Requirements Analysis Document

Proposed system

Functional requirements

Calender

The Calendar is used to keep track of events, e.g. Work calendar or Holiday calendar. The Calendar contains events specific to only itself. A new calendar is initiated by the Client with a name that described the calendar. The Calendar can be modified by the Client. A Calendar may contain one or more events. The system can contain multiple calendars.

Event

An Event has a start date and an end date. An Event belongs to a Calendar. Events are initiated by the Client and can also be modified by the Client. An Event may repeat on multiple dates, e.g. every week/month. Events have alarms that be set to notify the user

Client

The Client manages both the Calendars and the Events. The Client has access to all of his Calendars and Events and can therefore create, edit, share and delete Calendars and Events. The Client can change the view. The Client has the ability to Hide/Show different Calendars of their liking. The Client can synchronize the Calendars with the Server to they are easily accessible on different devices. The Client has the opportunity to search for specific events.

Server

The Server is responsible for keeping the Client's calendars up to date. When an Event or Calendar is shared the Server must send an invitation to the recipient(s).

Other user

The Other User receives invitations from the Client and can choose to either accept or decline the invitation.

View

The are four types of view: Day View, Week View, Month View, Year View. Besides the view can show one or more Calendars at the same time.

Nonfunctional requirements

Usability

- The user should have basic experience using computers.
- The user should be familiar with popular OS interfaces, e.g. Windows 7 or 8.
- The user should be able to use the system without little or no documentation.

Reliability

- It is important that the Calendars are synchronized.
- Restarting of the system is acceptable in the event of a failure.
- The system may lost at most five of the last changes made by the user in the event of failure.

Performance

- The system should be responsive. There should be little to no latency during actions.
- Server synchronization should take no longer than 1-2 minutes on an average home internet connection.
- The system should support different user accounts on a computer.

Supportability

- Extensions could be the support of other types of Calendar systems, such as GMail, Yahoo Mail etc

Implementation

- Internet connection is needed for synchronizing with the server. If no internet connection is available the data is only stored locally.

Interface

- The data is saved locally and on the server.

Operation

- The user manages the system.

Package

- The system is installed by the user.

System models

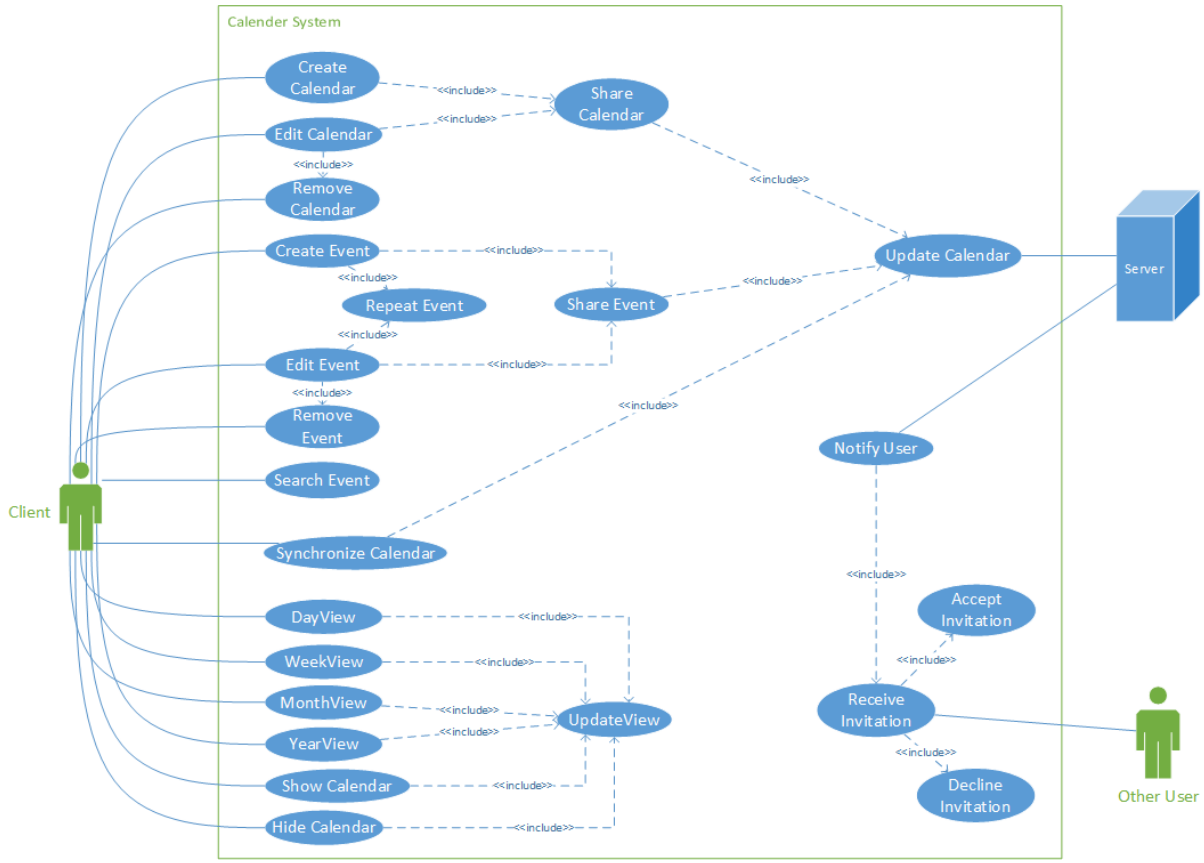
Scenarios

1: Nicolai just got an offer for a new job and wants to delete his old work calendar and replace it with a new one. Nicolai opens his calendar system and selects his old work calendar. He then searches for events he wants to transfer to his new calendar. He finds two events in the old work calendar he wants to transfer to his new calendar. Nicolai then creates a new calendar and creates his first event. While creating his second event, he gets a message saying his job offer is rejected. Luckily Nicolai didn't quit his old job, so he decides to synchronize his calendar to see if the old one is still on the server.

2: Peter has a bad day. Just as he is editing his calendar the lightning strikes down in the tree in the backyard and the power goes down and the computer crashes. Peter tries to reboot the computer but the computer will not start. Peter concludes the smoke from the hardware means his computer is now broken. The following week Peter gets himself a new computer. Peter installs the calendar system again and is annoyed that all his calendars are lost. But then Peter remembers that the calendars are stored on the server and that the calendars are not lost. Peter synchronizes his calendar system with the server and all his calendars are restored.

Use case model

Use case diagrams



Figur 0.1 – UML Use Case Diagram

Use case tabels

Use case name	**Create a repeating event**
Participating Actors:	Initiated by Client Communicates with Server
Flow of events: 1. The CLIENT creates a new event 2. The CLIENT Fills out event form 3. The CLIENT Clicks repeat 4. The CLIENT Selects interval 5. The CLIENT Saves Event. - SERVER Synchronizes calendar	
Entry conditions:	• Calendar must exist in the CLIENT's user profile
Exit conditions:	• Event must have been saved by the CLIENT • Calendar must be synchronized with the SERVER

Tabel 0.1 – Use case tabel 1

Use case name	**Share an existing event with another user**
Participating Actors:	Initiated by Client Communicates with Server and Other_User

Flow of events:

1. The CLIENT chooses an event.
2. The CLIENT clicks Edit Event button
3. The CLIENT clicks on the SHARE button.
4. The CLIENT enters the username or email of another user he would like to share his event with.
5. The CLIENT clicks the SEND button.
   - The SERVER sends an invitation to the OTHER_USER.
   - The SERVER sends a notification “Event shared” to the CLIENT.
6. The CLIENT reads the notification from the SERVER.
7. The CLIENT closes the notification.

Entry conditions:	• The CLIENT must have an existing event. • The CLIENT must be connected to the internet. • The CLIENT must know the email or username of the recipient.
Exit conditions:	• The OTHER_USER must have received the invitation.

Tabel 0.2 – Use case tabel 2

Use case name	**Set two reminders on an existing event**
Participating Actors:	Initiated by Client Communicates with Server

Flow of events:

1. The CLIENT chooses an event
2. The CLIENT clicks Edit Event button
3. The CLIENT clicks Set Reminder
4. The CLIENT selects two reminders
5. The CLIENT sets the time when the reminder should notify the client
6. The CLIENT chooses Reminder Type 'Email' for reminder 1 and Reminder Type 'Notification' for reminder 2
7. The CLIENT saves the event
8. The SERVER sends an email to the CLIENT when the first reminder is fired
9. The SERVER notifies the CLIENT when the second reminder is fired

Entry conditions:	• The CLIENT must have a calendar with a future existing event • The CLIENT must have a live connection to the internet
Exit conditions:	• The CLIENT must be able to add reminders to an event

Tabel 0.3 – Use case tabel 3

Analysis Object Model

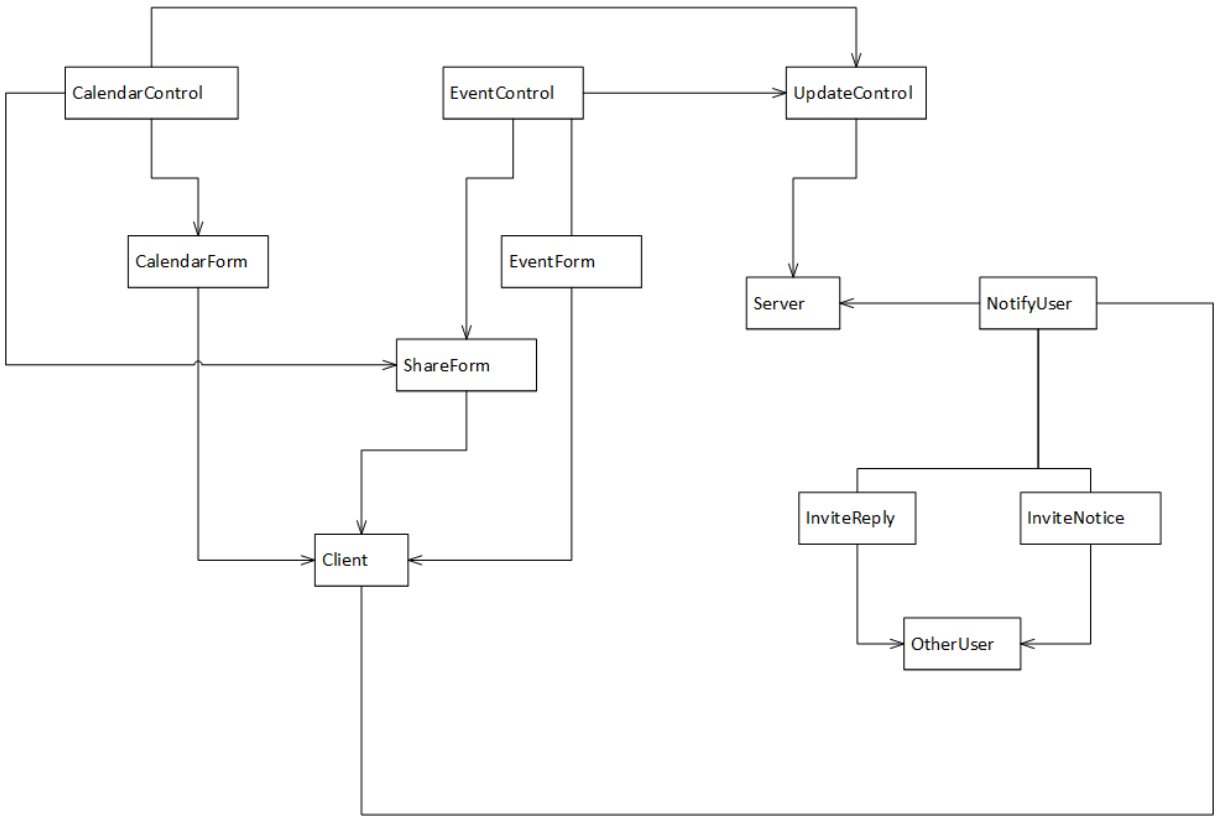
Identifying entity, boundary, and control objects

Identifying Entity Objects		
Entity Object	**Attributes & Associations**	**Definitions**
Client	• UserName	The actor of the system. He administrates the Calendars and Events in the system.
Calendar	• Name • SharedGroup • Color	A Calendar is a system used to keep track of events, e.g. Work calendar or Holiday calendar. The Calendar contains events specific to only itself. The system can contain multiple calendars.
Event	• Name • SharedGroup • Color • StartDateTime • EndDateTime • Alarm • Place • Description	An Event is an event belonging to only one Calendar. Events are used to set alarms or reminders.
OtherUser	• UserName	The actor who receives invitations. He has rights to the other client's calendar.
Alarm	• Type	Alarm notify the client within a given amount of time before the event start.

Boundary Object	**Definition**
CalendarForm	Form used by **CLIENT** to specify the properties of a **calendar** during creation or editing.
EventForm	Form used by **CLIENT** to specify the properties of an **event** during creation or editing.
ShareForm	Form used by **CLIENT** to specify the users that the client would like to share his calendar or event with.
InviteNotice	Notice received by **users** who have been invited to see an **event** or **calendar**.
InviteReply	Notice received by **CLIENT** that informs whether the user has accepted or declined the invite.
SynchronizedNotice	Notice provided by the **SERVER** when the **CLIENT** asked for synchronizing whenever it succeed or fail.

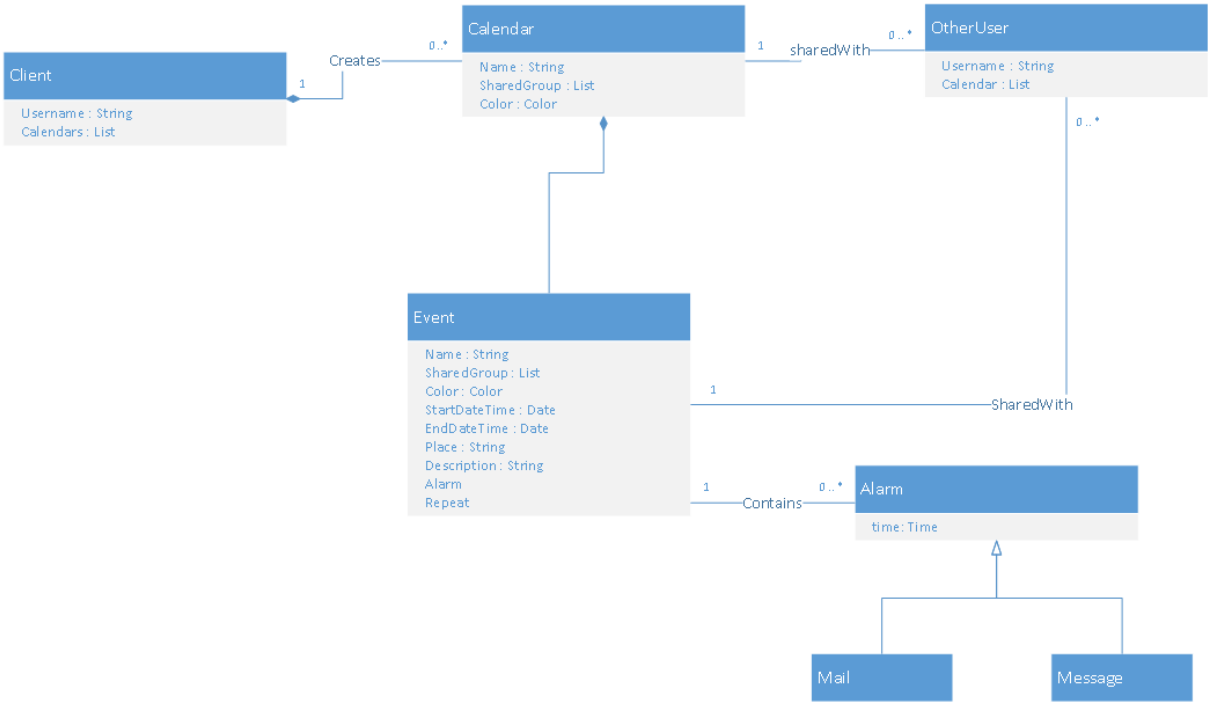
Control Object	**Definition**
CalendarControl	Manages the reporting function for whenever a **calendar** is created, edited og removed. When a **calendar** is created, edited or removed the CalendarControl creates a CalendarForm and present it to the **CLIENT**. It then creates a ShareForm and present it to the **CLIENT**. It submits the forms and then waits for the **SERVER** to automatically update(or if the **CLIENT** synchronizes). The UpdateControl takes over from here.
EventControl	Manages the reporting function for whenever an **event** is created, edited, or removed. When an **event** is created, edited or removed the EventControl creates a EventForm and present it to the **CLIENT**. It then creates a ShareForm and present it to the **CLIENT**. It submits the forms and then waits for the **SERVER** to automatically update(or if the **CLIENT** synchronizes). The UpdateControl takes over from here.
UpdateControl	Manages updates and synchronizations. When a form is submitted the UpdateControl takes over. It waits for the **SERVER** to automatically update or if the user manually synchronizes. When the **SERVER** updates the **calendar** the UpdateControl creates a SynchronizedNotice and displays it to the **CLIENT**.
NotifyUserControl	Manages invitation notices and replies. If a **calendar** or **event** is shared the NotifyUserControl creates an InviteNotice when the system updates. It sends the InviteNotice and creates a InviteReply and waits for the **OtherUser**'s reply. When the **OtherUser** give a reply it is displayed to the **CLIENT**.

Association Class Diagram This diagram is described in the Control Object Table. It illustrates the association between Entity objects, Boundary objects and Control Objects.



Figur 0.2 – Association Class Diagram

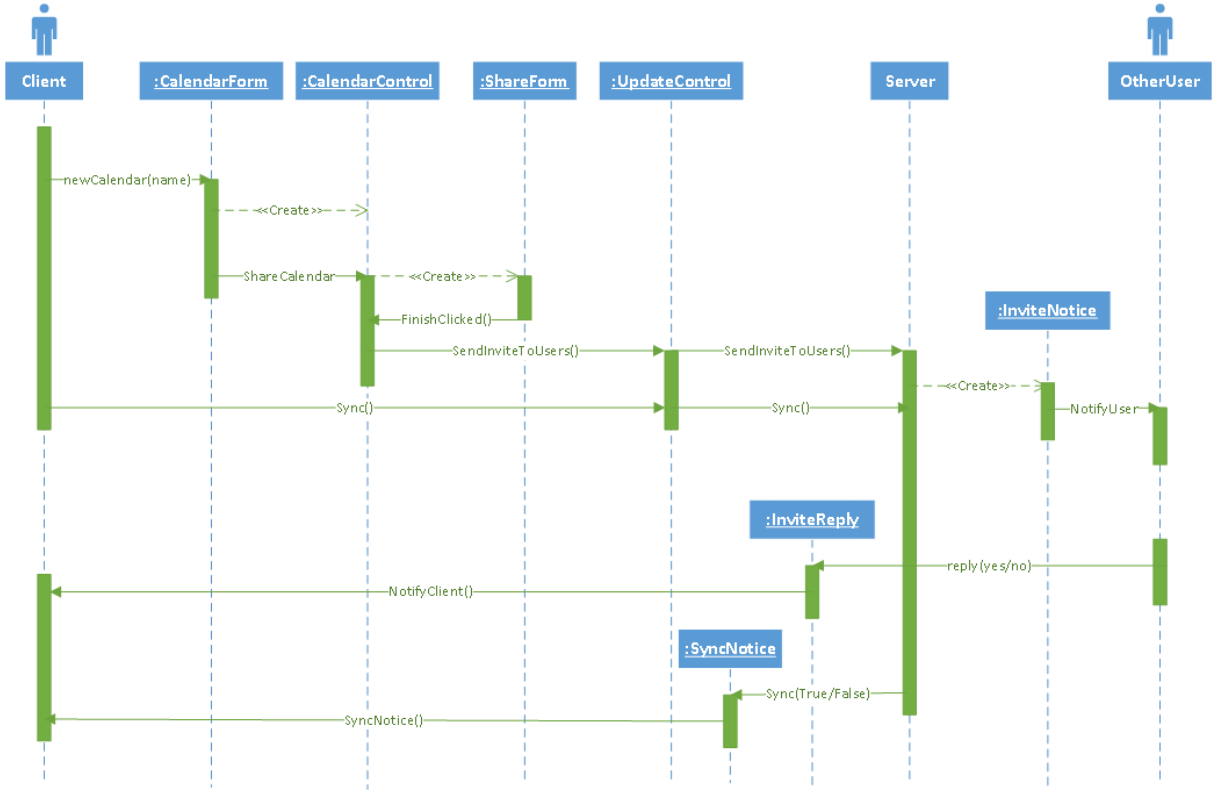
Class model



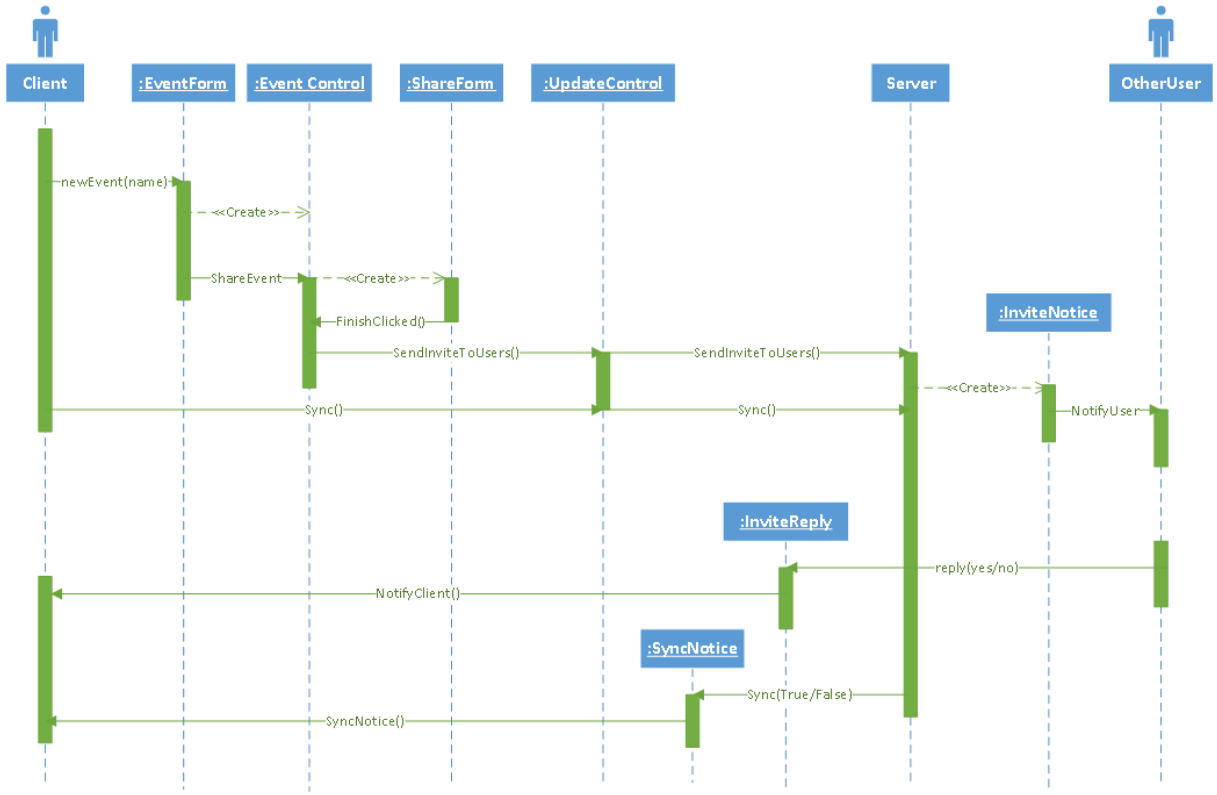
Figur 0.3 – UML Class Diagram

Dynamic model

Sequence Diagram

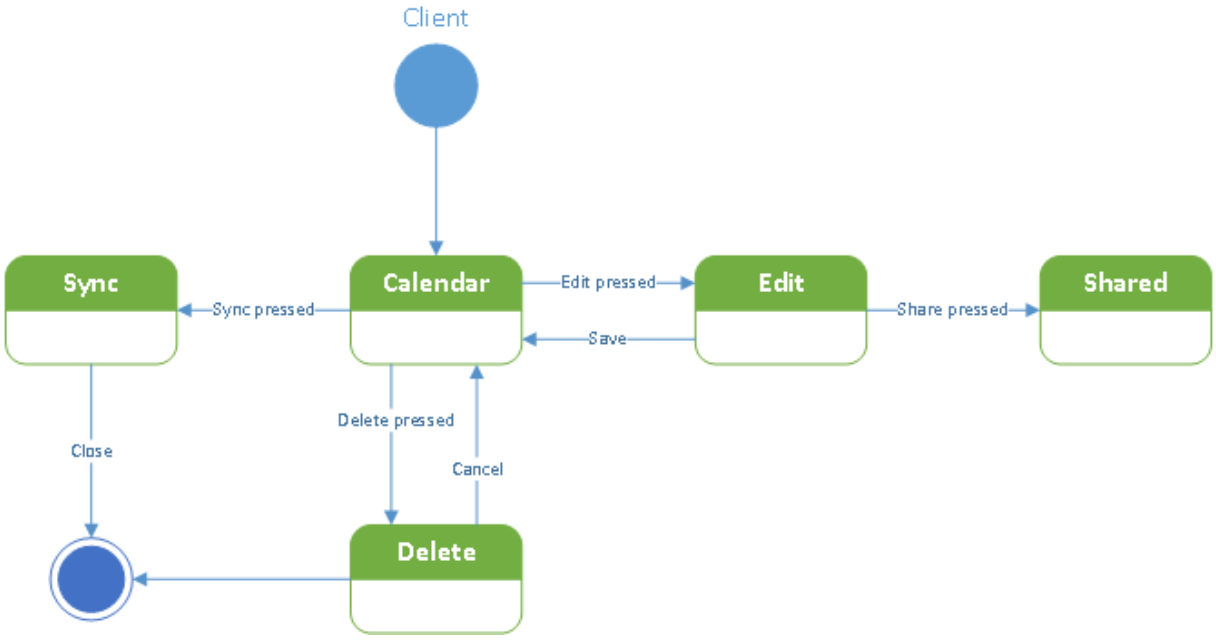


Figur 0.4 – Sequence Diagram Calendar Workflow

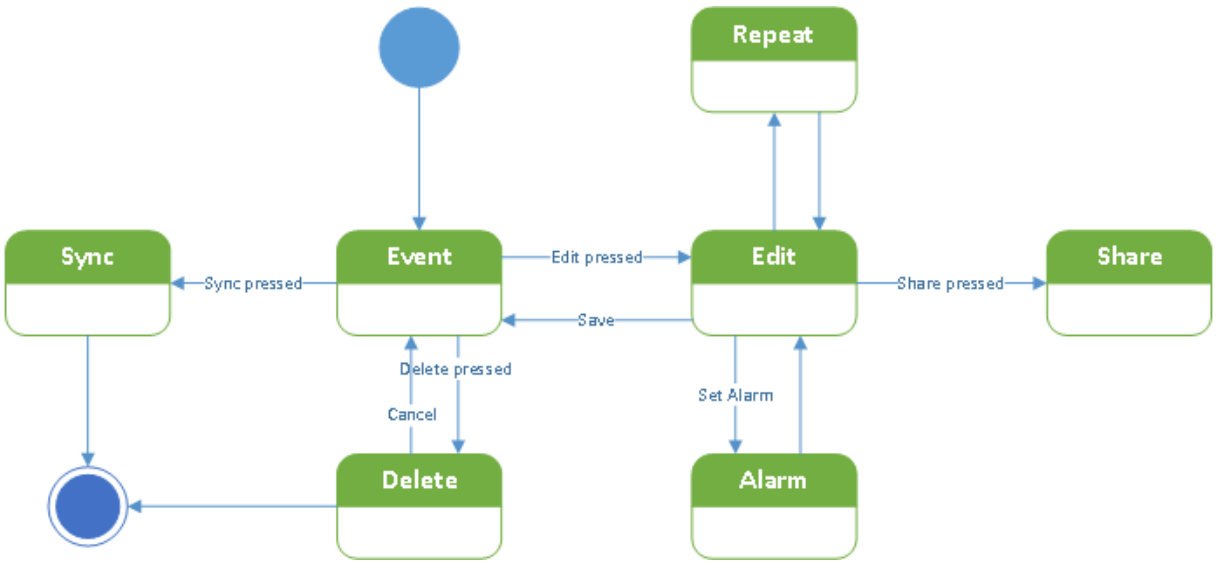


Figur 0.5 – Sequence Diagram Event Workflow

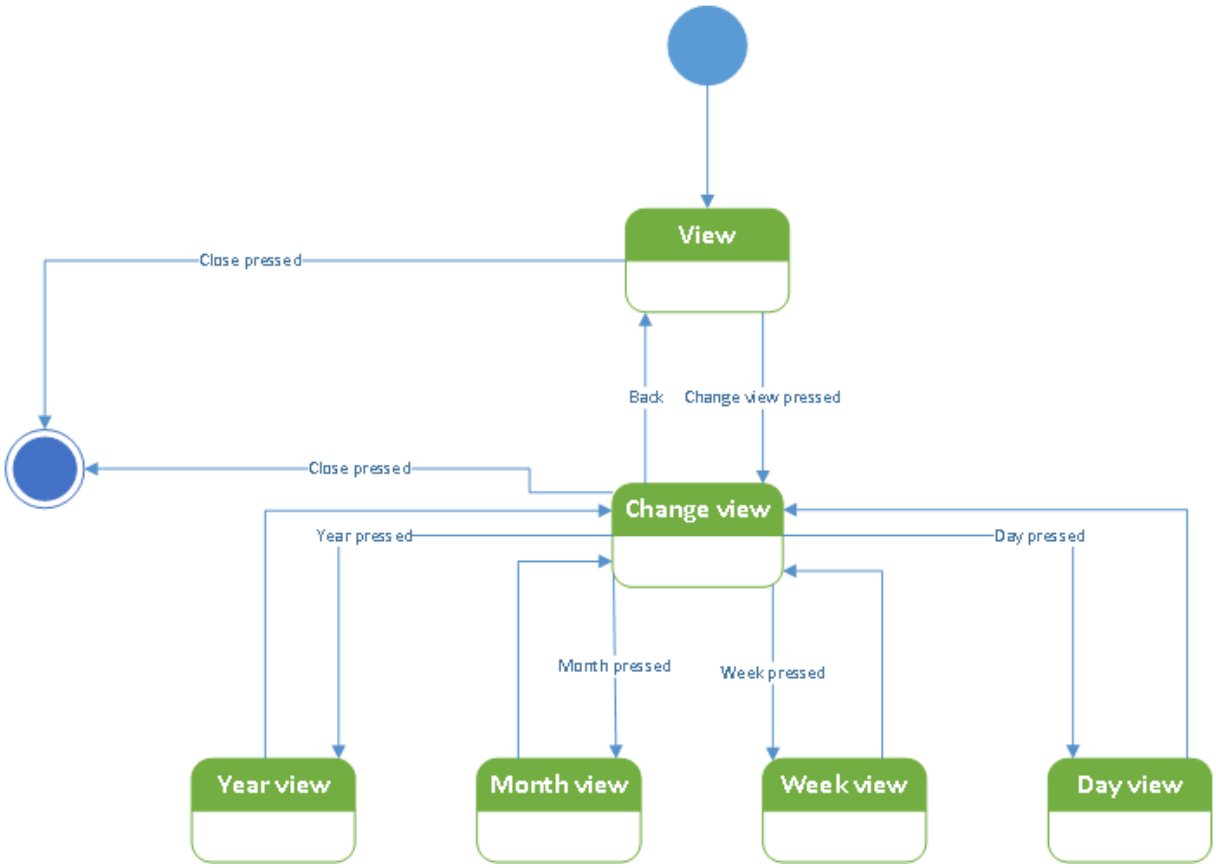
State Machine Diagram



Figur 0.6 – State Machine Calendar Object



Figur 0.7 – State Machine Event Object



Figur 0.8 – State Machine View Object