies with all the relevant fields can be approved to not cause any later issues.
    - Only relevant data should be presented on the website and no personal information.
    - Once the registration data is updated it should not contain any personal information of any members.
    - Only UOs with enough clearance can approve changes to societies or termination requests.
    - Once a society is terminated. Keep some data behind in case of human error in the request.

    - Post termination makes sure that members cannot find the society anymore and not be able to subscribe to it.
    - Only be able to notify relevant people on the operations of the societies. Have a mailing list and not randomly send out notifications to spam all UOs
- Performance
    - Society is to be available and, on the website, instantly after approval has been received.
    - Any changes to registration data or society data is to be updated and reflected on the website as soon as approval has been received.
    - Society is to be removed from the website instantly as soon as the request has been accepted.
    - Make sure that a mailing list is available for UOs that are able to be notified when there are requests or changes to make sure not every officer is spammed.
    - Once society has been deleted, keeping information for swift reinstatement if required or requested.
- Reliability
    - The management system should maintain a 99% uptime.
    - Requests should be sent across to UOs as soon as possible to give more available time for them to respond to the requests.
    - Reminders could be set up to give the UOs timeframes when they need to reply to requests. (e.g. when they are waiting on data or missing information)
    - Information being provided to the system should be escape checked and proofread to make sure there are no malicious attacks that could impact system uptime.
    - Societies should always have certain information linked such as contact information, to be able to contact the society leaders if there is any issues,
    - If a society has been inactive for a prolonged period. Attempt to contact the society leader and check if needed still. To reduce space usage on the system.
    - Removed data should only be accessible by UOs.
- Scalability
    - Having a limit to how many societies a society leader can have would save enough server space to allow for a greater variety of societies.
    - Having a payment system that students can use to generate the necessary funding a society needs to keep functioning
    - Expansion of the society information such as having operating times could provide more information to users for minimal impact on data storage.
    - Having an automated system for looking at all fields being filled and that they escape checked to streamline rejections on societies missing information.
    - The same automated system to check on changes to registration data must have all relevant fields filled in or the request is rejected instantly.
    - Have a captcha to make sure that a termination request is not submitted accidently.

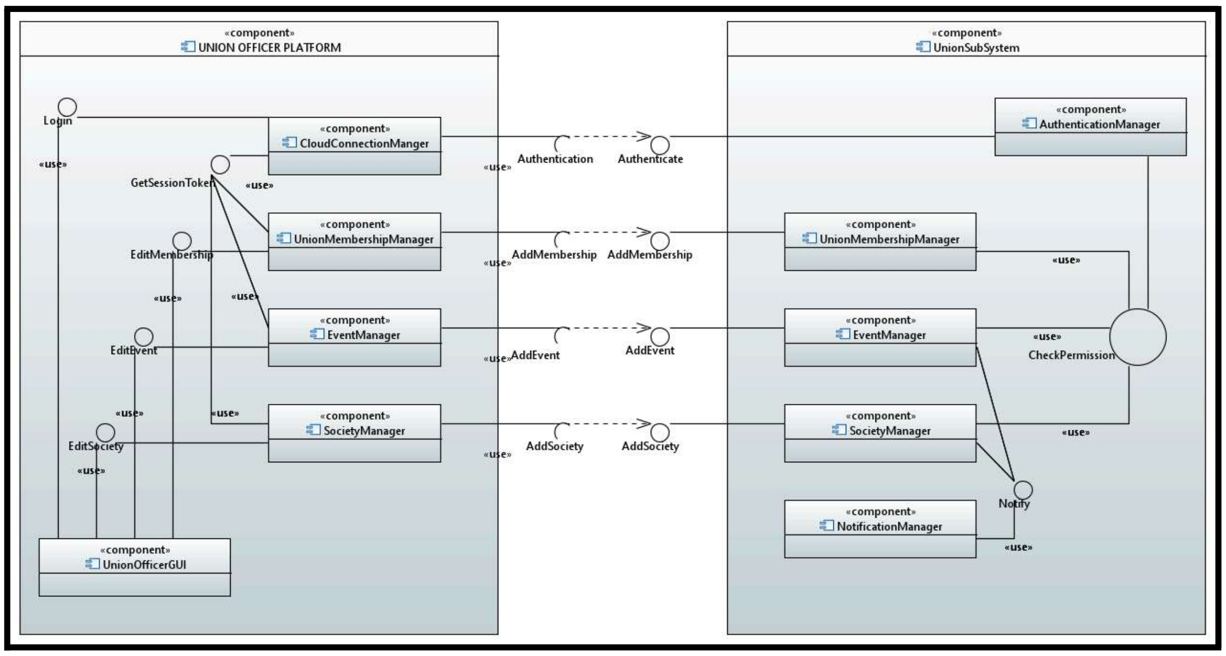
3A - Use Case Model:



3B - Activity Model:



4A - Component Diagram:


«component» UNION OFFICER PLATFORM
«component» UnionSubSystem
Login
«use»
GetSessionToken
«use»
«component» CloudConnectionManger
«use»
Authentication
Authenticate
«component» AuthenticationManager
EditMembership
«component» UnionMembershipManager
«use»
AddMembership
AddMembership
«component» UnionMembershipManager
«use»
«use»
«use»
«component» EventManager
«use»
AddEvent
AddEvent
«component» EventManager
«use»
CheckPermission
EditEvent
«use»
«use»
«component» SocietyManager
«use»
AddSociety
AddSociety
«component» SocietyManager
«use»
EditSociety
«use»
Notify
«component» NotificationManager
«use»
«component» UnionOfficerGUI

Documentation

UnionOfficerPlatform:

Component	UnionOfficerGUI
Description	Runs on application to provide the user interface for union officers to access the platform.
Stereo Type	Component
Required Interfaces	N/A
Provided Interfaces	-Login -EditEvent -EditUnionManagment -EditSociety

Component	CloudConnectionManager
Description	Runs on application to authenticate components and user logging in.
Stereo Type	Component
Required Interfaces	-Login
Provided Interfaces	-GetSessionToken -Authentication

Component	EventManager
Description	Runs on application to send requests to the cloud to manage the event details.
Stereo Type	Component
Required Interfaces	-GetSessionToken
Provided Interfaces	-EditEvent -AddEvent

Component	UnionMembershipManager
Description	Runs on Application to request cloud to make changes in status and data in union membership.
Stereo Type	Component
Required Interfaces	-GetSessionToken
Provided Interfaces	-EditUnionManagment -AddMembership

Component	SocietyManager
Description	Runs on application to send requests to the cloud to manage the society details.
Stereo Type	Component
Required Interfaces	-GetSessionToken
Provided Interfaces	-EditSociety -AddSociety

UnionSubSystem:

Component	AuthenticationManager
Description	Runs on cloud to authenticate users and services.
Stereo Type	Component - Service
Required Interfaces	-Authentication
Provided Interfaces	-CheckPermission

Component	Event manager
Description	Runs on cloud to process the changes to events.
Stereo Type	Component - Service
Required Interfaces	-CheckPermission
Provided Interfaces	-Add Event

Component	Union membership manager
Description	Runs on cloud to process the changes to union membership.
Stereo Type	Component - Service
Required Interfaces	-CheckPermission
Provided Interfaces	-Add Membership

Component	Society manager
Description	Runs on cloud to process the changes to society.
Stereo Type	Component - Service
Required Interfaces	-CheckPermission
Provided Interfaces	-Add Society

Component	Notification manager
Description	Runs on Cloud to send notifications of changes.
Stereo Type	Component - Service
Required Interfaces	N/A
Provided Interfaces	-Notify

Interfaces:

Name	Login
Provider	CloudConnectionManager

Operation	Signature	Login(UnionOfficerID)
	Function	Provides ID of union officer and logs into their account.

Name	Authenticate	
Provider	AuthenticationManager	
Operation	Signature	Authenticate(UnionOfficerID)
	Function	Passes login information across to the server to confirm the details are correct.

Name	GetSessionToken	
Provider	CloudConnectionManager	
Operation	Signature	CreateSessionToken()
	Function	Creates a session token for each other component on the officer platform.

Name	EditEvent	
Provider	EventManager	
Operation	Signature	EditEvent(EventID)
	Function	Updates the event details at the ID provided

Name	AddEvent	
Provider	EventManager	
Operation	Signature	AddEvent()
	Function	Sends a request to EventManager to create a new event.

Name	EditUnionMembership	
Provider	UnionMembershipManagement	
Operation	Signature	EditUnionMembership(UnionID)
	Function	Updates the membership details of a union at the provided ID

Name	AddMembership	
Provider	UnionMembershipManager	
Operation	Signature	AddMembership()
	Function	Sends a request to the server to add a membership entry to the database.

Name	EditSociety	
Provider	SocietyManager	
Operation	Signature	EditSociety(SocietyID)
	Function	Sends the update request to the society manager where the ID is provided.

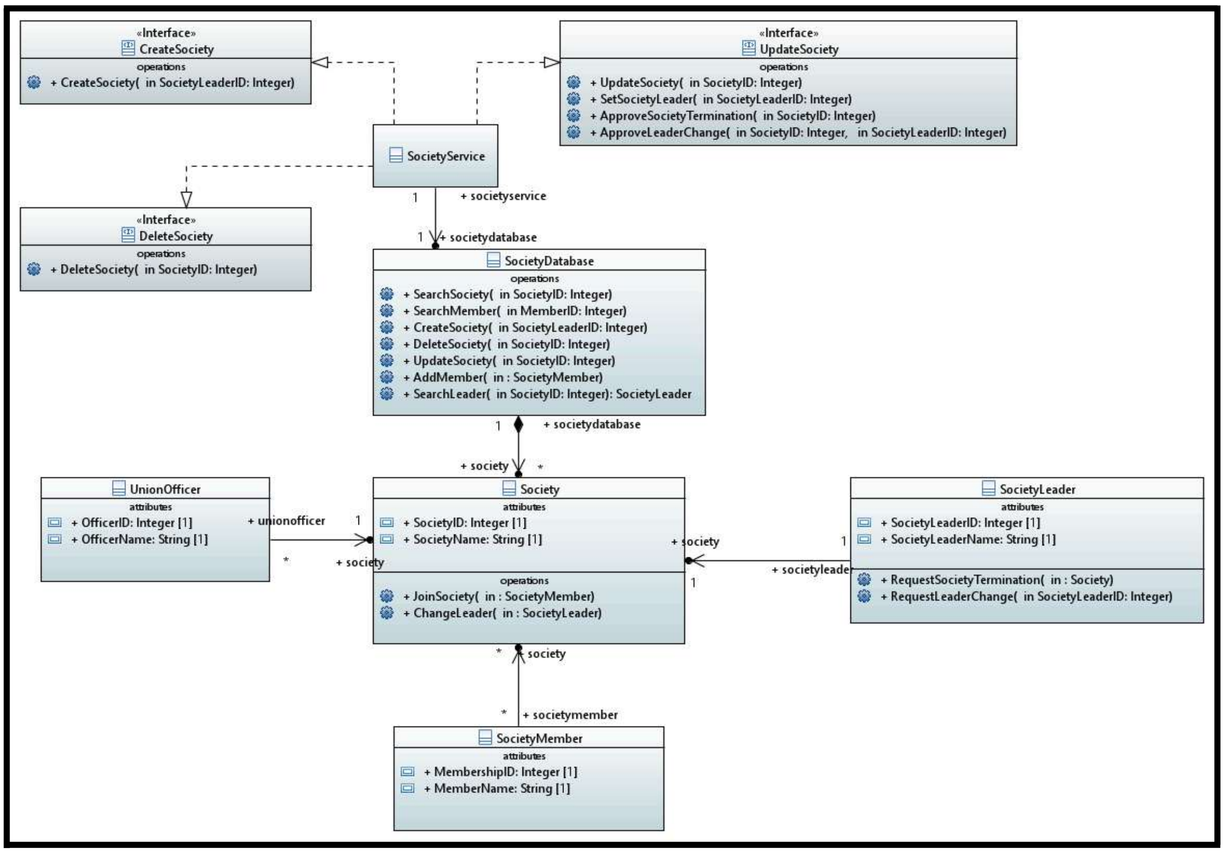
Name	AddSociety	
Provider	SocietyManager	
Operation	Signature	AddSociety()
	Function	Sends a request to the server to create a new society.

Name	CheckPermission
Provider	AuthenticationManager

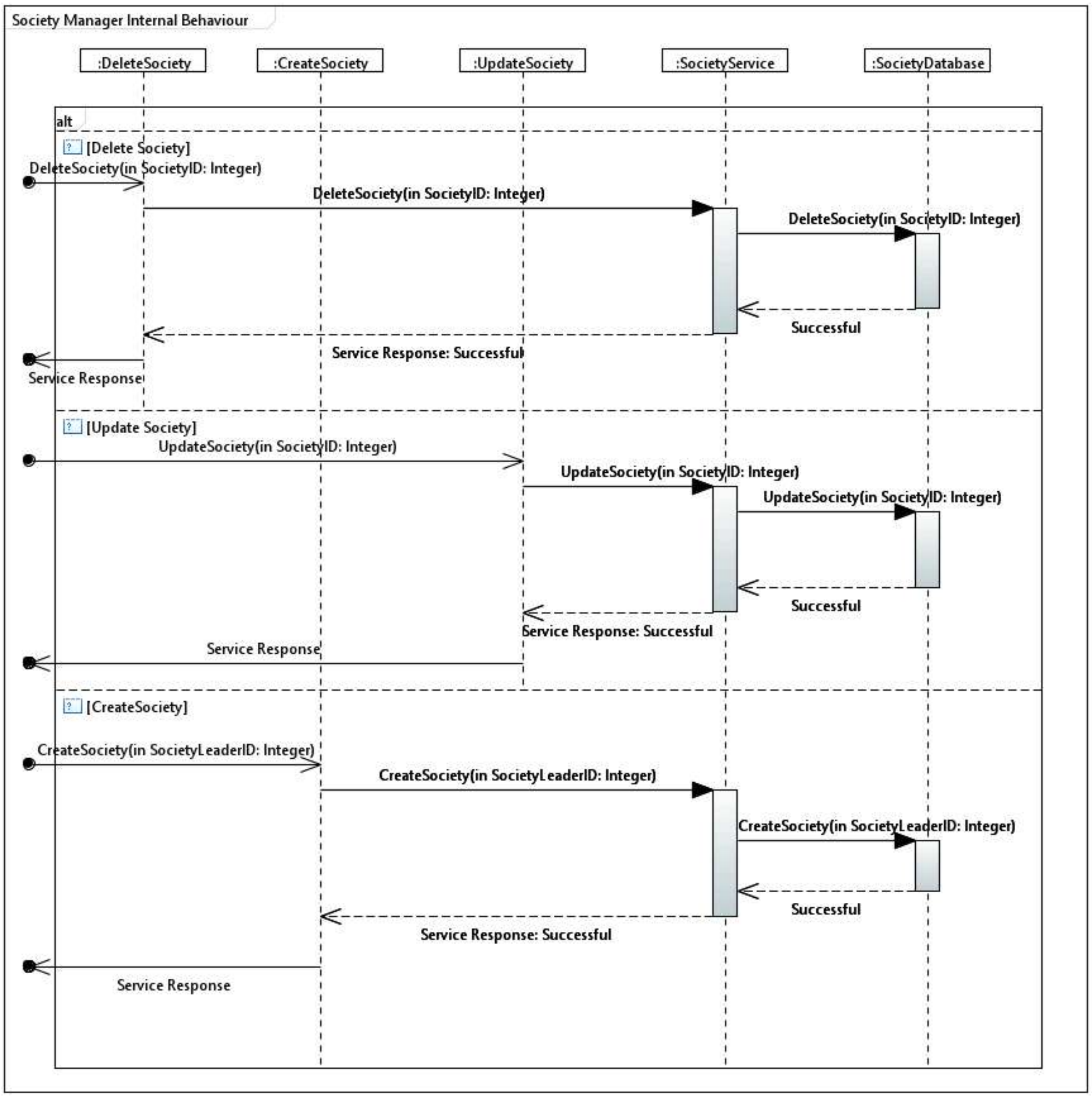
Operation	Signature	CheckPermission()
	Function	Confirms permissions are accepted before allowing service components to access data.

Name	Notify	
Provider	SocietyManager EventManager	
Operation	Signature	NotifySociety()
	Function	Sends out notifications regarding updates to the society to requirement members.
Operation	Signature	NotifyEvent()
	Function	Sends out notifications regarding events to a maillist.

5A - Structural Model, Class Diagram:


«Interface»
CreateSociety
operations
+ CreateSociety(in SocietyLeaderID: Integer)
«Interface»
UpdateSociety
operations
+ UpdateSociety(in SocietyID: Integer)
+ SetSocietyLeader(in SocietyLeaderID: Integer)
+ ApproveSocietyTermination(in SocietyID: Integer)
+ ApproveLeaderChange(in SocietyID: Integer, in SocietyLeaderID: Integer)
SocietyService
1
+ societyservice
«Interface»
DeleteSociety
operations
+ DeleteSociety(in SocietyID: Integer)
1
+ societydatabase
SocietyDatabase
operations
+ SearchSociety(in SocietyID: Integer)
+ SearchMember(in MemberID: Integer)
+ CreateSociety(in SocietyLeaderID: Integer)
+ DeleteSociety(in SocietyID: Integer)
+ UpdateSociety(in SocietyID: Integer)
+ AddMember(in : SocietyMember)
+ SearchLeader(in SocietyID: Integer): SocietyLeader
1
+ societydatabase
+ society
*
UnionOfficer
attributes
+ OfficerID: Integer [1]
+ OfficerName: String [1]
+ unionofficer
1
*
+ society
Society
attributes
+ SocietyID: Integer [1]
+ SocietyName: String [1]
operations
+ JoinSociety(in : SocietyMember)
+ ChangeLeader(in : SocietyLeader)
+ society
1
1
+ societyleader
SocietyLeader
attributes
+ SocietyLeaderID: Integer [1]
+ SocietyLeaderName: String [1]
+ RequestSocietyTermination(in : Society)
+ RequestLeaderChange(in SocietyLeaderID: Integer)
*
society
*
+ societymember
SocietyMember
attributes
+ MembershipID: Integer [1]
+ MemberName: String [1]

5B - Sequence Diagram:


Society Manager Internal Behaviour
:DeleteSociety
:CreateSociety
:UpdateSociety
:SocietyService
:SocietyDatabase
alt
[Delete Society]
DeleteSociety(in SocietyID: Integer)
DeleteSociety(in SocietyID: Integer)
DeleteSociety(in SocietyID: Integer)
Successful
Service Response: Successful
Service Response
[Update Society]
UpdateSociety(in SocietyID: Integer)
UpdateSociety(in SocietyID: Integer)
UpdateSociety(in SocietyID: Integer)
Successful
Service Response: Successful
Service Response
[CreateSociety]
CreateSociety(in SocietyLeaderID: Integer)
CreateSociety(in SocietyLeaderID: Integer)
CreateSociety(in SocietyLeaderID: Integer)
Successful
Service Response: Successful
Service Response