Calendar System

Calendar system for enterprise.

Inception

Vision

The purpose of the system is to allow users and managers to handle both events and tasks in a medium-sized company.

The calendar needs to encompass basic calendar functionality, including the following:

- Each event shall be repeatable, meaning a specific event shall occur each week, month etc.
- Each employee has a user.
- Each event shall be tied to a group or a single user.
- Each event shall have the option to have one or more categories attached, ex. Project X, Meeting.
- The system consists of three kinds of users; User, Group Manager and General Manager.
- Each group has a Group Manager, who can add users to or remove users from the group. General managers can also add users to or remove users from any group.
- A User can add/edit events for himself, a Group Manager can add/edit events for the group, and a General Manager can add/edit events for anyone.

The system needs to use an existing employee database. The employee database includes functionality to add or edit employees as well as current positions within the company.

Actors

User:

- UC.1a Create event
- UC.1b Edit event
- UC.1c Delete event
- UC.1d Invite user to event
- UC.1e Accept invitation to event

Group Manager:

- UC.2a Edit managed group

General Manager:

- UC.3a Create group
- UC.3b Edit any group
- UC.3c Delete any group

Use Cases

User

UC.1a Create event: A User needs to add an event to his calendar (e.g. to mark his absence from the office or occupation with a certain task). He uses the system to create an event, which will be associated with himself. He adds name, location, time, and tags to the event.

UC.1b Edit event: A User uses the system to alter an existing event (e.g. to change the timespan, location, or nature of the event).

UC.1c Delete event: A User uses the system remove an existing event (e.g. to cancel an appointment).

UC.1d Invite user to event: A User needs to include a co-worker in an event. He uses the system to invite the co-worker and awaits acceptance from the invited User.

UC.1e Accept invitation to event: A User uses the system to accept an invitation to an event.

Group Manager

UC.2a Edit managed group: A Group Manager needs to rename, add users to, or remove users from the group. He uses the system to rename, add

users to, or remove users from the group.

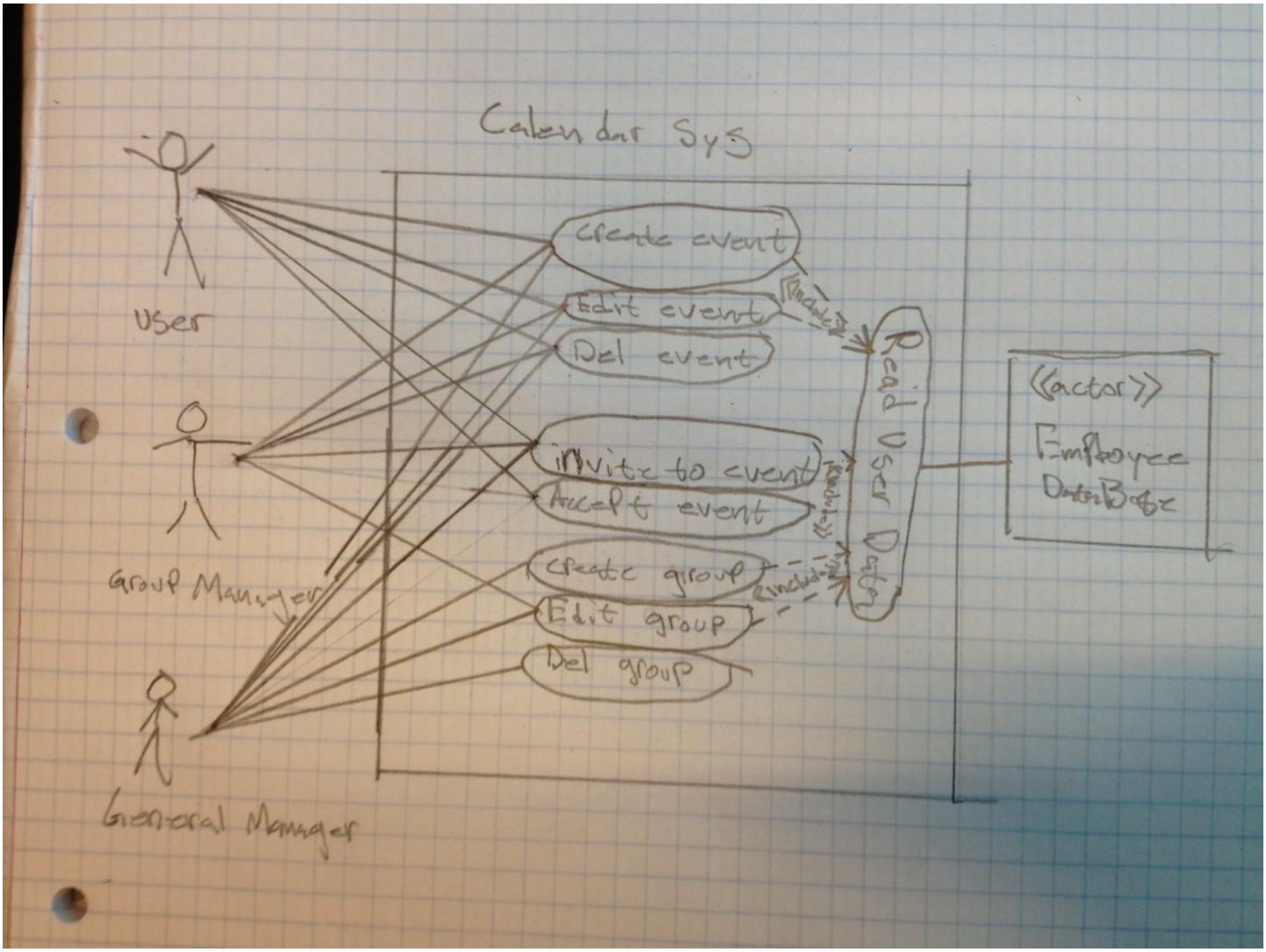
General Manager

UC.3a Create group: A group needs to be formed and a General Manager creates a new group. The General Manager uses the system to add an unknown number of users to the group and chooses one of these as a Group Manager. The system then sends a notification to each chosen user about their addition to the new group.

UC.3b Edit any group: As in UC.2a, but not limited to a managed group.

UC.3c Delete any group: A General Manager uses the system to delete a group.

Use Case Diagrams (fig. 1)



Glossary

User: Either a ordinary user or a general manager. A User exists in at least one Group and this Group represents the User throughout the system. **Group:** A collection of one or more users under one name. **Event:** An event in time associated with at least one Group. **Calendar:** A collection of Events concerning a specific User or Group.

Supplementary Requirements

Usability

Employee needs to be able to handle any task in the system, except addition of many users, in at most 2 minutes. This requirement needs to be fulfilled after at most 2 hours of training.

Reliability

The system must deliver 99% up-time during work hours. If a system failure occurs, the system must be up and running at the end of the following work day.

Performance

The system is to be located on-site and response time must therefore be instant. The exception is handling of many users, where a few seconds delay must be acceptable.

Supportability

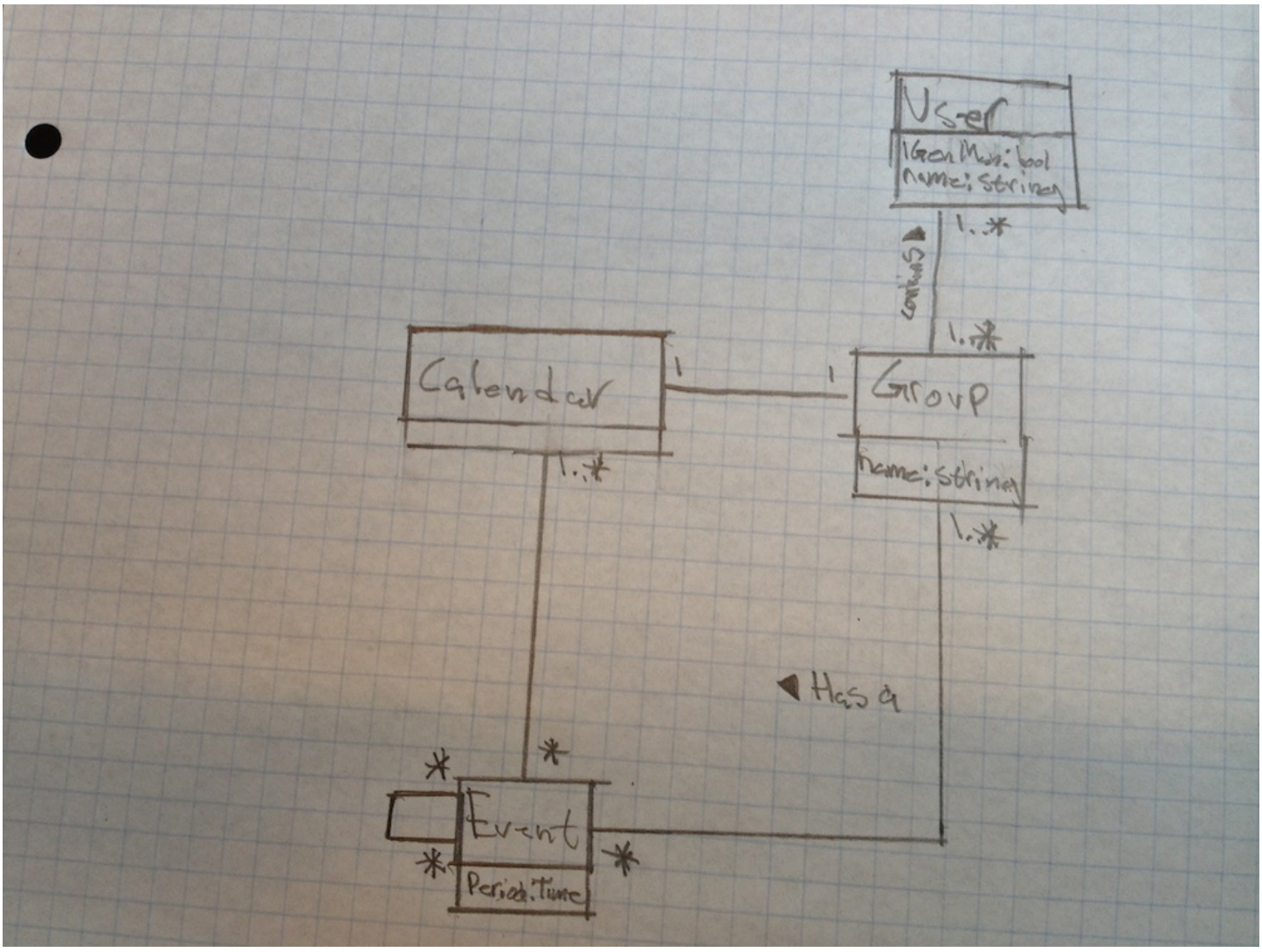
The system must be supported by a web interface, which also should be accessible from mobile devices, including iPhone 4S.

Other - Legal

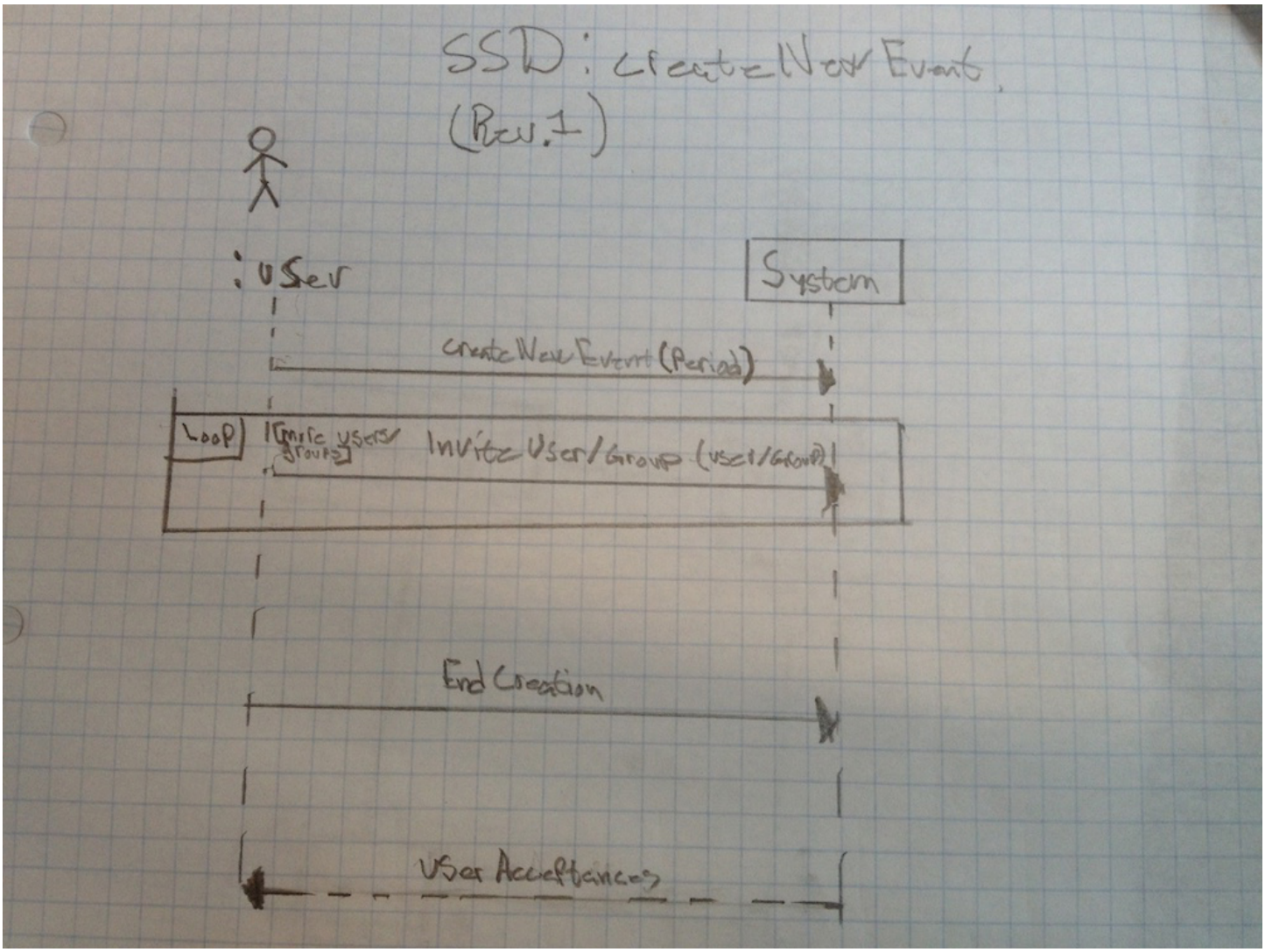
All sensitive personal data must be encrypted and only be accessible by General Managers.

Elaboration

Domain Model (fig. 2)



System Sequence Diagram (fig. 3)



Operations Contracts

createNewEvent OC1

Operation: createNewEvent(Period: Time).

Cross References: UC.1a Create Event (use case). SSD (rev. 1) Create New Event (system sequence diagram).

Preconditions: *None*

Postconditions:

- A Event instance *event* was created (instance creation).
- Conflicting events will only be created with the "caller"'s acceptance. Without acceptance the operation will leave no trace.

inviteUser/Group OC2

Operation: inviteUser/Group(User/Group).

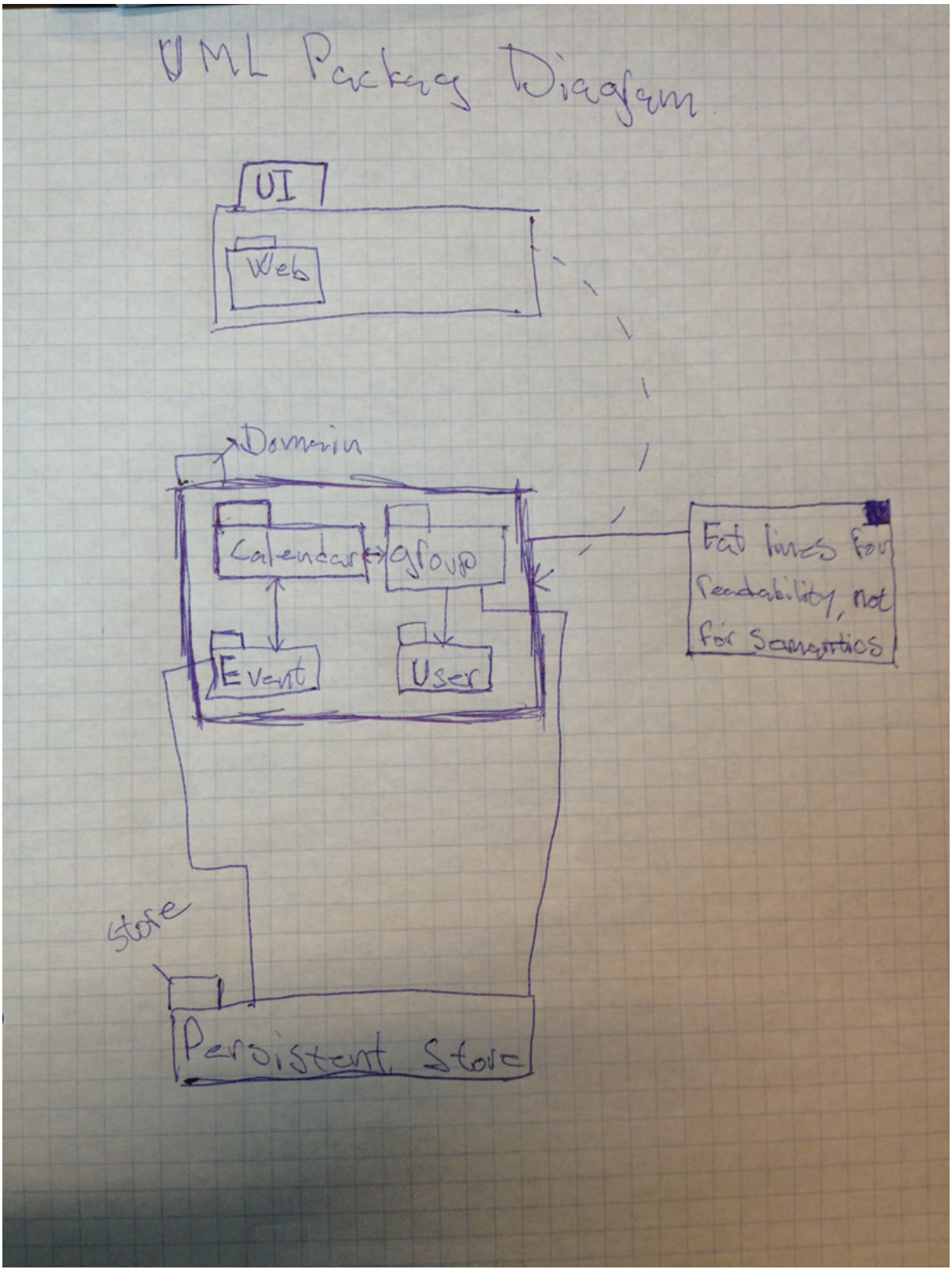
Cross References: UC.1d Invite user to event. SSD (rev. 1) Create New Event (system sequence diagram).

Preconditions: Event creation or editing underway.

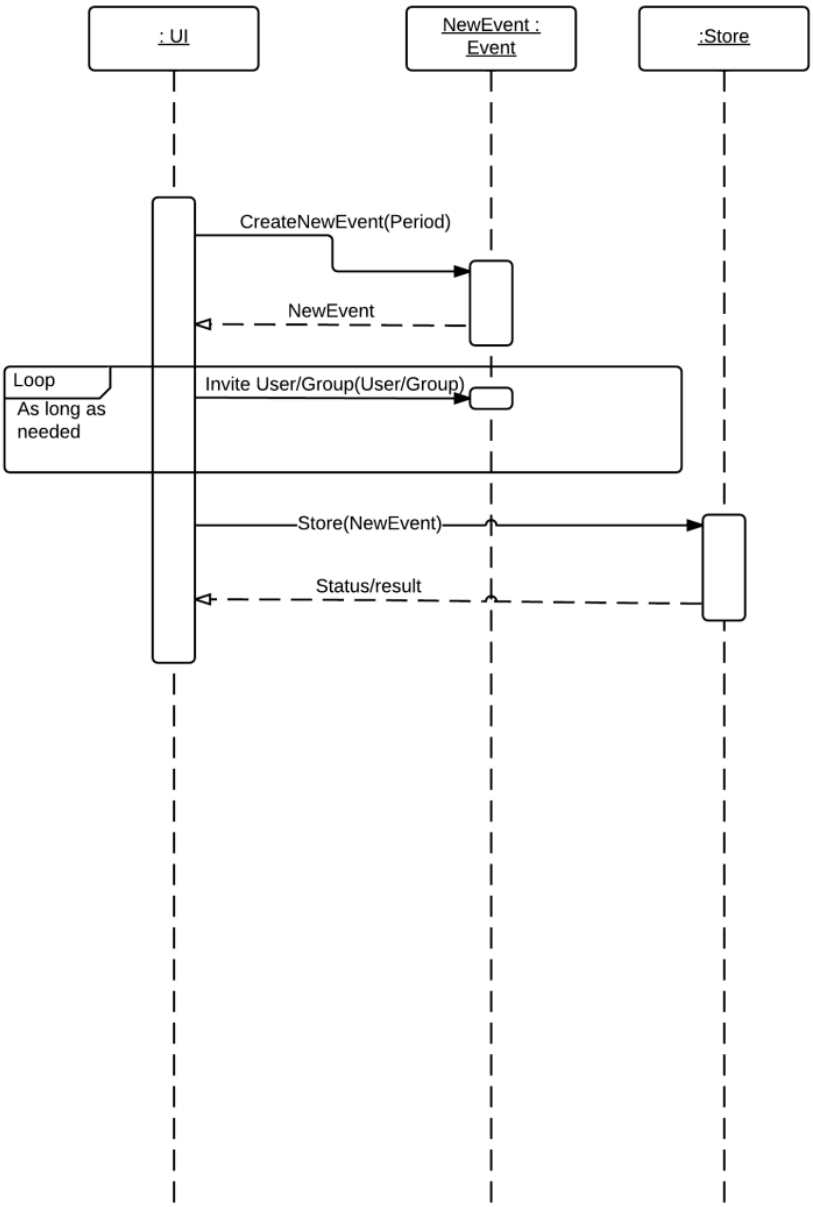
Postconditions:

- The specified users/groups have been associated with the event, or have been notified and the system is awaiting answers (this happens after the operation).

Package Diagram (fig. 4)



Interaction diagram (fig. 5)



(Lucidchart.com)

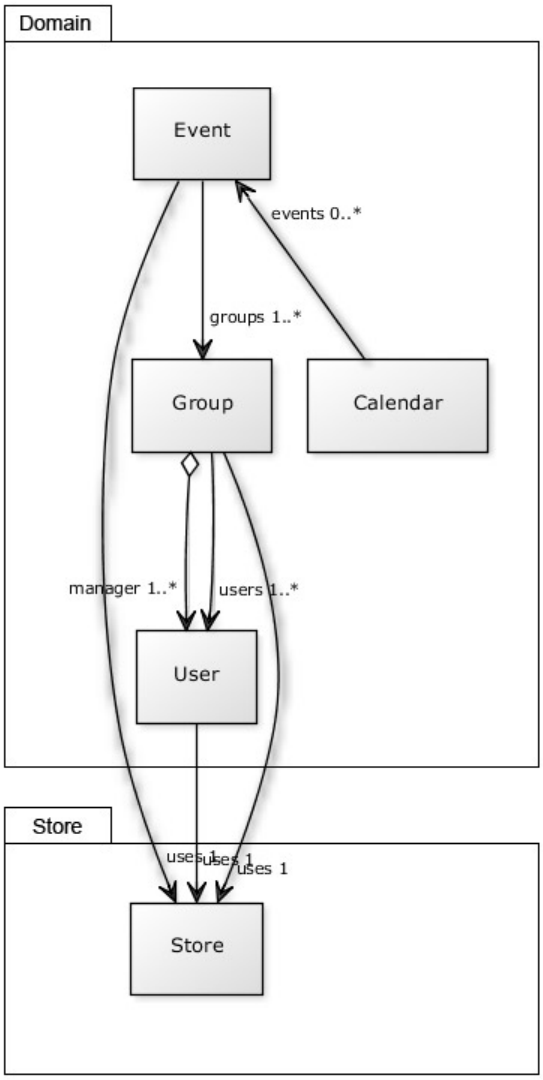
Architectural Discussion

Our system currently has no security, as we have no authentication or any sort of user login. Some sort of authentication interface would be needed to provide a multi-user system.

Our Store is a single point of failure, which could be avoided by providing several Stores, which in turn would probably lower performance and increase the complexity of the system overall.

We have chosen to always have at least one Group for any User. This is our choice instead of creating a common interface for User and Group and using a list of this interface in Events.

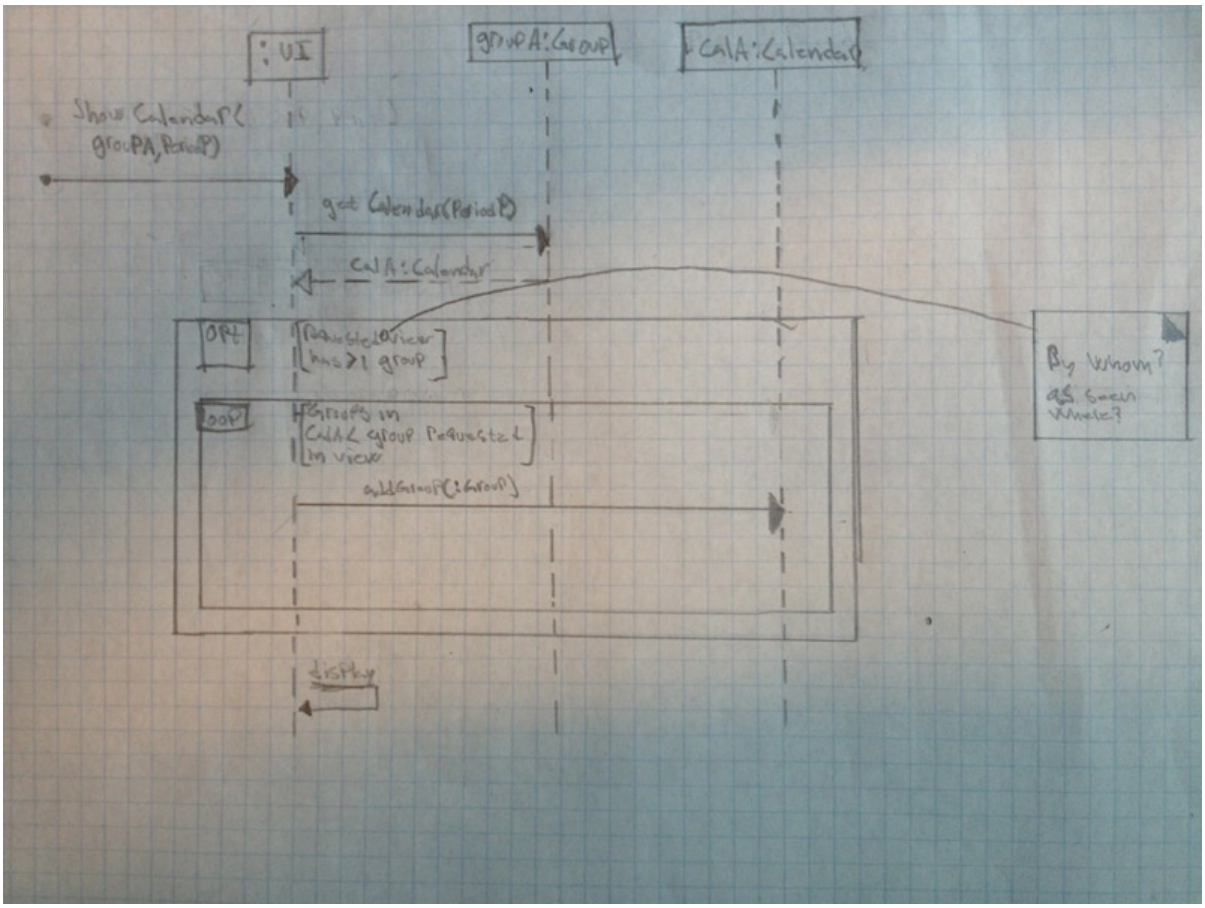
UML Package Diagram (fig. 6)



(yuml.com)

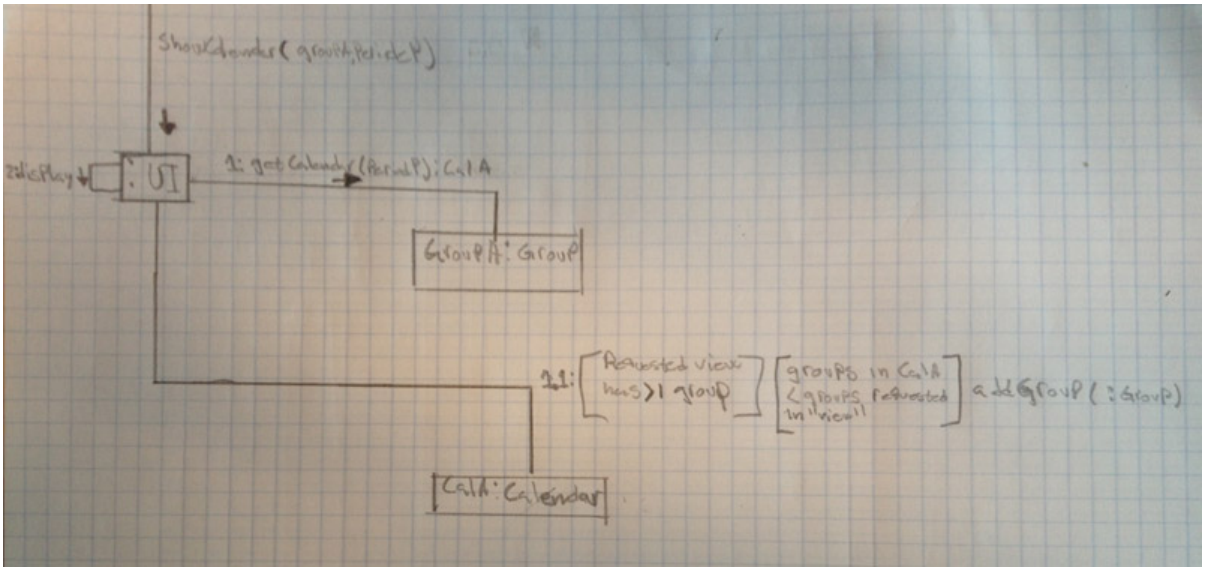
Interaction Diagrams

Sequence Diagram (fig. 7)



(Pencil, bitches!)

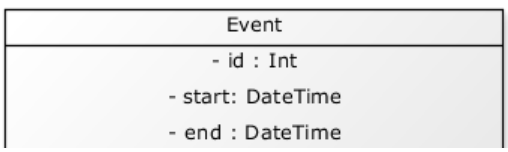
Communication Diagram (fig. 8)

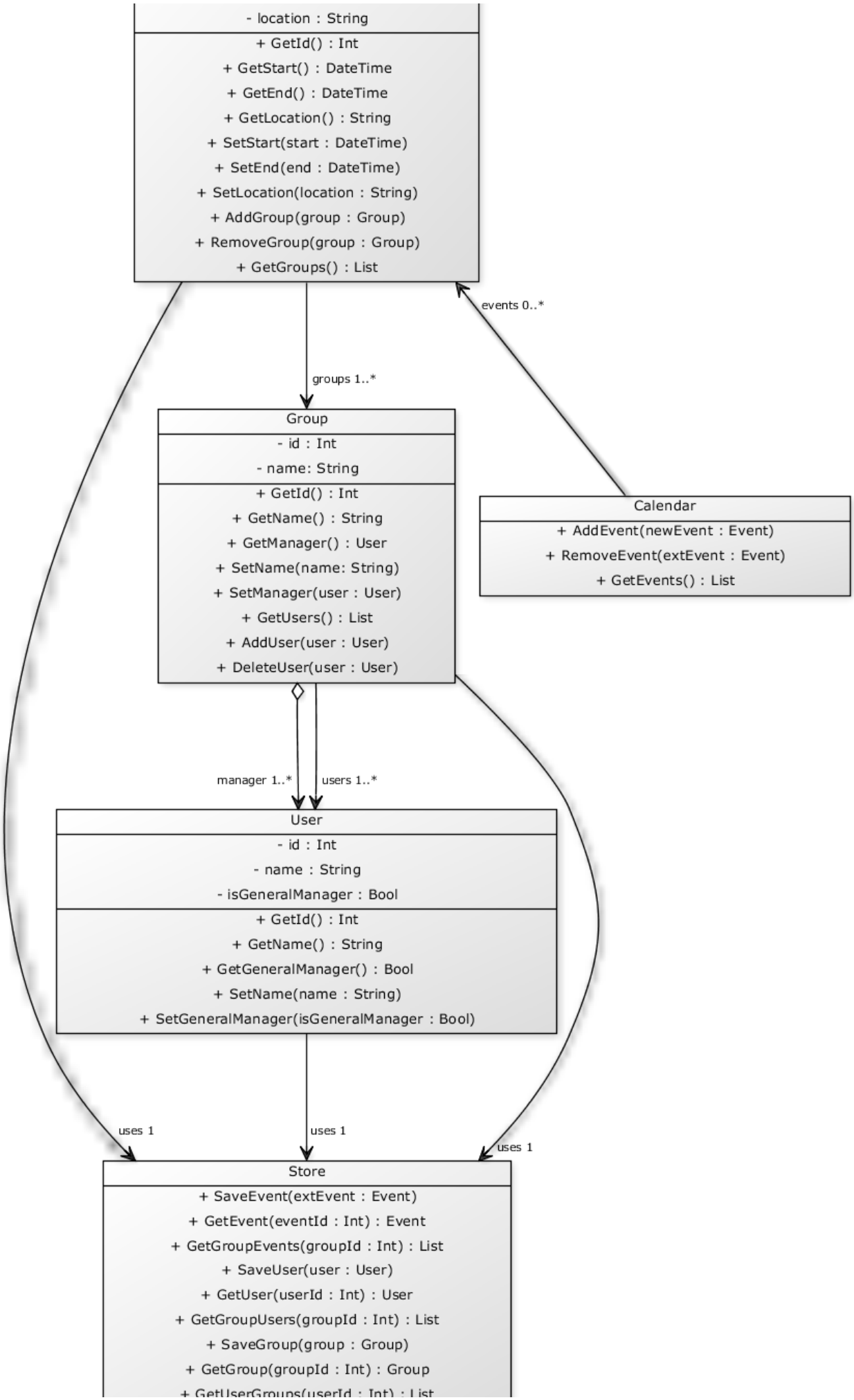


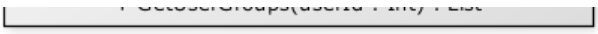
(Pencil, bitches!)

Reflection on the advantages and disadvantages of respectively sequence and communication diagrams: Sequence diagrams have a more rich notation, which is reflected by the rather ugly loop-inside-a-conditional syntax used with message 1.1. Although it seems like most of the missing notations could rather easily be adapted directly as they are for sequence diagrams. Communication diagrams are easily extended and rewritten, as they do not have the strict "left to right" structure of sequence diagrams. In turn, that might make them harder to assess. In our opinion, communication diagrams can be drawn much quicker than sequence diagrams. That might sound like the argument of a lazy person, but it might make the difference between making/updating a diagram, or just abandoning the documentation.

Design Class Diagram (fig. 9)




- location : String
+ GetId() : Int
+ GetStart() : DateTime
+ GetEnd() : DateTime
+ GetLocation() : String
+ SetStart(start : DateTime)
+ SetEnd(end : DateTime)
+ SetLocation(location : String)
+ AddGroup(group : Group)
+ RemoveGroup(group : Group)
+ GetGroups() : List
events 0..*
groups 1..*
Group
- id : Int
- name: String
+ GetId() : Int
+ GetName() : String
+ GetManager() : User
+ SetName(name: String)
+ SetManager(user : User)
+ GetUsers() : List
+ AddUser(user : User)
+ DeleteUser(user : User)
Calendar
+ AddEvent(newEvent : Event)
+ RemoveEvent(extEvent : Event)
+ GetEvents() : List
manager 1..*
users 1..*
User
- id : Int
- name : String
- isGeneralManager : Bool
+ GetId() : Int
+ GetName() : String
+ GetGeneralManager() : Bool
+ SetName(name : String)
+ SetGeneralManager(isGeneralManager : Bool)
uses 1
uses 1
uses 1
Store
+ SaveEvent(extEvent : Event)
+ GetEvent(eventId : Int) : Event
+ GetGroupEvents(groupId : Int) : List
+ SaveUser(user : User)
+ GetUser(userId : Int) : User
+ GetGroupUsers(groupId : Int) : List
+ SaveGroup(group : Group)
+ GetGroup(groupId : Int) : Group



(yuml.com)