

# **Project 1 - Gruppe 2**

## **Members:**

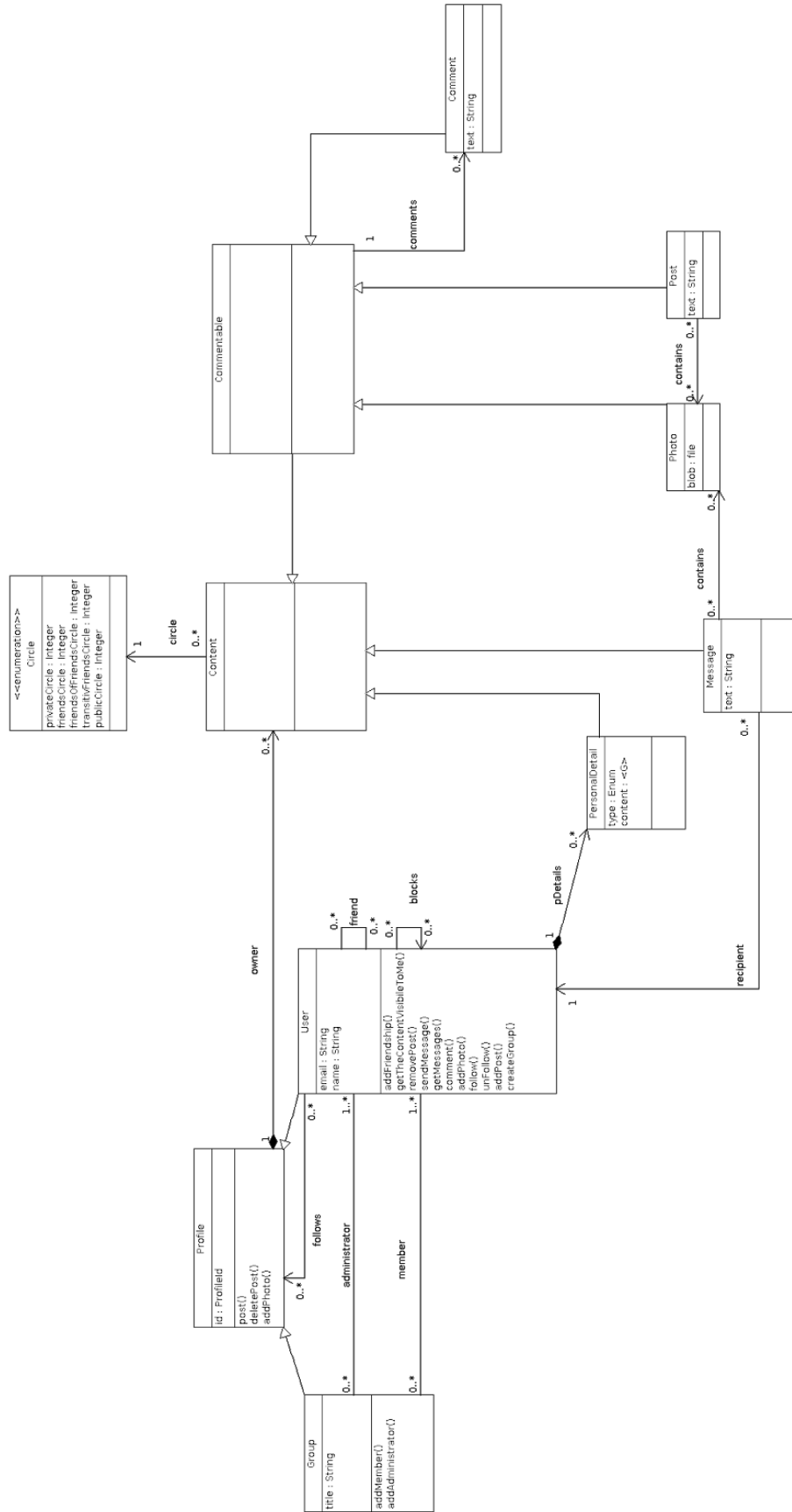
Valentin Trifonov  
Simon Ringeisen  
Jan Eberhardt

## **Assistant:**

Pavol Bielik

## **UML Class Diagram**

See next page....



## List of details that cannot be expressed by the UML model

- Comments are always owned by a User and not by a Group
- Any content posted by a group is always private (with respect to the group), so only members of the group can see it.
- If a user posts to a group, the owner of the post is the group
- Every admin of a group has to be a member too
- A user cannot block himself
- Friendship is symmetric but not reflexive
- Users cannot follow themselves
- Comment-chains have to have a non-comment content as a root
- The receiver and sender of

## Alloy Model

open util/integer

-----  
 -- Profile Signatures --  
 -----

abstract sig Profile {}

sig Group extends Profile {  
     administrator: some User,  
     member: some User  
 }

sig User extends Profile {  
     follows: set Profile,  
     friend: set User,  
     blocks: set User,  
     --canSee: set Content, -- for debug purposes - see below  
     pDetails: set PersonalDetail  
 }

-- Add this line (and the one in the user signature) to get arrows for the content each user can see:  
 --fact validCanSee {all u: User | all c: Content | c in u.canSee <=> canSee[u, c]}

-----  
 -- Profile Facts --  
 -----

fact friendship {all u1:User | all u2:u1.friend | u1 in u2.friend} -- friendship is symmetric  
 fact friendshipNonReflexiv {no u: User | u in u.friend}  
 fact blocks{no u:User | u in u.blocks} -- users cannot block themselves  
 fact follows {no u:User | u in u.follows} -- users cannot follow themselves  
 fact personalDetail {all pd: PersonalDetail | all p: Profile | pd in p.pDetails <=> pd.owner = p} -- Each PersonalDetail must be connected to exactly one user  
 fact administratorIsMember {all g:Group | g.administrator in g.member}  
 fact oneAdmin {all admin:Group.administrator | #{admin} > 0} -- There must be at least one administrator

-----  
 -- Content Signatures --  
 -----

```

-----

abstract sig Content{
    circle: one Int,
    owner: one Profile
}

abstract sig Commentable extends Content{
    comments: set Comment
}

sig Post extends Commentable{
    contains: set Photo
}

sig Photo extends Commentable{}

sig Comment extends Commentable{}

sig PersonalDetail extends Content{}

sig Message extends Content{
    recipient: one User,
    contains: set Photo
}

-----
-- Content Facts --
-----

fact message {all m:Message | m.recipient != m.owner}
fact commentCommentsOnOneThing{all com:Comment | {one con:Content | com in con.comments}} -- a comment belongs to
exactly one content
fact commentChainCannotStartWithComment{all com:Comment | one con:(Content-Comment) | com in con.^comments} -- at
the root of a comment chain there has to be a non-comment content
fact groupsDontPostComments{all c:Comment | no g:Group | c.owner = g}

-- posts created by groups count either as "private" (the group members, and only they, can see them) or as "public" (anyone
can see them).
fact groupPostPrivacy{all c:Content | c.owner in Group => (c.circle = 1 or c.circle = 5)}

-- the circle enum has numbers from 1 to 5, representing private/friends/friends of friends/friends chain/public, respectively
fact validCircle {all c: Content | c.circle >= 1 and c.circle <= 5}

-- note: a post CAN contain photos which are not visible to its owner (for example if their privacy setting has been changed since
the post has been created)

-----
-- Predicates --
-----

pred checkCirc3 {
    #{c: Content | c.circle != 3} <= 2 and #{c: Content | c.circle = 3} >= 2 and
    #{User} = 7 and #{PersonalDetail} = 1 and #{Post} = 5 and #{Photo} = 2 and {all u:User | #{u.friend}=2}
}

run checkCirc3 for 15

```

```

pred showSomeComments {
    #{Comment}>3 and #{Comment.comments}>0
}
run showSomeComments for 10

pred personalDetails {
    #{PersonalDetail} >= 5 and #{User} >= 3 and #{Group} >= 2 and {all u: User | #{u.friend} >= 1 and #{u.blocks} = 0}
}
run personalDetails for 9

-----
-- Assertions --
-----

check pDBelongsToOneUser {all disj u1,u2: User | no upd:u1.pDetails | upd in u2.pDetails}--Two user cannot have the same
personal detail
check twoContentsCannotHaveSameComment {all disj c1,c2: Content | no com:c1.comments | com in c2.comments}--Two
Contents cannot have the same comment
check groupsDontPostPD {all g:Group | all pd: PersonalDetail | not g in pd.owner}

-----
-- Exercises --
-----

-- Task C --

pred canSee[u: User, c: Content] {
    (
        -- circle logic
        ((c.circle = 1 => (u = c.owner) or (u in c.owner.member))
        and (c.circle = 2 => (u in c.owner.friend or u = c.owner))
        and (c.circle = 3 => (u in c.owner.friend.friend or u in c.owner.friend or u = c.owner))
        and (c.circle = 4 => (u in c.owner.*friend or u = c.owner)))
        or (c.circle = 5) -- anyone can see public content
        or (u = c.recipient) -- message recipients can always see the message
        ) and (not u in c.owner.blocks) -- EXCEPT if the content owner blocked them
    )
}
run canSee for 5

pred canModify[u: User, c: Content] {
    {u in c.owner} or {u in c.owner.administrator}
}
run canModify for 3

pred isOnNewsFeed[u: User, c: Content] {
    {canSee[u,c]} and {c.owner in u.follows}
}
run isOnNewsFeed for 3

-- Task D --

--1
check commentChainsAcyclic{all c:Comment | c not in c.^comments}

--2
check canModifyOnlyWhatCanSee{all u: User | all c: Content | canModify[u,c] => canSee[u,c] }

```

```

--3
-- This property is not true, as a user can post a content to a group while not being an administrator
-- and thus not being allowed, to change the content.

--4
-- This Property is not true, as the Circle for the Photo can be more restrictive, than the Circle of the Post itself.

--5
check groupHasMembers{no g:Group | #{g.member}=0}

--6
check allNewsFeedContentsVisible{all u:User | all c:Content | isOnNewsFeed[u,c] implies canSee[u,c]}

--7
check cannotSeeContentByAnyoneWhoBlockedThem {all u:User | all c:Content | u in c.owner.blocks => not canSee[u,c]}

-- Task E --

--1
pred showChainOfSizeFive {one c:Comment | #{c.comments.comments.comments.comments} > 0}
run showChainOfSizeFive for 6

--2
pred threeUsersSevenGroups {#{User}=3 and #{Group}=7 and {all disj g1,g2:Group | g1.member != g2.member}}
run threeUsersSevenGroups for 10

--3
pred fourUsersOneFriendNotTransitiv{#{User}=4 and {all u:User | #{u.friend}>0} and {some u1:User | some u2:(User - u1) | u1
not in u2.^friend}} -- {one u1,u2:User | u1 != u2 and u1 not in u2.^friend }}
run fourUsersOneFriendNotTransitiv for 10

--4
pred weird{some u1:User | some u2:u1.friend | some u3:u2.friend | some c:Post | some p:Photo | (#{Group} = 0) and u3 in
c.owner and u3 not in u1.friend and not u3 = u1 and c.circle = 3 and canSee[u1,c] and p in c.contains and p.owner in {User - u1
- u2 - u3}}
run weird for 10
-- added the group limitation for clarity

--5
pred photoWithPhotoNotByPoster{ #{Post} = 1 and #{Post.contains}=1
    and {some u:User | some post:Post | some photo:Photo | photo in post.contains
        and u not in photo.owner
        and u in post.owner
        and photo.circle != 5
        and photo.owner not in Group
        and post.owner not in Group} }
run photoWithPhotoNotByPoster for 3

--6
pred friendOfFriendPostWithPhoto{#{Post} = 1 and {some post:Post | some photo:Photo | some fr:post.owner.friend | post.circle
= 3 and photo in post.contains and photo.owner in post.owner.friend and canSee[fr, post] and not canSee[fr, photo]}}
run friendOfFriendPostWithPhoto for 5

```

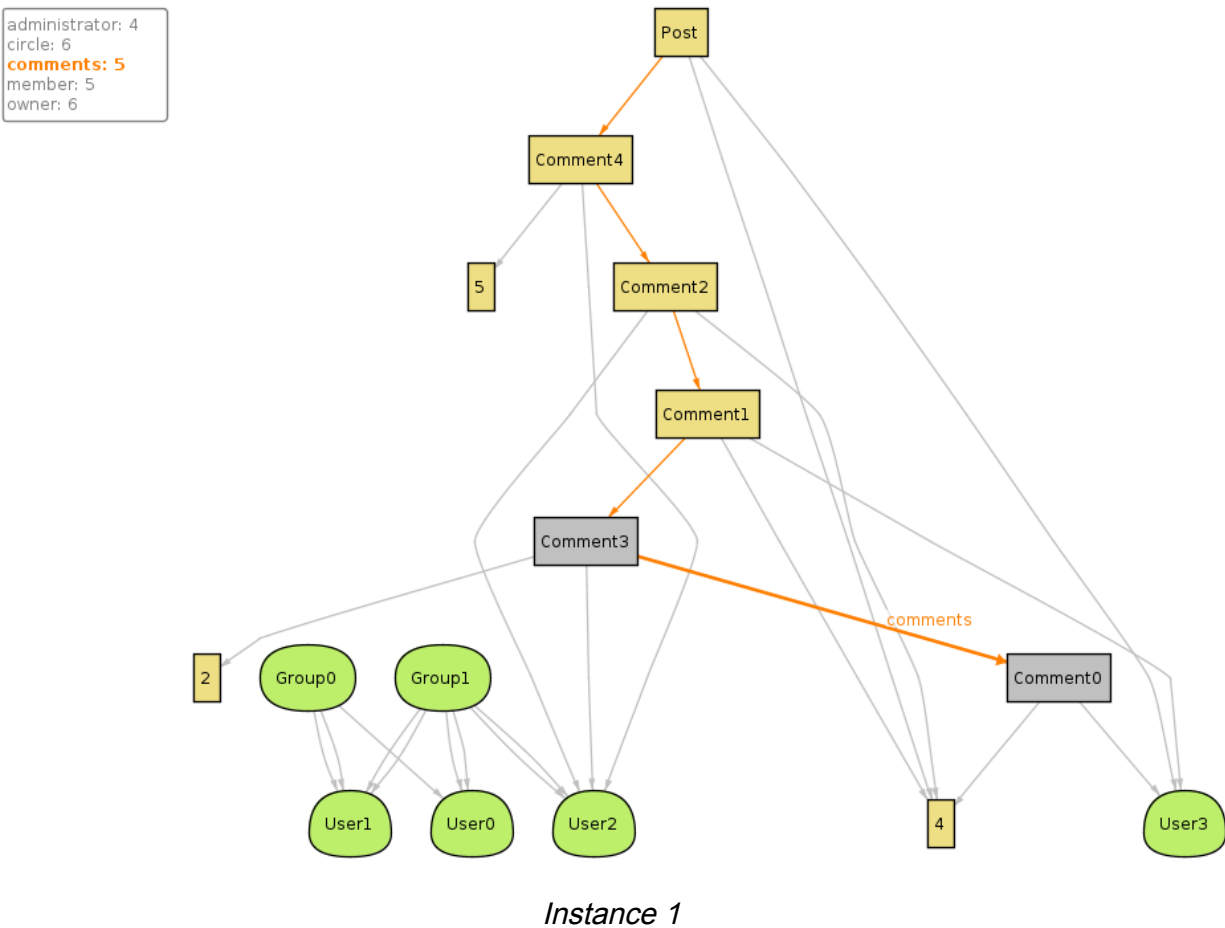
### List of Properties from task D unable to check

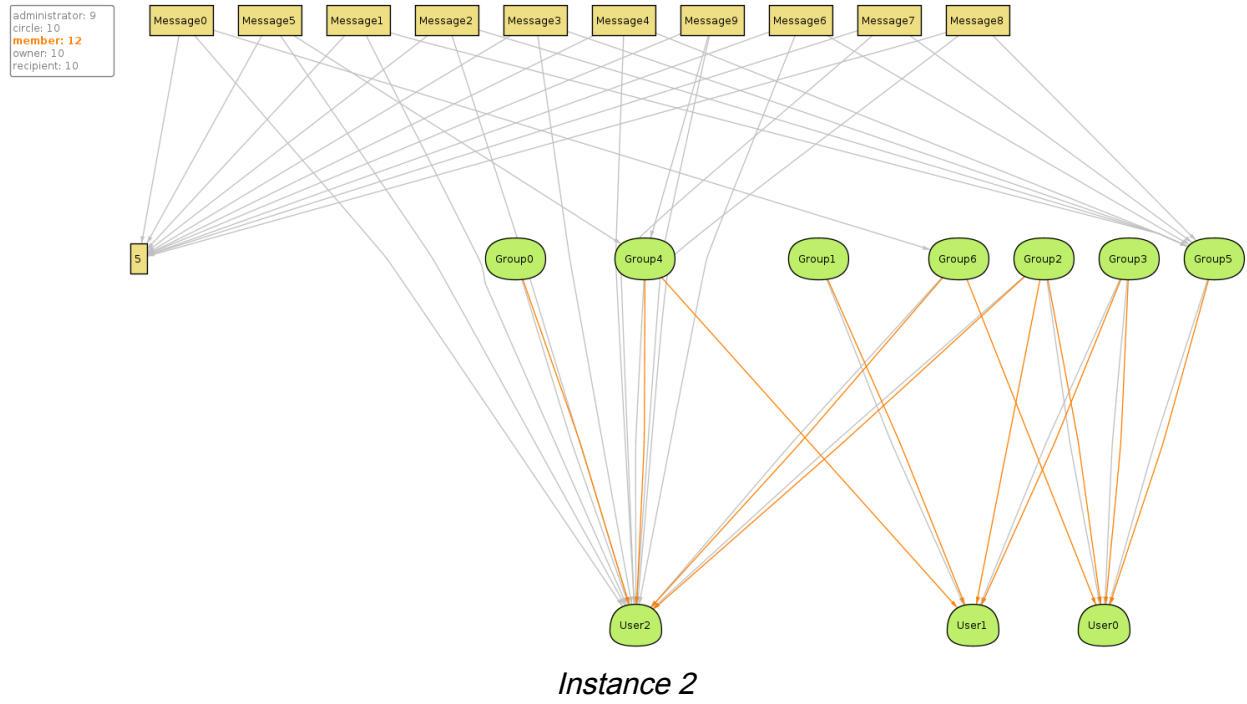
- Property 3 is not true, as a user can post a content to a group while not being an administrator and thus not being allowed, to change the content.
- Property 4 is not true, as the Circle for the Photo can be more restrictive, than the Circle of the Post itself.

### List of Instances from task E unable to produce

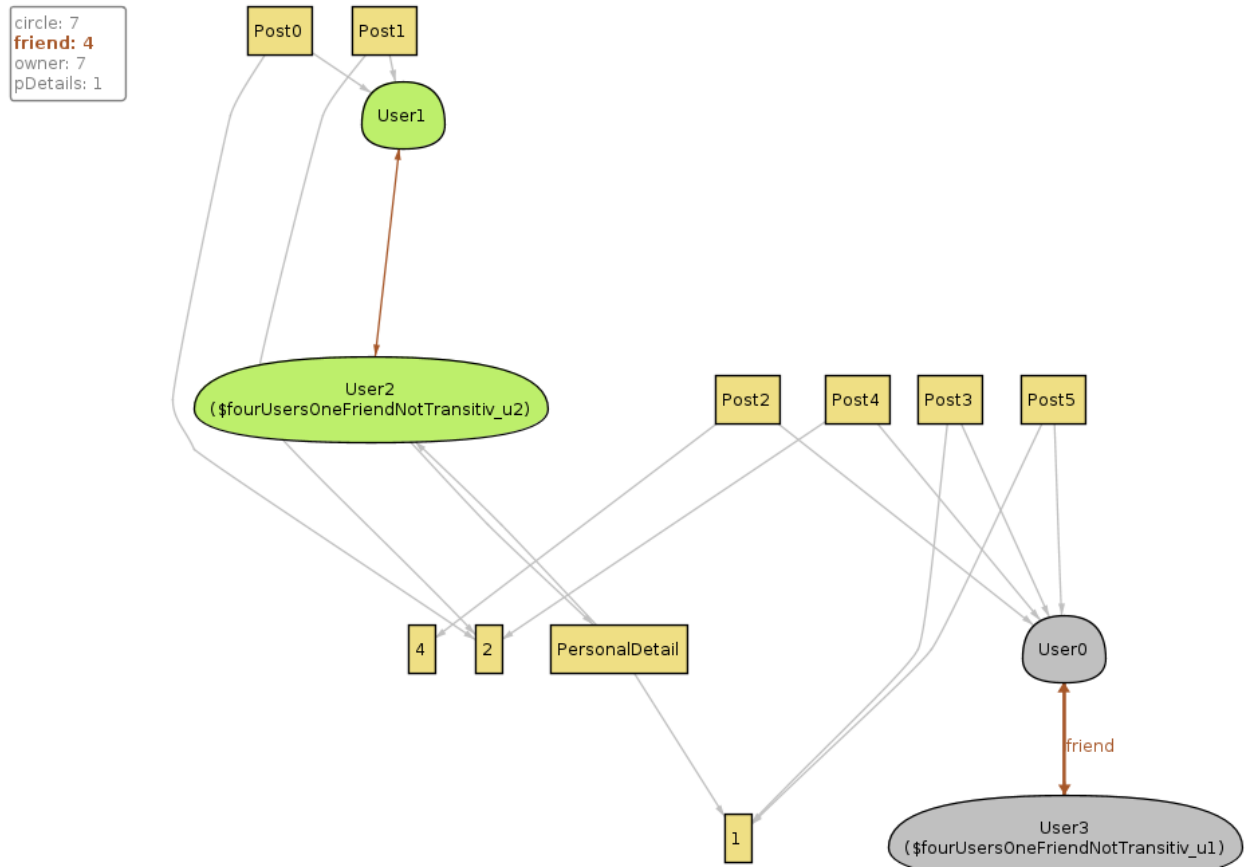
- We were able to produce all instances as you can see below...

### Diagrams of all generated instances from task E

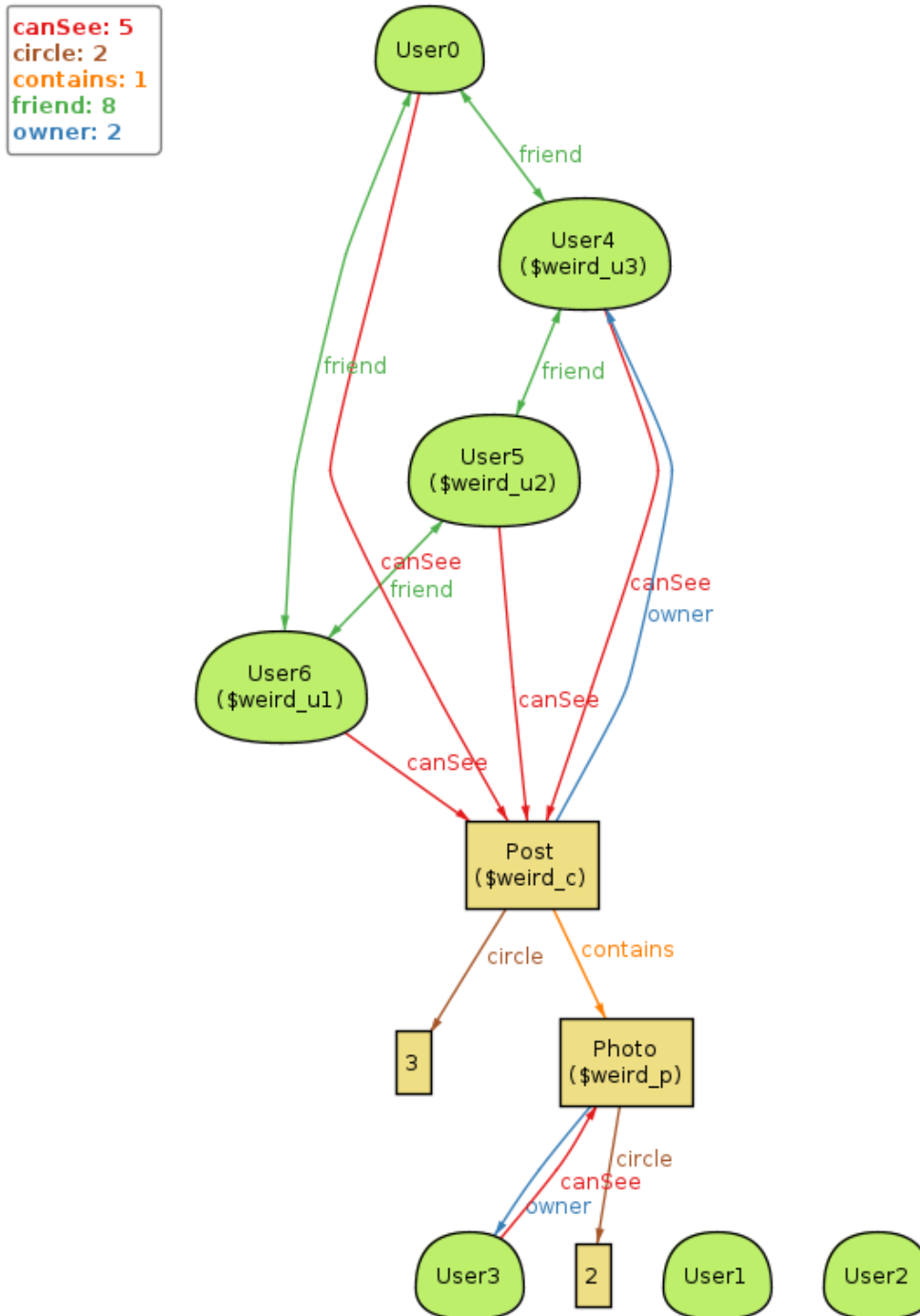






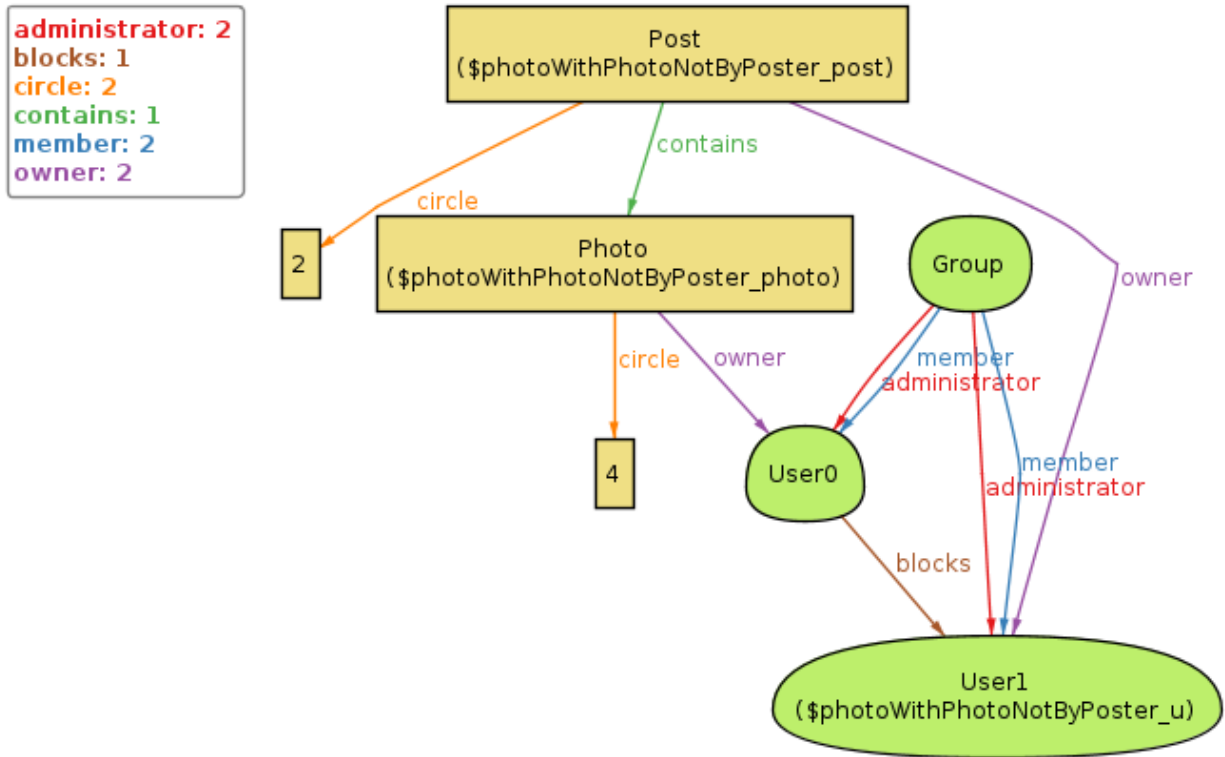


*Instance 3*

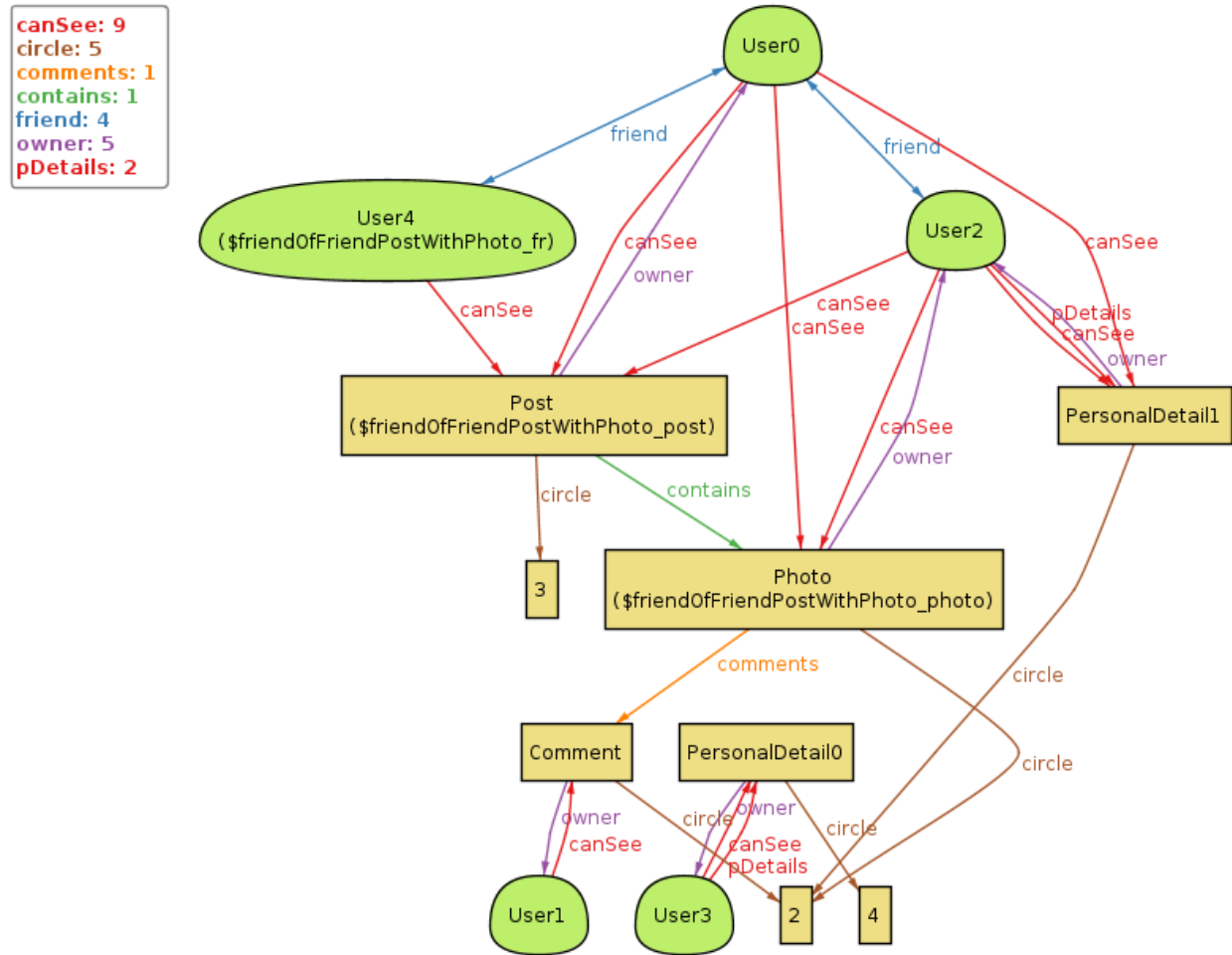


#### Instance 4

(here we made the canSee predicate visible and removed all the groups to simplify the diagram...)



Instance 5



*Instance 6*

*(here we made the canSee predicate visible...)*