System Design Document

Revision History

Revision	Date	Author(s)	Description
1.0	21.10.14	anud, ntho, sjri	Revision: Removed non-functional requirements from Design goals. Updated component diagrams. More coherent.
1.01	22.10.14	ntho, sjri	Revision: Added more design goals: Low operating cost, Scalability, High Availability
1.02	22.10.14	ntho, sjri	Revision: Updated the access control matrix: Removed the objects Global, Local and Account and replaced them with Entity objects found in the RAD document. Renamed the Client actors to User as it seemed more intuitive
1.02	22.10.14	anud	Revision: Updated Subsystem decomposition, Hardware/Software mapping and Persistent data management: Coherent subsystem names. Removed syncManagement and shareManagement. Removed Component Diagram 0.3 and replaced Component Diagram in subsystem decomposition with Component Diagram 0.4

Indholdsfortegnelse

Kapitel 1

Introduction

1.1 Design goals

Low operating cost
To minimize the need for advertisement, the cost of running the system should be minimized. This also leads us to select free or open-source components.

High availability
Unexpected crashes and interruptions of the system can lead to a lot of frustration for the user. Therefore if the system should crash the data should not be lost. The system should do live synchronization with the server and the local storage.

Usability
The Calendar system should be designed with the goal in mind that it should be fairly easy to use by the Client.

Response time
The Calendar system should be quite responsive to user input, and commonly make actions in a matter of milliseconds.

Extensibility
The Calendar system should support the opportunity to extend the program and make it interact with different calendar programs such as Google Calendar, Exchange and iCal.

Scalability
Scalability in terms of number of calendars and events. The response time of the system should not degrade with the number of calendars or events.

Kapitel 2

Current software architecture

Kapitel 3

Proposed software architecture

3.1 Overview

subsectionArchitectural style
We decided to use the MVC pattern as the basic architectural style for the Calendar system.

The MVC pattern separates data and user interface and has Controller as a intermediated component. The pattern is used when there are multiple ways to view and interact with data.

MVC is a abbreviation for Model View Controller. The software is divided in these components. The components are defined as following:

- **Model** handles data.
- **View** is attached to model and presents the data.
- **Controller** is the link between model and view.

To get an overview of the MVC pattern we outlined the pros and cons.

Pros

- low coupling
- high cohesion
- code maintenance
- code reuse

Cons

- code complexity
- development time

In our Object Oriented Analysis an Analysis Object Model was defined where we identified:

- Entity objects (*model*)
- Boundary objebts (*view*)
- Control objects (*controller*)

The Analysis Object Model is a simplification of MVC pattern. It made it eaiser to set up the MVC pattern and build further on the Object Oriented Design.

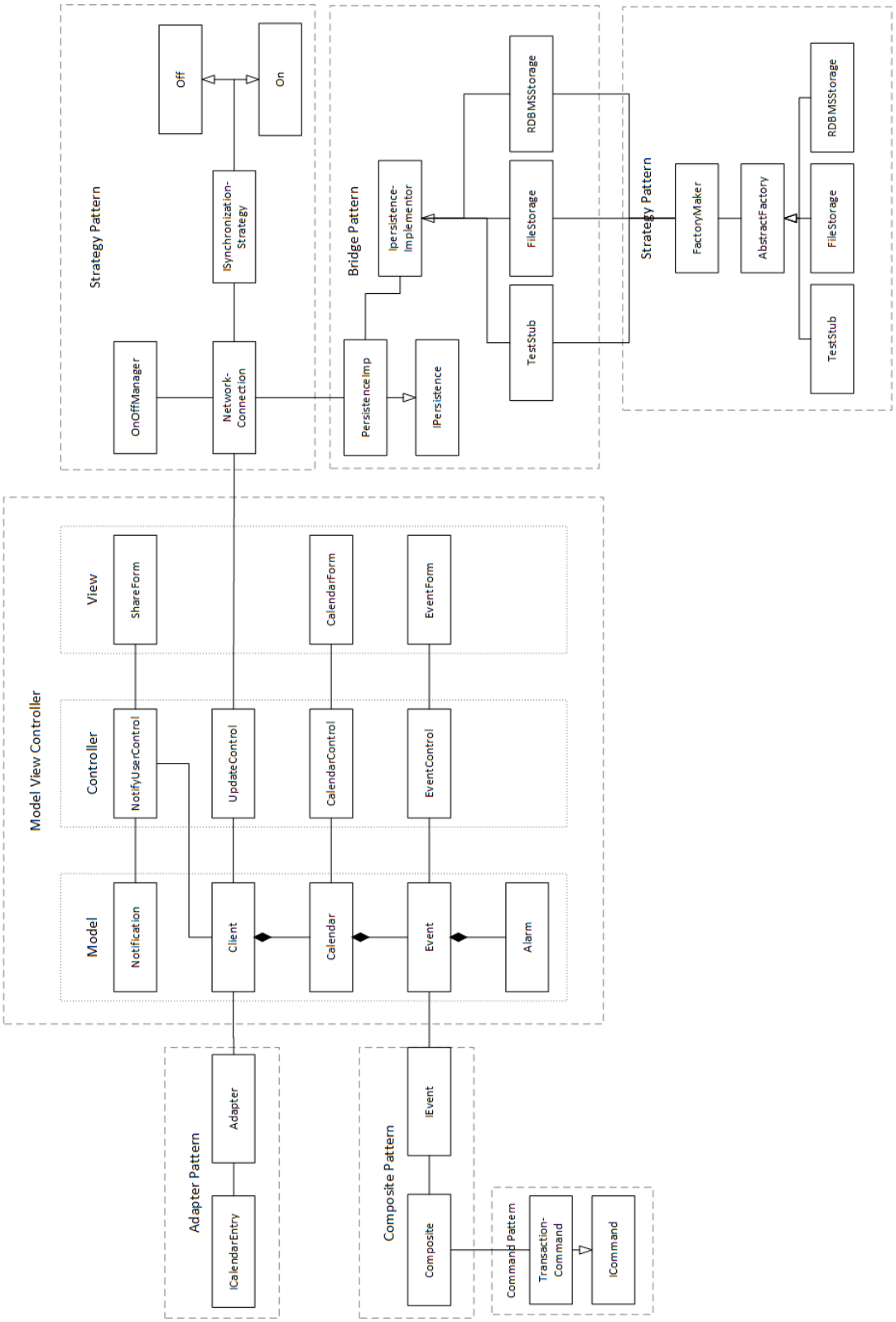
UML Class diagram

Our main program is in the MVC pattern.The Model is used to keep the information about Calendar, Events, Client and Alarms. The Controller deals with modification of the model, e.g. synchronize and share events. The view consist of our main view and our forms, these forms are used by the client to input information that the controller then saves in the model or database. When we want to synchronize our Calendar we need to connect to the database, here is where the strategy pattern comes in. The Strategy pattern observes whether we have a connection to the internet or not. Then the information will be sent on to the bridge pattern. By now we know which kind of storage we would like to use. When the right one is selected the storage will be built by our factory pattern. Furthermore we have implemented the adapter pattern, this has an outgoing connection from google calendar. The information that the google calendar wants to extract will then be sent to the ICalendarEntry and further on to the right adapter that will handle the transformation to our system. We have also implemented Composite pattern but it is a very weak implementation because we do not think that it has any relevance in our program. We already had the repeat event implemented in our own system. The composite pattern has a connection to the events in our model. The composite pattern is used for repeating events.
The UML Class diagram is on next page.

3.2 Subsystem decomposition

The subsystem CalendarApplication is the system where most of the program is placed. This system is responsible for all the local events happening in the system. This means that it would handle creation of the calendar, events and provide the necessary GUI for the local program.

This subsystem has two connections outside the subsystem itself. The first one goes to "ShareInterface"that has all the necessary functions that the CalendarApplication

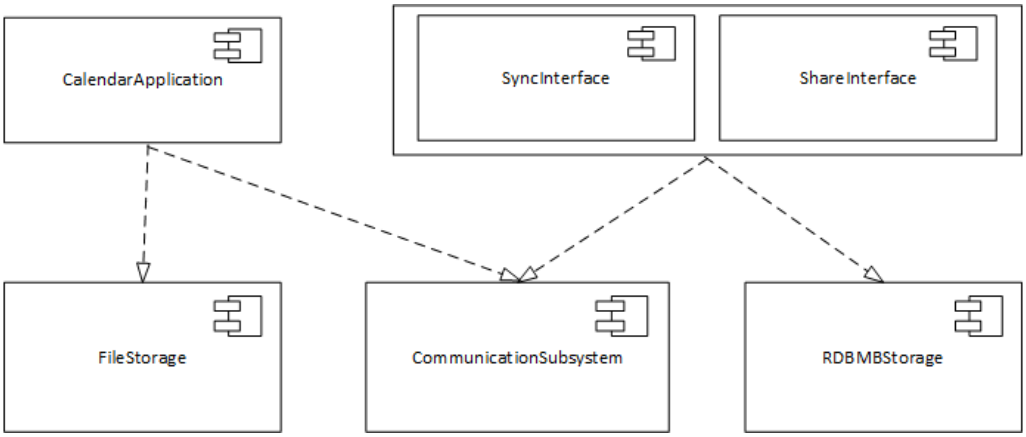


Figur 3.1 – UML Class diagram

system could ask for when it is about to share a calendar or an event. The other outgoing connection is to the SyncInterface. This is an interface that holds all the needed method calls to synchronize the calendar.

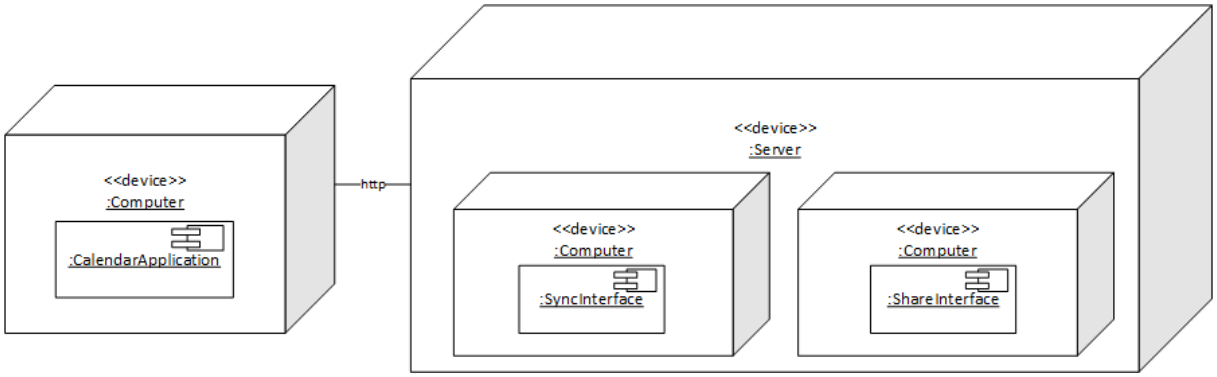
The ShareInterface has to do with sending the information to the right people and using the right systems. The reason why we made this an interface is that it has been a commonly thing to share through facebook and other social media platforms. Therefore it would be nice to easily have the opportunity to extend the share option so that it would support e.g. Facebook.

The subsystem SyncInterface is where we keep our calendars up to date, where we got the information, of whom the calendar belongs to, and which users that are invited to see it. The SyncInterface keep the client calendars synchronized and handles the updates between the local calendar and the online ones. This is also behind an interface so that we can easily exchange it. That would turn to our advantage so the maintenance and upgrades of the system could be done easy. You simply exchange the subsystem part with a new one. E.g. if we wanted to update our synchronization part with a newer version of a data base program, the SyncInterface is exchanged, but the rest could stay intact.



3.3 Hardware/software mapping

Our Calendar system is run by two devices. The user program run on the user computer, and the synchronized parts will be on the server together with our share system. The two devices will have a connection to each other, we have not yet decided which kind of connection it would be.



Figur 3.2 – UML Deployment diagram

Now that we have mapped our program to the hardware that is used to run our system we dicoverred that some informations had to be handled between the computer and the server. Therefore we have made up a Component Diagram with the subsystems. If we study the 'CalendarApplication' the 'SyncInterface' and the 'ShareInterface', we find that they all have instances of the same classes. This information kept in these instances has to be transfered from the computer to the server via a network connection. Here comes the 'CommunicationSubsystem' in play, this is our subsystem that handles the information and connection between the 'SyncInterface' and the 'CalenderApplication'. 'CommunicationSubSystem' also handles the connection to the 'ShareSubSystem'.

3.4 Persistent data management

Persistent data

FileStorage is responsible for storing the data from the calendar on the computer. This functionality is used before the system is synchronized. When synchronized, the data is stored in RDBMStorage, and the data in FileStorage is deleted.

RDBMStorage is responsible for storing all data from CalendarApplication in a database for the server.

Selecting strategy

- In the FileStorage the data is stored as flat files in plain text.
- Data in RDBMStorage is stored in af Relational Database.

3.5 Access control and security

In fact that our program have multiple users it good to clarify what functions that are available for the different users in the three states a Calendar can be. The three states that our calendar can be in is.

- The Global: This Calendar is visible to all users, that subscribe, but it can only be edited by the Client that made the Calendar.
- The Local: This Calendar is the one that the Client have shared with his friends, family og coworkers. Both the Client and the subscribing clients have the opportunity see and edit events. The Client that created the Calendar, off course as the only one has the possibility to delete the Calendar.
- The Account: The Calendar that can only be seen and edited by the Client that it was created by.

With those informations in the back of our head we made an Access Matrix to get a better view of our Objects Actors functions or method calls available at the different states of a Calendar.

Object Actors	**Client**	**Calendar**	**Event**	**Subscribing User**	**Alarm**
User	edit() delete()	getCalender() setCalender() subscribe() unsubscribe() getSubscribedUsers() save() delete()	getEvent() setEvent() subscribe() unsubscribe() getSubscribedUsers() save() delete()		getAlarm() setAlarm() removeAlarm()
Server		synchronize()			
Other User		requestEditAccess() subscribe() unsubscribe()	requestEditAccess() subscribe() unsubscribe()		

Figur 3.3 – Assess control matrix

3.6 Global software control

In OOSE section 7.4.4 there is mentioned three types of control flow paradigms.

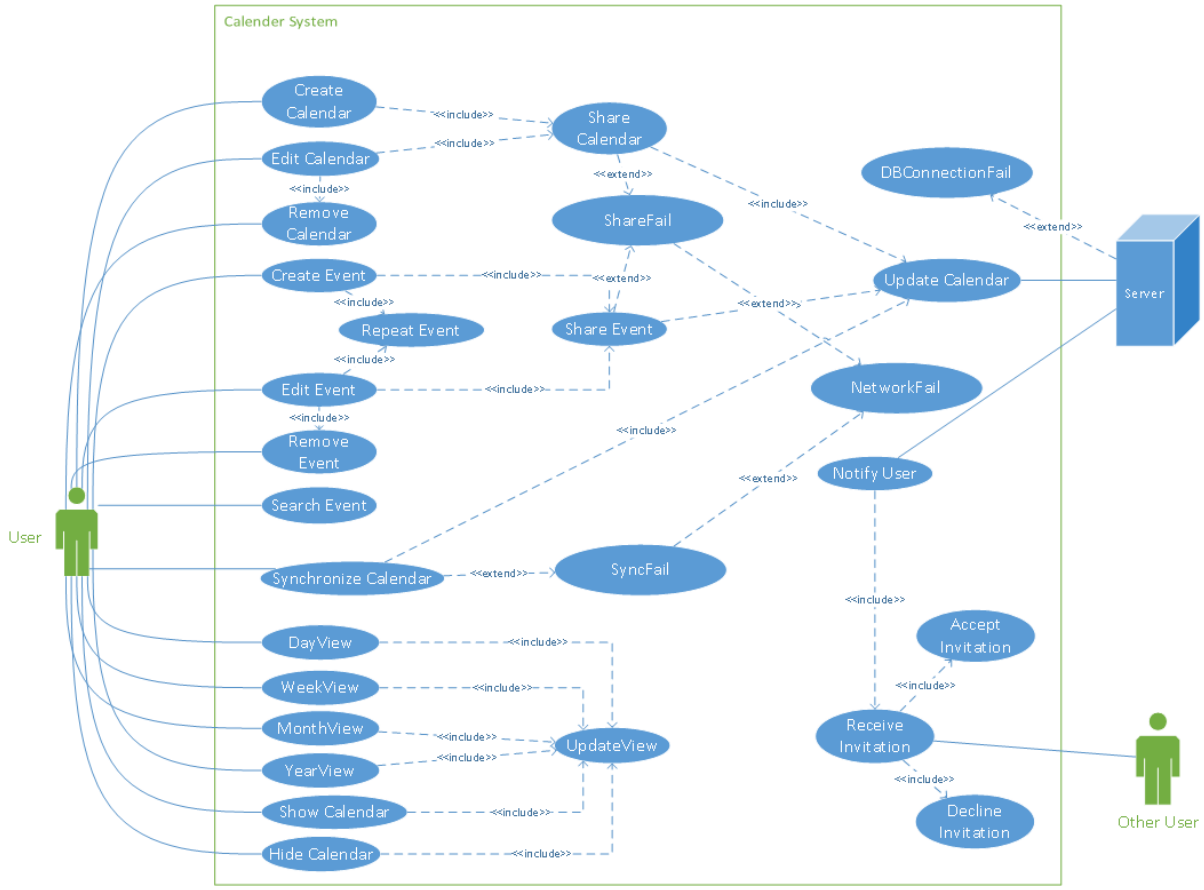
- Procedure-driven control: Operations will wait for an input from actor whenever data is needed.
- Event-driven control: The control flow is determined by events.
- Threads: This is a concurrent version of procedure-driven control.

When having multiple components in a system
Events determines the control flow of the system. The design of the control flow will be event-driven.

3.7 Boundary conditions

Start program To start the program you must be on a computer with the program on it. Then you just have to run the file. If you want the program to have the full functionality at synchronize the calendar, you must have a network connection.

Shutdown program To shut down the program you must be in the program, and be able to click on the lock down button. We have also clarified some boundary use cases that we have put in our use case diagram, to handle some different boundary exceptions.



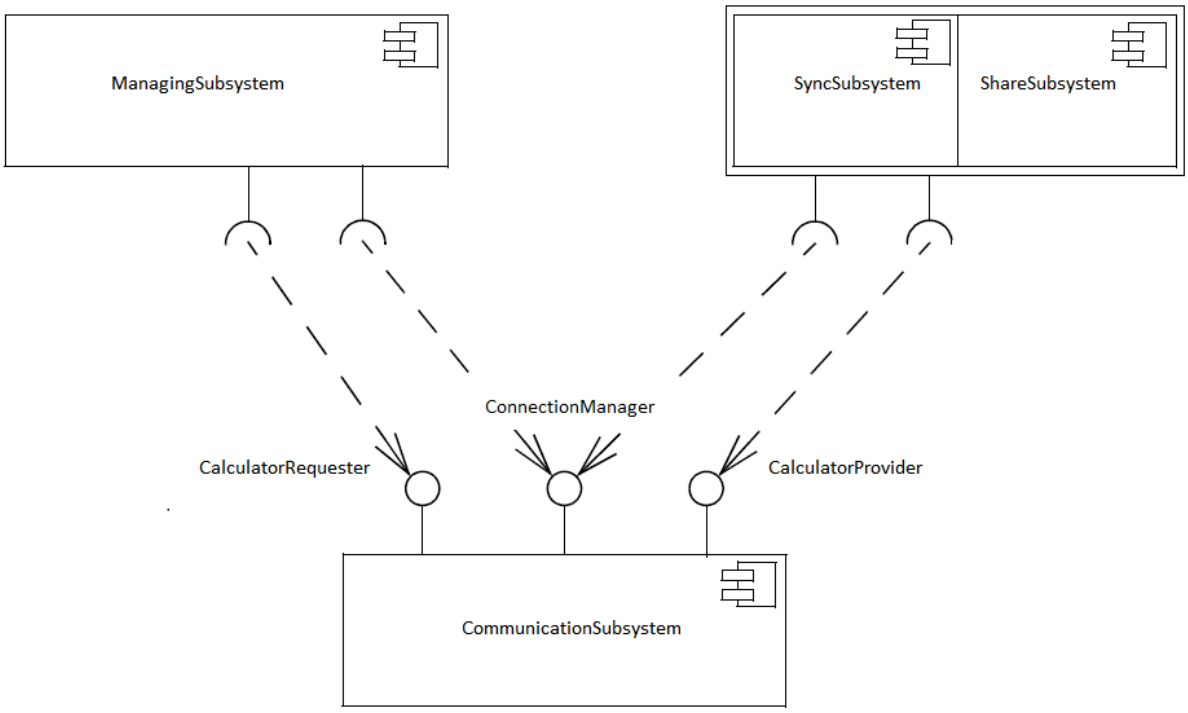
Figur 3.4 – Updated UML Use Case diagram with boundary conditions

Kapitel 4

Subsystem services

Subsystem services

- ConnectionManager allows a subsystem to register with the CommunicationSubsystem and ControlSubsystem and communicate.
- CalculatorRequester allows a subsystem to request a list of available calendars and select among them.
- CalendarProvider allows a subsystem to provide a list of calendar that are available for the client and the subscribing clients.



Figur 4.1 – Subsystem decomposition with services (UML Component diagram)

Kapitel 5

Glossary