Contents

1 Introduction

1.1 Purpose of the system

The purpose of the calendar system is to make a calendar system for a workplace, capable of sharing entries and organizing meetings etc. The system is intended to be used in a workplace environment, where people know each other, or at least are acquainted in some way. Thus the program has no need for contact lists and blocking of people, since it is used by people working together, to organize meetings etc.

1.2 Design goals

Usability: The system should be easy to learn, since it is can be used by a wide array of people with different backgrounds and different levels of expertise, it should be hard to learn since it will not be a small group of experts using the system.

Fault tolerance: The system should be fault tolerant to loss of connectivity with the calendar server. Since the centralised server might die, it would be a problem if every user completely loses access to their calendar on such an occasion.

The system should use a low amount of bandwidth. Since the users most likely will have a client running all the time while working, it should require a low amount of bandwidth as to not slow down the entire network.

The client part of the system should be compatible with older windows versions.since the system is to be used in a work environment, where the upgrade time may vary a lot, it should be compatible with all the commonly used versions of windows, to make it usable by most companies.

1.3 Definitions, acronyms, and abbreviations

1.4 References

1.5 Overview

2 Proposed software architecture

2.1 Overview

The following chapter will go through: the chosen architecture patterns and design of the calendar system, the design patterns used to implement it and how the control flow of the program.
The figure below shows the class diagram of the proprosed system:

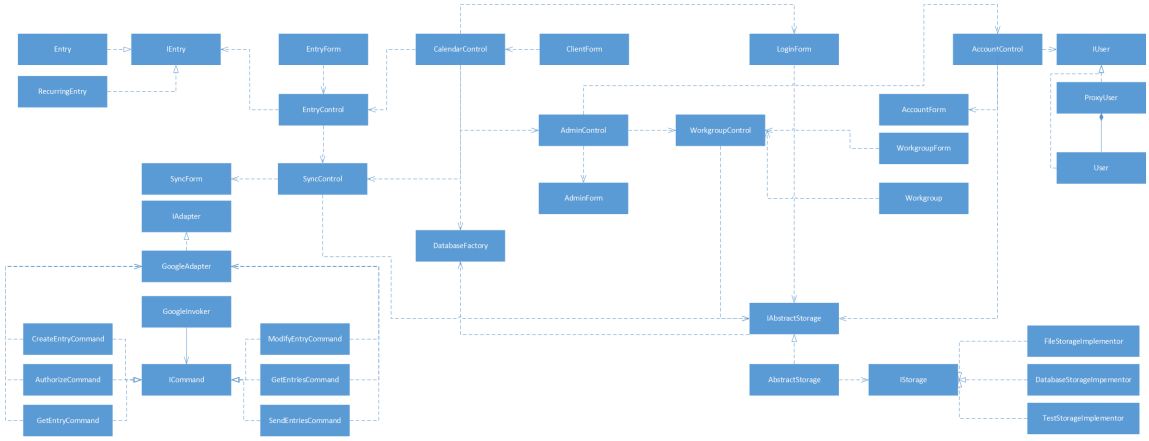

Figure 1: Subsystems and their decomposition

2.2 Subsystem decomposition

The figure shows the subsystem decomposition of the User part of the program. The system is decomposed into subsystems which has logic handling one thing each, which means there's a subsystem for entries, accounts, workgroups and sync. There's a subsystem for handling the client (meaning the general UI) and directing the control flow. Then there's a storage subsystem handling all interactions with the different storages (file and server storage). Finally there is the server subsystem holding the parts of the logic and model, handled by the server. Control, model and view are still separated, but the logic has been split across multiple subsystems each handling something different.

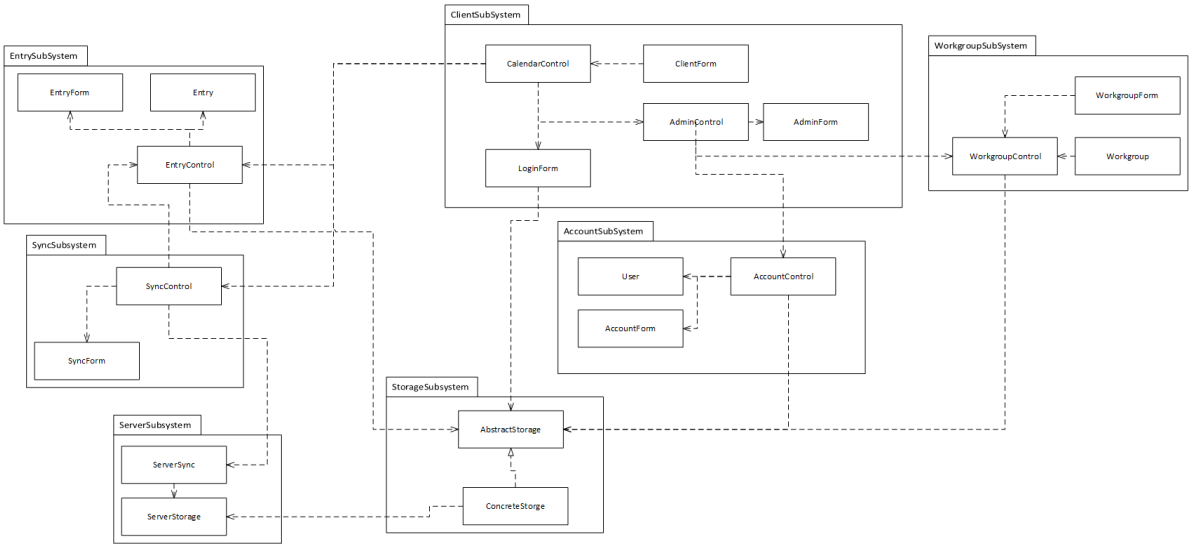

Figure 2: Subsystems and their decomposition

2.3 Hardware/software mapping

The figure shows the mapping of subsystems to hardware. All of the subsystems except ServerSubsystem are part of the client part of the system, meaning it runs on the computer a User is using. The StorageSubsystem runs on the client, communicating with the ServerSubsystem and in cases when the client is offline, uses the local cashed version of the calendar. The ServerSubsystem runs on the centralised server, allowing all clients to use it's functionality.

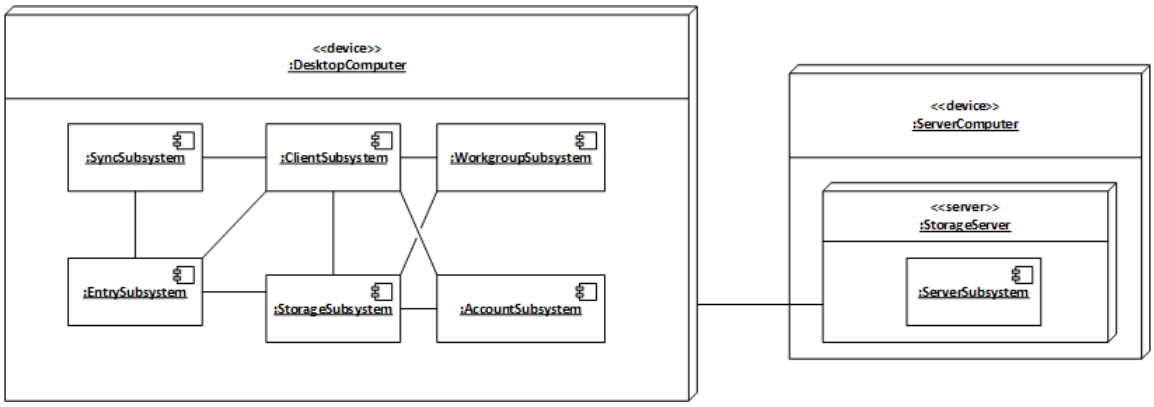

Figure 3: Hardware/software mapping

2.4 Persistent data management

Identifying persistent objects
The calendar system works with four entities, which need to be stored: The administrator and user entities, which need to be persistent, so accounts doesn't have to be made every time the system starts again. The workgroups also have to be saved since they would be useless unless persistent. Since it's a calendar, the entries also need to be persistent, else it wouldn't be a calendar, which is used to write down and remember events.

Selecting a storage strategy
A mixed strategy has been chosen for the system, between relational database and flat file. The relational database will save all persistent information, which is then available online, while the flat file will save all relevant information to the specific user locally. This makes the system able to function off- and online. The relational database will be updated when online while also updating the local file.
The figure below shows a RDBMS of our entities and their relation, showing how a database should store the data.

Figure 4: OO to RDBMS mapping

2.5 Access control and security

There are only two kinds of actors that will be using the Calendar System: User and Administrator. Below is the access matrix, which specifies which parts of the Calendar System the two different actors will have access to. The User is able to manage entries and manage which other calendars to sync their calendar with. A User is only able to add Users and Workgroup to an Entry, but is not able to change them. The Administrator can manage Users and Workgroups, but doesn't have anything to do with Entries and Sync. The Server syncs a Users calendar with another calendar server, and as a part of that it is able to manage Entries.

2.6 Global software control

The server should be using threads as to allow multiple clients connected, updating their calendar at the same time. While the local client should be

Objects Actors	User	Workgroup	Entry	Sync
User	addUserToEntry	addWorkgroupToEntry	manageEntry	manageSync
Administrator	manageUser	manageWorkgroup		

Figure 5: Access matrix for the Calendar System

event-based, since it's simpler than threads and there is no reason for the clients to use threads.
The figure below shows the subsystem decomposition with services:

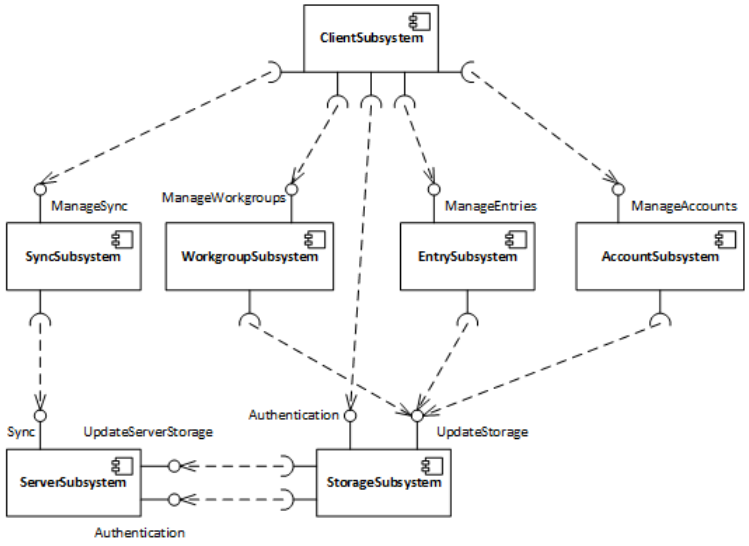

Figure 6: Subsystem services

2.7 Boundary conditions

Configuration use cases:

InstallClient	An administrator creates the calendar server, allowing the creation of new administrators and users, who can make use of the systems functions.

Start-up and shutdown use cases:

StartCalendarServer	An administrators starts the calendar server. As soon as the server is up and running, users will be able to synchronize their local calendar with the servers.

ShutDownCalendarServer	An administrator stops the calendar server. Any changes made to entries, by users will instead be saved in the local storage and uploaded when the server becomes available again.

Exception use cases:
The calendar system can experience two major classes of system failures

- A network failure between the client and the server.
- A failure causing the calendar server to unexpectedly terminate.