Muhammad Faisal Muhammad email: mfaisal@aucegypt.edu

Groups Management: Package Implementation Details

github.com/dedis/student_19_groupmgmt

Members in large societies have various interests and beliefs that hence generates tendency for members to get closer with other members with similar interests and beliefs. Moreover, such tendency to stay with members with similar interests comes from various reasons such as maybe trying to organize activities with common interests or trying to call other to share their beliefs or interests. As more people with same interests start to get closer, new groups start to emerge. Groups can then take many forms such as political parties where its members seek to issue laws that seems best according to their agenda. It can also take the form of an NGO, startup, or similar groups where some society members wants to do something collectively. In this document, we discuss what some useful groups features and how to implement them.

1 Group formation and management

The need to form groups results from the need of some society members to work together on some tasks and therefore some problems would arise such as how to organize these tasks and how to distribute the work with each other.

Group A Group is an organized set of society members working together for some common interests and goals.

1.1 Separation of roles

A first approach where all people would meet together to discuss the important points and then decide how to proceed with their plans may seem impractical specially in large groups. For example, not all members would have the same free time slots and there is no need for a member to go through every single detail themselves. Moreover, in order for a group to execute their task successfully and efficiently, they better distribute roles for each group of members according to that group expertise and how much they would fit for such role. Moreover, although some roles can be done by many members correctly without special skill set or special expertise such as routine work that just needs some minor human work or also verifying that someone is eligible to become a group member, other tasks can be much more complicated. For example, The task of managing financial assets such as buildings owned by the group or deciding how to put a budget for the group's various activities needs someone with special expertise in finance for example.

Therefore, it may seem more convenient if similar activities can be grouped together to be controlled from some subset of the group members who may seem more suitable for such task.

Role A role is a subset of a group members who authorized to manage some group activities because of their common skill set or expertise.

1.2 Membership management

Each group should be able to manage their members whether by members addition or removal to ensure that all the current members in the group are useful and valuable working members. Such as managing the group financial assets, managing the group members and the membership of members in each role is an important activities that should have its rules and conditions clear, transparent, and managed by a role that have reliable group members.

1.3 Decision making

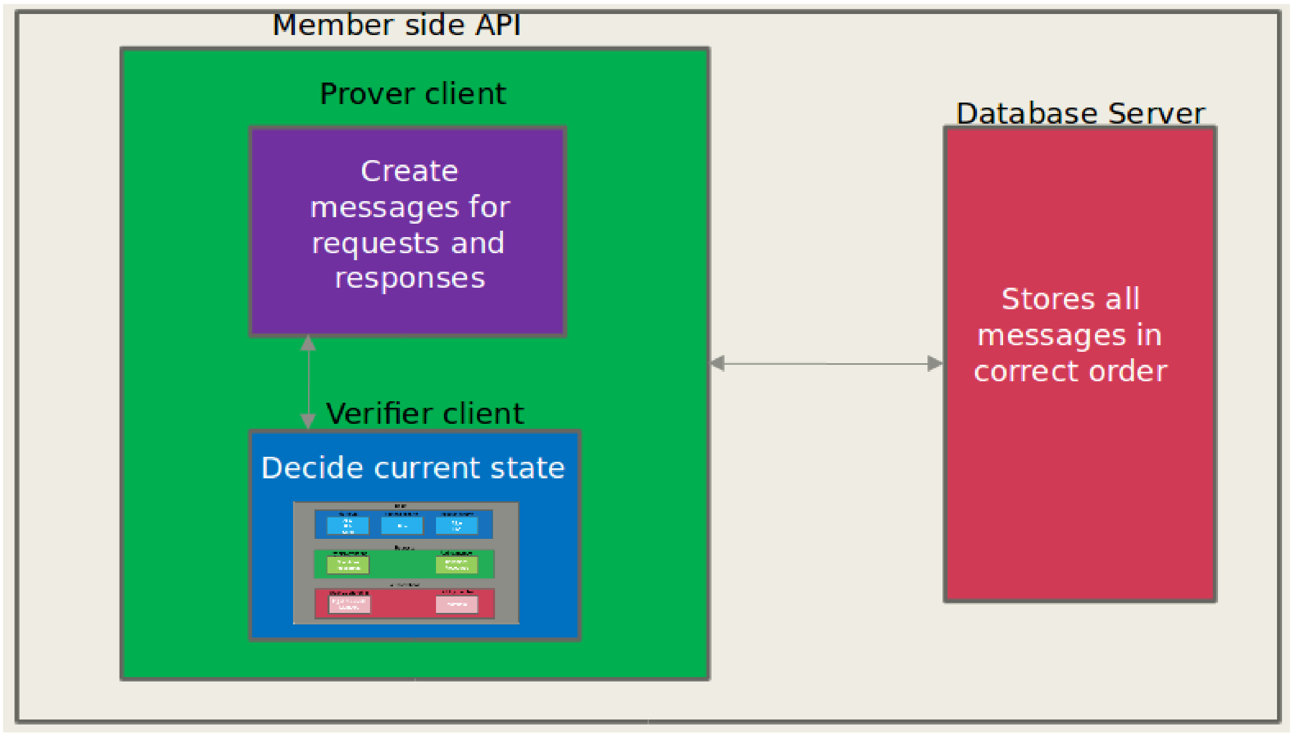
Although as said before some activities would require more expertise than others, every decision taken in a group must be a result from some collective members' action to ensure minimum malicious decisions and also to avoid potential coercion. A simple way to do this is to ensure that every decision is approved from the majority of members who are managing such type of decisions. In other words, if there are two roles who are managing decisions related selling some building, then to proceed with selling such building, it is a must to get approval from more than half the members in these two roles.

1.4 Rules Flexibility

A group management system should be able to count for the changes that may result in the policies or rules controlling the group. In other words, it should be working towards making the target of the system to provide a set of rules to control decisions made through the system, yet provide a way to dits such rules.

2 Design and Implementation

A system has been designed to count for the previously mentioned points. This package implements the client side of this system and will be described in details in the next subsection. The other part of the system, the trusted storage has been implemented as a shared object in the memory. However, for the system to be fully functional and be deployed for public use, that simulated storage should be turned into a passive trusted storage responsible for storing all received messages with the correct timestamp. Moreover, the package for the client side is written in go and will be discussed in the following subsections.


Member side API
Prover client
Create messages for requests and responses
Verifier client
Decide current state
Database Server
Stores all messages in correct order

Overall structure of the system.

2.1 Client side API

This can be considered the upper layer of the system and consists from two parts, the prover client and the verifier client. Each user must have a pair of keys and their public keys stored the in the trusted storage. The pair of keys is stored through the prover class which is called the User class. This class has the main functions to be able to generate messages that can be related to the user to issue some request or reply to such a request according to the need as it will be explained later.

The main structure of the User class is as follows:

```
type User struct {
    UserInfo
    GroupMemberInfos map[string]GroupMemberInfo
}

type UserInfo struct {
    UserID     string
    PublicKey  interface{}
    PrivateKey interface{}
}

type GroupMemberInfo struct {
    PendingRequests []Message
    SignRequest     func(request Request, userInfo UserInfo) interface{}
    SignResponse    func(response Response, userInfo UserInfo, userIDs []string)
}
```

It mainly has two main functions in order to communicate with the trusted storage and send verifiable messages;

```
func (user *User) CreateRequest
    (groupName string, requestName string, args ...interface{})
    RequestMessage

func (user *User) CreateResponseMessage
    (groupName string, requestNumber int, answer bool)
    ResponseMessage
```

Moreover, the client side has another main part which is working as a verifier. It is important that each group member be able to verify all the messages done by the other members to know the current state of the group. The verifier class is called the Group class.

The group class has the following structure and can be considered just a layer to verify the validity and signer of each message in the system.

```
type Group struct {
    BasicGroup

    Messages        map[string]Message
    AllMessages     []interface{}
    PendingRequests map[string]map[string]EmptyStruct
}
```

Each message related to the group must be added in the correct order after retrieval from the trusted storage to the group through the following two functions. If a message passes this layer successfully, it will be echoed to the lower layer called the BasicGroup class to apply the message effect.

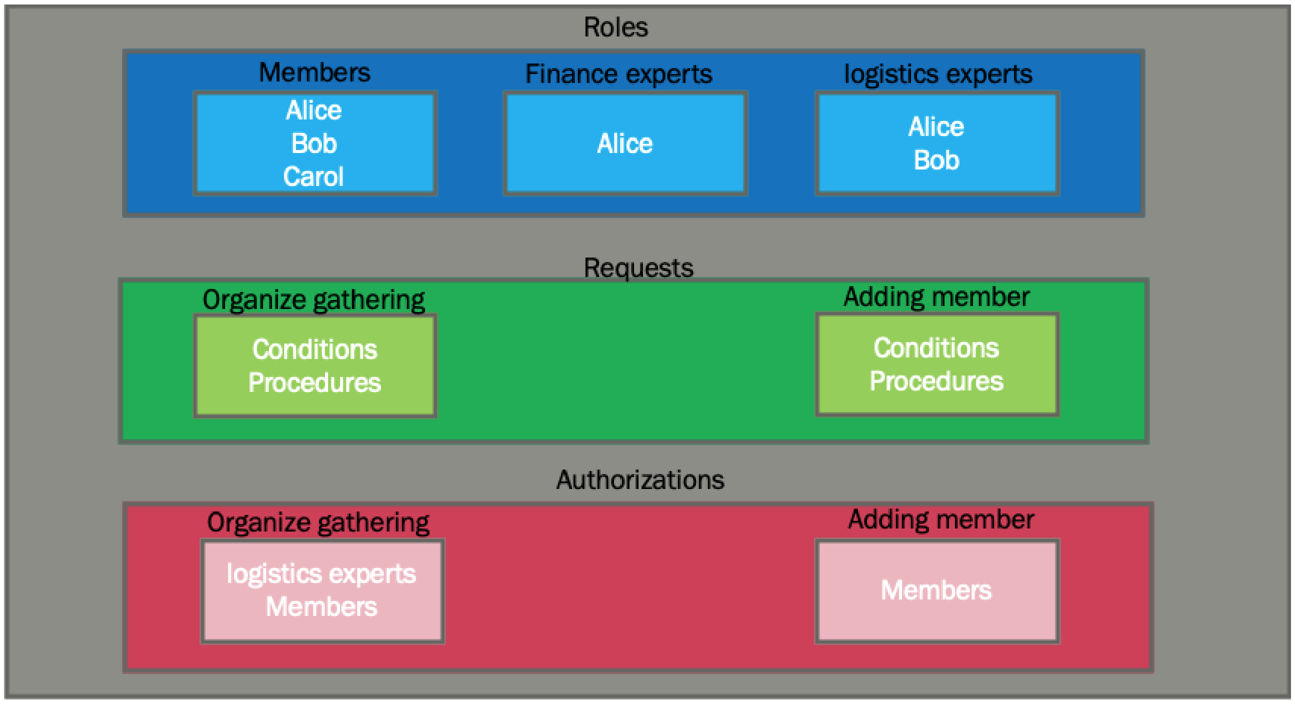
```
func (group *Group) AddRequest(requestMessage RequestMessage) RequestStatus
func (group *Group) AddResponse(responseMessage ResponseMessage) MessageStatus
```

2.2 BasicGroup Structure

The basic group class or structure is the class responsible to apply changes to the group structure. In other words, this is the class that has the functionalities to add members, remove members or such things of group features. However, this functions should not be called directly. However, they will be called a correct message was received that request the call of such functions. This means that the verifier part of the client side is first a layer that decides whether a change should happen or not and then another layer and implements the logic of how a change happens.

Please, note the following definitions that will be also explained in detail later.

1. **User:** Any one with access to the system.
2. **Member:** User that has a role in the group.
3. **Role:** A subset of the group members which gets authorization to handle specific types of requests.
4. **Request:** a question accompanied with some data to be passed to the members of some specific roles to get a yes/no answer.
5. **Delegated member:** Member belonging to a role that can handle the mentioned type of requests.



An example of a group structure

The proper way to do a change is through registering a request type and specifying which function to be called in case such request is a success while allowing some parameters to be passed that to be called function. So first a request type is created, then a certain role is given the privilege to reply these requests. This can be done using the following function.

```
func (group *BasicGroup) AddRequestType(requestType string, managingRoles []stri
    successFunction func(args ...interface{}) interface{},
    verifyRequest func(request Request, signature interface{}) RequestStatus,
    verifyResponse func(responseMessage ResponseMessage, previousResponses map[s
    (MessageStatus, map[string]ResponseMessage))
```

There are more functions to ensure more easiness dealing the system. For example adding roles, adding members in roles, merging roles and such possible manipulation functions for roles, request, and aut

3 Blockchain instead of database

In order to decentralize the system's database, a blockchain can be used then to store both the requests and responses.

However, this requires the following:

1. The order, in which transactions are chosen to be included in the block, should be modified such that responses should be included before including new requests.
2. If a role's members are controlling the addition of a another role's member; Responses or requests from the controlling roles should be added to the block before those of the controlled roles.
3. Not keeping these constraints would for example allow that members pending removal be somehow still able to answer requests.

4 Future plan

1. Finding a way for delegated members to change a malicious response if any happens.
2. Researching on how an election process can happen as a way of choosing delegated members (can we for example use delegated members' previous decisions to decide on the future of their continuity in their role through some reputations' system).