POLITECNICO DI MILANO

# Distributed C4C

## Requirements Analysis and Specification Document

Written by:

| | |
|---|---|
| Sara Magliacane | 735968 |
| Francesco Pongetti | 735339 |

Milan, 29th of November, 2008

# Introduction

The system main purpose is to provide a platform to reach distributed agreements in a close community. There are two main kinds of agreement that the community wants to reach:

- on when to arrange coffee breaks, and
- on the preparation of birthday presents

# Description of the problem

The specification document provides the requirements in a very informal and ambiguous way through a natural language description. The first thing to do is to formalize all the information that can be obtained from this document.
We review all the tasks the system is supposed to handle in a correct and robust way:

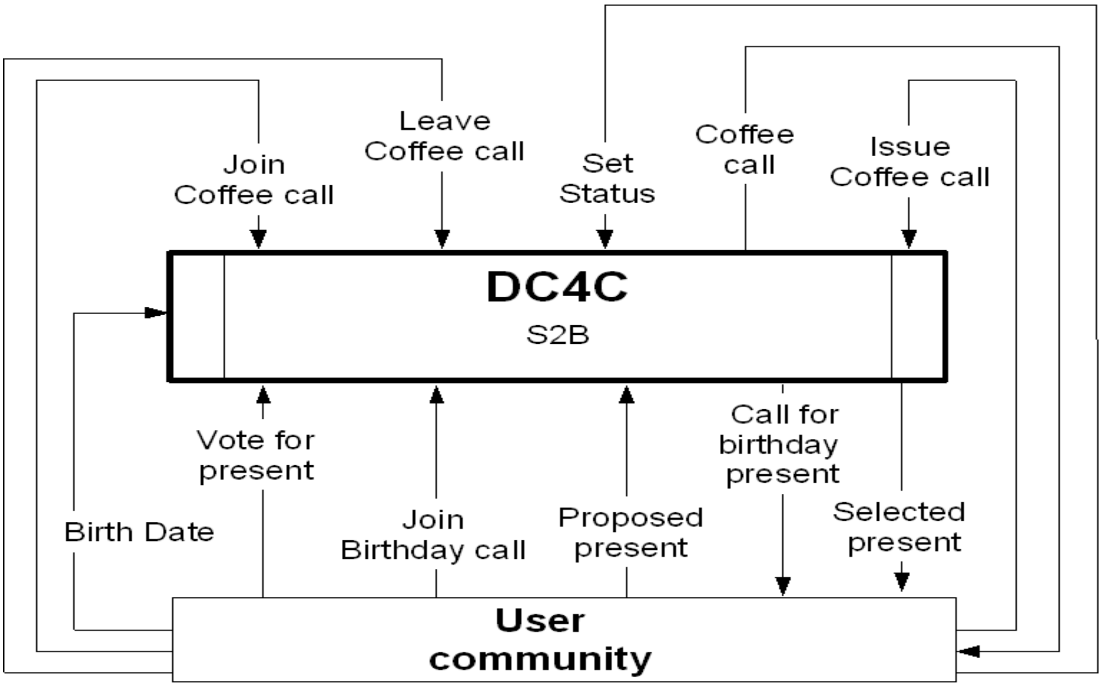
### *Arrange Coffee Calls functionalities:*

1) Any user can either issue a new coffee break call or join an existing one.
2) There are two types of calls for coffee in the system:
   1. *timed calls:* the user wants to have a coffee at a specific time no matter who comes with him
   2. *shared calls:* the user wants to wait that a specified amount of people are available to have a coffee with him; the coffee break starts as soon as the right number of people is reached
3) The users must specify in which place they want to have the coffee; two calls referred to differed places are to be considered always unrelated.
4) Users can also join an existing coffee break call, both timed or shared.
5) When either the specified time has come or the specified number of people have been gathered the coffee break starts for all the participants of the call.
6) When the participants are back to work, they have to signal it to the system.
7) In the shared calls the people already taking coffee at the specified location are also counted as participating to the coffee break, it doesn't matter in which call they joined in first place.
8) Any user can see which other users are logged at any time.

### *Organize birthday presents functionalities:*

1) The system will have to handle the choice of a birthday present for the users of the community.
2) One week before the birth date of a user, the system notifies all the other users of the community that a member's birthday is approaching. Thus the system creates a "call for birthday present".
3) Every user that has received a call can choose to join it or not.
4) If an user chooses to join a call, then he can do two things: vote for a present or propose a new present.
5) If an user wants to propose a new present, he has to provide a description and the price of the present.
6) Two days before the birthday the system automatically closes the call and the most voted present is selected.
7) If a present reaches the absolute majority of votes in the community, it is selected as a birthday present, even if the closing date has not yet been reached.

Using the System Context Diagram we can show the previous information in a graphical form that synthesizes the problem description.



## Assumptions

During requirements analysis we found some lacks in the given specification. In this section we list these lacks with a description of the way we chose to solve them. In order to perform a correct analysis of the problem we had to make some assumptions, thus adding additional hypothesis to the provided domain description. Obviously, all these assumptions will be used through the entire system development phase.

General assumptions about space and time :

- State-of-the-art technology does not allow a person to be in two different places at the same time, nor does the system.
- It is assumed that the users reach the locations of their coffee breaks, moving from their desks, in a non-relevant time.

Specific assumptions about the system:

| |
|---|
| In the specification it is not stated if there are some limitations on the number and duration of coffee breaks. |
| We assume that there are no limitations to the number of coffee breaks a person can participate to, nor to the number of people drinking coffee at the same location. We also assume that the duration of the coffee breaks is no limited, since a coffee break cannot be interrupted by external events (not even other concurrent coffee breaks), but only from the user himself. |

| |
|---|
| In the specification document it is not stated anything about the users' account management. |
| We assume that these features are not requested by the system. Furthermore the possibility of removing |

users is not contemplated as is related to the introduction of an administrator user. There is also no possibility for the users to retrieve or change the password.

In the specification it is stated that users should be notified of the starting of a shared call coffee break, but nothing is specified about the timed calls.

Notification for timed calls are assumed as useful in reminding the users they are supposed to have a coffee break.

In the specification it is stated that there are different places for coffee breaks, but nothing more is specified.

We  assumed that the coffee break locations are fixed. This assumption may be removed in further releases by introducing an administrator which would add/remove some locations that are new or temporarily out of order.

In the specification it is not stated if a user can leave a joined coffee call.

We assume that the users have the possibility to join or leave the coffee calls at any time, but always before the calls are started. Furthermore if all participants leave a coffee call, then it is deleted.

In the specification document it's written that *“users should specify their birth dates”*, so it isn't clear if every user must specify his birth date during the registration process or if this information isn't mandatory.

We assume that users can or not, provide their birth date to the system during the registration process.  If they didn't specify their birth dates, they won't have their birthday presents managed by the system, anyway they can participate to other presents.

In the specification document it isn't stated if an user can vote for more than one proposed present.

We assume that users can vote for at most one present at the time.

In the specification document it isn't stated if an user can modify his vote, while a birthday call is still open.

We assume that an user can modify his vote until the birthday call isn't closed.

In the specification document it isn't stated if an user can propose more than one present.

We assume that users can propose as many presents as they want.

In the specification document it isn't stated if an user can vote and propose a present in the same birthday call.

We assume that an user can either vote or propose a present in the same birthday call. If an user proposes a present the system automatically counts a vote from the user for that present.

In the specification document it isn't stated what to do when nobody proposed a present for a birthday call until the end of the call.

We assume that in this case the system closes the call two day before the birthday, reporting to the users that joined to the call that the present hasn't been chosen.

In the specification document it isn't stated what to do when more than one presents have received the same number of votes.

We assume that in this case the system, two day before the birthday, reports to the users that have joined the call, that more than one presents have received the same number of votes. Then the system automatically chooses the least expensive present and closes the birthday call. If two or more presents have received the same number of votes and have the same price, then the system selects the present proposed first.

In the specification document it isn't stated what to do when an user joins a birthday call but he doesn't vote for any present nor proposes a new present.

If the user, at the closing time of the call, hasn't still voted or proposed anything, the system considers that the user will accept any present selected by other users.

In the specification document it isn't stated what to do when an user was born on 29th of February.

We assume in this case that the system controls if the actual year is a leap. If it isn't, the system considers the 1th of March be the same day as 29th of February. Otherwise it considers the 29th of February as a different day.

In the specification document it isn't very clear what has to be considered the “absolute majority”.

We assume that if the total number of users of the system is X. The absolute majority is equal to (X-1)/2. We have to subtract 1 from X because the user whose the birthday is, doesn't participate to the choosing of the present.

## Further assumptions and disambiguation:

- We assume that a user is always sincere about his current status and notifies whether he is working or busy.
- We assume that the users could have joined a shared coffee call and yet be busy at the moment the coffee break should start. This is an obvious consequence of the asynchronous nature of a shared call.

  Forcing the users to have the coffee break could in some cases lead to a particular situation in which the user is immediately notifying the system he is back at work, without even leaving his desk. In this case the users actually going to take the coffee will not be joined by the exact number of users the shared call is supposed to provide.

  - ***Proposal mechanism:***

    We first chose to solve this ambiguity by adding the possibility for users to decline a coffee break proposal. As soon as the shared call is ready, the system will send a coffee proposal to all the subscribers of the call. Each of them can accept or decline the coffee proposal. If the shared call constraints are still satisfied, the system will notify all the accepting users.

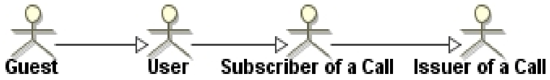
  - ***User can set status to Busy:***

    After some analysis we decided that the trade-off between the complexity we were adding with the previous solution was too high for our purposes, so we assumed that a user can set his status to Busy. In this case he will not be counted in the Quorum of the Shared Calls he joined.

This functionality increases the system complexity, but also its flexibility and user friendliness.

- If a subscriber of a coffee call is set as Busy or having a coffee break at another location and the call is triggered, he does not attend the related coffee break. Even if he returns back to work, he has lost his chance for that particular coffee break.
  The latter decision is also motivated by the assumption " the less (coffee) breaks , the more productivity" :)
- We also assume that it is not possible to have a birthday present which has not received any vote, because the system automatically counts the vote of the user who proposed the present.
- If a user proposes a present and he has already voted, then the system automatically modifies his vote to the proposed present.
- It was suggested that there should be a reasonable time interval between two consecutive coffee breaks, in order to allow the users to work, but this functionality was not implemented in this release.

## Actors



| | |
|---|---|
| **Guest:** | If a user hasn't logged in, the system considers him as a Guest. Therefore he can only register himself to the system compiling the registration form directly from the homepage. After successful registration the user will be able to access to the system with his own username and password. |
| **User** | When a Guest logins into the system by providing his correct username and password, he becomes a User. The User is the main actor in this system, he can issue a timed or shared calls or join existing coffee calls. If he wants he can participate to any existing birthday call. After his login, he can always logout. |
| **Subscriber:** | When a User joins an existing  call, he becomes a Subscriber of the call. This role extends the previous one. As a Subscriber he can leave the call at any time, or accept/decline a coffee proposal. |
| **Issuer:** | When a User issues a timed or shared call, he becomes the Issuer of the call. This role extends the previous one. As an Issuer he decides the parameters of the new call, for example the time and location of a timed call, or the quorum of people needed to trigger a shared call. For our purposes the Issuer is just the first Subscriber. |

## Glossary

- *Coffee Break* :  the main activity managed by the subsystem "Arrange Coffee Break". It consists of a social gathering around a cup of coffee.
- *Location* : the places where you can have a coffee. Examples include "Coffee Machines in the second floor" or "Bar X".
- *Status :* Every User can have be Working, Busy or at a Coffee Break in a particular Location.

- *Issue a Coffee Call* :  A User can issue a Coffee Call to invite other users or to notify the system he is going to a Coffee Break. As a side effect he becomes a Subscriber of the Call.
- *Join/Leave a Coffee Call* : At any time another User can join an existing Coffee Call, thus becoming a Subscriber of the specific Call. As a Subscriber he is free to leave the Call, whenever he wants to.

- *To trigger a Coffee Call* : When a Coffee Call is triggered, its Subscribers go to the Coffee Break.

- *Timed Call* :
  An Issuer of a Timed Call decides :
  - the BreakTime (the time of the Coffee Break)
  - the Location of the Call

  At any time before the BreakTime any User can join the Call, thus becoming a Subscriber of it, or leave it. At BreakTime all the current Subscribers receive a Notification and go to the Coffee Break.

- *Shared Call* :
  An Issuer of a Shared Call decides :
  - the Quorum (the threshold of users needed to trigger the call)
  - the Expiration Time (after which the not triggered call expires)
  - the Location of the Call

  At any time before the Call is triggered, any User can join the Call, thus becoming a Subscriber of it, or leave it.

  When number of Available Subscribers and the number of Coffee Drinking Users at the specified Location is equal to the Quorum,

- *Notification* : The system notifies the Subscribers of a Call that they shall have a Coffee Break.
- *Available Subscriber* : A Subscriber whose status is Working.

- *Coffee Drinking Users* : All the Users currently at having a Coffee Break at a specific Location.

- *Compatible Shared Call* : Supposing to have more than one concurrent Shared Call. A Shared Call S2 is compatible with a Shared Call S1 if the number of Subscribers in S2 can complete the Quorum of S1 and conversely the number of Subscribers in S1 can complete the Quorum of S2. The compatibility is a n-ary relation, that means that it applies not only to couples of Shared Calls, but to any subset of the current Shared Calls.

- *Call for birthday present* : the event generated by the system one week before the birthday date of an user. This event is reported to all the users except to the one who is celebrating his birthday.
- *Proposed present* : a present proposed by an user participating to a call for birthday present.

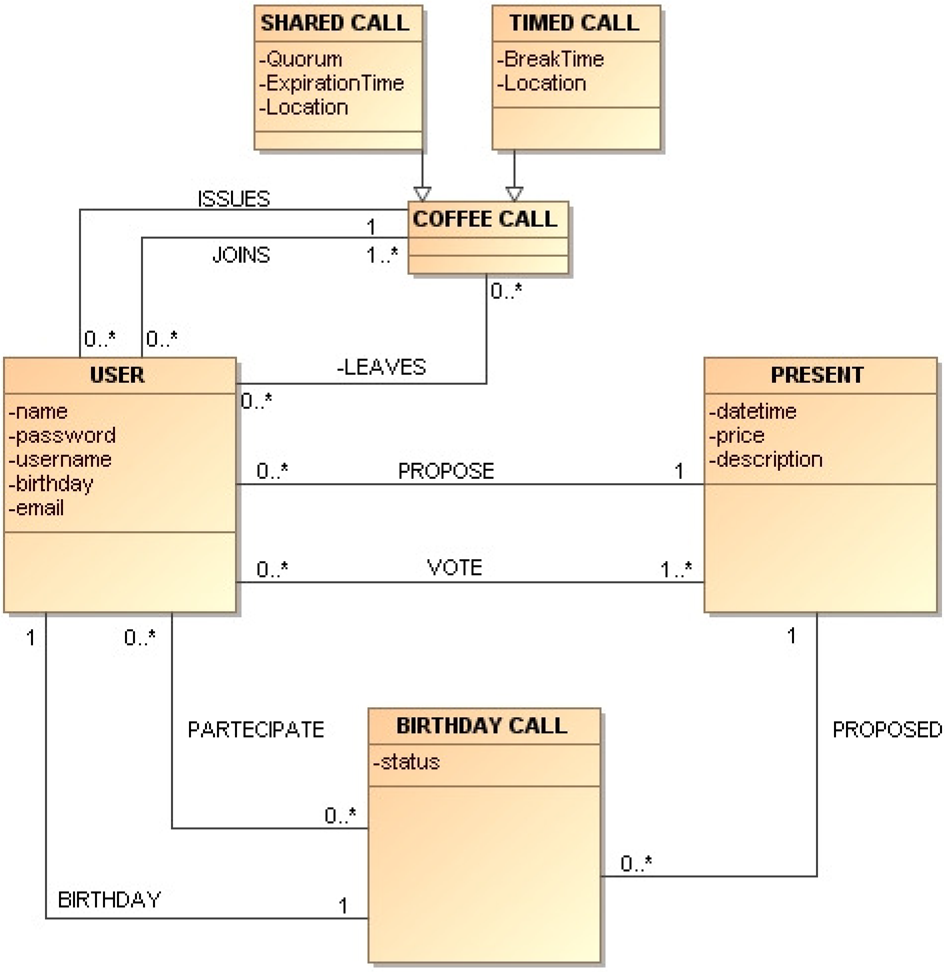
- *Join a birthday call* : when an user receives a call for birthday present, he can decide to participate to the choice of the present by joining the call.

## Conceptual Class Diagram

Through the class diagram it is possible to see the relationships between the different entities described in the specification document. In this phase of the development we are not going in to describe the system in detail, because this is the purpose of the design phase. Clearly in this phase the class diagram is a kind of conceptual model that may be modified in future.

Description of the entities:

- ***USER:*** registered user that uses the DC4C system. This entity has six attributes: name, surname, birthday, username, password, email
- ***BIRTHDAY CALL:*** a "call for birthday present" automatically generated by the system one week before the birth date. The entity has one attribute: status (open | close)
- ***PRESENT:*** all the present proposed by the users. The entity has three attributes: price and description and datetime
- ***COFFEE CALL:*** a "call for coffee" that can be issued by any user; it is a generalization of a TIMED and SHARED CALL
- ***TIMED CALL:*** a coffee call that is triggered on a specific time, no matter how many users have joined it. The entity has two attributes, BreakTime and Location
- ***SHARED CALL***: a coffee call that is triggered when a threshold of users have joined it. The entity has three attributes, Quorum ( the threshold of users needed to trigger the call), Expiration Time (after which the not triggered call expires) and Location

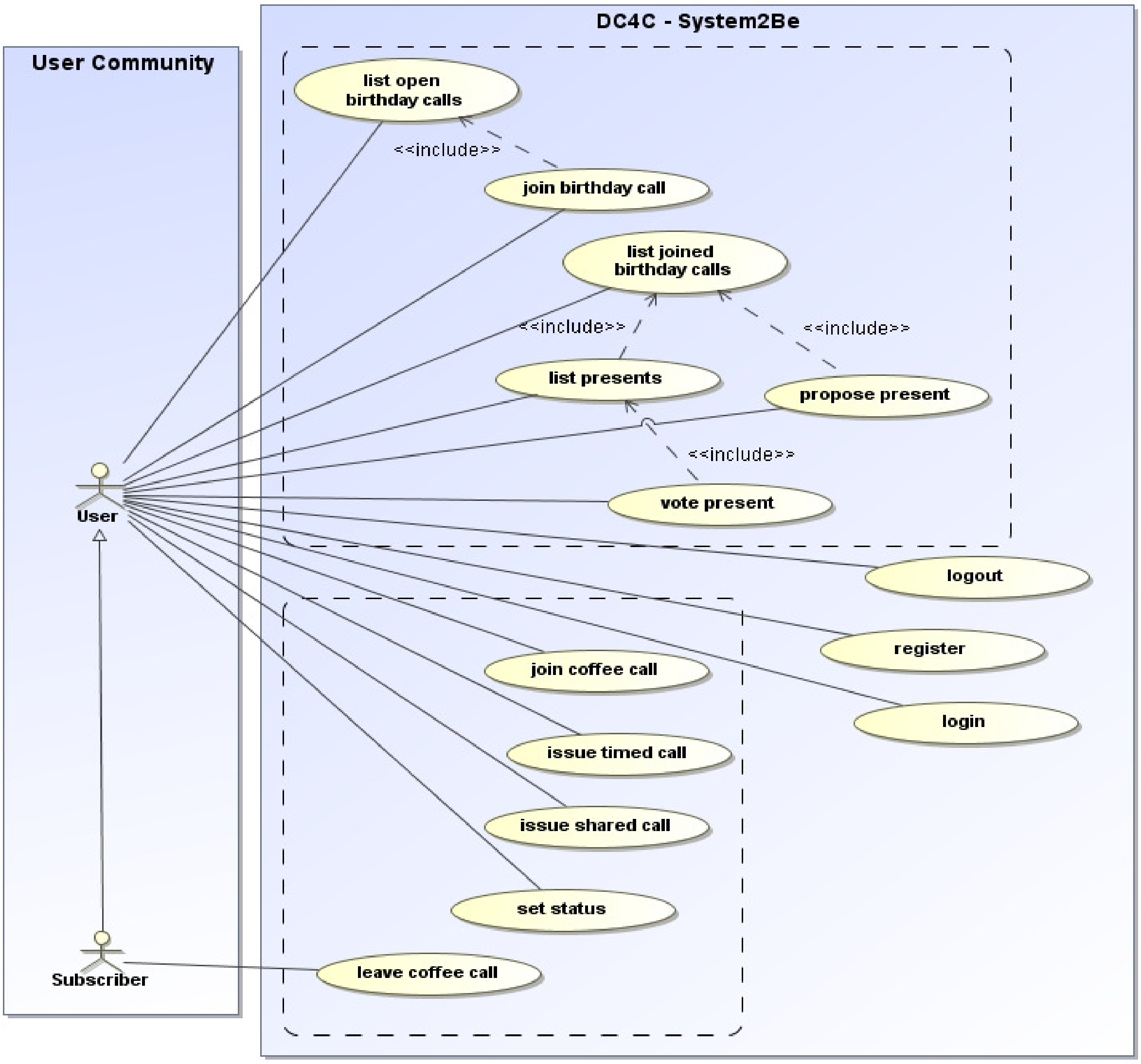


Description of the relationships:

- ***PROPOSE:*** an user can propose a present for a "call for birthday present"
- ***VOTE:*** an user can vote for an existing present
- ***PARTECIPATE***: an user can join to a "call for birthday present"
- ***BIRTHDATE:*** every "call for birthday present" is created starting from the birth dates of the users
- ***PROPOSED FOR:*** every present is proposed for a specific "call for birthday present"

- *ISSUES:* a user can issue a coffee call to invite other users or to notify the system he is going to a coffee break
- *JOINS:* at any time an user can join an existing coffee call
- *LEAVES:* at any time an user can leave a coffee call he had joined

## Use Cases



## User Use Cases

### 1. Register

Description: A Guest wants to register in the system

Participating actors: Guest (G1)

Pre-conditions: G1 must not be already registered in the system

Event flow:

1. G1 accesses the homepage and selects the form New User
2. G1 chooses his username and password
3. If he wants, he can also specify his email and birthday

Post-conditions: G1 becomes the registered User U1

Exceptions: If the username G1 has chosen in the second step is already in use, we iterate the second step

## 2. Login

Description: An User wants to login into the system

Participating actors: User U1

Pre-conditions: U1 must be already registered in the system

Event flow:

1. U1 accesses the homepage and selects the Login form
2. U1 types his username and password and submits them to the system
3. U1 can access his personal page and choose the “Arrange coffee breaks” or “Organize birthday presents” services

Post-conditions: U1 is logged into the system

Exceptions: If the username and password are not correct, we iterate the second step

## 3. Logout

Description: An User wants to logout from the system

Participating actors: User U1

Pre-conditions: U1 is logged into the system

Event flow:

1. U1 clicks on the button Logout in his personal page.

Post-conditions: U1 is logged out the system

## 4. Issue a Timed Call

Description: User U1 issues a Timed Call

Participating actors: User U1, Subscriber S1

Pre-conditions: U1 must be already logged into the system

Event flow:

1. From his personal page U1 selects the Coffee Call Tab and clicks the Issue Timed Call button
2. U1 specifies a selected time T and a selected Location P in the Issue Timed Call Form, becoming Subscriber S1 of the Call

Post-conditions: U1 is now Subscriber S1 of the selected Timed Call

## 5. Issue a Shared Call

Description: User U1 issues a Shared Call

Participating actors: User U1, Subscriber S1

Pre-conditions: U1 must be already logged into the system

Event flow:

1. From his personal page U1 selects the Coffee Call Tab and clicks the Issue Shared Call button
2. U1 specifies a Quorum Q, Location P and a Expiration Time T in the Issue Shared Call Form, becoming Subscriber S1

Post-conditions: U1 is now Subscriber S1 of the selected Shared Call

## 6. Join a Coffee Call

Description: User U1 joins a Coffee Call

Participating actors: User U1, Subscriber S1

Pre-conditions: U1 must be already logged into the system

Event flow:

1. From his personal page U1 selects the Coffee Call Tab
2. U1 sees all the issued Coffee Calls from the list in the Coffee Call Tab

3. U1 selects a Coffee Call and clicks on the Join Button, becoming Subscriber S1

Post-conditions: U1 is now Subscriber S1 of the selected Coffee Call

## 7. Set Status

Description: User U1 notifies the system if he is Working or Busy

Participating actors: User U1

Pre-conditions: U1 must be already logged into the system

Event flow:

1. From his personal page U1 selects his current Status from a list
2. U1 selects Busy from the list and clicks on the submit button

Post-conditions: User U1 is listed as Busy

## 8. View who is Online

Description: User U1 wants to see which other users are online

Participating actors: User U1

Pre-conditions: U1 must be already logged into the system

Event flow:

1. From his personal page U1 selects Who is Online Tab

Post-conditions: User U1 sees the list of all other logged users

## 9. List open birthday calls

Description: An User wants to see the list of open “call for birthday present”

Participating actors: User U1

Pre-conditions: U1 must be already logged into the system

Event flow:

1. U1 accesses to the homepage and click on the “list birthday calls” button

Post-conditions: the list of open “calls for birthday present” is displayed to U1. For each call, the list contains the following informations:

1. The name of the user whose birthday is
2. The birth date

Exceptions: if there isn't any open call, an information message is displayed to U1

## 10. Join birthday call

Description: An User wants to join a “call for birthday present”

Participating actors: User U1

Pre-conditions: There must be at least one open “call for birthday present”

Event flow:

1. U1 sees the list of open “calls for birthday present”
2. U1 selects one “call for birthday present” from the list
3. U1 clicks on the “join” button

Post-conditions: U1 must be signed as participating to the selected “call for birthday present”

## 11. List joined birthday calls

Description: An User wants to see the list of “calls for birthday present” he have joined

Participating actors: User U1

Pre-conditions: U1 must be already logged into the system

Event flow:

1. U1 accesses to the homepage and click on the "joined birthday calls" button

Post-conditions: the list of joined "calls for birthday present" is displayed to U1. For each call, the list contains the following information:

1. the name of the user whose the birthday is
2. the birth date
3. the status of the call (open | close)
4. if the call is closed, the selected present

Exceptions: if there isn't any joined call, an information message is displayed to U1

## 12. List presents

Description: An User wants to see the list of presents for a selected "call for birthday present"

Participating actors: User U1

Pre-conditions: U1 must be signed to at least one at least one "call for birthday present"

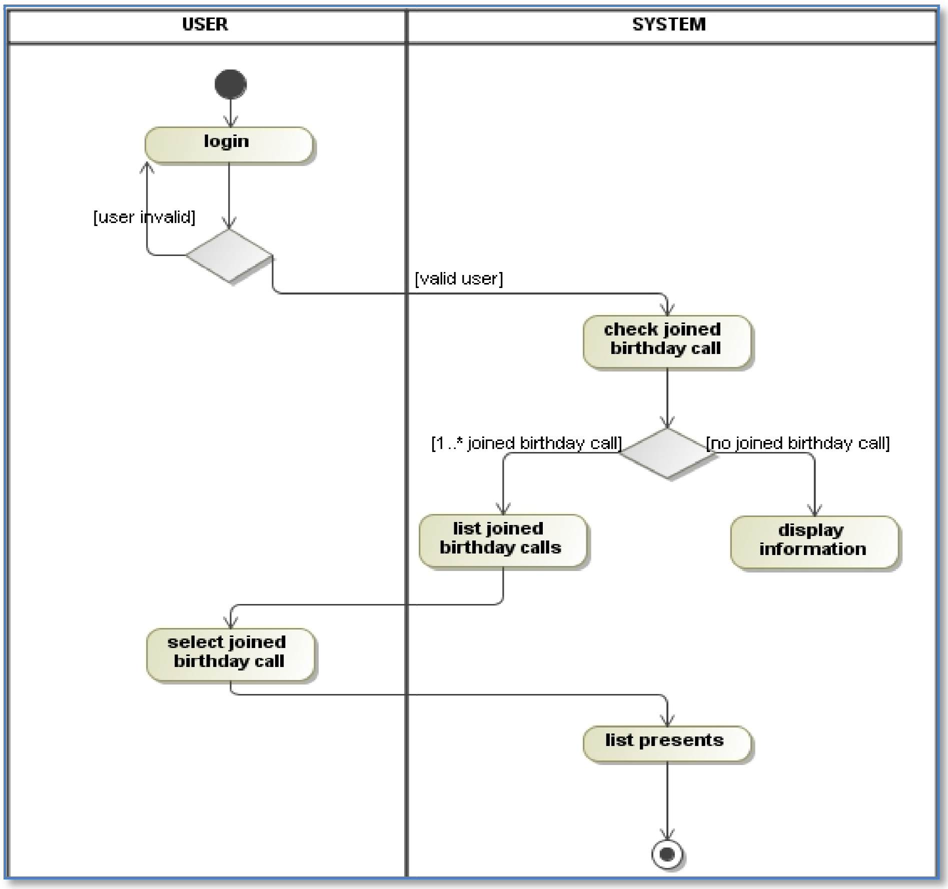
Event flow:

1. U1 sees the list of joined "calls for birthday present"
2. U1 selects one "call for birthday present" from the list
3. U1 clicks on the "show presents" button

Post-conditions: a list of presents for the selected "call for birthday present" is displayed to U1. For each present, the list contains the following informations:

1. the votes that the present has received so far
2. the name of the user that proposed the present
3. the description
4. the price

Exceptions: if there isn't any present, an information message is displayed to U1


USER
SYSTEM
login
[user invalid]
[valid user]
check joined birthday call
[1..* joined birthday call]
[no joined birthday call]
list joined birthday calls
display information
select joined birthday call
list presents

## 13. Vote present

Description: An User wants to vote for a specific present

Participating actors: User U1

Pre-conditions: U1 must be signed to at least one “call for birthday present” and there must be at least one present for the selected call

Event flow:

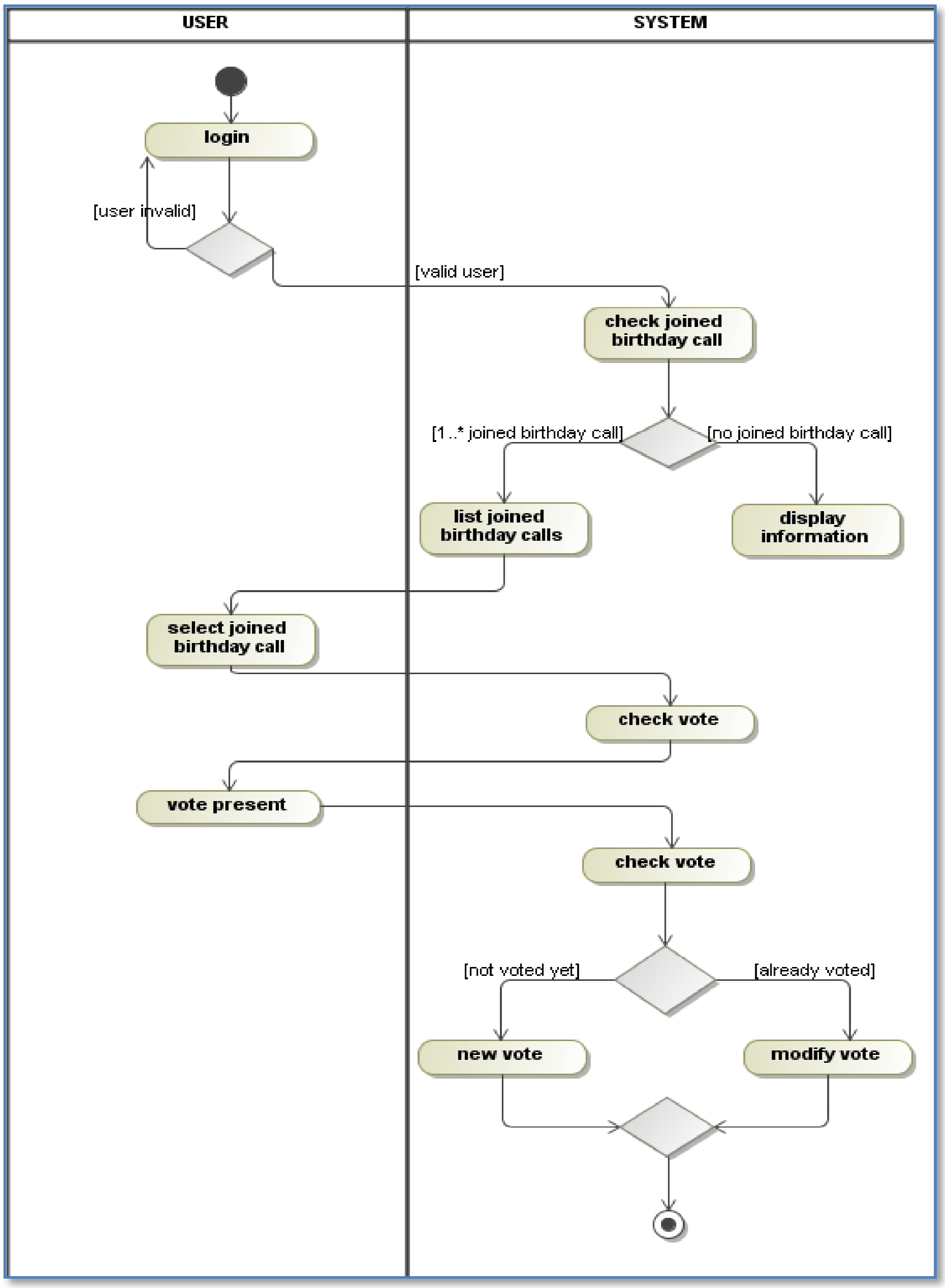
1. U1 sees the list of presents for a selected “call for birthday present”
2. U1 selects one present from the list
3. U1 clicks on the “vote” button

Post-conditions: one vote is added to the count for the selected present

Exceptions: there are a few possible exceptions:

1. The selected “call for birthday present” is closed, so it's not possible to vote
2. U1 has already voted for a present, thus the system asks to the user if he wants to modify his vote to the new present.

   In all case an information message is displayed.



## 14. Propose present

Description: An User wants to propose a present for a selected "call for birthday present"

Participating actors: User U1

Pre-conditions: U1 must be signed to at least one "call for birthday present"

Event flow:

1. U1 sees the list of joined "call for birthday present
2. U1 selects one "call for birthday present" from the list
3. U1 click on "propose present" button
4. U1 inserts price and description for the present

Post-conditions: the present is added to the list of presents for the selected "call for birthday present"

Exceptions: there are a few possible exceptions:

1. The selected "call for birthday present" is closed, so it's not possible to propose new presents

   In all case an information message is displayed.

## Subscriber Use Cases

### 1. Leave a Coffee Call

Description: Subscriber S1 leaves a Coffee Call

Participating actors: User U1, Subscriber S1

Pre-conditions: S1 must be a Subscriber of the selected Coffee Call

Event flow:

1. From his personal page S1 selects the Coffee Call Tab
2. S1 sees all the issued Coffee Calls from the list in the Coffee Call Tab
3. S1 selects the Coffee Call he wants to leave and clicks on the Leave Button

Post-conditions: U1 is not longer Subscriber S1 of the selected Coffee Call

## More complex Scenarios

### Scenario 1

Name: Simple Timed Call

Description: User U1 issues a Timed Call, user U2 joins the Timed Call

Participating actors: Users U1, U2, Subscribers S1, S2

Pre-conditions: U1 and U2 must be already logged into the system

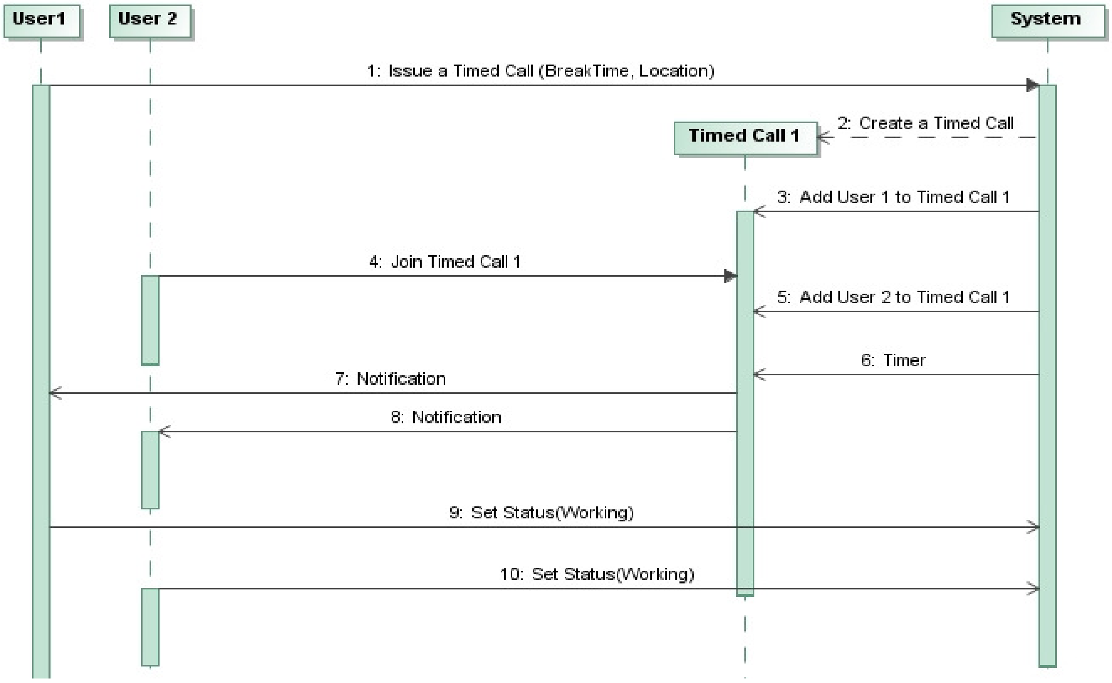
Event flow:

1. U1 issues a Timed Call for a selected time T and a selected Location P, becoming Subscriber S1
2. U2 joins U1's Timed Call before time T, becoming Subscriber S2
3. At the specified time S1 and S2 receive a Notification and have a Coffee Break at Location P. The Call is deleted.
4. When each of the users U1 and U2 return to work, he notifies he's Working.

Post-conditions: U1 and U2 have had a Coffee Break and are Back to Work

Exceptions:

1. S1 or S2 leaves the call. The other Subscriber still goes to the Coffee Break.
2. Both S1 and S2 leave the call. The call is deleted.



## Scenario 2

Name: Simple Shared Call

Participating actors: Users U1, U2, Subscribers S1, S2

Pre-conditions: U1 and U2 are already logged into the system

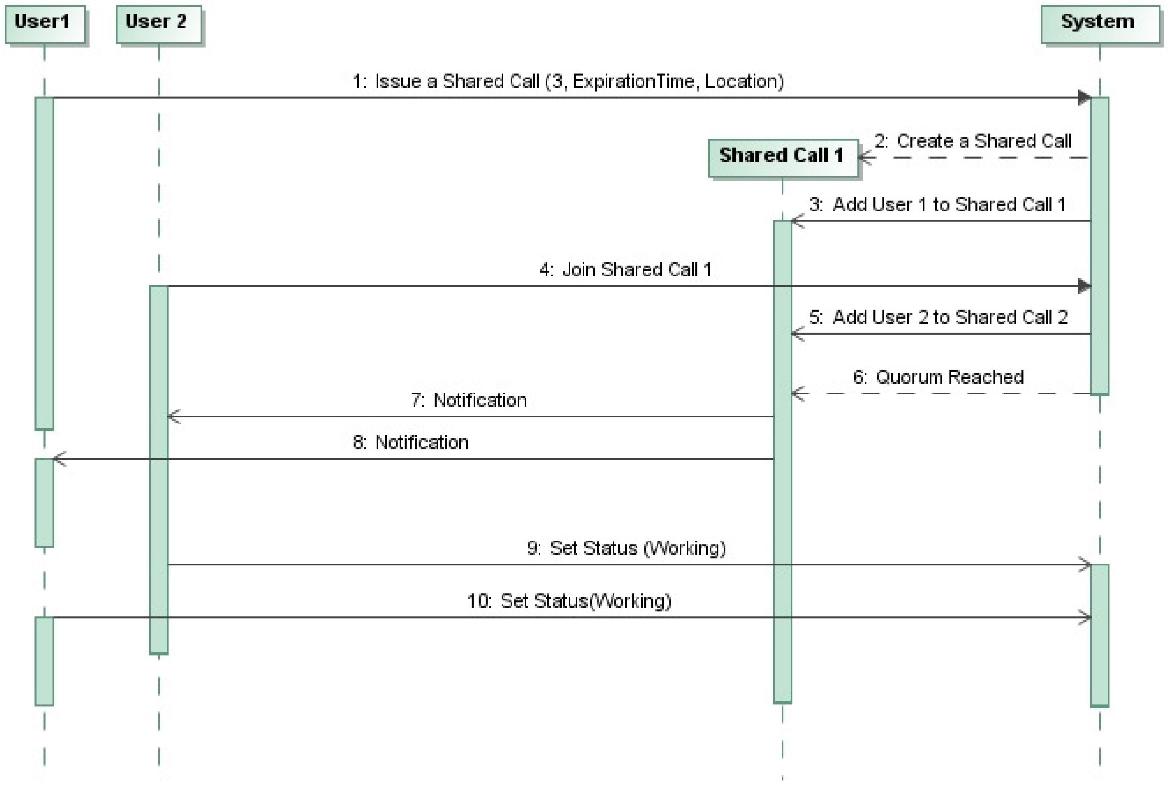
Event flow:

1. U1 issues a Shared Call with a Quorum of 2, Location P and a Expiration Time T, becoming Subscriber S1
2. U2 joins U1's Shared Call at time x before T, becoming Subscriber S2
3. The Quorum is reached, so the system sends a Notification to S1 and S2
4. S1 and S2 have a Coffee Break at place P, the Call is deleted
5. When each of the users S1 and S2 return to work, he notifies he's Working again

Post-conditions: U1 and U2 have had a Coffee Break and are Back to Work

Exceptions:

1. One of the Subscribers, for example S1 sets his status to Busy. The Quorum is not reached until at least one User joins the Call, issues a Compatible Shared Call or has a Coffee Break, increasing the number of Coffee Drinking Users.



## Scenario 3

Name: A Shared and a Timed Call

Description: User U1 issues a Shared Call (3) at Location P, user U2 issues a Timed Call at the same Location, user U3 joins the U1's Shared Call

Participating actors: User U1, U2, U3, Subscribers S1, S2, S3

Pre-condition: U1, U2 and U3 are already logged into the system; no Coffee Drinking Users at Location P

Event flow:

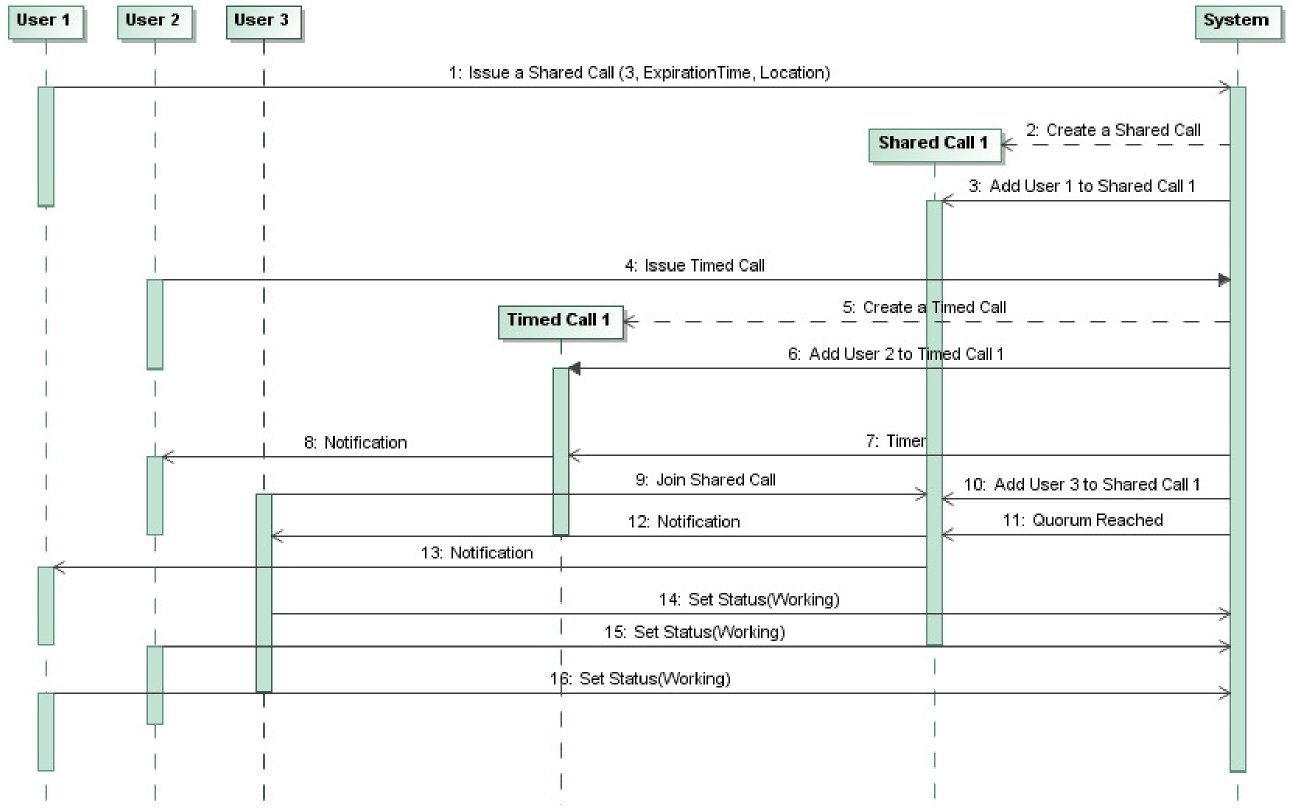
1. U1 issues a Shared Call with a Quorum of 3 and Expiration Time T; U1 becomes Subscriber S1

2. U2 issues a Timed Call with Location P and Break Time x (before T); U2 becomes Subscriber S2
3. At time x S2 receives a Notification for his Timed Call and has a Coffee Break at Location P
4. The Timed Call is deleted
5. U3 joins U1's shared call at time y, before T; U3 becomes Subscriber S3
6. At time y the Quorum is reached, so S1 and S3 receive a Notification
7. S1 and S3 have a Coffee Break in P and the Shared Call is deleted, U2 is still there
8. When each of the users U1, U2, U3 return to work, he notifies he 's Working again

Post-conditions: U1, U2 and U3 have had a Coffee Break and are Back to Work

Exceptions:

1. One of the Subscribers, for example S1 is Busy. The Quorum is not reached until at least one User joins the Call, issues a Compatible Shared Call or has a Coffee Break, increasing the number of Coffee Drinking Users.



## Scenario 4

Name: Two Compatible Shared Calls

Description: User U1 issues a Shared Call (2) for Location P, user U2 issues a Shared Call (2) for the same Location

Participating actors: Users U1, U2, Subscribers S1, S2

Pre-conditions: U1 and U2 already logged into the system; no Coffee Drinking Users at Location P and no other concurrent Calls

Event flow:

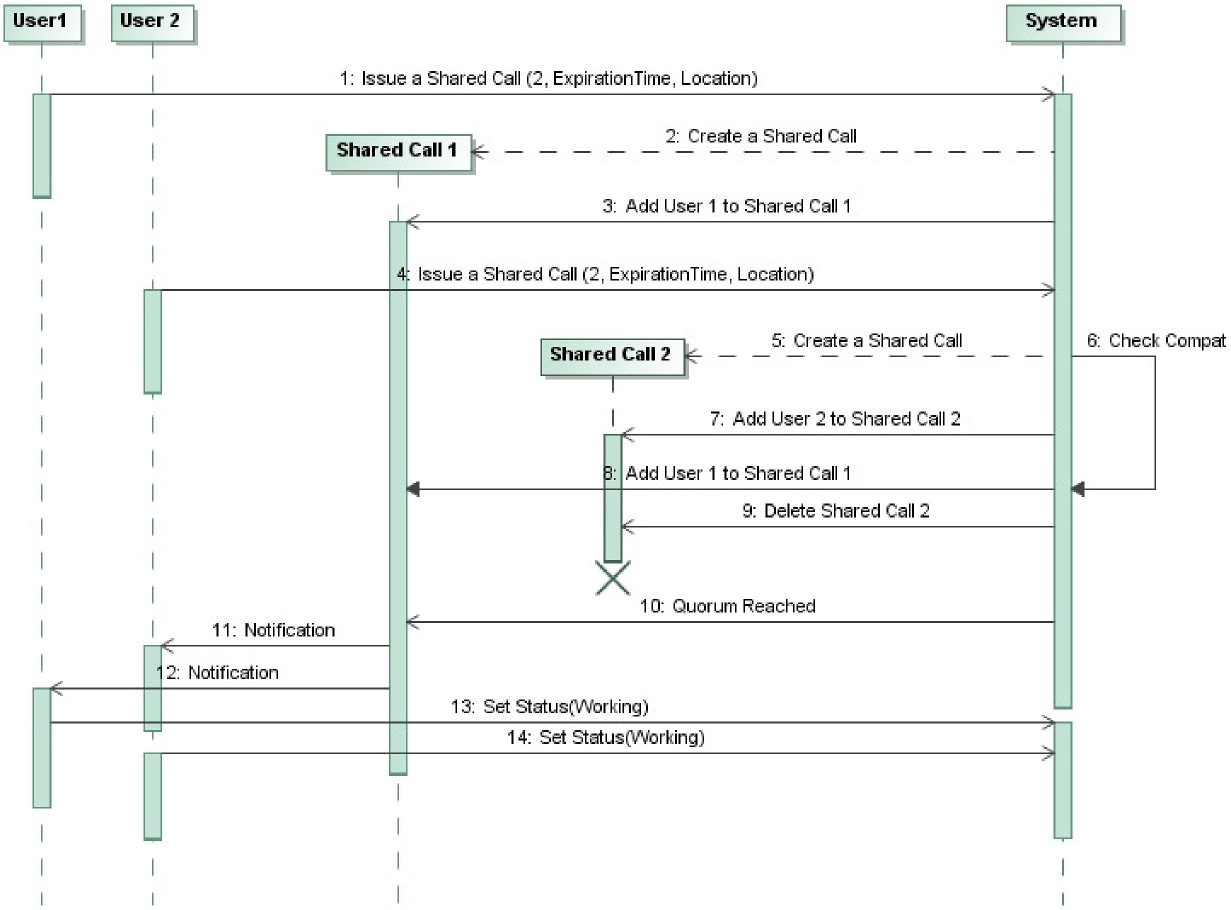
1. U1 issues a Shared Call for Location P (SCP1) with a Quorum of 2 and an Expiration Time T1, becoming Subscriber S1 of SCP1
2. U2 issues a Shared Call for Location P (SCP2) with a Quorum of 2 and an Expiration Time T2, becoming Subscriber S2 of SCP2
3. SCP1 is Compatible with SCP2, both Quorums are completed

4. The system sends to each of S1 and S2 a Notification
5. S1 and S2 have a Coffee Break at Location P and the Call is deleted

Post-conditions: U1 and U2 have had a Coffee Break together and are Back to Work

Exceptions:

1. If SCP1's Quorum is 3, then the two Calls are no longer Compatible, because SCP2's Subscribers do not complete SCP1's Quorum. In this case, no one has Coffee Break.



## Scenario 5

Name: Lost Shared Call

Description: User U1 issues a Shared Call (3) for Location P, user U2 issues a Timed Call for another Location Q and also joins U1's Shared Call, user U3 and user U4 join the U1's Shared Call

Participating actors: Users U1, U2, U3, U4, Subscribers S1, S2, S3, S4

Pre-conditions: U1, U2, U3 and U4 already logged into the system; no Coffee Drinking Users at Location P and no other concurrent Calls

Event flow:

1. U1 issues a Shared Call for Location P (SCP1) with a Quorum of 3 and an Expiration Time T, becoming Subscriber S1
2. U2 issues a Timed Call for Location Q at time x and joins SCP1, becoming a Subscriber S2 of SCP1
3. At time x S2 receives a Notification and has a Coffee Break at Location Q, so he is not an Available Subscriber of SCP1; the Timed Call is deleted
4. U3 joins SCP1 at time y1 before T, thus becoming S3 of SCP1
5. The sum of the number of Available Subscribers and the Coffee Drinking Users at Location P is lesser than the Quorum, so SCP1 is not triggered
6. U4 joins SCP1 at time y2 before T, thus becoming S4 of SCP1

7. The Quorum of SCP1 is reached, so the system sends to each of them a Notification
8. S1, S3 and S4 have a Coffee Break at time z in Location P and SCP1 is deleted
9. S2 comes back to work after time z, but SCP1 is deleted, so he can't join **S**1, S3 and S4 at Location P

Post-conditions: U1, U3 and U4 have had a Coffee Break together and are Back to Work, U2 has had his Coffee Break too.

Exceptions:

1. If U2 returns to work before the sixth point, he becomes an Available User and he is counted in the Quorum, so SCP1 is triggered before time y2
2. If Location Q and Location P are the same, at the eighth point S1, S3 and S4 reach S2 at the Coffee Break. S2 is not counted as an Available User, because his status is not Working, but he is counted as a Coffee Drinking User at that Location.

## Scenario 6

Description: Some Users use DC4C to organize the birthday present for another user of the community

Participating actors: User U1, User U2, User U3, User U4, User U5

Note: X is the U1's birthday

Event Flow:

1. One week before X the system creates a new "call for birthday present"
2. Six days before X, U2,U3, U4 and U5 access to their homepages and control if there are new "calls for birthday present", then U2,U3,U4 and U5 join to the call for U1's birthday
3. Four days before X, U2 proposes a present A that costs Y
4. Four days before X, U3 proposes a present A' that costs Y'
5. Three days before X, U4 votes for the present A.
6. Two days before X, the call for U1's birthday is closed and present A is selected

## Scenario 7

Description: Some Users use DC4C to organize the birthday present for another user of the community but two presents reach the same number of votes

Participating actors: User U1, User U2, User U3, User U4, User U5, User U6

Note: X is the U1's birthday

Event Flow:

1. One week before X the system creates a new "call for birthday present"
2. Six days before X, U2, U3, U4, U5 access to their homepages and control if there are new "calls for birthday present", then U2 and U3 join to the call for U1's birthday
3. Five days before X, U6 accesses to his homepage and controls if there are new "calls for birthday present", then U6 joins to the call for U1's birthday
4. Four days before X, U2 proposes a present A that costs Y
5. Four days before X, U3 proposes a present A' that costs Y', with Y'>Y
6. Three days before X, U4 votes for the present A and U5 votes for the present A'
7. Two days before X, the call for U1's birthday is closed. The present A and the present A' have received the same number of votes, so the presents A is selected because is cheaper.

## Scenario 8

Description: Some Users use DC4C to organize the birthday present for another user of the community, one present receives the absolute majority of votes before the closing of the birthday call

Participating actors: User U1, User U2, User U3, User U4, User U5, User U6, User U7, User U8

Note: X is the U1's birthday

Event Flow:

1. One week before X the system creates a new “call for birthday present”
2. Six days before X, U2, U3 and U8 access to their homepages and control if there are new “calls for birthday present”, then U2, U3 and U8 join to the call for U1's birthday
3. Five days before X, U4,U5, U6 and U7 access to their homepages and control if there are new “calls for birthday present”, then U4, U5, U6 and U7 join to the call for U1's birthday
4. Four days before X, U2 proposes a present A that costs Y
5. Four days before X, U3 proposes a present A' that costs Y'
6. Three days before X, U4 and U5 vote for the present A.
7. Three days before X, U6 votes for the present A
8. After the U6's vote, the call for U1's birthday is closed because present A has already received the absolute majority of the votes. Present A is selected.

## Scenario 9

Description: Some Users use DC4C to organize the birthday present for another user of the community, but nobody proposes a present

Participating actors: User U1, User U2, User U3

Note: X is the U1's birthday

Event Flow:

1. One week before X the system creates a new “call for birthday present”
2. Six days before X, U2 and U3 access to their homepages and control if there are new “calls for birthday present”, then U2 and U3 join to the call for U1's birthday
3. Two days before X, the call for U1's birthday is closed without selecting a present.

## Scenario 10

Description: Some Users use DC4C to organize the birthday present for another user of the community, two users modify their votes

Participating actors: User U1, User U2, User U3, User U4, User U5, User U6

Note: X is the U1's birthday

Event Flow:

1. One week before X the system creates a new “call for birthday present”
2. Six days before X, U2, U3, U6 and U5 access to their homepages and control if there are new “calls for birthday present”, then U2, U3, U6 and U5 join to the call for U1's birthday
3. Five days before X, U4 accesses to his homepage and controls if there are new “calls for birthday present”, then U4 joins to the call for U1's birthday
4. Four days before X, U2 proposes a present A that costs Y
5. Four days before X, U4 votes for the present A.
6. Three days before X, U3 proposes a present A' that costs Y'
7. Three days before X, U2 modifies his vote from A to A'

8. Three days before X, U4 modifies his vote from A to A'
9. After the U4's vote, A' reaches the absolute majority of votes

# ALLOY models

In order to keep the models simple, we decided to develop two separate models in Alloy, one per each macrofunctionality the system has to provide.

## Coffee Calls Model

### Static Model

We first define all the atoms composing our system: CoffeeCalls (Timed and Shared) and Users. We add some additional signatures that describe the primitive types in our systems, such as Strings and Locations, and a signature Status, that will be linked to the Users.

```
module coffeecalls

sig String{}
sig Location {}

abstract sig Status {}
one sig Busy, Working extends Status{}

sig User {
        id: one String,
}

abstract sig CoffeeCall{
        located: one Location,
}

sig TimedCall extends CoffeeCall{
        Breaktime: one Int
}

sig SharedCall extends CoffeeCall{
        Quorum: one Int,
        ExpirationTime: one Int

}{Quorum>1
   ExpirationTime>1
}
```

Then we define the relationships between them. After some experiments we decide to use a signature System, describing the current status of the system. The set of subscribers of each Call is defined in the System as it can change in different Systems; the same for the status of the User.

```

//the signature designing the current status of the system
abstract sig System{

        registeredusers: set User,                          //set of all the users registered into the system
        loggedusers: set User,                              //set of all the users logged into the system
        activecalls: set CoffeeCall,                        //set of all the calls active in the system
        currentTime: one Int,                               //current time of the system
        status: loggedusers -> one (Status+Location),       //status of each loggeduser…Working, Busy, Location1
        subscribers:CoffeeCall lone->set User,              //all the subscribers of each CoffeeCall
        sub_available: CoffeeCall lone->set User            //all the working subscribers of each CoffeeCall
}
```

```
{          currentTime>0

           //if a CoffeeCall is active in this system, then it must have at least a subscriber
           all c: activecalls|#(c.subscribers)>0

          //all the working (that means not busy or alreadya t the coffee) subscribers must be obviously subscribers
           all c: CoffeeCall|c.sub_available in c.subscribers
           all c:CoffeeCall, u:c.subscribers|(u.status=Working)<=> (u in c.sub_available)

          //if a CoffeeCall is not active in this system, then it must have at no subscribers
           all c: CoffeeCall| (c not in activecalls)=>(#(c.(subscribers))=0)
}

one sig System0, System1 extends System{}
```

Now we have to constrain our entities to some basic rules. We divide the rules in chunks of facts, one per each entity.

```
fact CoffeeCalls{

//All active calls must have a Break/Expiration time after the currentTime of their system
all s:System, t: ((s.activecalls)&TimedCall)|t.Breaktime>=s.currentTime
all s:System, sh: ((s.activecalls)&SharedCall)|sh.ExpirationTime>=s.currentTime

//No timed call with the same time and the same place
all s, t: TimedCall |( t.located = s.located&& t.Breaktime = s.Breaktime) => s=t

//No coffee call with more subscribers than the quorum
all s:System, c: s.activecalls|#(c.(s.sub_available)) <=c.Quorum

}

fact Users{

//All CoffeeCalls must be activecalls in at least one System
all c: CoffeeCall, disj s, s': System| c in (s.activecalls+s'.activecalls)

//All the users in all systems must be registered
all u:User, s: System|u in s.registeredusers

//All users must have different ids
all s: System| no disj r1,r2: s.registeredusers| r1.id= r2.id

//If a user is a subscriber of an activecall in the system, then he must be logged
all s : System, c: s.activecalls, u: c.(s.subscribers)|u in s.loggedusers

}
```

## Dynamic Model

The way we defined our static model was significantly biased by the fact we were planning a dynamic evolution of our system. For this purposes we declare an important predicate, Triggerable, which is true for all the CoffeeCalls that can be started in the specific System.

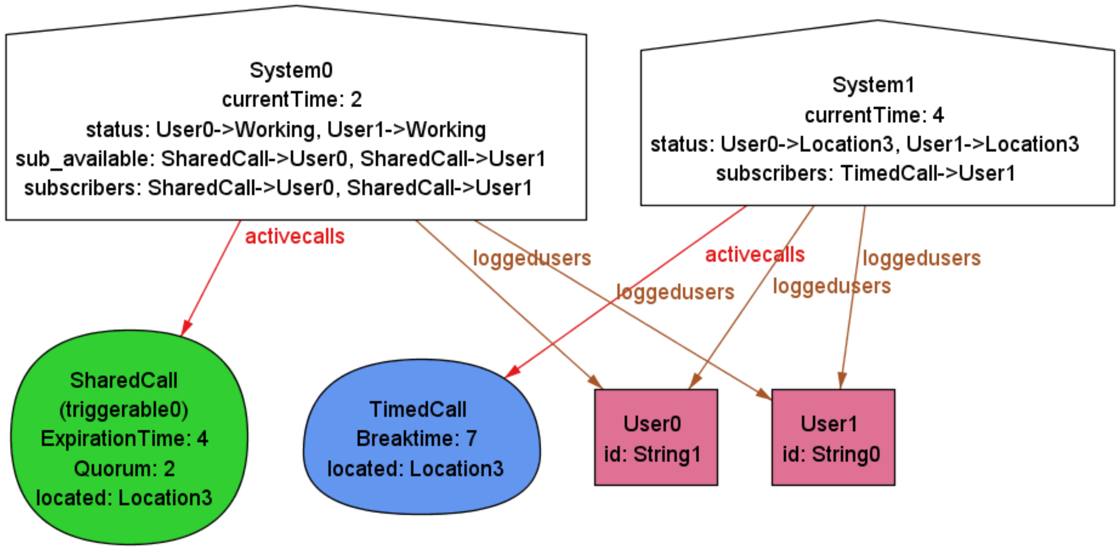
```
//a timed call is triggerable, if it is an active call and its Breaktime is equal to the current time of the system
pred TriggerableT(s:System, t:TimedCall){
(t=s.activecalls) && (t.Breaktime= s.currentTime)
}

//a shared call is triggerable, if there is at least a Quorum number of available subscribers
pred TriggerableS(s:System, t:SharedCall){
(t=s.activecalls) &&(t.Quorum= #(t.(s.sub_available)))
}

pred Triggerable(s:System, t:CoffeeCall){
#(t & SharedCall)>0 => TriggerableS[s,t] else TriggerableT[s,t]
}
```

The idea of a previous and a current state is modeled by restricting our model to two states and adding some additional constraints. The model we have built can now predict the behavior of the system in time.

```
fact Systems{
//System0 is before System1
System0.currentTime<System1.currentTime

//all calls that can be triggered in System0 are taking place in System1
all u:User, c: CoffeeCall|(Triggerable[System0,c] && (u in c.(System0.sub_available)) )=> u.(System1.status) = 
c.located
}
```

We define a pred show to have some visual representation of the system.

```
pred show{

//not everybody working, but at least one :)
some u:User| #(u.(System.status)&Location) >0
some u:User| #(u.(System.status)&Working) >0

//at least one SharedCall and one TimedCall
some c:CoffeeCall| #(c&SharedCall)>0
some c:CoffeeCall| #(c&TimedCall)>0

//at least a triggerable call in System0
some triggerable0:SharedCall | TriggerableS[System0,triggerable0]
}
```

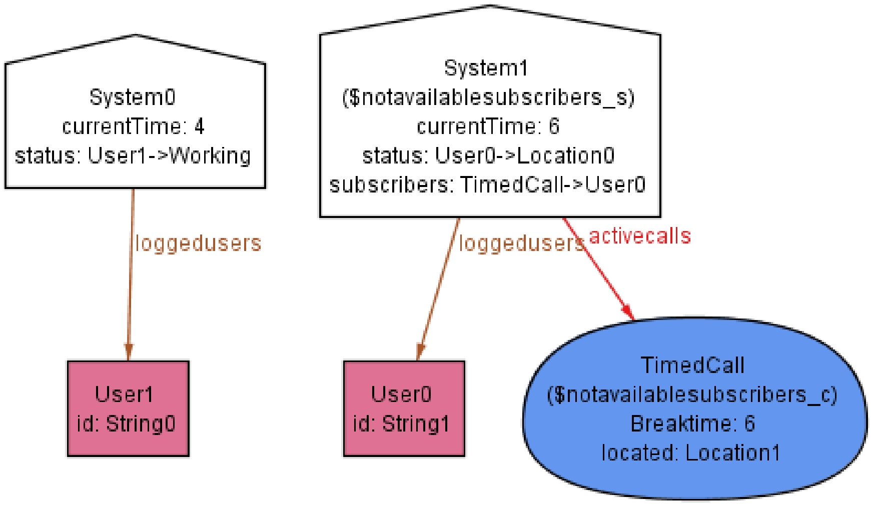


We can see that in System0 there is a triggerable call named SharedCall, that has Quorum 2 and two available subscribers User0 and User1 and is located in Location3. In System1 both of these Users are in Location3.

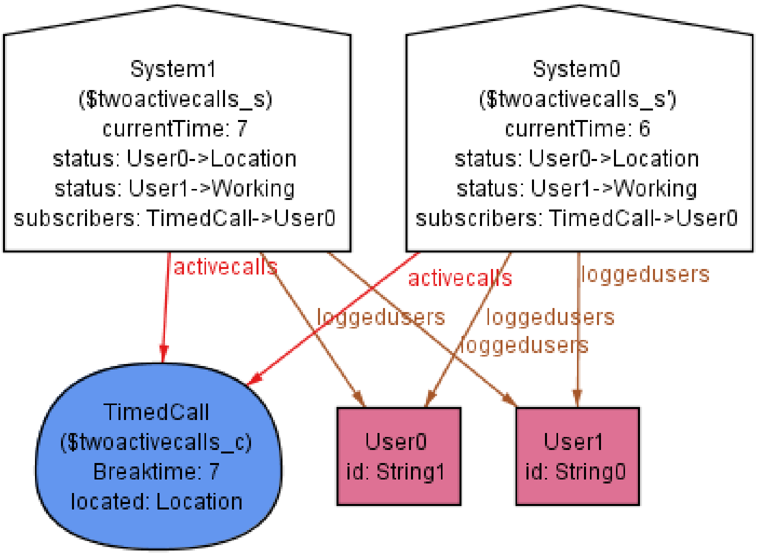
## A few assertions

We define some assertions to test if the system is behaving correctly. First we check if there is the possibility to have a non available subscriber, as it should be. Next we will check if a call can be active in two systems.

```
assert notavailablesubscribers{

//are there some subscribers who are not available?
all c: CoffeeCall, s : System| c.(s.subscribers)=c.(s.sub_available)

}
check notavailablesubscribers
```



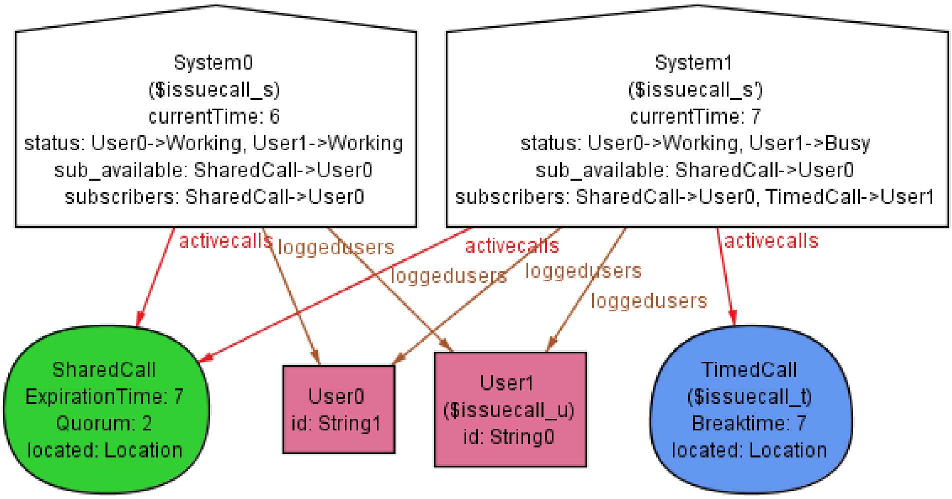
```
assert twoactivecalls{

no c: CoffeeCall, disj s, s' : System|  (c in s.activecalls) <=>(c in s'.activecalls)

}

check twoactivecalls
```



## Operations

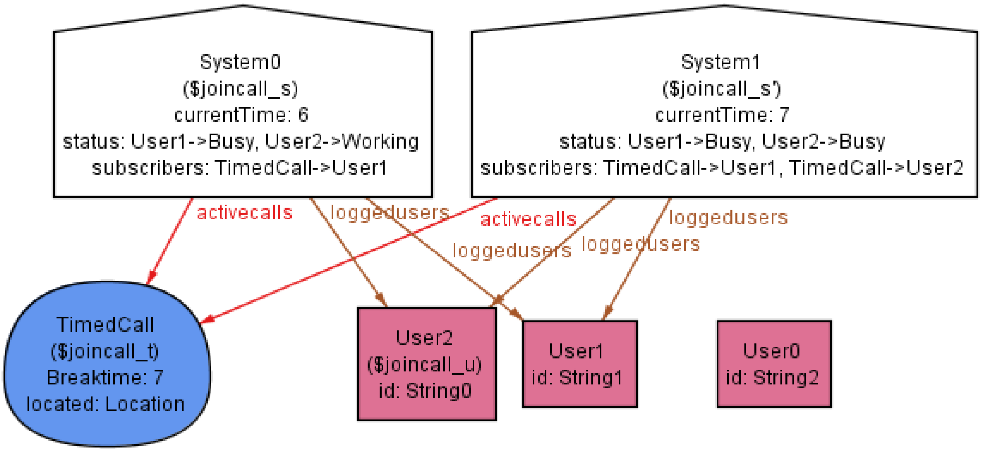
We test our model by defining some of the most important operations in our system: Issue Call, Join Call and SetStatus.

```
pred issuecall(s, s': System, t: CoffeeCall, u: User){

// precondition
t not in s.activecalls
u in s.loggedusers
s = System0

//postcondition
s'.activecalls = s.activecalls + t
s'.subscribers= s.subscribers + t->u

}
```



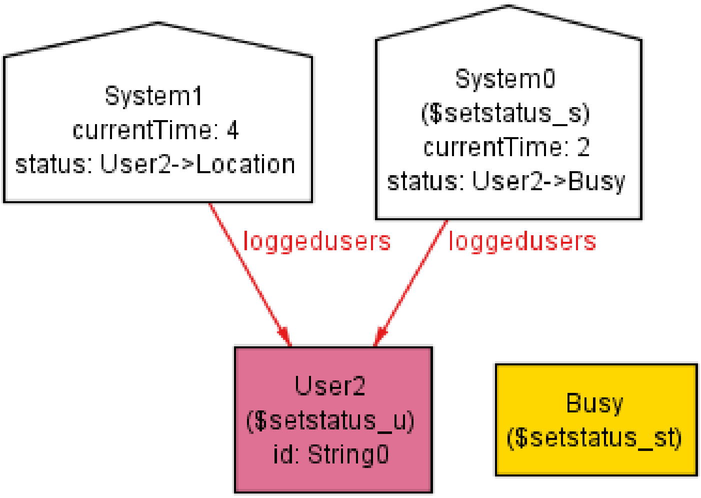
We can see that User1 issued the TimedCall, that was not existing in System0 and in System1 he is a subscriber of it.

```
pred joincall(s, s': System, t: CoffeeCall, u: User){

// precondition
u in s.loggedusers
u not in t.(s.subscribers)
t in s.activecalls
t in s'.activecalls
s = System0

//postcondition
s'.subscribers= s.subscribers + t->u

}
```



User2 joined the existing TimedCall, and now he is a subscriber of it.

```
pred setstatus(s: System,u: User, st:Status){

// precondition
u in s.loggedusers

//postcondition
u.(s.status) = st

}
```



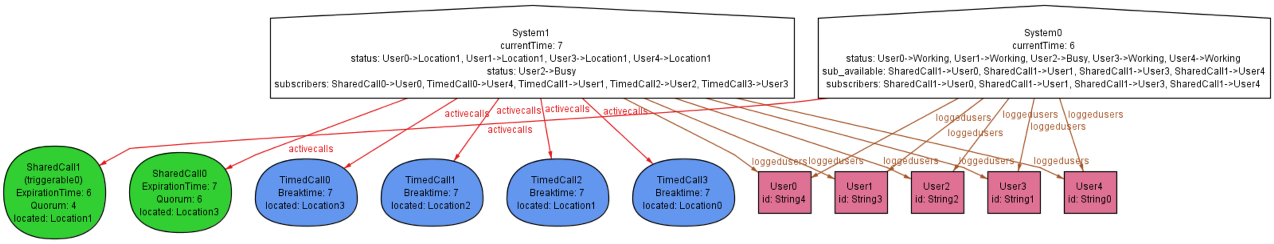
User2 set his status to Busy, so in System0 he is Busy. In this predicate we don't distinguish if System0 is before System1.

## More complexity

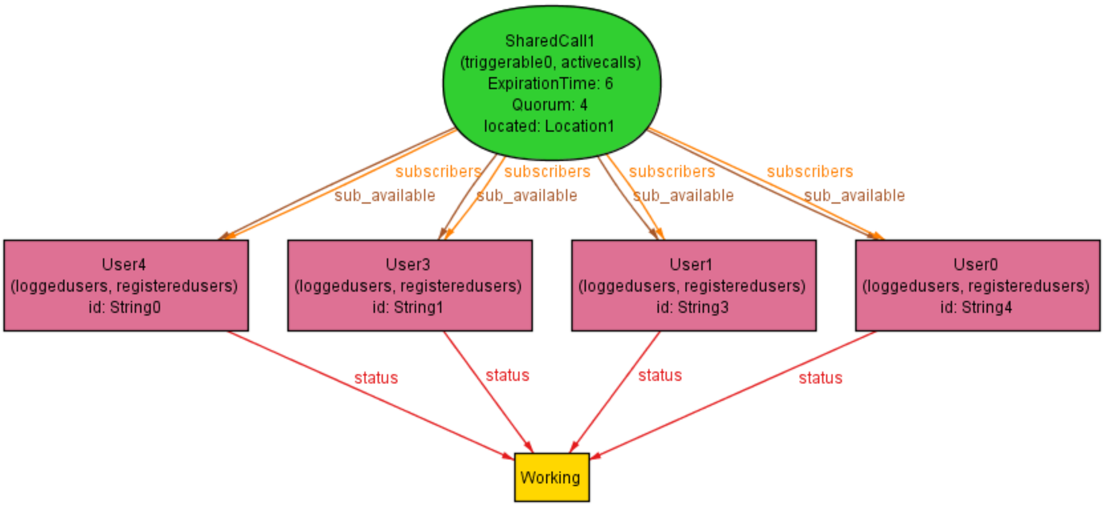
What happens if we add some cardinality constraint to our basic predicate show like #User>4 and #CoffeeCall>5 and then write a command like this?

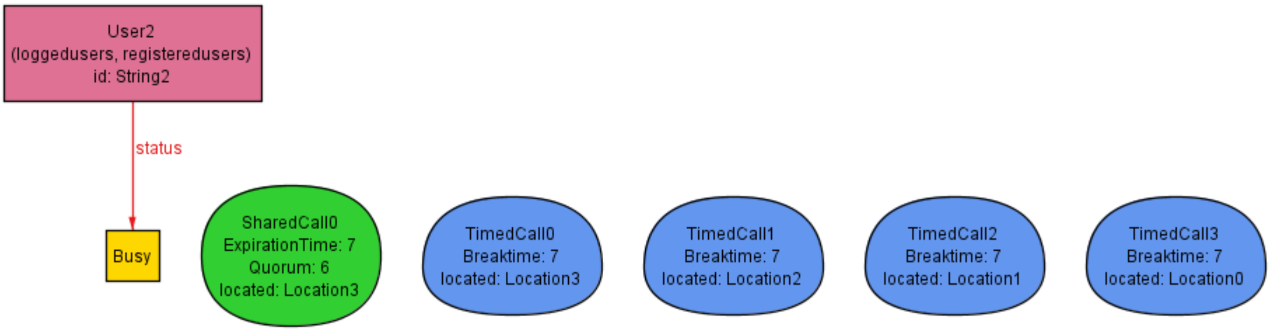
```
run show for 20
```

We obtain a really complex model, like this:

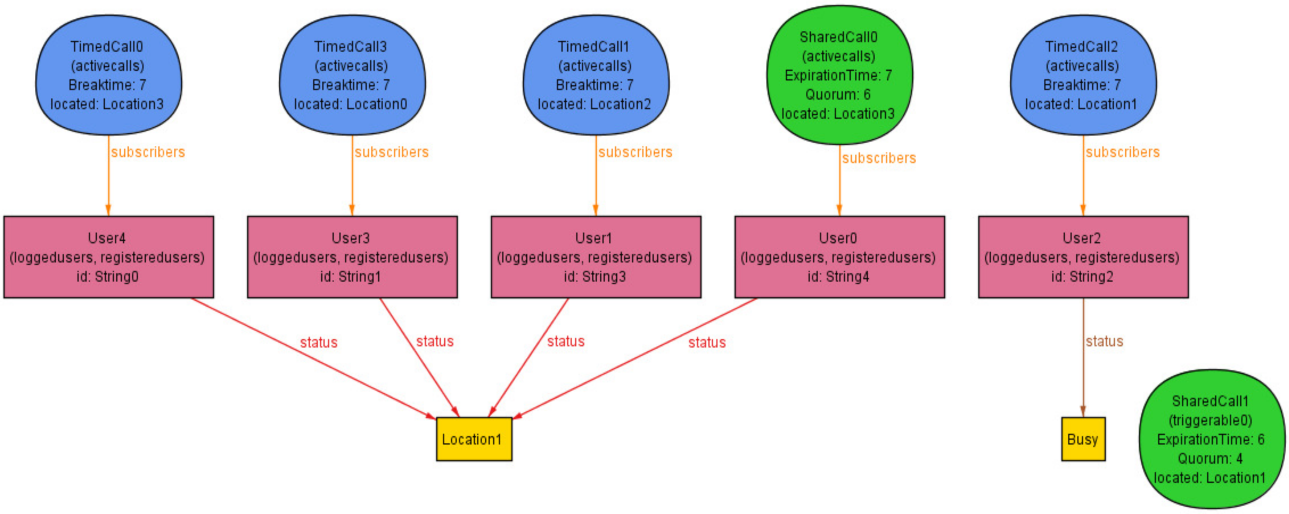


The only way to manage complexity in this case is to project the model over the System entity. In this case we obtain two different graphs, one per each System. For istance we will show the graph of System0:





This is the graph for System1. We can notice that the SharedCall1 which was triggerable in System0 “sent” all its available subscribers (User0, User1, User3 and User4) in Location1.



## Organize Birthday Presents Model

As before, we first define the entities we already described in the class diagram: USER, BIRTHDAY_CALL and PRESENT.

```
sig string {} // generic string
sig date{} // generic date
sig double{} // generic double value

// only two possible status for the birthday call: open or close
sig opn {}
sig close {}


// entities in the conceptual model

sig USER {
          username: string,
          birthday: lone date,
          partecipate: set BIRTHDAY_CALL
}

sig PRESENT {
          price: double,
          description: string,
          proposed_by:one USER,
          proposed_for:one BIRTHDAY_CALL,
          votedby: set USER
}

sig BIRTHDAY_CALL {
```

```
            status: one  (opn+close),
            birthdayof: one USER,
}
```

Next we constrain our entities with a set of rules.

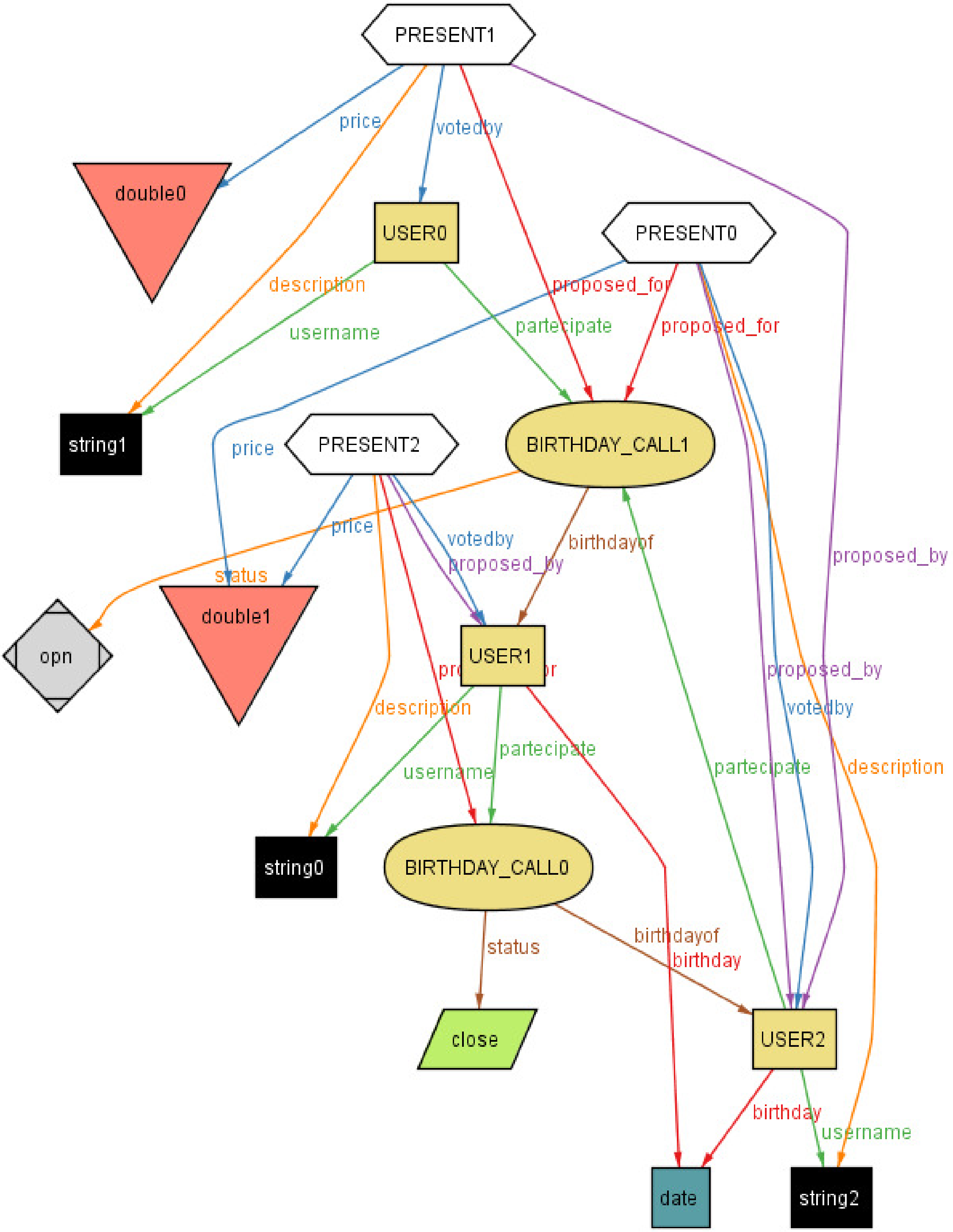
```
fact RULES{

// no two users with the same username
            no disj u1, u2: USER | u1.username = u2.username

//there is no user participating to his own birthday call
             no u: USER, b: BIRTHDAY_CALL | u in b.birthdayof and b in u.partecipate

// there is exactly one birthday call for each user's birthday
            (all u:USER | #u.birthday =1 implies #(birthdayof.u)=1)
            (all u:USER | #u.birthday =0 implies #(birthdayof.u)=0)

//only user participating to a call can propose a present for the call
            (no u:USER, p:PRESENT | u in p.proposed_by and no(u.partecipate & p.proposed_for))

// only user participating to a call can vote for a present proposed for the call
            (no u:USER, p:PRESENT | u in p.votedby and no(u.partecipate & p.proposed_for))

// if an user proposes a present than he has to vote for at least a present of the call
            (all u:USER, p:PRESENT | u in p.proposed_by => u in (proposed_for.(PRESENT.proposed_for &
p.proposed_for)).votedby)

// an user can vote at most one time for a specific birthday_call
            (all u:USER, b:BIRTHDAY_CALL | b in u.partecipate implies #(proposed_for.b & votedby.u)=1)
}


fact cardinalities{
            #opn = 1
            #close = 1
}
```

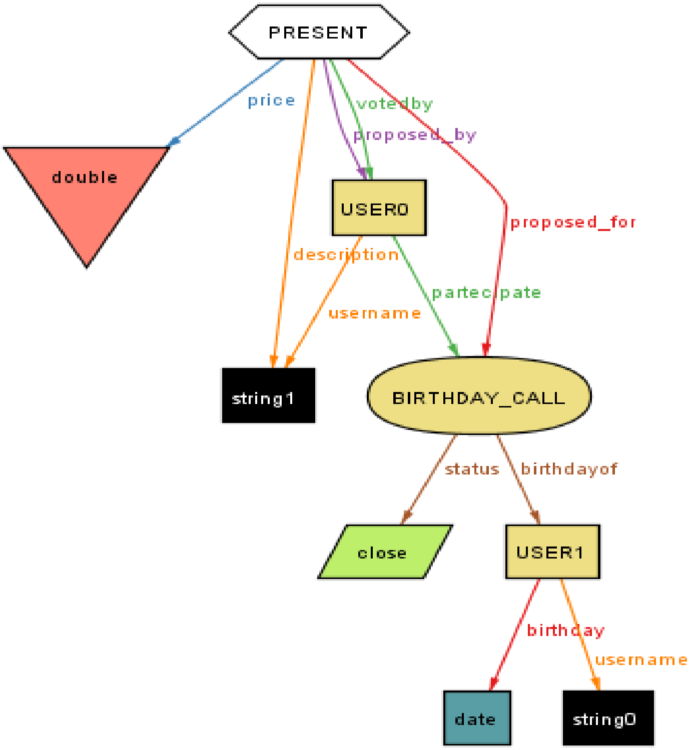
Now we have to check if the model is consistent. We define the predicate show in order to have a non-trivial visual representation of our model.

```
pred show() {
            #PRESENT >2 // at least three present
            #BIRTHDAY_CALL>1 // at least two birthday call
            #USER >2 // at least three user
}

// an user participating to more than one birthday call
pred umbc() {
            one u:USER | #(u.partecipate)>1
}

// check for error in the model. Is it possible to participate to a birthday call without specifying his own birthday?
pred partec() {
            one u:USER | some(PRESENT.proposed_by & u) and some(PRESENT.votedby & u)  and no(u.birthday)
}

// it's impossible to vote more than one time and participate to only one birthday call
// so it's impossible to vote two or more time in a specific birthday call
assert morevote {
            no u:USER | #votedby.u=3 and one(u .partecipate)
}

check morevote for 3 // no counterexample found!!
run partec for 3         // istance found
run show for 3           // istance found
run umbc for 3           // istance found
```

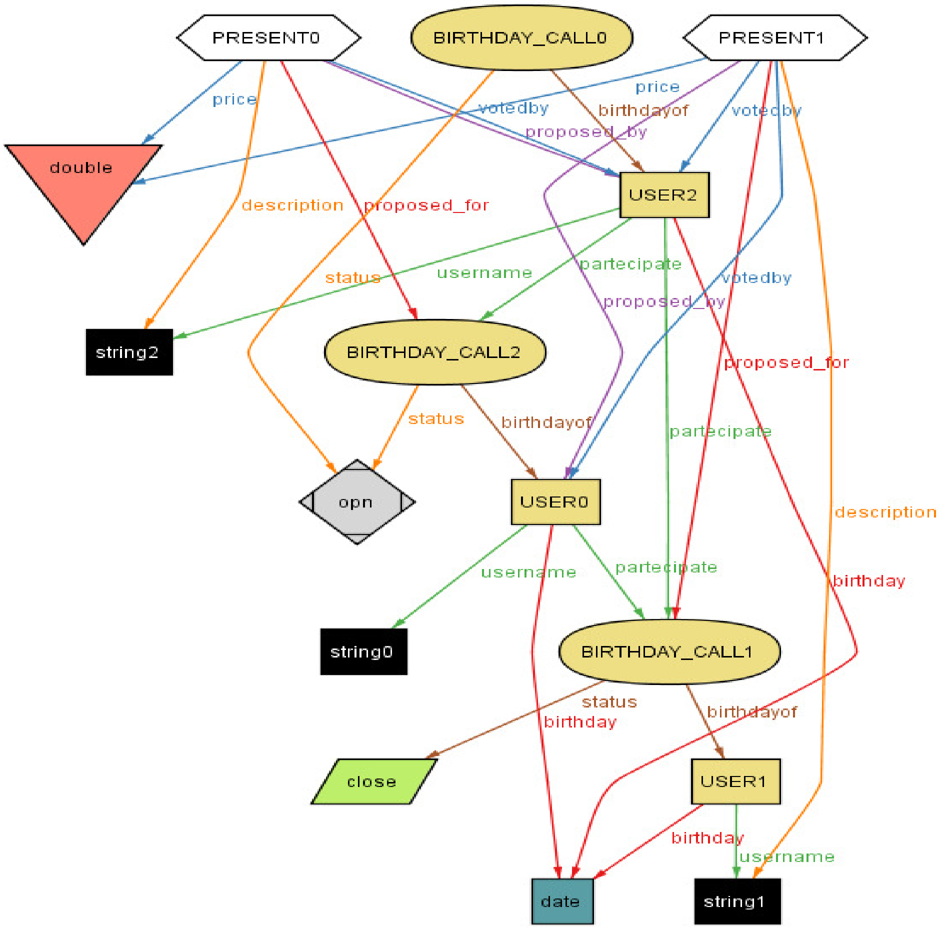
## RUNNING SCREENSHOT

We present some screenshots of the Alloy analyzer for each of the predicates we already mentioned. For instance a possible universe generated by the Alloy analyzer from the model running *“run show for 3”.*





Another possible universe generated by the Alloy analyzer from the model running *"run partec for 3"*



A possible universe generated by the Alloy analyzer from the model running *"run umbc for 3"*