SWIM v2

Requirements Analysis
and
Specification Document

Authors:

Affetti Lorenzo

Canidio Andrea

November 27th 2012

Summary

1. Introduction

1.1 Description of the given problem

We will project and implement SWIMv2 (Small World hypothesIs Machine v2, later called simply SWIM), which is a kind of social network with the aim to put through people and help the user to find the right person for the right job.

The system that will be developed has to allow the registration of a new user with all his personal information (like name, age, abilities and experiences to form a kind of CV that have to be stored), his log in, the possibility to request a friendship to another user and the possibility to search through the database for a desired user and view his personal information to understand if he can resolve his specific problem.

There is also an administrator that creates an initial set of available abilities and that can accept new ad hoc abilities added by the users.

1.2 Goals

If we think about possible users, we think that SWIM has to provide this main features:

- Registration of a person to the system;
- Searching for help and giving help;
- Creating a special bound with some users;
- The possibility to choose the best person for each job (once found help);
- Contacting another person;
- Sharing personal information (e.g. contacts, abilities).

1.3 Domain Properties

We suppose that these conditions hold in the analyzed world:

- When a person asks for help, a lot of people can answer to him, but only one will really help him;
- A person can evaluate only a received performance;
- A person requests help only if he really needs it;
- If a person evaluates someone's work, he is impartial;
- If a person declares an ability, he really has it (or, at least, he thinks so) and in general a person declares only true and correct personal information about himself;
- A person can help someone only if he has the ability needed;
- A person always has at least one ability.

1.4 Glossary

First of all we have to define some words that will be used in our documents.

- **Ability**: generically, something that somebody is able to do. With this we mean that, if, for instance, I state that I am a Math Teacher, then I am supposed to at least being graduate at the high school. It is not enough to know how to do sums. This mechanism will be automatically implemented by the possibility to give a feedback on someone's work (see below).
- **Friendship**: adding a user to your friends list, here, is like putting him into favorites. In fact you can look for a help filtering your research with your friends list and also send or receive message from him.
- **Feedback**: a mark on someone's work. After someone finishes his work, I can give him a feedback: if I am satisfied the feedback will be high, otherwise not.
  We think that the basic idea is that the worker has to take negative feedbacks

as a way to improve his work and not as useless critics. In the same way, it is not worth to declare false abilities, because you will never be taken into consideration by anyone: a person that gives a hundred performance as a cooker and has a very low feedback, could appear in the bottom of the list of cookers and will lose a lot of work possibilities.

- **User:** for user we mean a person already registered in the system, so that has a profile, asks for friendship to another user and can use all the functionalities described below (see Functional Requirements).

- **Guest:** a guest is a person that probably for the first time accesses the system or that hasn't already signed up. Guest has less power in the system than a user, his functionalities are reduced and can only search for users and, obviously, he can sign up.

- **Administrator:** the administrator of the system is the person allowed to manage requests of new abilities by users.

- **Help request**: It's a public or private request made by an user that is looking for someone to do a job. It can be accepted by another user or stay pending. Note that only users can publish help requests and accept them.

- **Performance**: It is the work carried out by an user A that answers to a help request posted by an user B, in this case we will say that user B fulfills the performance asked by user A.

- **Wall**: a notice-board, an area where users can post their help requests and where other users can reply.

- **Post:** as a verb, the action to add another help request (or to reply to) to the wall. As a noun, everything that is written on the wall.

1.5 Assumptions

There are few points that are not very clear in the specification document, so we will have to assume some facts. We assume that:

- There is a well-defined set of abilities that the social network can offer. This set is initialized by the administrator of the system. When a user wants to add an ability which is not in the initial set, the system sends a request that can be accepted or denied by the administrator himself.

- As for us it wasn't so clear if the system should be only a place where an user can access to people's contacts (e-mails, phone, etc.) or it should be similar to social networks we use every day (so with instruments that provides you the possibility to share information through the world) we define to ways to obtain help from users:
    - An user or guest that needs help can view the contacts of people showing a determinate capability and contact them using only their personal information (so not using system features).
    - *An user can post his help request into the system, in this way his request can be visible to all users.*
    - *An user can contact another user directly using a service of messaging (provided by the system itself).*

It is important to say that the points in italic are advanced functionalities that we would like to implement to have a more complete system, but, before, we want to carry out all the basic set of functionalities in order to have a complete product.

Just to be ready for a possible implementation in the project, we'll carry on these functionalities in all the project's phases.

We hope to have enough time to reach this goal.

1.6   Proposed system

We propose a web platform, that will give to linked people the services described below.

Users will be able to access the wall and post help requests on it. This way it will be easy to control all the help requests posted through the world or by friends and even answer to them. They will also be able to modify their profile information (add abilities etc.).

The administrator will be given a special module to manage new ability requests and so on.

Server will generate different user experiences and consequentially different pages basing on the actor that's using the platform.

1.7   Identifying Stakeholders

Our "financial" stakeholder is the professor who gave us this didactical project. Our professor needs are to have, within the end of the semester, a product that at least works and we think the main thing that interests the professor is to show that we have understood the development process in all its parts and that we can carry out a project from the beginning to the end passing from all its development phases.

So we want to show that we can identify requirements and specifications, design our web application, implement it and then test it, providing all the documents that developers use in developing real software.

It still remains that we have to focus on some functionalities that the professor gave us to have an application that works.

We think that our typical user will be an occasional one, because our platform is intended to work this way. Once a user finds help it is very probable that he will

not use the platform until he needs help another time (and this doesn't happen every day).

1.8 Other considerations about the system

We want to add these other considerations because they are about our thoughts about how SWIM should be once terminated.

So we think SWIM has to be:

- Easy to use: we will try to "make it simple", in the sense that we will try to offer a large set of functionalities, but in the way to let a "normal user", for instance, not to spend too much time to post on the wall, or find a user near his place.
- Nearly stable: it has to grant some basic functions in a stable way. For example there has not to be any fault in registering a new user.
- Have a nice look and feel: we will try to make a well designed application in the sense that we hope it will be nice to see and will fit to any user.

2. Actors Identifying

The actors of our informative system are basically three:

- **Guest**: a person that has not registered and can only exploit basic functionalities such as looking for help.
- **User**: a person that has registered and so has provided his personal information and a set of abilities.
- **Administrator**: the responsible for the web application. The only one that can accept new abilities from users.

3. Requirements

Thinking that the domain properties, written in paragraph 1.3, holds, from the goals, written in paragraph 1.2, we can derive our requirements.

We write below, for each goal, what we can derive:

1. **Registration of a person to the system:**
   - The system has to provide a sign up functionality.
2. **Searching for help and giving help:**
   - The system will made possible to search for users by ability;
   - The system will allow users to share help requests;
   - The system will allow to see help requests and to post them on the wall, and will give the possibility to reply to each of them.
3. **Creating a special bound with some users:**
   - The system will allow users to be friends (in the system);
   - The system will highlight friendship information.

4. **The possibility to choose the best person for each job (once found help):**
   - The system will provide a feedback mechanism, in this way everyone will be able to choose a user starting from his reliability;
   - The system will offer the possibility to choose a reply to a help request.
5. **Contacting another person:**
   - The system will made possible to users to find, in another profile, some mandatory profile information (e.g. name, surname, at least one ability);
   - The system will give a service of messaging between friends.
6. **Sharing personal information (e.g. contacts, abilities):**
   - The system will allow users to browse profiles of other users.

3.1 Functional Requirements

Once found the main features that SWIM has to give, we can find some functional requirements concerning each defined actor:

- **Guest**: he can only exploit basic functionalities, so he can only:
  - Search for users of the system and view their profiles;
  - Sign up.
- **User**: he can:
  - Log in;
  - Modify his profile information;
  - Search for users of the system and view their profiles;
  - Send a friendship request to a user or reply to a received friendship request;
  - Post a new help request or reply to a pending help request;
  - Provide a feedback on a received performance;
  - Ask the administrator to add a new ability to the system;

- o Send a message to another user.
- **Administrator**: he can:
  - o Log in;
  - o Answer to requests to add new abilities from all users;
  - o Add new sets of abilities.

3.2 Non Functional Requirements

3.2.1 User Interface

The interface of our application it is thought to be used via web. It will be better if it can support also a mobile version (we don't know if we'll implement this solution) to reach the most part of the users in every way.

It has to provide an unified log in page that provides each user a link to its profile page, allowing only to access the functionalities the user can use. Also it has to provide all the functionalities provided by the platform in the first page after the log in, so the usage of the platform can be quicker and easier.

We want that a user that for the first time connects to the platform can understand in a few time how the platform works, so we decided to inspire ourselves to other more popular social networks and to maintain a minimal user interface, in order not to overload the user with no useful details.

The pages will contain forms to be completed for the needed functionalities (like posting an help request) and also checks to the inputs, in order to always control the flow of the user experience.

Our platform will contain two main pages which we want to focus on in order to understand its usability. The first page is the profile page in which user's information will be displayed. We can see below a first sketch of the page.

The second page is the help request page. This page will be available once a user posts a help request into the wall and then he browses his help requests. Both the creator of the help request and other users will be able to display it. Obviously there are some little differences because, for the creator, there will be the commands to choose who, in the set of the repliers, will do the performance; for another user, instead, there will be displayed the commands to reply to the help request.

In this sketch we show the view of a user that browses a help request posted by another user:

Home | Profile | Messages | Wall   Search for users   Logout

Title of the Request   Reply
Requested Ability
Date, Hour
Place
Description
Lorem ipsum dolor sit amet, maiores ornare ac fermentum, imperdiet ut vivamus a, nam lectus at nunc. Quam euismod sem, semper ut potenti pellentesque quisque. In eget sapien sed, sit duis vestibulum ultricies, placerat morbi amet vel, nullam in in lorem vel. In molestie elit dui dictum, praesent nascetur pulvinar sed, in dolor pede in aliquam, risus nec error quis pharetra. Eros metus quam augue suspendisse, metus rutrum risus erat in.

Who Applied
Sancho Panza
Lionel Messi
Alan Turing

3.2.2 Documentation

We will draft these documents to well-organize our work in the way to do in a fewer time the best work as possible:

- **Project Plan**: to define tasks and to show our organization and the timings of this process.

- **RASD**: Requirement Analysis and Specification Document, to well-understand the given problem and to analyze in a detailed way which are our goals and how to reach them defining requirements and specification.

- **DD**: Design Document, to define the real structure of our web application and its tiers.

- **JavaDoc comments in the source code**: to make anyone that wants to develop the platform or do maintenance on it understand the code.

- **Installation Manual**: a guide to install SWIM.

- **User Manual**: a guide to use SWIM.

- **Testing Document**: a report of our testing experience of another SWIM project.

3.2.3 Architectural considerations

We will use J2EE platform with a database in which all system's information will be stored.

What is this information will be clearer watching the Class Diagram given below, because it could be considered as a basis for our database, but the precise ER Diagram will be drawn in the DD.

An Internet connection is needed to use SWIM and also a recent web browser.

4. Specification

We write down here some specifications for SWIM that will state clearer how we can reach the goals listed above:

- The administrator can refuse to add a new ability, for example because it is indecent or rude, or because another one very similar already exists;
- An user can see the wall and posts from the all world;
- Only a user can post on the wall or reply to a post on it;
- Posts will be well structured. With this we mean that a user can't post anything he wants, but only a help request. We mean that the help request will have a defined structure that has to be followed to post on our platform (like a date and an hour, a detailed description, etc.);
- Every help request will have a particular state with some consequences:
    - Pending: the user that posted is waiting for an answer, anyone can reply to;
    - Closed: the user is receiving a performance from another one and it is not possible to answer anymore;
- A user can filter the wall in the way to see only posts posted by his friends or posts that requires, , for instance, a cooker or posts related to a particular city.
- A user can post a help request and then he can choose a reply as the one he thinks is the best. Then he can only give a feedback to this reply. We think that this is the best way to inform the user that posted the best reply that he is the chosen for that job;
- After a user has chosen the best reply, the corresponding help request will be considered closed and no one will be able to reply to it;
- Only users that are friends will be able to exploit the messaging functionality;

17

- Every help request will refer only to one ability.

5. Scenarios Identifying

Here are some possible scenarios of SWIM:

- Mark's piano is out of tune and he couldn't find anyone that could solve his problem in a week. So he opens his laptop thinking that he wants to find in the internet something that will help him in finding help. He search Google for "find piano tuners" and among the voices, he finds SWIM. So he clicks on it and he discovers this new web platform. In the home page of the platform there is a good explanation of what it does, so he decides to have a try. He searches for a piano tuner and a list of persons appears. He finds Bob in the list, who has a good feedback, and from his personal information he discovers him living quite near to him. He decides to contact him using his e-mail address that is available in his profile.

- Mark needs a catering service for his birthday feast. So he remembers that he used SWIM some weeks before to find help. Now it is the second time he uses it, so he decides to sign up in the system, so he gives a set of personal information and declares a new ability (marketing manager). Mark discovers that, now that he has signed up, he can exploit the wall, where there are many help requests made by other users of the platform. So he decides to post his own help request and starts creating it. As he needs a catering service he puts it in the request and chooses as day the next Sunday and, as place, his house (specifying his address). He puts also a description of the problem and the acceptable fees. In an hour there are several answers to his request and he notices Anna (viewing feedbacks), who seems very trustable and has an high feedback and chooses her reply. Mark is astonished by the difference of being registered or not, because, if you are registered, you can be helped by people that maybe you will never

contact by yourself. Mark now thinks that the registration to SWIM really adds a lot of value to the system.

- Mark suddenly remembers that his profile information is a bit poor. So he logs in to SWIM and goes to his profile page. He adds an avatar, a mobile phone number and his address in order to make easier to contact him. Then he wants to add the "Pianist" ability but when he adds it, right next to it, an exclamation mark appears, meaning that the ability doesn't exist in the system and needs to be approved by the administrator. After some time, the administrator, Andrea, logs in and finds some pending requests. He finds that user Mark asked him to add the new ability "Pianist" and confirms it. So, from now on, the ability will be visible to other users in Mark's profile and anyone will be able to add it to his profile.

  This fact reminds Andrea that he didn't thought of musical abilities, so he adds to the system new abilities such as, "Singer", "Dancer", "Saxophonist", etc..

- Mark is logged to the platform and looks if there is something new on the wall, so he filters it by ability and city, and discovers that Luke needs a pianist for a marriage near his house. So he answers to the post hoping to be chosen. He also goes to Luke's profile page and he decides to send him a friendship request. Luckily Luke is on line and suddenly confirms the friendship request. Now Mark and Luke are friends and they can exploit the messaging service. So they start to write each other, at first just to know each other, then talking about Mark's ability to find out if he is the right person to play the piano at Luke's marriage. After deciding that Mark's have enough experience for his marriage, Luke chooses Mark in his help request.

- Mark finally receives the performance of the catering service from Anna. He logs into SWIM, he accesses his profile and he finds out his help request. Then he opens it and chooses to send a feedback. He compiles the mark with a 4 (over 5) and writes a brief comment with a title. Then he submits the feedback.

6. UML Models

6.1 Use Case Diagram

We can derive some use cases from the scenarios identified in the previous paragraph:

- Search for a user;
- Sign up;
- Log in;
- Post a help request;
- Search help requests into the wall;
- Reply to a help request;
- Choose an answer to a help request;
- Modify profile information;
- Add an ability to the profile;
- Send a friendship request;
- Reply to a friendship request;
- Provide a feedback;
- Send a message to a user;
- Receive a message from a user;
- Add a new ability to the system.

We decided to split the Use Case Diagram into smaller ones because we wanted to make the situation clearer. We can provide some "macro Use Cases" below (this is not a Use Case Diagram, but only a diagram that helps the reader to understand the composition of the diagrams drawn below):
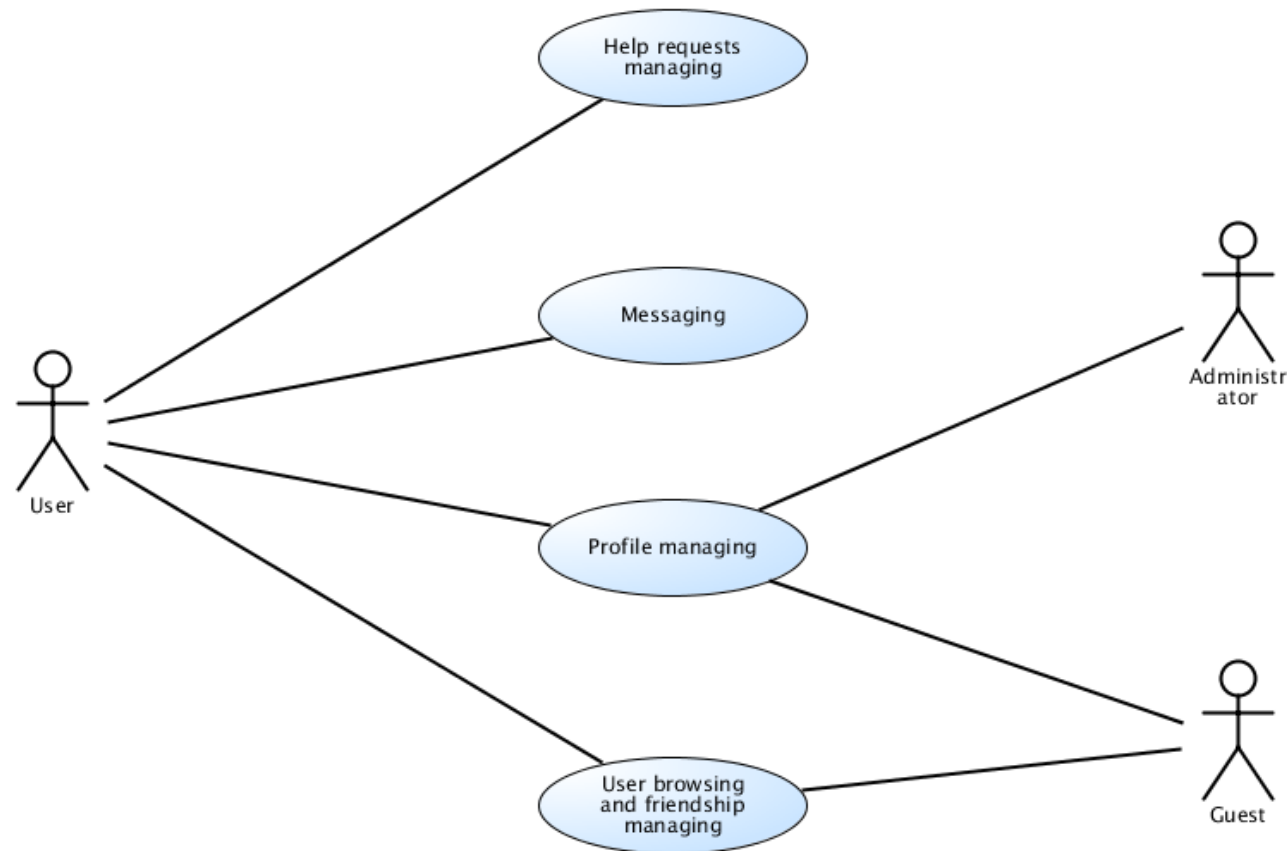
6.2   Use Cases Description

We describe in a detailed way below the main use cases. Some use cases that extends other use cases and some others are omitted because they are really similar to others. It is important to understand that all references to "pages", "buttons" or "input forms" are only hypothesis to make the situation clearer and to help the reader to draw a visual picture in his mind of what we are talking about, real pages and page structures will be well defined in the Design Document.

We refine here the use case "Log in", because it is in all other sections:

Name	Log in
Actors	User or administrator.
Entry Conditions	The user has successfully signed up to the system.
Flow of events	- The user\administrator opens the home page of the platform; - The system shows him the page; - The user\administrator enters his e-mail address and password in the input form provided; - The user\administrator clicks the button "log in". - The system shows the profile\administrator page.
Exit conditions	There are no exit conditions.
Exceptions	The information inserted in the form is wrong, an error message is shown.

6.2.1 Help Requests Managing

Name	**Search for help requests**
Actors	User.
Entry Conditions	The user is logged in and he is in the wall page.
Flow of events	- The user clicks on a link or a button that says "Search for Help Request"; - The system shows him the wall again with a form for searching; - The user fulfills the input form in which he specifies the possible conditions of his search: Ability and\or Friends and\or City; - The user starts the search clicking on a button; - The system shows the wall page filtered.
Exit conditions	The page is reloaded filtered as the user wants.
Exceptions	The search gives no result. An error message ("No result found") is shown. Then the user clicks on "OK" and the wall page is shown.

Name	**Post a help request**
Actors	User
Entry Conditions	The user has to be logged in and has to be in the wall page.
Flow of events	- The user clicks on the button "Post a Help Request"; - The system shows him a page where there is a form to be completed with the information of the help request; - The user fulfills it, writing the subject of the post, the ability(ies) needed, the place, the expiring date, etc.; - The user clicks on the button "Post". - The system reloads the wall page containing the new information.
Exit conditions	The data is now stored and Wall page is reloaded containing the new post.
Exceptions	The user doesn't fill some mandatory fields. In this case the system shows an error message.

Name	Reply to a help request
Actors	User.
Entry Conditions	The user is logged in and he views the help request. The user hasn't already replied to the help request chosen.
Flow of events	- The user clicks on "Reply"; - The system shows the reply page which contains a text field which the user can fulfill with an answer. - The user clicks on "Post"; - The system reloads the page containing the new information.
Exit conditions	The page is reloaded and the new data stored.
Exceptions	The user that replies doesn't have the right ability. In this case the system shows an error message.

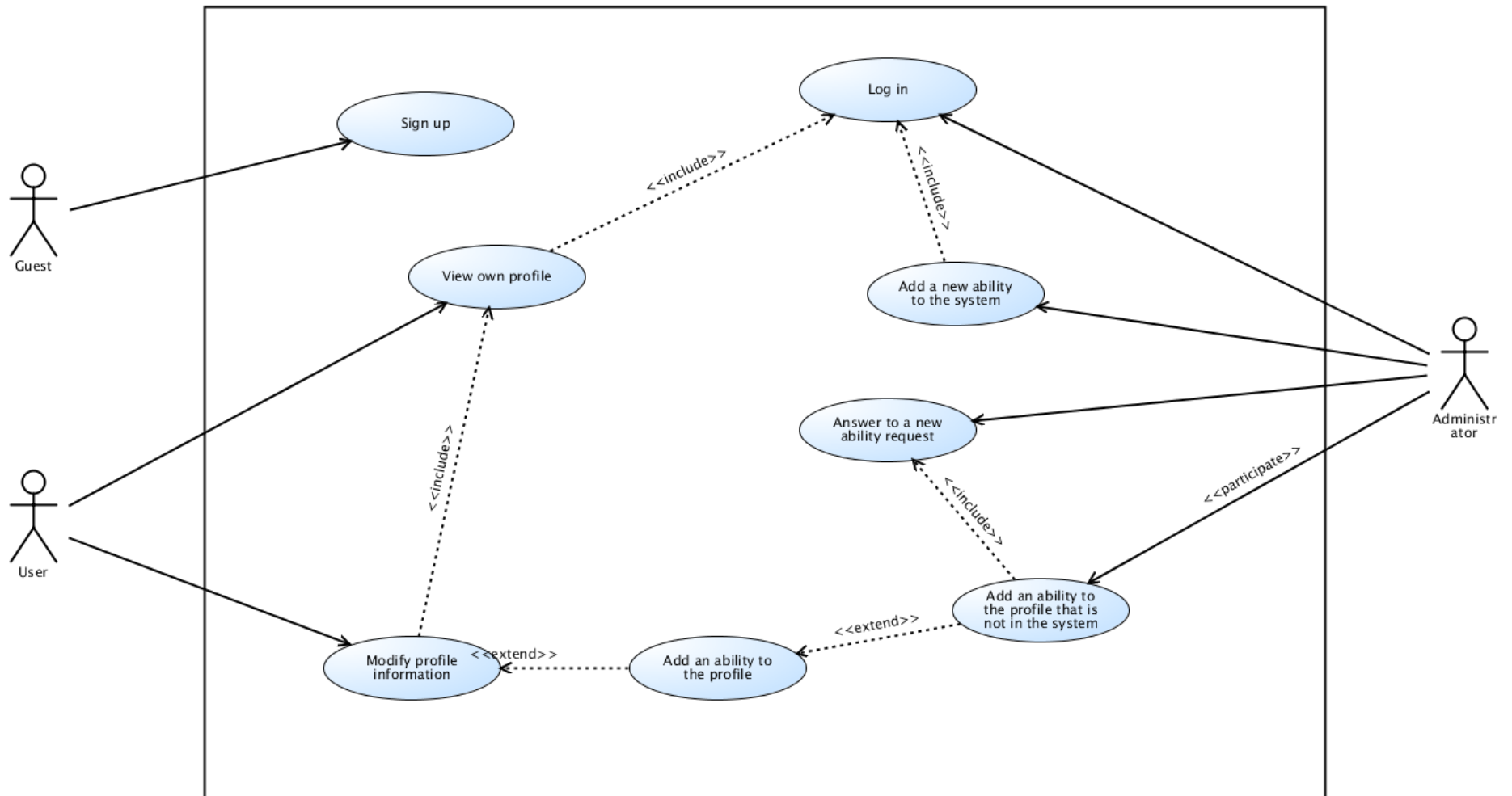| Name	Choose an answer to a help request
Actors	User.
Entry Conditions	The user has to be logged in; The user has posted a help request; The user received at least one reply to his help request; The user hasn't already chosen a reply. The user views the help request.
Flow of events	- The user clicks on something like "Choose" on the side of a reply; - The page is reloaded.
Exit conditions	The page is reloaded with the correct information, the information is stored.
Exceptions	There is no exception.

Name	**Provide a feedback**
Actors	User.
Entry Conditions	The user has to be logged in. The user has chosen from the answers to his help request another user. The user has already received the performance requested. The user views the help request.
Flow of events	• The user clicks on "Give a Feedback" on the side of the chosen reply; • The system loads a page containing an input form which the user can fill with the mark of the performance and a description; • The user fulfills the form and clicks on "Ok"; • The system informs the user of the result of the operation; • The system reloads the help request page.
Exit conditions	The page is reloaded with the right information. The information is stored.
Exceptions	There are no possible exceptions.

6.2.2 Messaging

Name	Send a message to a user
Actors	User.
Entry Conditions	• The user has to be logged in; • The two users involved have to be friends; • The user has to view the conversation with the other user.
Flow of events	• The user clicks on the button "Send a message"; • The system loads a window containing the previous messages and a form for sending a new one; • The user fills the form with the message and clicks on "Send"; • The system shows a message containing the result of the operation to the user;
Exit conditions	• The "message window" is closed; • The message is stored and notified to the other user.
Exceptions	There is no exception.

6.2.3 Profile Managing

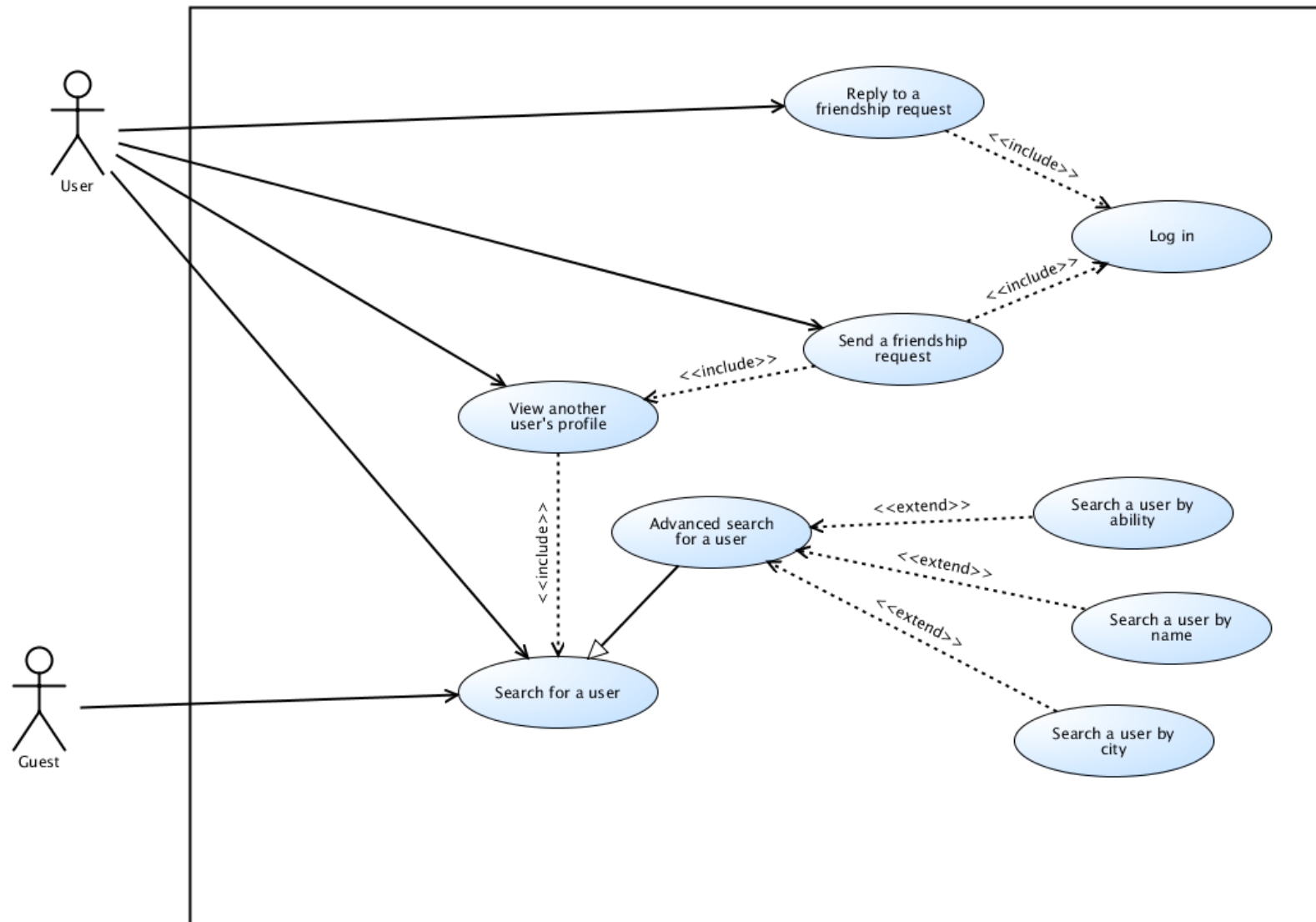| Name	**Modify profile information**
Actors	User.
Entry Conditions	The user has to be logged and be into his profile page.
Flow of events	• The user can click on "Modify"; • The system loads a page with a form that allows to modify profile information; • The user modifies what he wants; • The user clicks on "Save Changes"; • The system reloads the profile page and shows the result of the operation.
Exit conditions	The new information is stored and accessible by users. The profile page is reloaded and updated with the new information.
Exceptions	• The user doesn't click on "Save Changes" and loads another page. In this case data is lost; • The user adds an ability which is not yet in the system; • The user has emptied some mandatory fields. In this case an error message is shown.

Name	Add a new ability to the profile that is not in the system
Actors	User.
Entry Conditions	The user has to be logged and he has to be into his modification profile page.
Flow of events	• The user adds an ability that is not in the system; • The user clicks on "Save Changes"; • The system shows a message and asks to proceed; • The users clicks on "Yes"; • The system reloads the profile page and shows the result of the operation.
Exit conditions	The system notifies the administrator.
Exceptions	• The user doesn't click on "Save Changes" and loads another page. In this case data is lost; • The user clicks on "No". The system loads the modified profile page.

Name	Add a new ability to the system
Actors	Administrator.
Entry Conditions	The administrator is logged in as an administrator and he views the administrator home page.
Flow of events n° 1	• The administrator clicks on a button that tells "add ability"; • The system loads a page containing an input form; • The administrator fills the input form with the information about the ability, such as the name, the avatar and so on and clicks on "save"; • The system informs the admin of the result of the

	operation.
Flow of events n° 2	• The system notifies the administrator that somebody asked to add a new ability; • The administrator clicks on the notification; • The system loads a window which contains details about the new ability; • The administrator confirms the new ability; • The system informs the admin of the result of the operation;
Exit conditions	The home page of the administrator is reloaded and the information is stored.
Exceptions	In flow of events n° 2 the administrator can refuse to add the ability. In this case data is not updated.

Name	Sign up
Actors	Guest.
Entry Conditions	The guest hasn't already signed up and he views SWIM's home page.
Flow of events	- The user clicks on the button "sign up"; - The system shows a page which contains an input form. The input form asks for some personal information and an e-mail address; - The guest fulfills the input form and clicks "sign up"; - The link redirects him to SWIM, the system confirms the registration.
Exit conditions	The information about the guest is stored into the database in the way to grant his log in. The log in is now possible.
Exceptions	The guest enters a not valid password, or doesn't fill some mandatory fields. In this case the page is reloaded showing an error message.

6.2.4 User Browsing and Friendship Managing

Name	Search for a user
Actors	Guest or user.
Entry Conditions	The user has to view the home page of SWIM.
Flow of events n° 1	- The guest\user types the name of a user in a search field to have a quick search; - The system shows the results.
Exit conditions	There are no exit conditions.
Exceptions	The research gives no results. The user\guest clicks "search" without entering any information.

Name	**Advanced search for a user**
Actors	Guest or user.
Entry Conditions	The user has to view the home page of SWIM.
Flow of events	- The user\guest clicks on "advanced search"; - The system shows a page with an input form with different fields. For instance the name of the user, abilities, contacts or the city in which he lives; - The guest fulfills the form and clicks "search"; - The system shows the results; - The guest clicks on the user he is looking for; - The system shows his profile.
Exit conditions	There are no exit conditions.
Exceptions	The research gives no results. The user\guest clicks "search" without entering any information.

Name	Send a friendship request
Actors	User.
Entry Conditions	The user has to be logged in. The user have to be in the profile page of the other user. The user and the other have not to be already friends.
Flow of events	• The user clicks on user's name; • The system loads his profile; • The user clicks on the button "Add to friends"; • The system shows a message like "Request sent"; • The system notifies the other user of the friendship request.
Exit conditions	The friendship request is sent to the right user and the information is stored.
Exceptions	There is no exception.

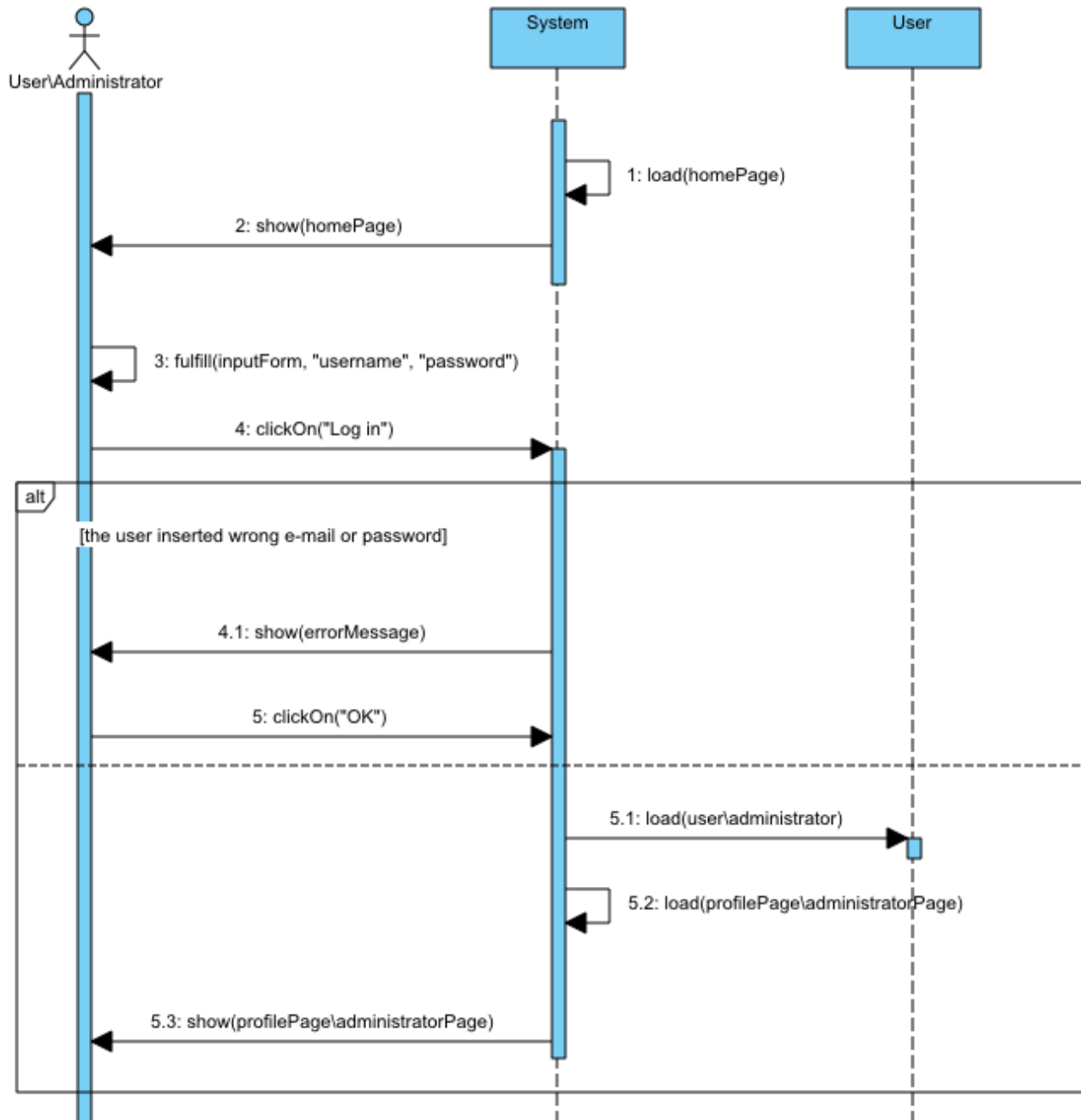| Name	Reply to a friendship request
Actors	User.
Entry Conditions	The user has to be logged in. A friendship request has been sent to the user.
Flow of events	• The system notifies a new friendship request; • The user proceeds to see the notification; • The system shows to the user the request; • The user clicks on accept or deny; • The system shows a message window with the result of the operation;
Exit conditions	The new information is stored.
Exceptions	There are no possible exceptions.

6.3 Class Diagram

Now that we have refined our use cases we can draw a class diagram of our system:
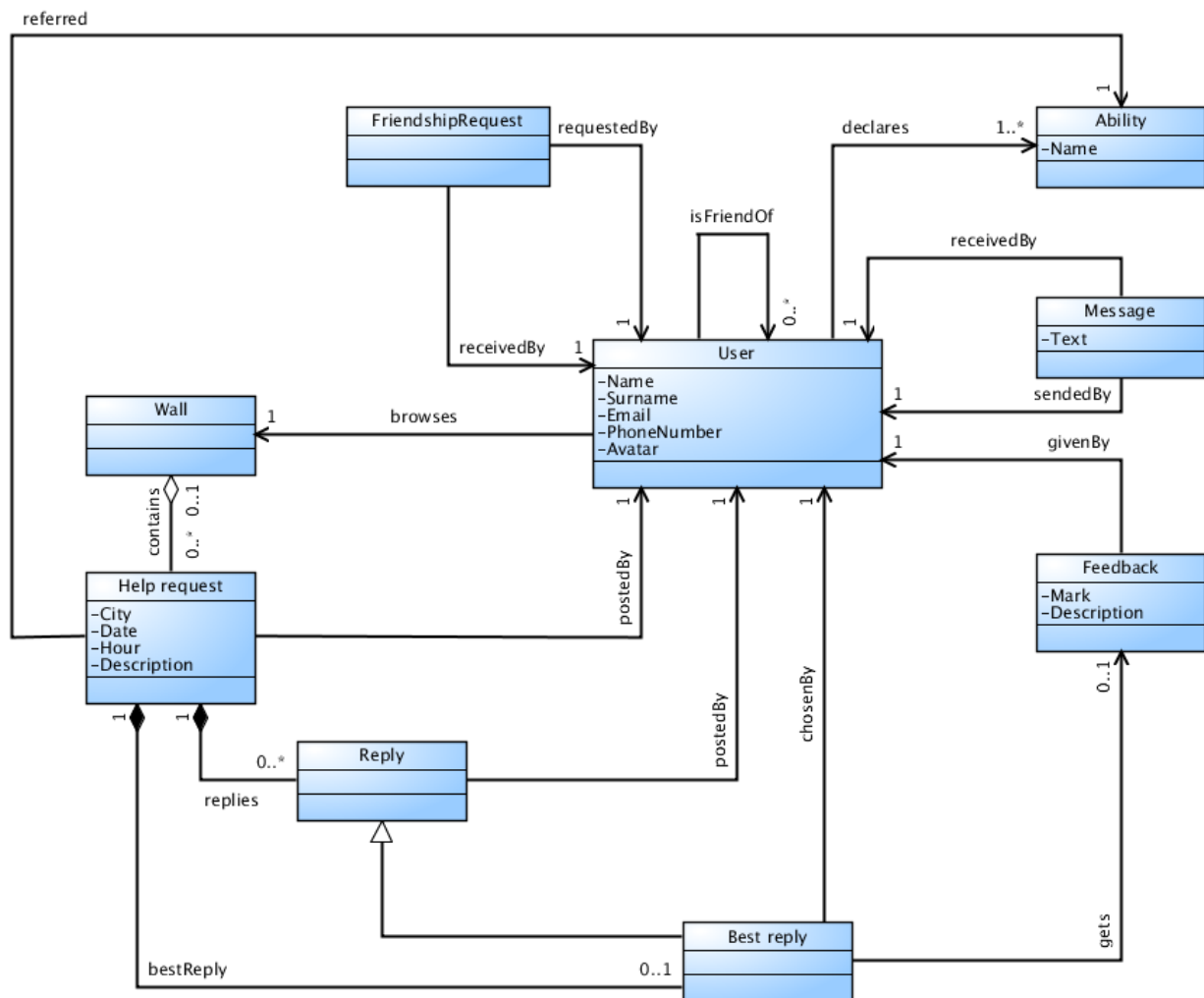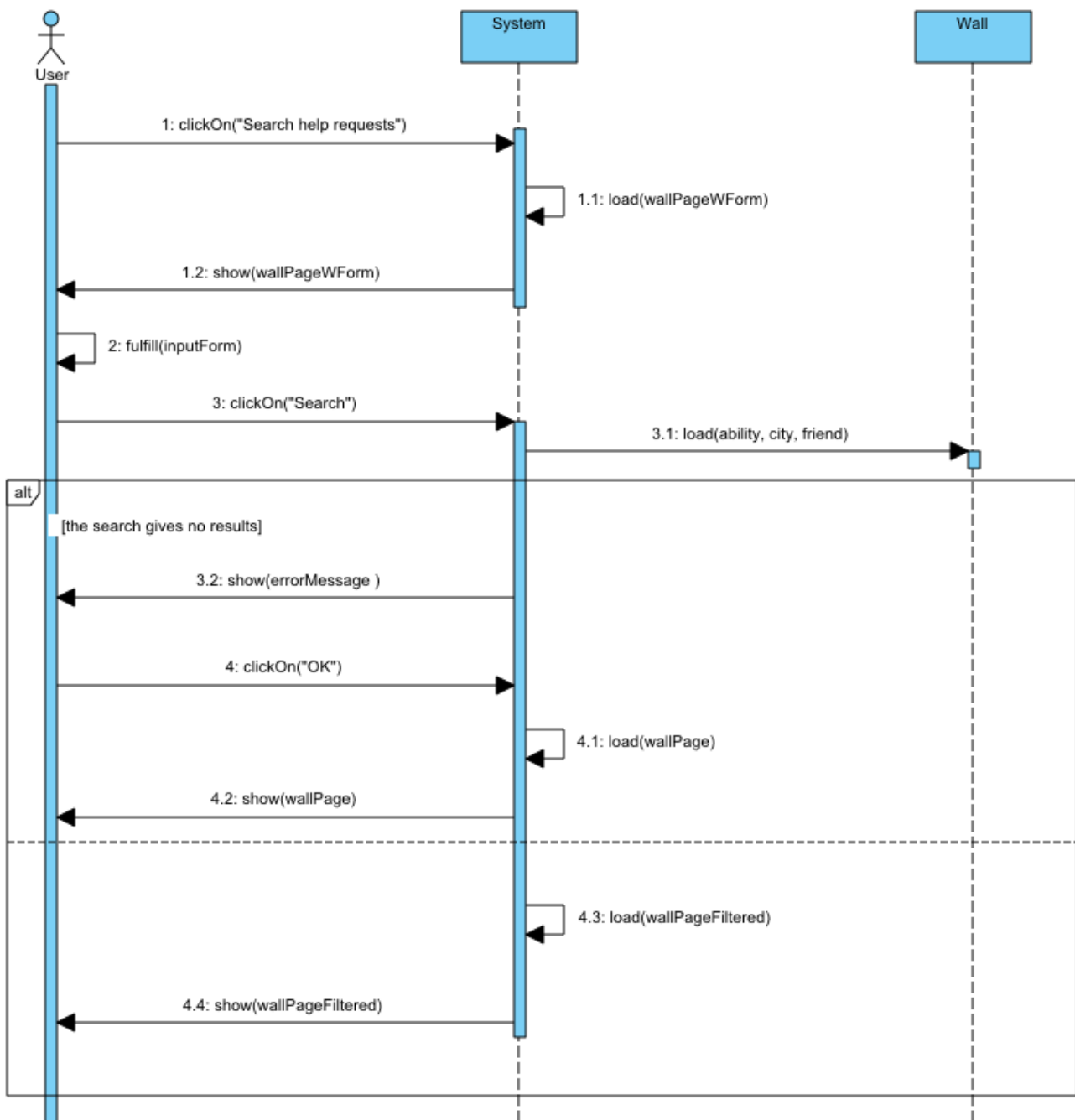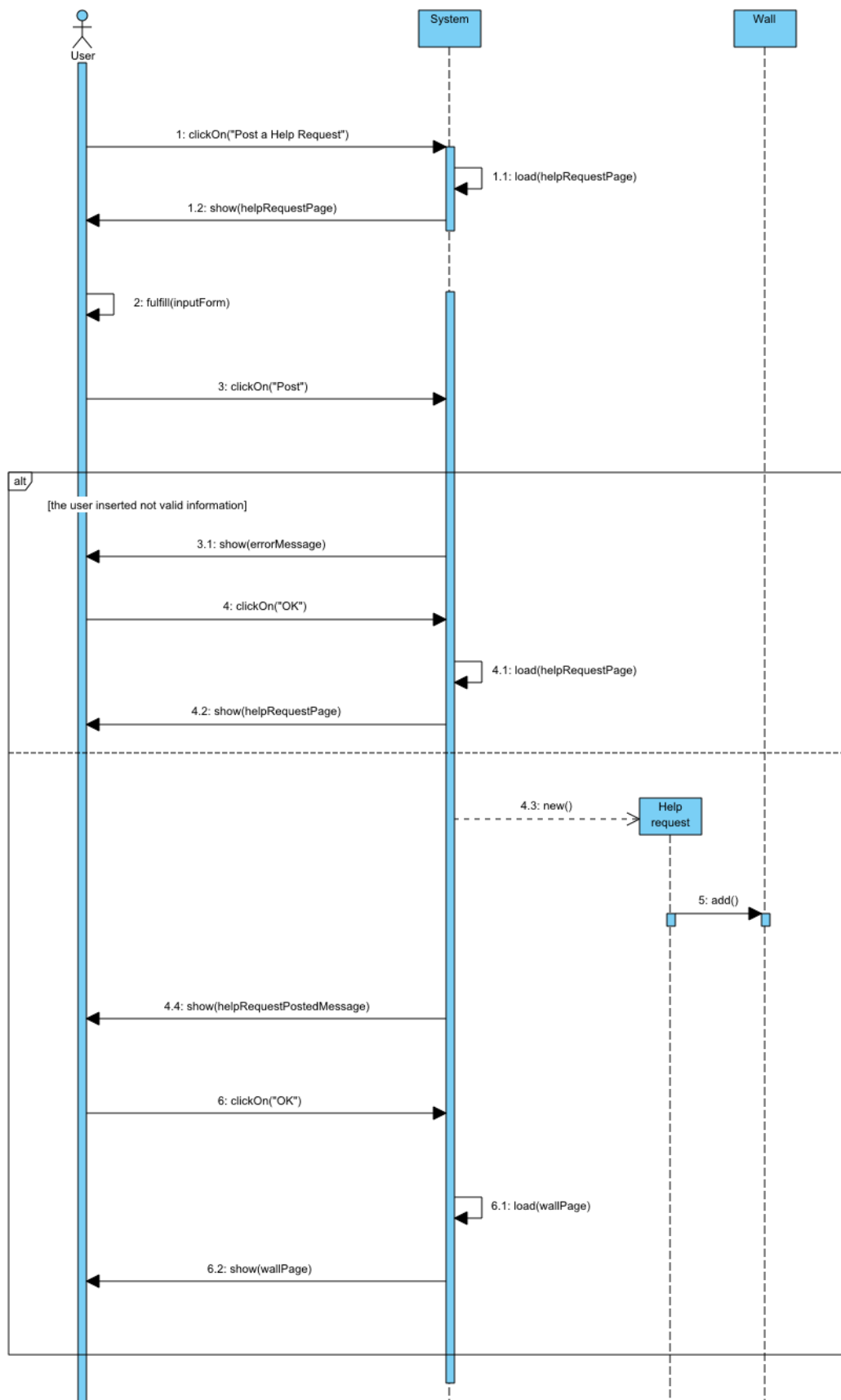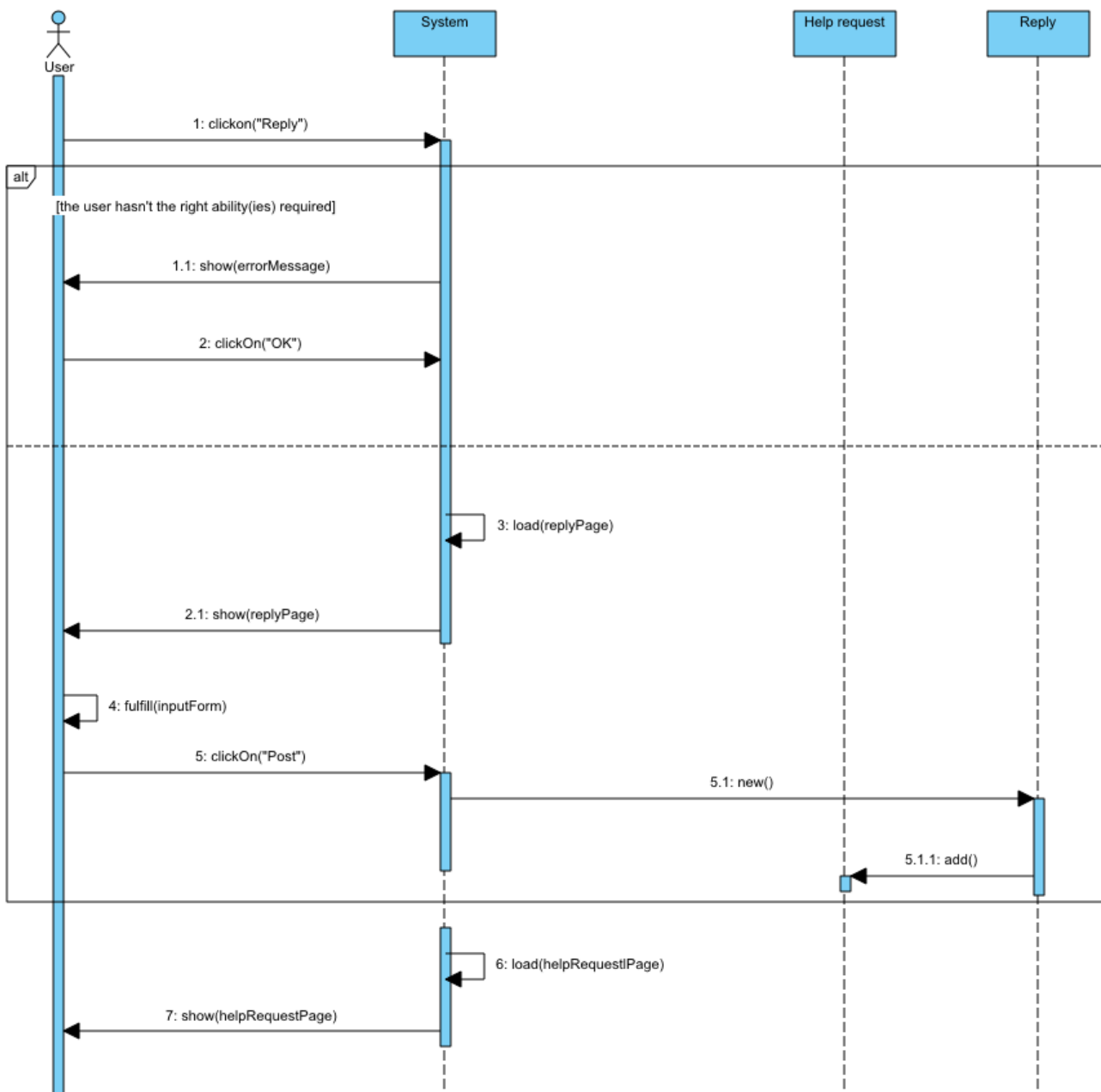
6.4 Sequence Diagrams

6.4.1 Log in
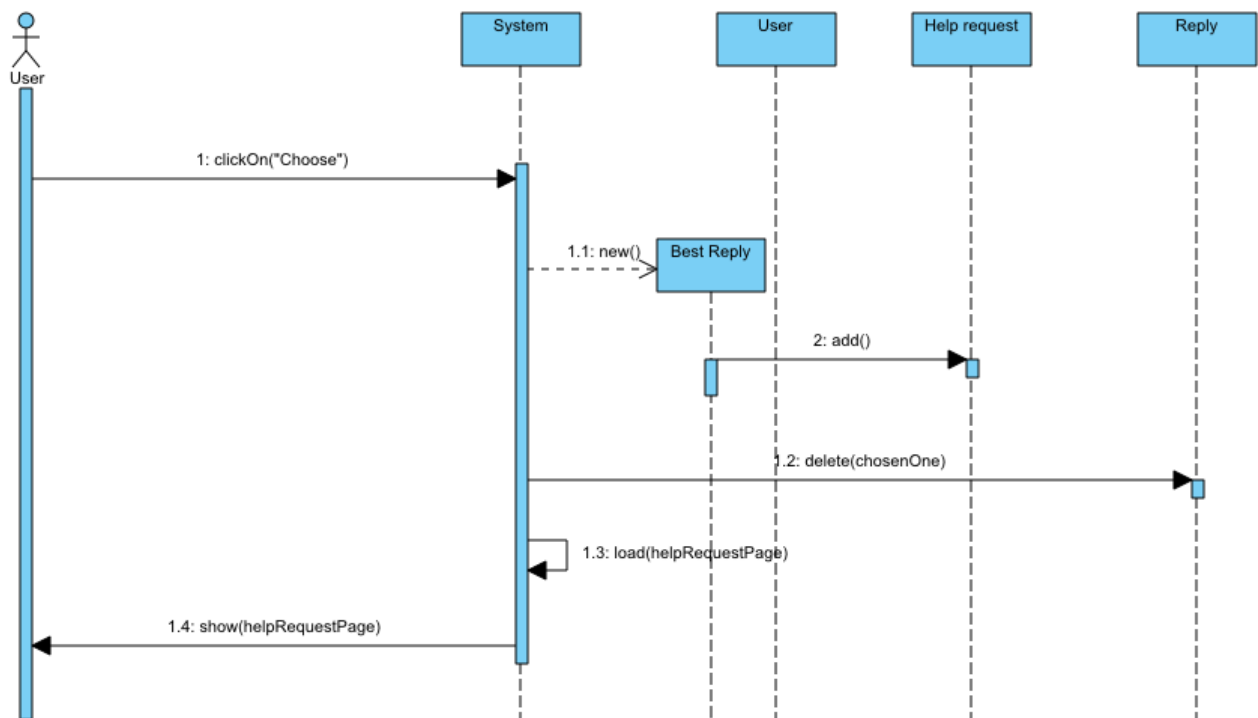
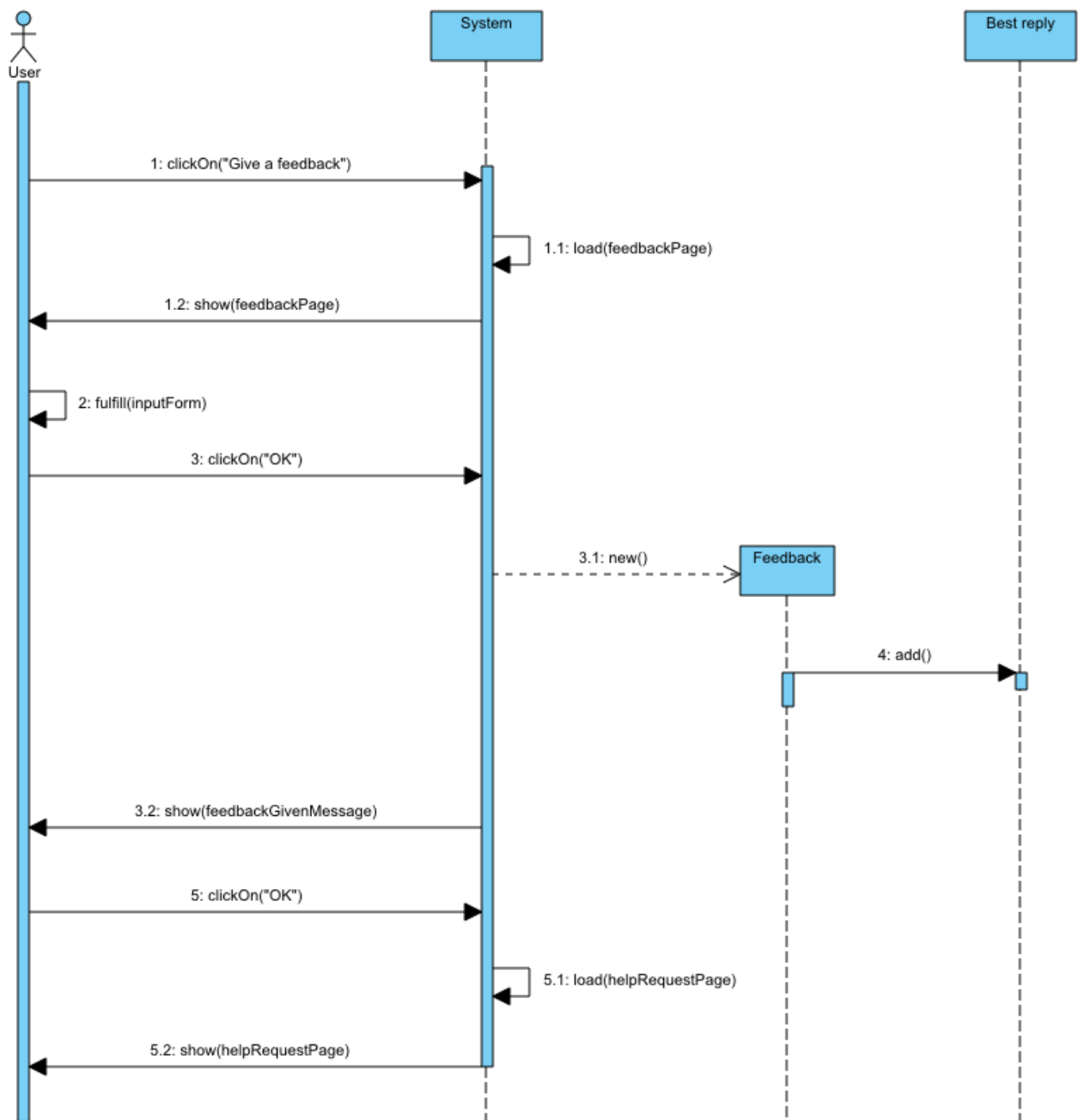## 6.4.2 Search for help requests

6.4.3 Post a help request

6.4.4 Reply to a help request
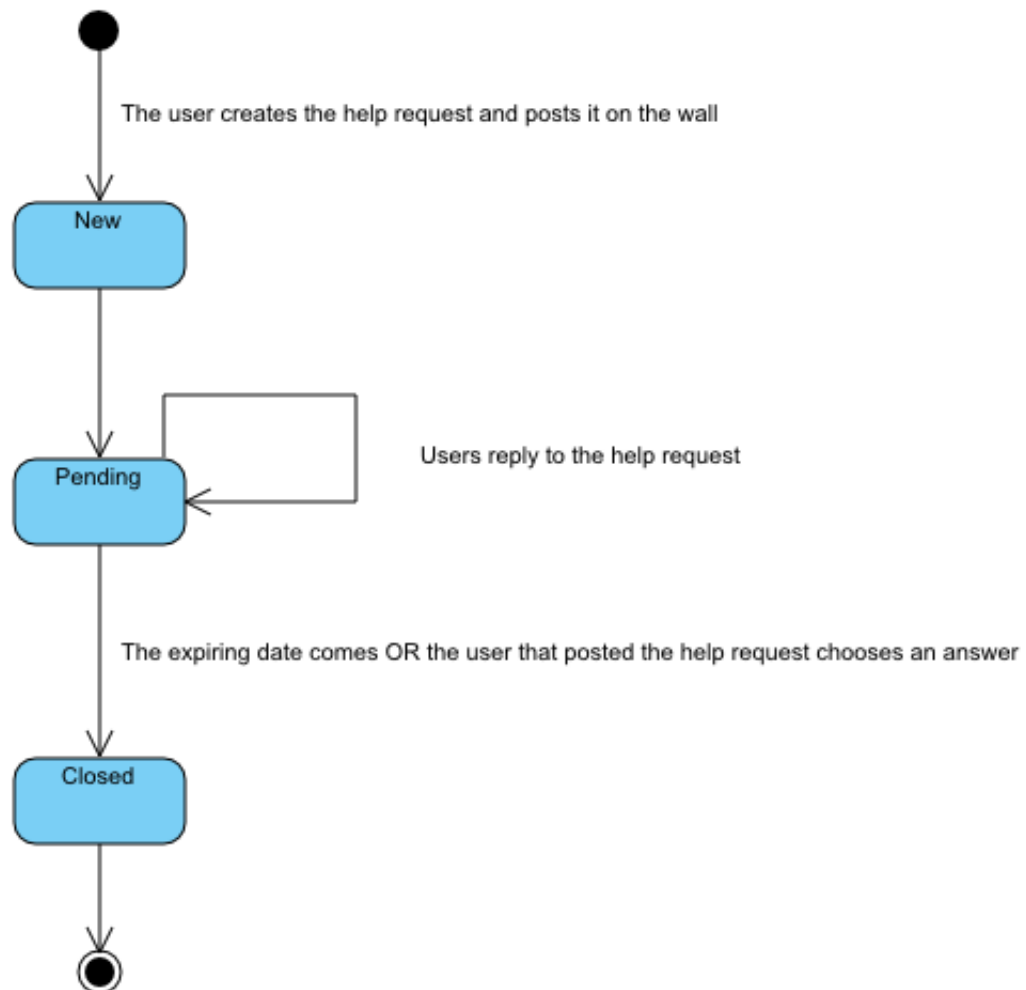
6.4.5 Choose the best answer to a help request

6.4.6 Provide a feedback

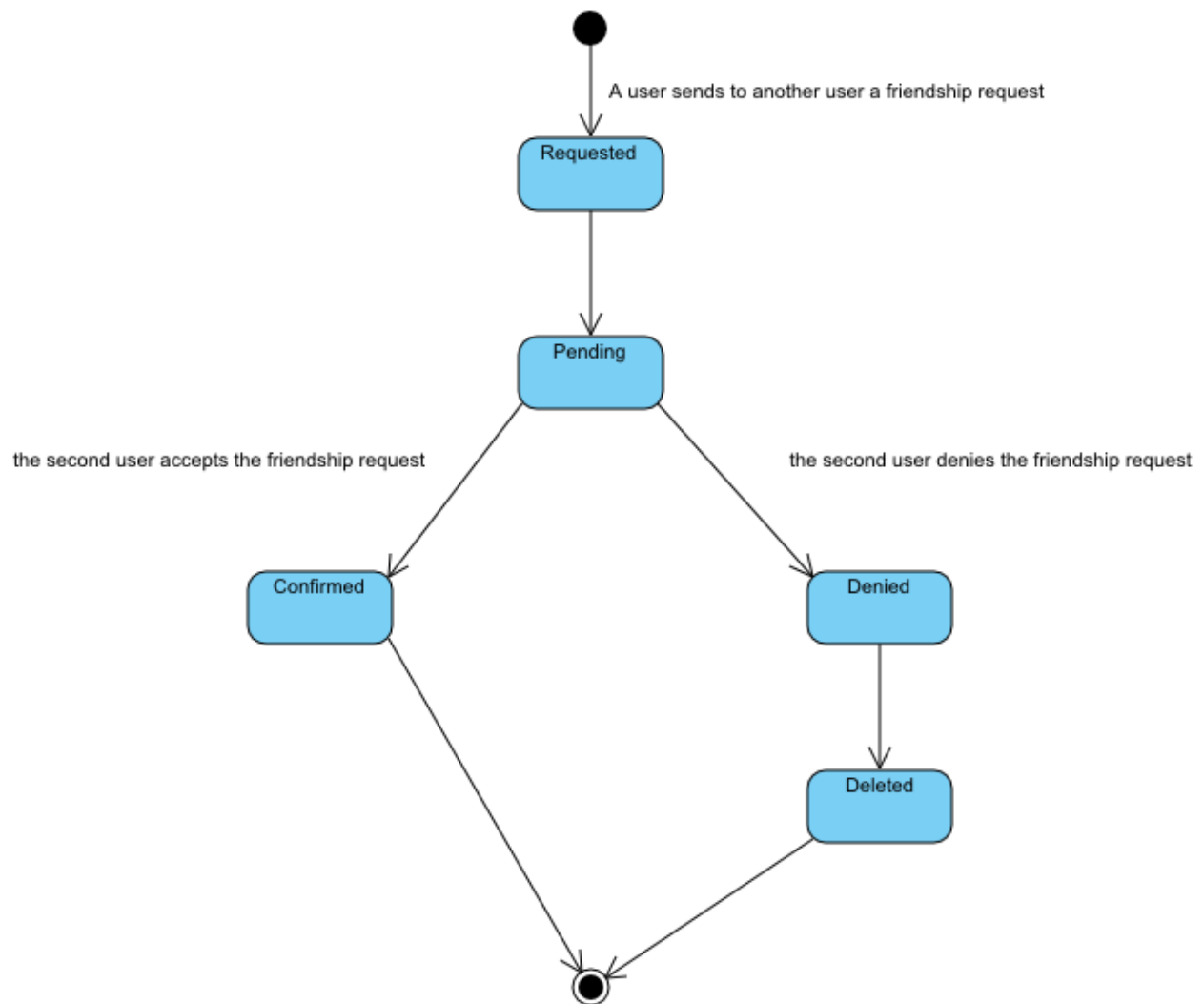# 6.5 State chart Diagrams

6.5.1 Help request Class

The user creates the help request and posts it on the wall

New

Pending

Users reply to the help request

The expiring date comes OR the user that posted the help request chooses an answer

Closed

6.5.2 Friendship Request Class

A user sends to another user a friendship request

Requested

Pending

the second user accepts the friendship request

the second user denies the friendship request

Confirmed

Denied

Deleted

7. Alloy Modeling

In this paragraph we try to understand if our Class Diagram can be consistent using Alloy Analyzer. We report below the code used and some World generated by our predicates just to let understand that our model is consistent.

```
module SWIMv2

//SIGNATURES

sig Ability{}

sig User{
    abilities: set Ability,
    friends: set User
}{
    #abilities>0
}

sig Message {
    sender: one User,
    receiver: one User
}

sig FriendshipRequest{
    sender: one User,
    receiver: one User
}

sig HelpRequest{
    owner: one User,
    replies: set Reply,
    referredAbility: one Ability
}

sig Reply{
    owner: one User
}

sig BestReply extends Reply {
    choser: one User,
    gets: lone Feedback
}

sig Feedback {
    owner: one User
}
```

```
//FACTS

fact friendshipProperties {
     //If A is a friend of B, then B is a friend of A
     all disj u1, u2: User | u1 in u2.friends implies u2 in u1.friends
     //A cannot be his own friend
     no u: User| u in u.friends
     //If A and B are friends, then there's no FriendshipRequest between
     //them
     all disj u1, u2: User | u1 in u2.friends implies no
          req: FriendshipRequest | u1 = req.sender && u2 = req.receiver
}

fact friendshipRequestProperties {
     //A cannot send a FriendshipRequest to himself
     no req: FriendshipRequest | req.sender = req.receiver
     //No multiple requests
     no disj req1, req2: FriendshipRequest |
          req1.sender = req2.sender && req1.receiver = req2.receiver
     no disj req1, req2: FriendshipRequest |
          req1.sender = req2.receiver && req1.receiver = req2.sender
}

fact messageProperties {
     //A cannor send a message to himself
     no u: User | some m: Message | u = m.sender && u = m.receiver
     //if A and B are messaging, then they has to be friends
     all disj u1, u2: User | (some m: Message |
          m.sender = u1 && m.receiver = u2) implies u2 in u1.friends
}

fact helpRequestProperties {
     //A User cannot Reply more than once in a HelpRequest
     no disj r1, r2: Reply | some h: HelpRequest |
          r1 in h.replies && r2 in h.replies && r1.owner = r2.owner
     //A HelpRequest can have at most one BestReply
     no disj br1, br2: BestReply | some h: HelpRequest |
          br1 in h.replies && br2 in h.replies
}

fact replyProperties {
     //All Replies must belong to exactly one HelpRequest
     all r: Reply | one h: HelpRequest | r in h.replies
     //The User who posts the HelpRequest cannot answer with a Reply
     all h: HelpRequest | no r: Reply |
          r in h.replies && r.owner = h.owner
}

fact bestReplyProperties {
     //All BestReplies must belong to exactly one HelpRequest
     all br: BestReply | one h: HelpRequest | br in h.replies
     //Only the User that onws the HelpRequest can choose the BestReply
     all br: BestReply | one h: HelpRequest |
          br in h.replies && br.choser = h.owner
}
```

```
fact feedbackProperties {
    //No Feedback without a BestReply
    all f: Feedback | one br: BestReply | f = br.gets
    //The User who provides a Feedback is the one who chooses the
    //BestReply (and consequentially the one who posts the HelpRequest)
    all f: Feedback | one br: BestReply |
        f = br.gets && f.owner = br.choser
}

//ASSERTIONS

assert NoSelfMessaging {
    no u: User | some m: Message | m.sender = u && m.receiver=u
}

check NoSelfMessaging

assert NoMessageIfNotFriends {
    no disj u1, u2: User | u1 not in u2.friends && some m: Message|
        u1 = m.sender && u2 = m.receiver
}

check NoMessageIfNotFriends

//PREDICATES

pred showFriendship(){
    #FriendshipRequest<3
}

run showFriendship for 5 but 0 HelpRequest, 0 Message

pred showMessagging(){}

run showMessagging for 5 but 0 HelpRequest

pred showHelpRequest(){
    #Reply>#BestReply
    #BestReply>#Feedback
}

run showHelpRequest for 4 but exactly 6 Reply, 2 BestReply, 1 Feedback,
    0 FriendshipRequest, 0 Message

pred show(){}

run show for 3
```

And here the report of Alloy Analyzer:

```
6 commands were executed. The results are:
  #1: No counterexample found. NoSelfMessaging may be valid.
  #2: No counterexample found. NoMessageIfNotFriends may be valid.
  #3: Instance found. showFriendship is consistent.
  #4: Instance found. showMessagging is consistent.
  #5: Instance found. showHelpRequest is consistent.
  #6: Instance found. show is consistent.
```

8. Worlds Generated

In this paragraph we report some worlds generated by Alloy Analyzer in order to make understand that our model is consistent.

We report below a general world generated. Further specific representations are reported below in order to better understand the characteristics of the model.
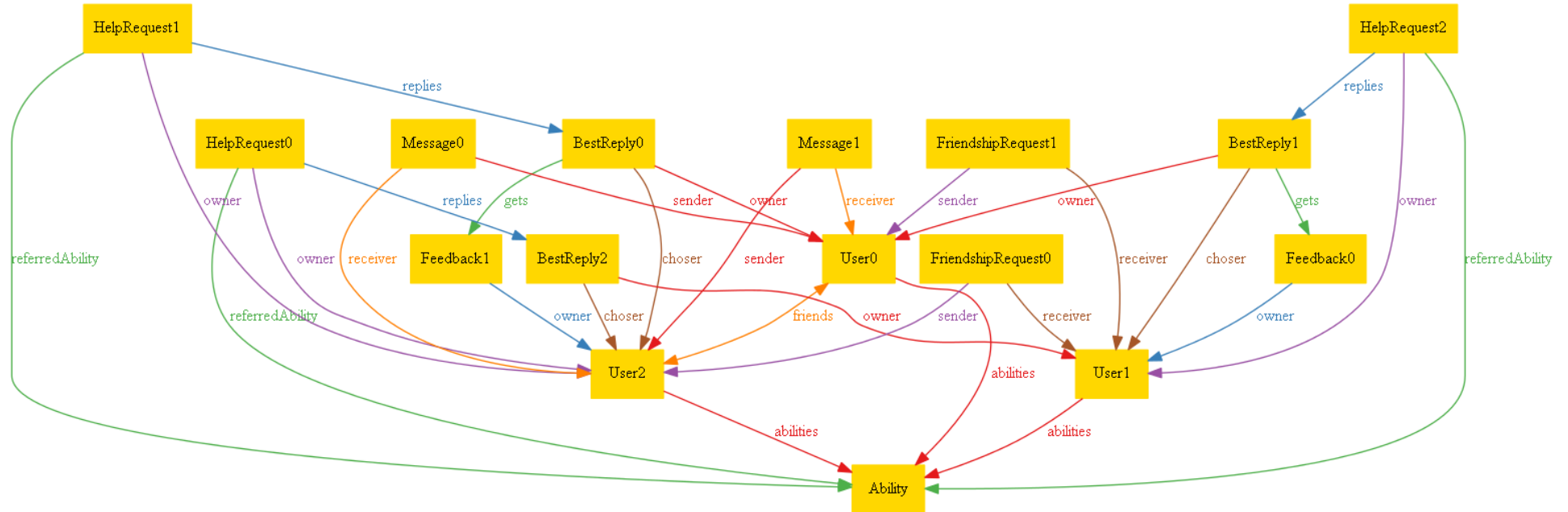
8.1 Friendship Consistency

In this paragraph is well reported that two users can be friend or there can be a Friendship Request between them or there is no "friends" relationship between them.

These three situations are mutually exclusive.
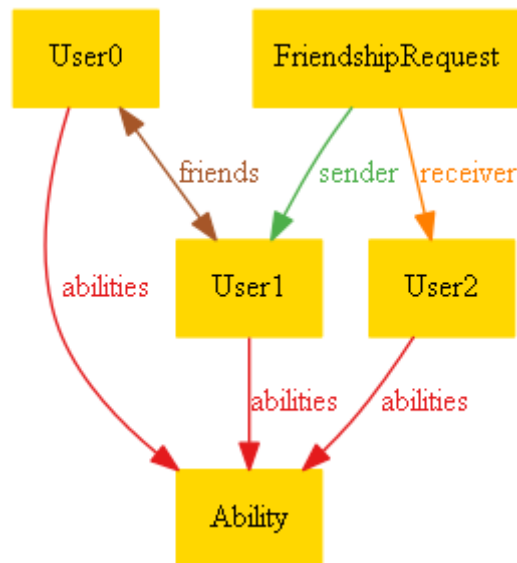
8.2 Messaging Consistency

Here is clear that a person can message with another only if they are friends. In other cases no message can be sent.
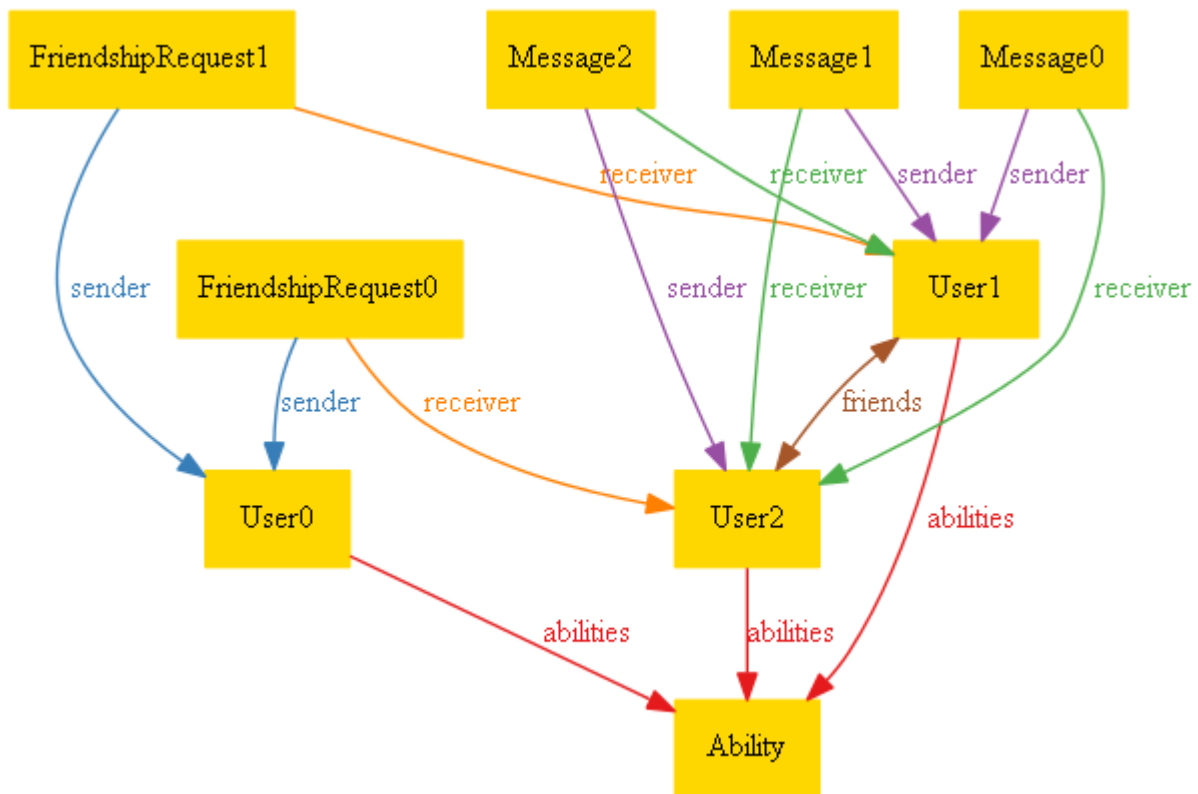
8.3 Help Requesting Consistency

We can see below the Help Request sequence of possible states:

- Requests with no Reply;

- Requests with Replies;

- Requests with Replies and a Best Reply;

- Requests with a Best Reply and a Feedback.
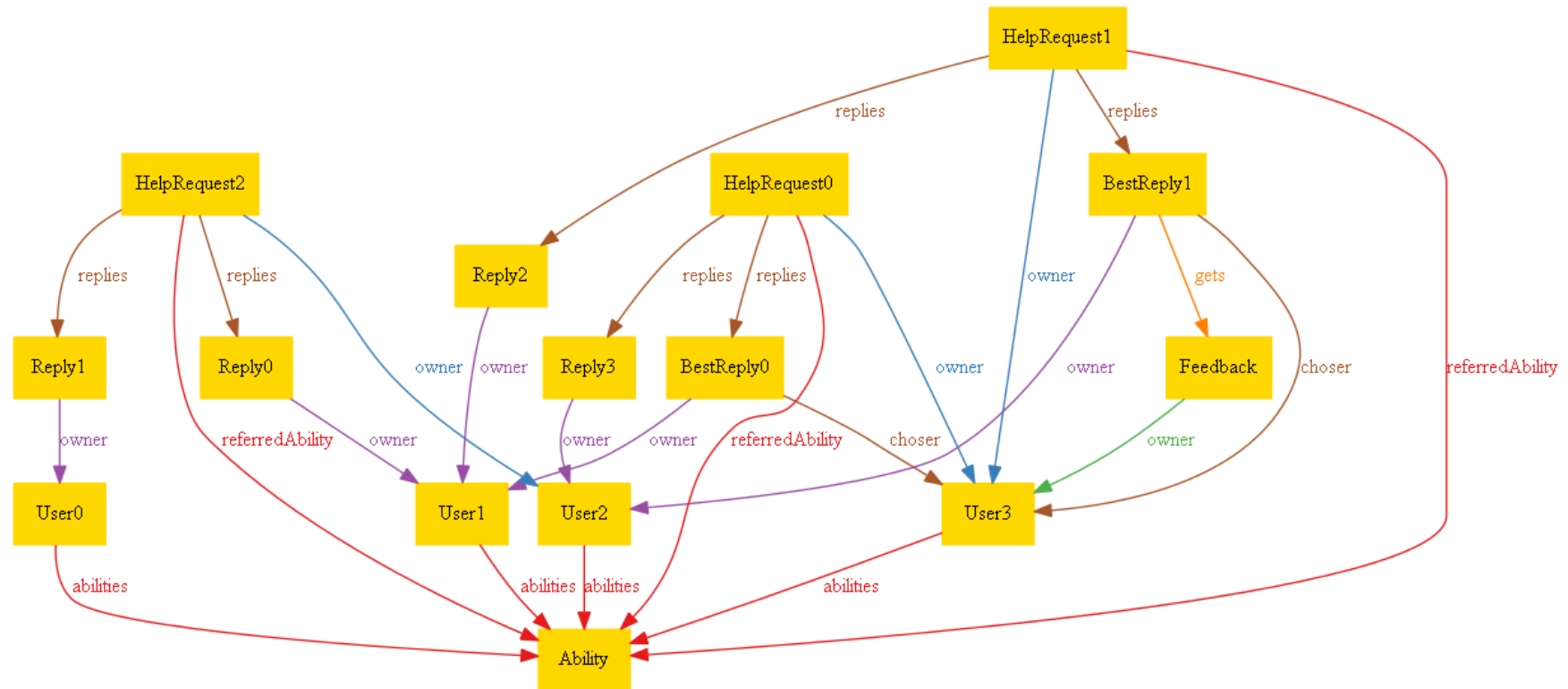
9. Used Tools

The tools we used to create this RASD document are:

- Microsoft Office Word 2007: to redact and to format this document;

- GoMockingBird Platform: to create the UI sketches;

- Signavio Platform: to create Use Cases Diagrams and Class Diagrams;

- Visual Paradigm 10 Community Edition: to create Sequence Diagrams and State Charts;

- Alloy Analyzer 4.2: to prove the consistency of our model;

- GraphViz: to format the worlds generated by Alloy Analyzer.