IT University of Copenhagen

BDSA 2014

Calendar System Requirement Analysis Document

Assignment 38

Anders Wind Steffensen - awis@itu.dk
Christopher Blundell - ppma@itu.dk
Pierre Mandas - cnbl@itu.dk

October 21, 2014

Contents

CHAPTER 1

Introduction

1.1 Purpose of the system

The purpose of the Calendar system is to have a standard client-server calendar system like MS Exchange or iCal. The system must provide the user with the ability to save, edit events and be able to retrieve their calendar on any computer with the system installed. The system should furthermore provide the user with multiple kinds of overviews of the events he has planned, and prompt the user with a notification if an event has a notification time.

1.2 Scope of the system

The system will be created over a short amount of time and therefore will have some limitations.
Features

- Create event
- Edit event
- Two kinds of overview of events: Monthly and Weekly
- Notifications
- Get local time from global time.
- Filter events by tags.
- Get access to events from any Windows 7 or 8 pc

1.3 Objectives and succes criteria of the project

For this project to be succesfull all the features described in the scope of the project must be met. Furthermore the non-functional requirements must work as guidelines for how well the project has met its success criteria.

1.4 Definitions, acronyms and abbrevations

None at the time.

1.5 References

None at the time.

1.6 Overview

CHAPTER 2

Current System

CHAPTER 3

Proposed System

3.1 Overview

3.2 Functional requirements

3.3 Nonfunctional requirements

In this section is the first draft of the non functional requirements written. It does not include, operation, interface or legal, since these subjects were irrelevant.

Usability
User must know how to operate a computer and being familiar with the windows interface standard will help, but should not need any training to use the system. Documentation of how to use the program should be included an available in the program.

Reliability
Should the system crash, all userdata should still be available on next launch, this could be done by uploading the data to the storage as soon as the user has created or updated it. The system should at worst loose the data concerning an event the user was creating during the crash. Force restarting should not be acceptable and most exceptions must be caught and handled runtime. Data losses cant be 100% prevented but it must be a focus in the development

---

Performance
The system should be able to hold a large number of events(around 20 events pr day), without slowing the system down with more than 1 second loading in between. Furthermore most user actions must not be met without long waiting times(again up to 1 second), and heavy work should run in the background and thereby not disturbing the user. The user may have to wait a bit when logging into the system, but no more than 15 seconds with a connection speed of a minimum 2mb down / 2mb up, so the program can retrieve data from the storage.

Supportability
Updating by reinstalling is acceptable in this project, but the system should be easy to extent for a programmer.

Implementation

The testing will probably be a little scarce due to the small amount of time available to develop the system, but Key components must be tested thoroughly. The Calendar is a lightweight system, and stores data on the cloud, so there spacewise a maximum of around 1gb should be achievable. Additionally, in it's current form the CalendarSystem is only runnable on Windows OS.

3.4 System model

Scenarios

Scenario: createCalendarEntry

Actors:USER, DATASTORAGE

Flow:
1. USER gets invited to the most rad party of the year in the hall of his high school by his friend Alfred.
2. USER wishes to be reminded of this rad party on Friday at 3 pm., and decides to enter the EVENT into a CALENDARSYSTEM.
3. USER opens CALENDARSYSTEM, opens his Party-CALENDAR, selects Friday the 2nd. in the CALENDAR-OVERVIEW, and is prompted to enter information about TIME, DESCRIPTION (3pm., rad highschool party with alfred.
4. CALENDARSYSTEM stores the information to the PAC-ClOUD.

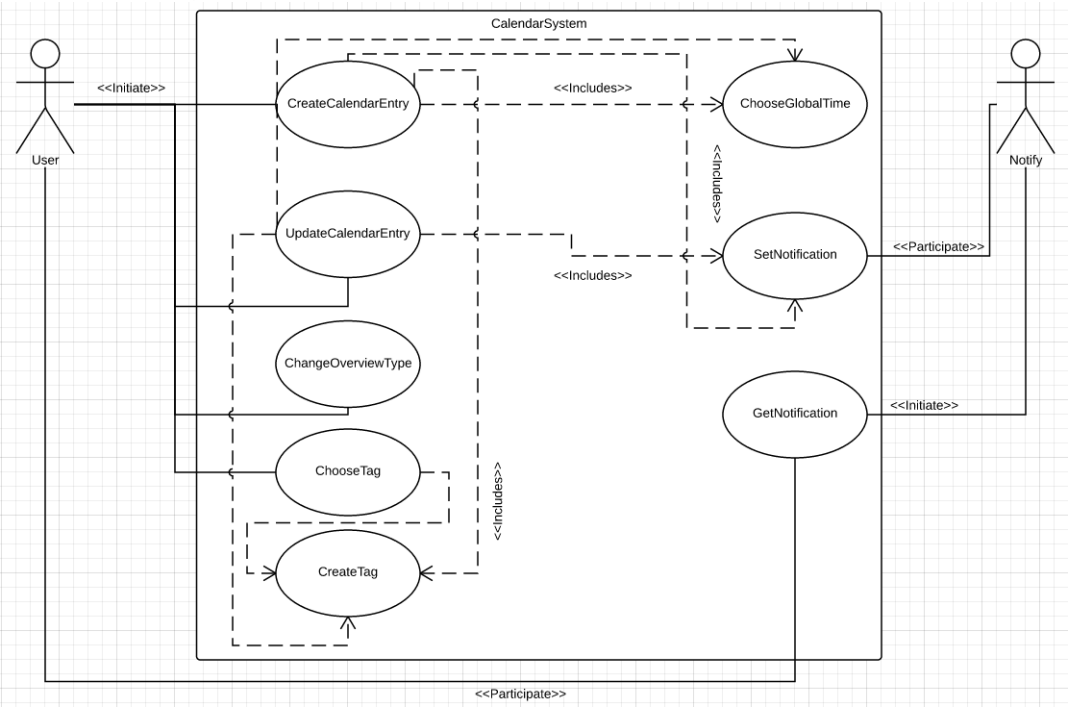
Scenario: updateCalendarEntry

Actors: USER, DATASTORAGE

Flow:
1. USER discovers that his rad party event was moved from Friday the 2nd. at 3pm to Saturday the 3rd at 7pm. due the host Alexis' parent being home on that Friday, because their cruise on the titanic had reported possible delay.
2. USER is currently logged on to the CALENDARSYSTEM. USER locates the event, and opens the EVENT on Friday the 2nd.
3. USER then changes the TIME and DATE to Saturday the 3rd. at 7pm, proceeds to save the EVENT, and the EVENT is saved to the PAC-CLOUD.

Use Case Models

User case diagram:



Here you can see a short description of the three user cases CreateTag, ChooseTag and GetNotifikation

User Case name: CreateTag

Participating Actors:
Initiated by User
Communicates with PAC-CLOUD data storage

Flow of events
User initiates the CreateTag
User chooses a name for the tag
- Exceptional case: User cancels the creation.
User accepts and finish

Entry Condition

The user has logged into the CALENDARSYSTEM
The user is in the tag drop down menu OR creating an Event or Updating an Event

---

EXIT Condition
The user has created a tag or cancelled the creation

User Case name: ChooseTag

Participating Actors:
Initiated by User
Communicates with PAC-CLOUD data storage

Flow of events
User opens the TAG drop down menu
- Exceptional case: CreateTag use case - user wants to create a new tag
- Exceptional case: user closes the dropdown menu
User chooses between his tags or <All>
The Calendar overview is updated with all events with the tag.

Entry Condition
The user has logged into the CALENDARSYSTEM

EXIT Condition
The user has picked a tag or choosen to keep the current tag

User Case name: GetNotifikation

Participating Actors:
Initiated by CALENDARSYSTEM
Communicates with PAC-CLOUD data storage
Prompted to the user

Flow of events
CALENDARSYSTEM creates a popup with a notifikation message of the event
User reads the notification

Entry Condition
The user has logged into the CALENDARSYSTEM
CALENDARSYSTEM has a notifikation from an event set at a specific time.

EXIT Condition
User closes the notification

Initial object analysis

The table describes the predicted classes and their purpose and relations. This will be elaborated on when the class diagram will be added in assignment 38.

Event
A collection of information such as DATE, TIME, DESCRIPTION, NOTIFICATION, TAG.

Calendar
Holds an abitrary number of EVENTS

User
A person who uses the system, who wants to get an overview of the events he/she has.

Data Storage
A storage unit who can send and retrieve CALENDARs for USERs

Notification
A message prompting the USER with information about an upcoming EVENT

Tag
An one-worded description of an EVENT - used to group simular kinds of events together.

Notifier
A collection of notifications that will be popped and shown to the user when a time threshold has been exceeded.

CalendarView
A visual representation of dates and events the USER

EventView
A visual representation of a single event and the data in it to the USER

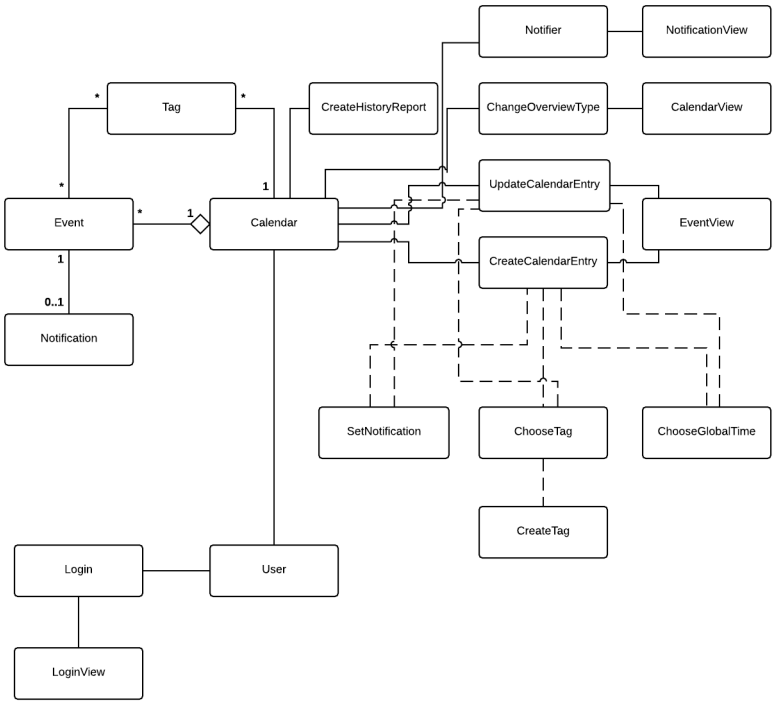
NoticeView
A visual representation a notification to the USER

LoginView
The login screen where the USER will input login info

Analysis Object Model (Static Model):
This table contains our Entity, Control and Boundry objects, that are being used in our system, at the moment. These objects are being used to make our class diagram

Entity	Control	Boundry
Event	CreateCalendarEntry	CalendarView
Calendar	UpdateCalendarEntry	EventView
User	ChangeOverviewType	NoticeView
Notification	ChooseTag	LoginView
Tag	CreateTag	
	SetNotification	
	GetNotification	
	CreateHistoryReport	
	CancelEntry	
	Login	

This is our class diagram. All our Entity, Control and Boundry objects are represented in our class diagram, and our class diagram shows how our objects are connected to each other.

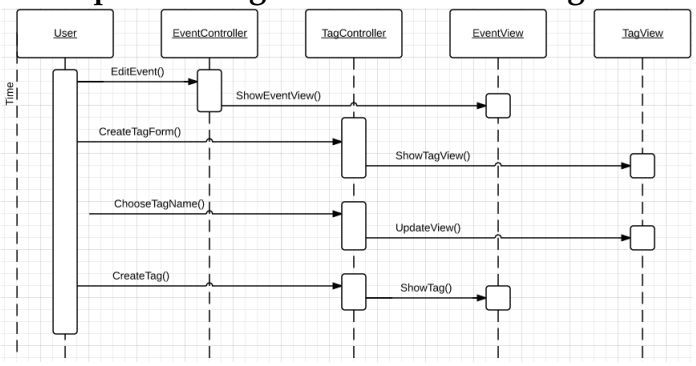


Dynamic Model

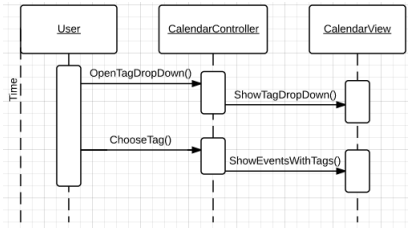
Sequence Diagrams

In this section, sequence diagrams have been created from our use cases. We have the following sequence diagrams:

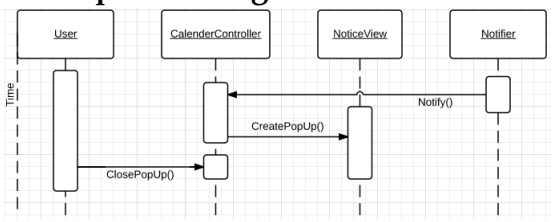
Sequence Diagram for our CreateTag use case:



Sequence Diagram for our ChooseTag use case:

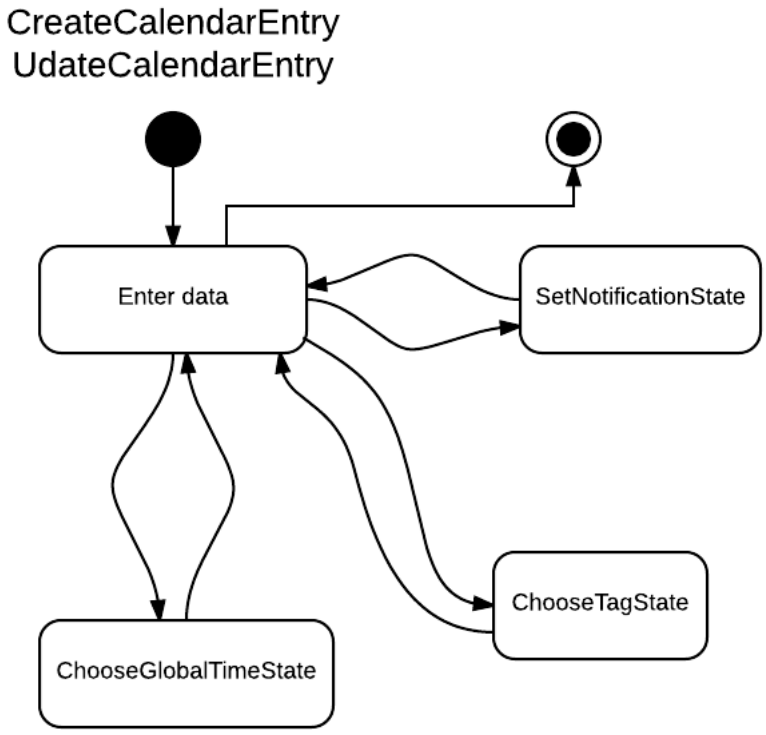


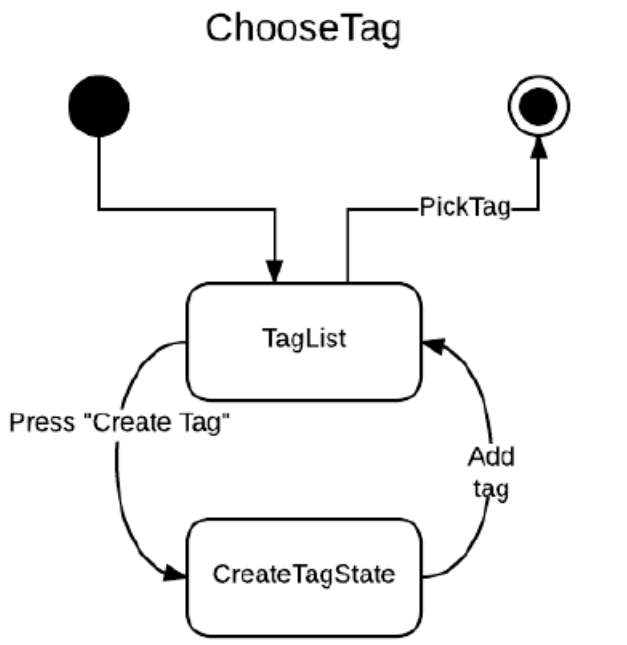
Sequence Diagram for our GetNotification use case:

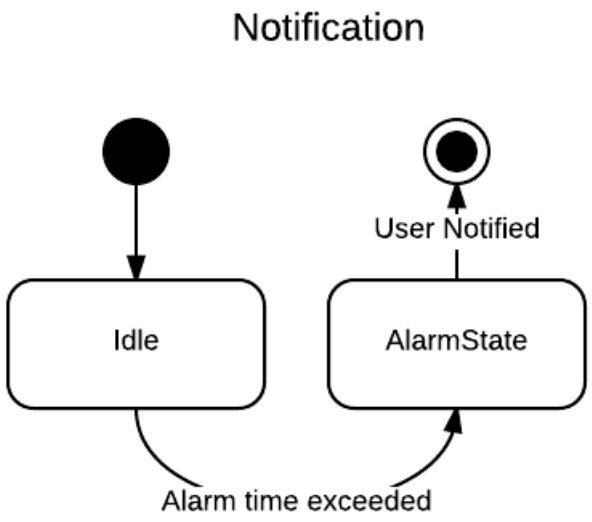


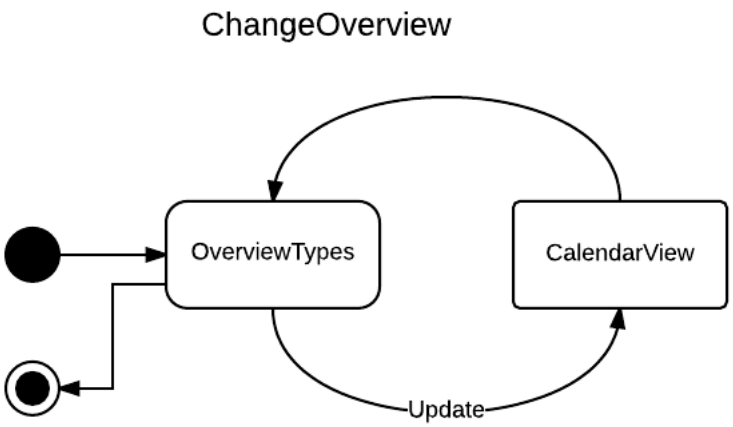
State Machine Diagrams

In this section, state machine diagrams have been created for some of the objects described in the initial object analysis. Some of the diagrams have states where they wait for the other object to respond. In these cases the state will be named as following: ObjectNameState.




ChooseTag
PickTag
TagList
Press "Create Tag"
Add
tag
CreateTagState


Notification
User Notified
Idle
AlarmState
Alarm time exceeded



User Interface

CHAPTER 4

Glossary