Politecnico di Milano
Facoltà di Ingegneria dell'Informazione

Engineering Software 2 Project

MPH
Manage Project Homework
Part I: RASD

2011/2012

Prof. Di Nitto

Andrea Gandini
Andrea Mambretti
Guendalina Milano

Index

1 Introduction

1.1 General purpose

The main goal of Manage Project Homework, from now on simply MPH, is to create a software platform, which will make easier the management of academic courses which plan a project development.

1.2 Glossary

In RASD document some terms will often occur, so we will explain their meanings:

Group	Set composed by one or more students who decides to face the project development together.
Project	Set of informations, that is specifications and deadlines defined by professor in order to clarify the requests.
Deliverable Mark	Outcome of the evaluation process made by professor with respect of the material produced by a given group, in a given deliverable.
Deliverable	Single part of a project characterized by a deadline and composed by one or more files.
Project Mark	Arithmetical average among each deliverable marks.

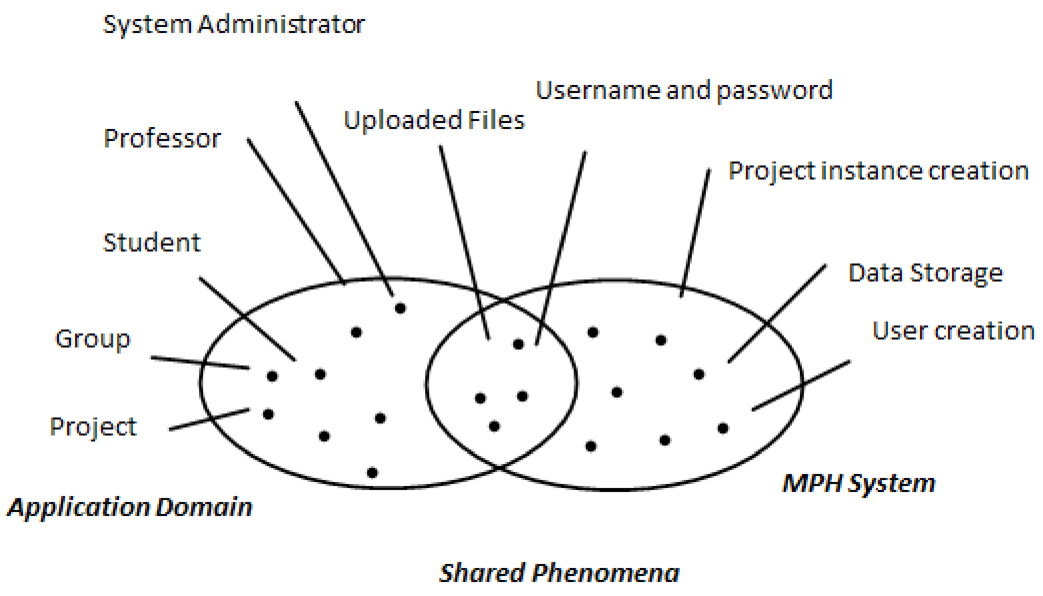
2 Previous support

At the moment there's no computerized process able to facility MPH tasks.
Normally professors are used to publish project documents on their personal website.
Students deliver their material both manually and through platform suggested by professor, which are outside Politecnico such as Origo.

3 Suggested support

The platform we want to design aims to be an alternative solution inside Politecnico, who can directly manage through university system administrators.
Each user, both professor and student can access MPH platform using their access credentials so that they can display only a set of restricted functionalities according to user profile.

3.1 *Jackson-Zave approach*



4 Application Domain

Application domain stands for the part of real world where a general system will work and be placed, in our case the university world.
The domain is composed by different entities and relationships among these and finally the stakeholders.

4.1 *Entities*

From the problem analysis these entities came out:

- Person
- File
- Project
- Group
- Deliverable

4.2 *Stakeholders*

MPH development will affect mainly three kind of users:

- Professors,who will be able to collect more easily all material produced by groups, store it without wasting space and paper and have an automatic way to give evaluations.
- Students,who will be facilitated to manage the material delivery and the project development.
- University, who aims at improving the available tools for students and professors and in consequences at widening the opportunity to introduce itself to the world.

4.3 Relationship

- Each group is composed by one or more student, at most three.
- Each project is divided into different deliverables.
- Each deliverable is composed by at least one file
- Each project is characterized by a name e a deadline.
- Each file is referred to only one deliverable.
- Each group can be subscribed to only one project.

5 Requirements

5.1 Functional requirements and actors

Actor	Professor
Functionalities	• Project publication and deliverables and correlated deadlines definition • Group list and profile visualization • Material download • Project evaluation and consequent marking • Sharing materials among group authorization • Groups management

Actor	Student
Functionalities	• Published project visualization • Personal profile creation • Group profile creation with respect to all constraints • Group member invitation • Group subscription to the desired project • Personal informations access • His/her own group informations access • File update • Requested files upload • Other groups materials visualization only if professor authorized it.

Actor	System administrator
Functionalities	• User “Professor” authorization

5.2 Non functional requirements

5.2.1 User interface and human factor

The system will offer web oriented interfaces,which allow all kind of user to use easily his specific functionalities.
Different interfaces are displayed according to the roles played by the specific user.

5.2.2 Documentation

In order to better explain the project features we provide the following documents:

- Requirements Analysis Specification Document(RASD)which is addressed to the final users,that is student and professor, and to the developers.
- Design Document(DD), addressed strictly to developers.
- User's handbook.
- Code documentation and test cases

5.2.3 Hardware considerations

MPH is developed on a J2EE platform and so it will be composed of JSP, Servlets and JSF. It also uses a Database to store the information of the entire application.
It will be necessary to have a virtual machine and a web-browser installed on all machines.

5.2.4 Error handling and extreme conditions

The system considers all the not valid inputs and error caused by mistake of upload and download. It isn't based on quickness but on consistency and reachability.

5.2.5 Security

It is necessary to provide an authentication mechanism through username and password, so that each kind of user can access only to his specific functionalities.
In advance a professor must be enabled by an administrator.

6 Specifications

6.1 Specification definitions

- Each student can access to MPH platform only after having created a profile and through a username and a password.
- Professor has to give an evaluation to each deliverable
- At each deliverable professor has to define the number of needed files.
- If files are uploaded beyond the deadline, the related deliverable is marked as “late” and it will receive a penalty proportional to the delay.
- Each deliverable is evaluated with a score between 1 and 10.
- At the end of the project the not presented deliverables are considered having score 1.
- A group can display other groups uploaded files, group informations and group component profiles only if professor authorized it.

6.2 *Verify specific:ALLOY*

To verify the correct description of various scenarios it was used Alloy by Alloy Analyzer version 4.2.
Following they are presented the code of the model with description for every command, some predicates to verify the dynamic action and some assertions for the specification of our model. Finally, some instances of a realistic world that may be present.

6.2.1 Mode of Model

```
  //SIGNATURE

sig Deadline {}

sig Deliverable_Deadline {}

sig Vote {}

sig ID {}

sig File {}

abstract sig Person {
     id: one ID
}

sig Student extends Person{
     group_membership: set Group
}

sig Professor extends Person{
     assign_vote: set Group,
     enabled: lone Administrator,
     published_projects: set Project
}

sig Administrator extends Person{
     enable: set Professor
}

sig Group {
     vote: one Vote,
     active_project: one Project,
     visible_groups: set Group,
     member: some Student
}
```

```
{
     #member <= 3
}

sig Upload {
     file_ref: one File,
     by: one Group,
     visible_to: one Professor
}
sig Project {
     deadline: one Deadline,
     deliverable: some Deliverable
}

sig Deliverable {
     previous, follower: lone Deliverable,
     deadline_deliverable: one Deliverable_Deadline,
     file_delivery: some File
}


//FACT

fact {

//There aren't two different Person with same ID
     no p, p': Person | p != p' && p.id = p'.id

//Every File must belongs to one and only one Deliverable
     all f: File { one d: Deliverable | some
     (f & d.file_delivery)}

//Every Deliverable belongs to one and only one Project
     all d: Deliverable { one p: Project |
     some (d & p.deliverable)}

//Every Project must belongs to one and only one Professor
     all p: Project { one t: Professor |
     some (p & t.published_projects)}

//Student cannot, with different group, participate to the
same project
     no s: Student, g, g': Group | g != g' &&
     g in s.group_membership && g' in s.group_membership
     && g.active_project = g'.active_project

//Every Project have one and only one Deadline
     all d: Deadline { one p: Project | some
     (d & p.deadline)}
```

```
//Every Deliverable have one and only one
Deliverable_Deadline
     all s: Deliverable_Deadline { one f: Deliverable |
     some (s & f.deadline_deliverable)}

//Every Group have a Score
     all v: Vote { one g: Group | some (v & g.vote)}

//If some Student is signed to a Group, that Group have
that Student like member
     all s: Student, g: Group | (g in s.group_membership)
     => (s in g.member) all g: Group, s: Student |
     (s in g.member) => (g in s.group_membership)

//Every Group doesn't see itself (it's implicit)
     no g: Group | g in g.^visible_groups

//Each group receives a mark only if it has been
subscribed to a project
     no p: Professor, g: Group | #g.active_project = 0 &&
     g in p.assign_vote

//Group receives a mark only from the professor owner of
its active project
     all g: Group | #g.active_project = 1 => one p:
     Professor | g.active_project in p.published_projects
     && g in p.assign_vote


//Two different Administrator can't enable same Professor
     no a, a': Administrator, p: Professor | p in a.enable
     && p in a'.enable&& a != a'

//Only Professor enabled can be publish Projects
     all p: Professor | #p.published_projects > 0 =>
     one a: Administrator | p in a.enable all p:
     Professor, a: Administrator | p in a.enable => a in
     p.enabled

//Duality realtionship enable/enabled
     all p: Professor, a: Administrator | a in p.enabled
     => p in a.enable

//Groups can Upload a file referenced to a File of its
Project
     no u, u': Upload | u != u' && u.file_ref =
     u'.file_ref && u.by = u'.by
```

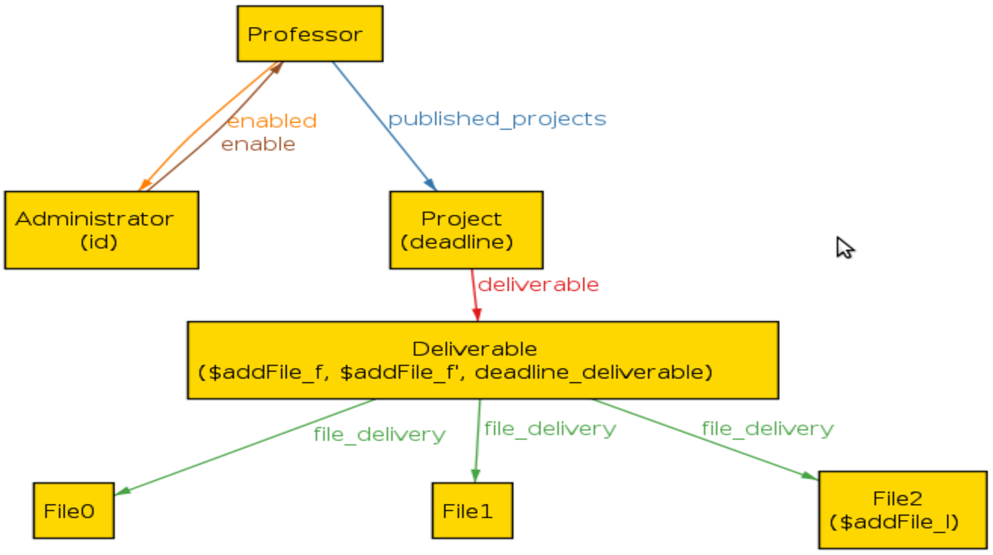
```
//An Upload can refer only to a File of the same Project
of the Group who make the upload
     no u: Upload, g: Group | g in u.by && u.file_ref
     not in g.active_project.deliverable.file_delivery

//Deliverable can't be a follower of itself
     no f: Deliverable | f in f.^follower

//If Project has at least two Deliverable, these have some
previous/follower relationship
     all p: Project | #p.deliverable > 1 => all f:
     Deliverable | f in p.deliverable  => some (f.previous
     + f.follower)

//Duality relationship previous/follower by Deliverables
     all f, f': Deliverable | f.previous = f' <=>
     f'.follower = f

//Two Deliverables reference different Project can't have
any follower/previous relationship
     no f, f': Deliverable, p, p': Project | p != p' &&
     f != f' &&     f in p.deliverable &&    f' in
     p'.deliverable && (f' in f.previous || f in
     f'.previous)

//Every Upload can be downloaded from a Professor owner of
the Project
     all u: Upload, g: Group, p: Professor |
     g.active_project in p.published_projects &&
     g in u.by => u.visible_to = p
     }
```

6.2.2 Predicates

Two example of predicate (are omitted some relations/attribute for simplicity).

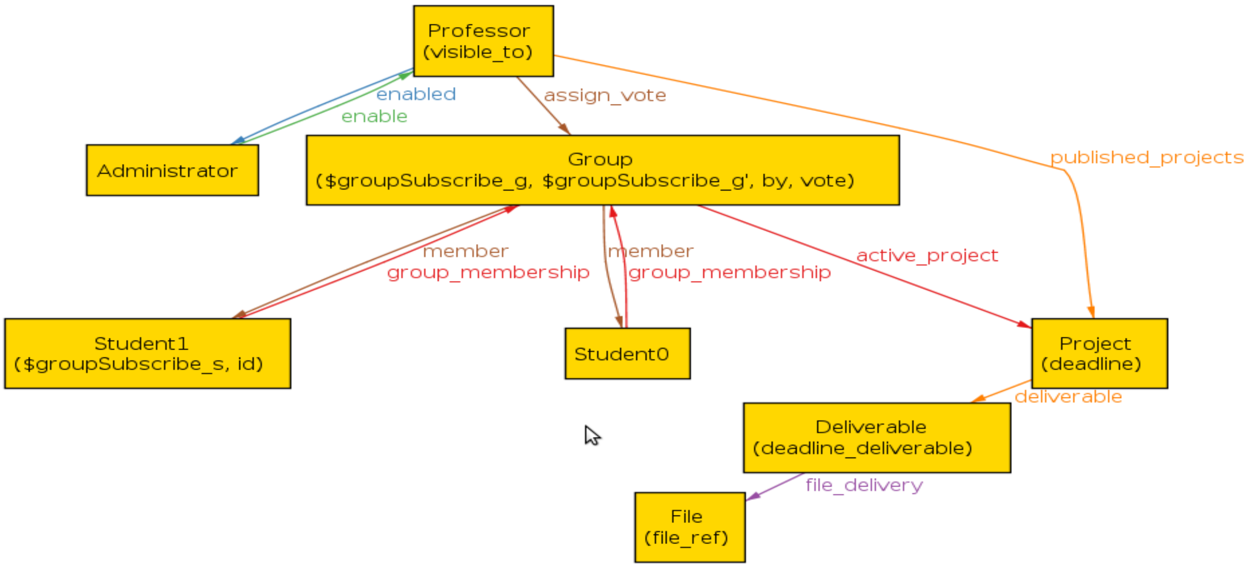
The first one controls if it's possible to add a file to an existing deliverable: it's possible and it can be checked by the instance of a new file (File2) added to the project.

```
pred addFile(f, f': Deliverable, l: File){
     f'.file_delivery = f.file_delivery + l
}
run addFile
```



The second predicate checks if a student can subscribe to a group that has already another student. The group after the predicate present two student (newest is Student1).

```
pred groupSubscribe(s: Student, g, g': Group){
     g'.member = g.member + s && #g.member > 1
}run groupSubscribe
```

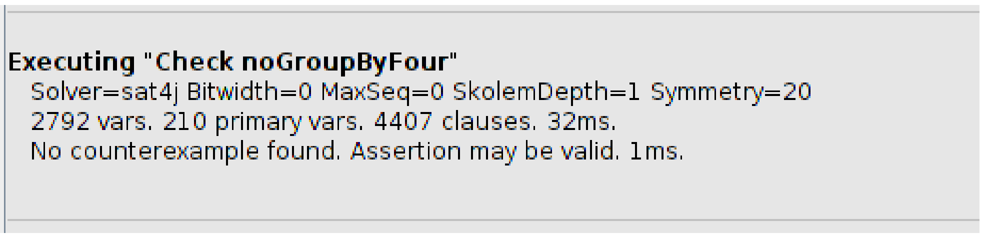


6.2.3 Assertions

Assertions represent some specific the model must have. Here three fundamental assertions (here too are omitted some relation/attribute for simplicity).

The first one check a group can't has more than three student subscribed to it.

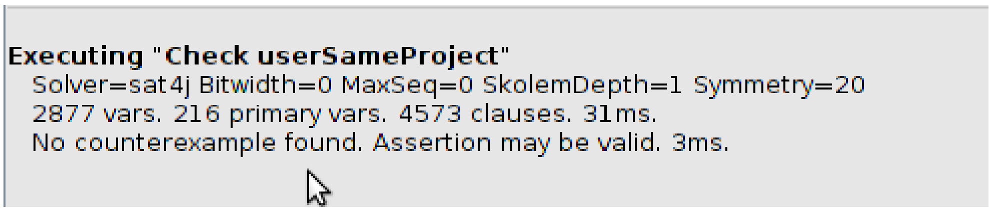
```
assert noGroupByFour {
     all g: Group | #g.member <=3
}
check noGroupByFour
```



```
Executing "Check noGroupByFour"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  2792 vars. 210 primary vars. 4407 clauses. 32ms.
  No counterexample found. Assertion may be valid. 1ms.
```

This assertion controls a student can't partecipate to the same project by two different groups.

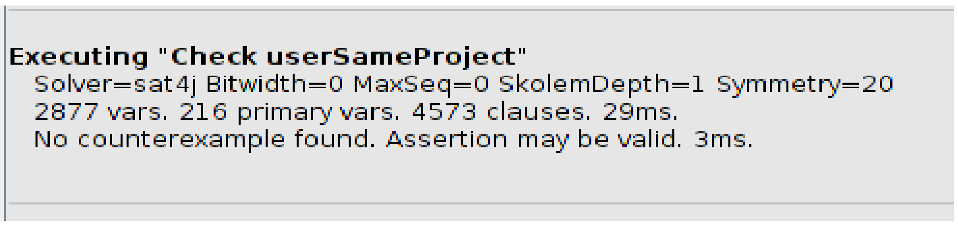
```
assert userSameProject {
     no s: Student, g, g': Group| g in
s.group_membership &&
     g' in s.group_membership && g.active_project =
g'.active_project && g != g'
}
```



```
Executing "Check userSameProject"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  2877 vars. 216 primary vars. 4573 clauses. 31ms.
  No counterexample found. Assertion may be valid. 3ms.
```

The last assertion check the relation of the same deliverable: this ensures there's no cycle between deliverables.

```
assert noCycleDeliverable{
     no f: Deliverable | (f in f.^previous || f in
f.^follower)
}
check noCycleDeliverable
```



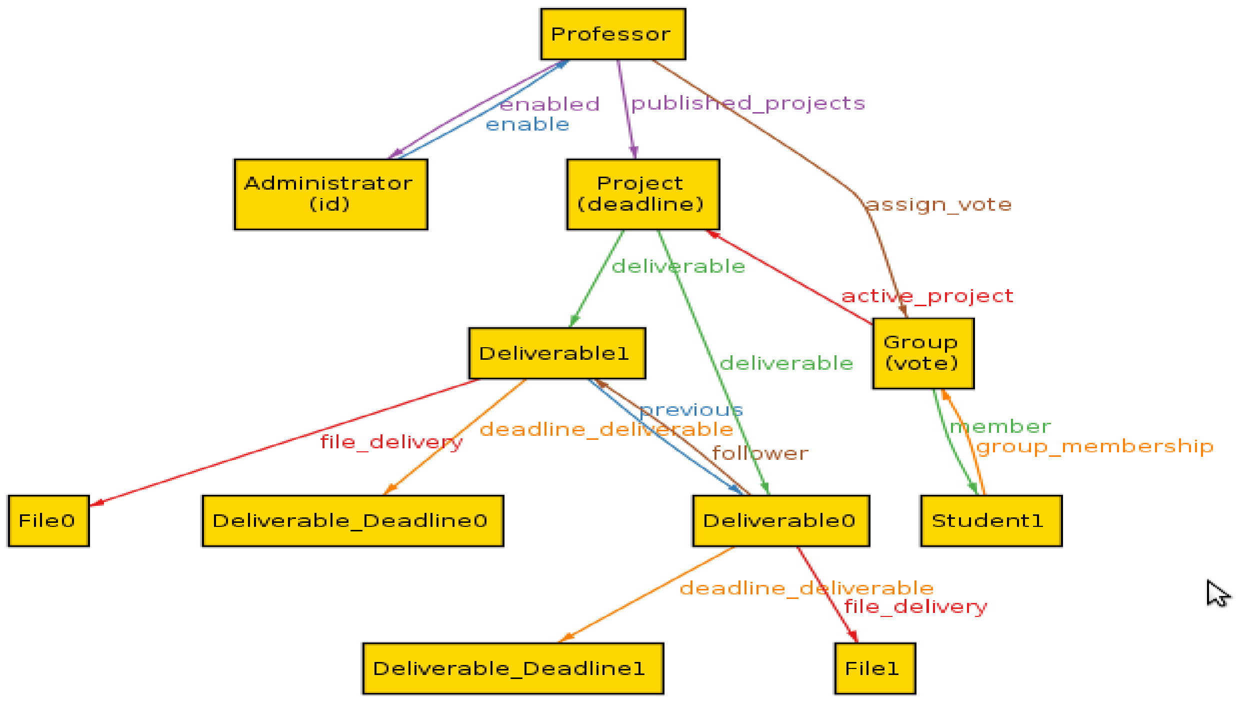
```
Executing "Check userSameProject"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  2877 vars. 216 primary vars. 4573 clauses. 29ms.
  No counterexample found. Assertion may be valid. 3ms.
```

6.2.4 Instances

Cocluding the Alloy description, here some example of the world can be present in the application.
Note the attribute ID for each Person, Vote for each Group, Deadline for each Project and Deliverable_Deadline for each Deliverable are represented by a substring in each object.

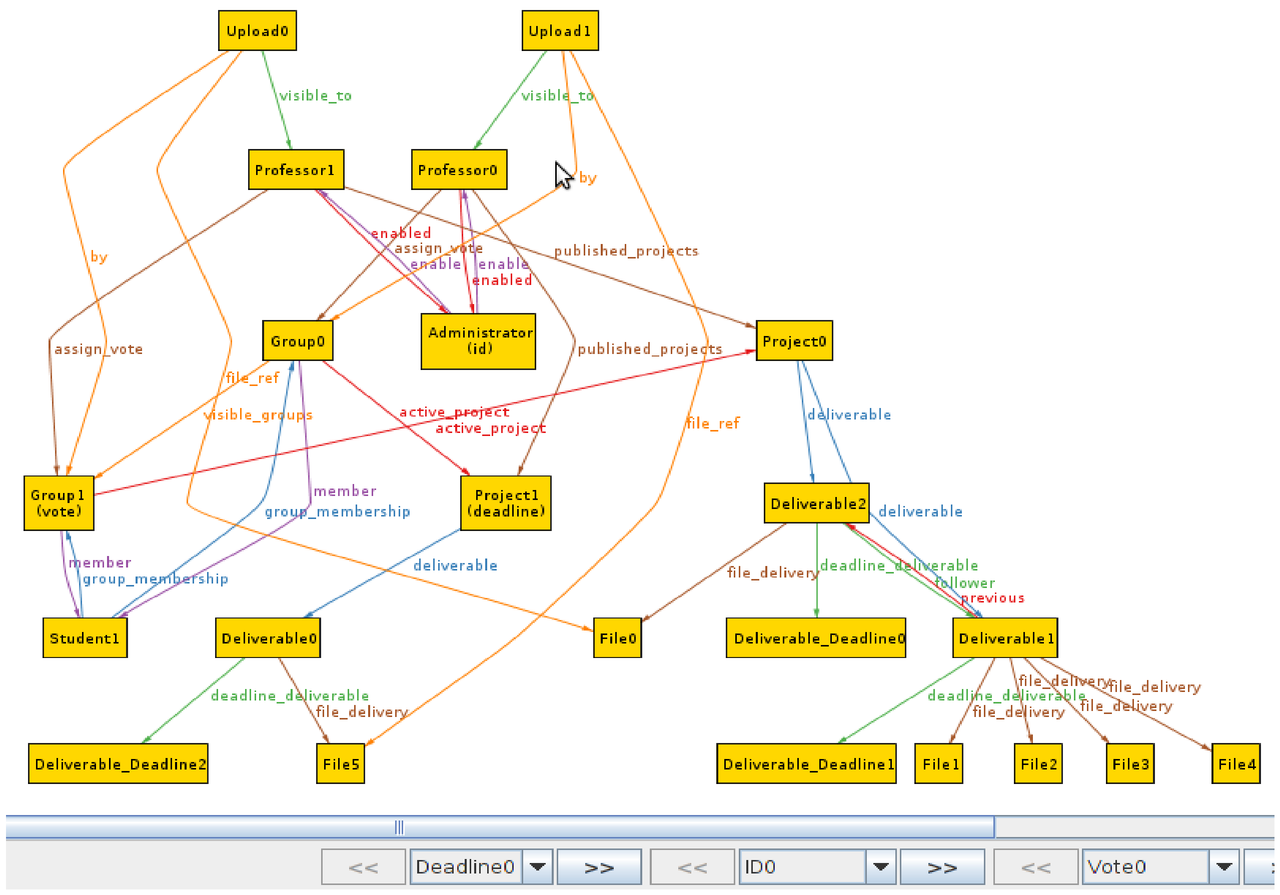
```
pred show() {
     #Student = 1
     #Professor > 0
     #Group > 0
     #Deliverable = 2
     #Project = 1
     #File = 2
}
run show for 6
```

Example 1

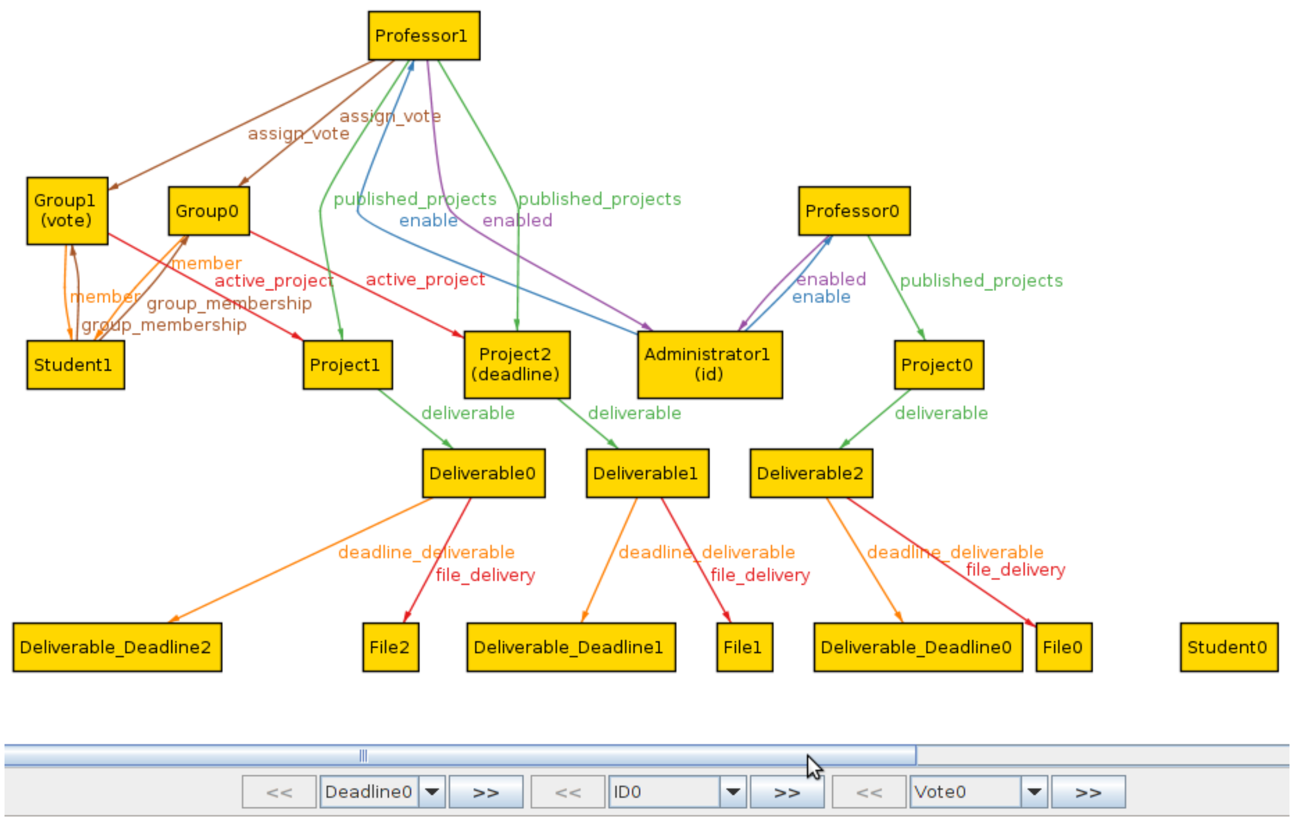


In this example, a professor enabled by an administrator publish a project formed by two deliverables, each one has its deadline and one file. At this one has subscribed a group composed by one student.
The Example 2 show a student subscribed to different group each subscribed to a different project. One administrator enable the professor who publish each one a project. There's a student who's not participate to any project.
The Example 3 shows a particular complicated instance: there's two groups, each one make an upload reference to a file in the project where the group are subscribed. One project have two deliverables have a relationship of follower/previous. In this case too, an administrator enable the professor who can downloaded the file uploaded by only the group in reference of a project's professor.

Example 2



Example 3



7 System Models

7.1 Scenarios

Here they are some typical scenarios grouped by actor.

7.1.1 Professor

Scenario Name	Project publication
Participating Actors	Prof. Smith, professor
Events Flow	1. Prof. Smith logs in using username and password. 2. The system recognizes the user and displays him his specific interfaces with the corresponding functionalities. 3. Prof. Smith launches “Project Publication” functionality. 4. The system asks him to fill a form in with project description. 5. Prof. Smith inputs data and confirms clicking on “Publish!” button. 6. The system notifies the operation success.

Scenario Name	Project marking
Participating Actors	Prof. Smith, professor
Events Flow	1. Prof. Smith logs in using username and password. 2. The system recognizes the user and displays him his specific interfaces with the corresponding functionalities. 3. Prof. Smith launches “Material download” functionality. 4. Prof. Smith analyses documents and produces a marking. 5. Prof. Smith launches “Project marking” functionality. 6. The system asks him to fill a form in with the decided marking. 7. Prof. Smith inputs “7” and confirms clicking on “Mark!” button. 8. The system notifies the operation success.

Scenario Name	Authorization among groups
Participating Actors	Prof. Smith, professor
Events Flow	1. Prof. Smith logs in using username and password. 2. The system recognizes the user and displays him his specific interfaces with the corresponding functionalities. 3. Prof. Smith displays group 2 details. 4. Prof. Smith launches “Sharing materials among group authorization” functionality. 5. The system asks which group 6. Prof. Smith inputs “group 5” and confirms clicking “Ok” button. 7. The system notifies the operation success.

7.1.2 Student

Scenario Name	Registration
Participating Actors	Jack, student
Events Flow	1. Jack visits “www.mph.polimi.it”. 2. Jack click on Register link 3. The system displays a form with two field:username and password. 4. Jack inputs his username and password 5. The system states that there is already a student with this username and ask for a new one. 6. Jack inputs a new username and his password. 7. The system notifies the profile creation.

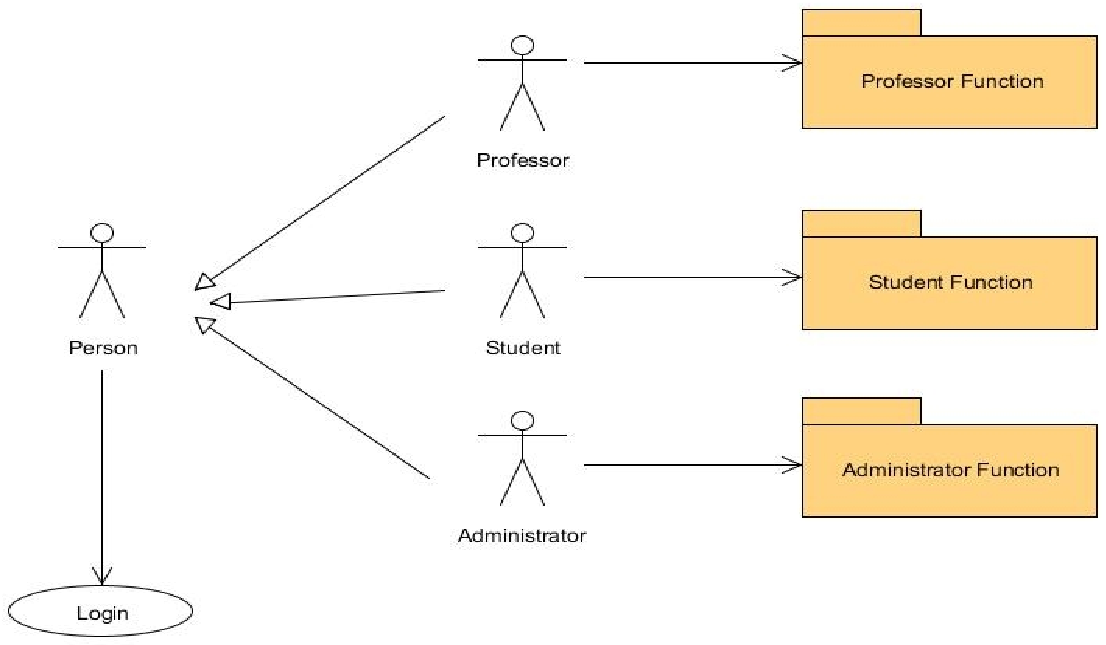
Scenario Name	Project Visualization
Participating Actors	Jack, student
Events Flow	1. Jack logs in using username and password. 2. The system recognizes the user and displays him his specific interfaces with the corresponding functionalities. 3. Jack launches “Visualize your project”. 4. The system displays the project presentation .

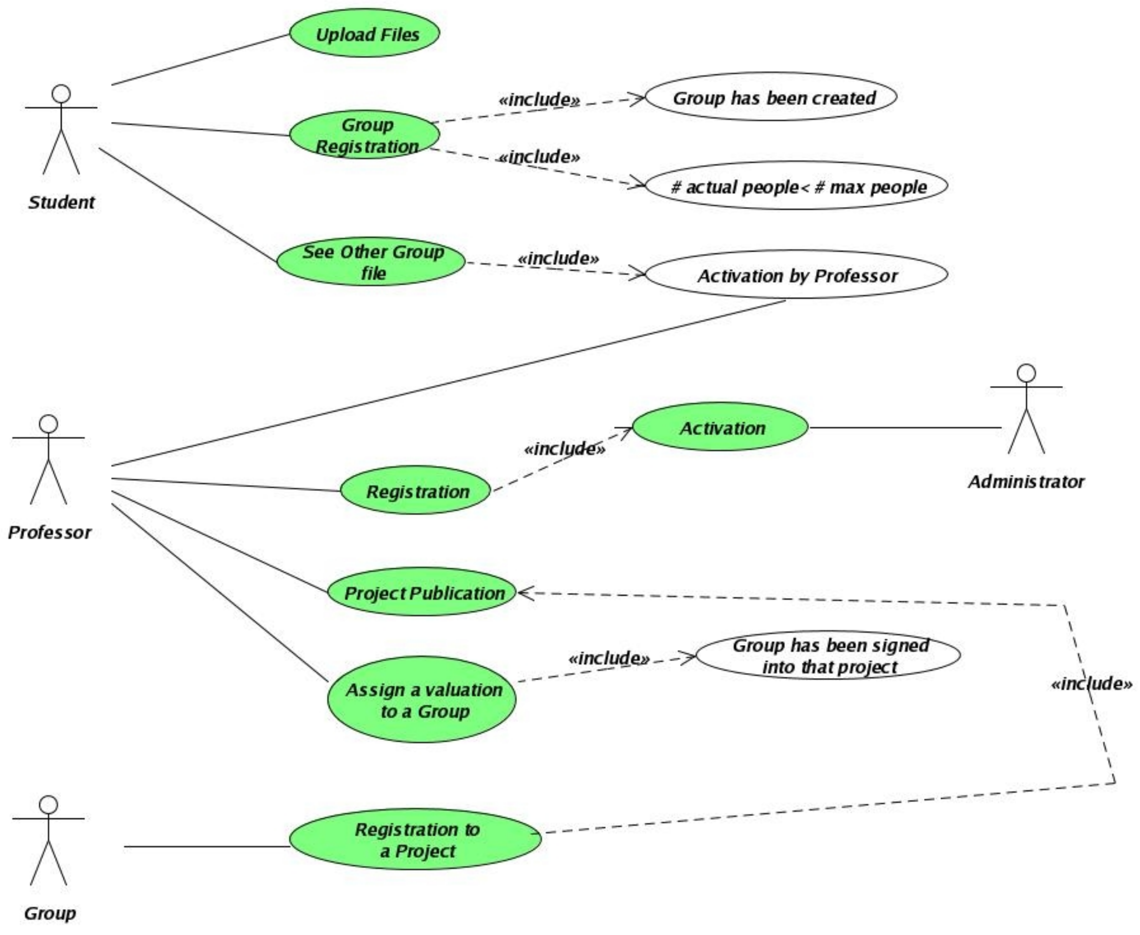
Scenario Name	Material Upload beyond deadline
Participating Actors	Jack, student
Events Flow	1. Jack logs in using username and password. 2. The system recognizes the user and displays him his specific interfaces with the corresponding functionalities. 3. Jack launches “Upload files”. 4. Jack selects the files. 5. The system notifies the upload has done beyond the deadline and marks the files as “late”.

7.1.3 System administrator

Scenario Name	User “Professor” Authorization
Participating Actors	Mr Green, system administrator, Prof. Smith, professor
Events Flow	1. Mr Green receives a authorization request from Prof. Smith. 2. Mr Green checks that Prof. Smith have right to get a “Professor” profile and finds out he can. 3. Mr Green enables Prof. Smith's profile 4. Mr Green communicates to Prof. Smith the creation success.

7.2 Use Cases

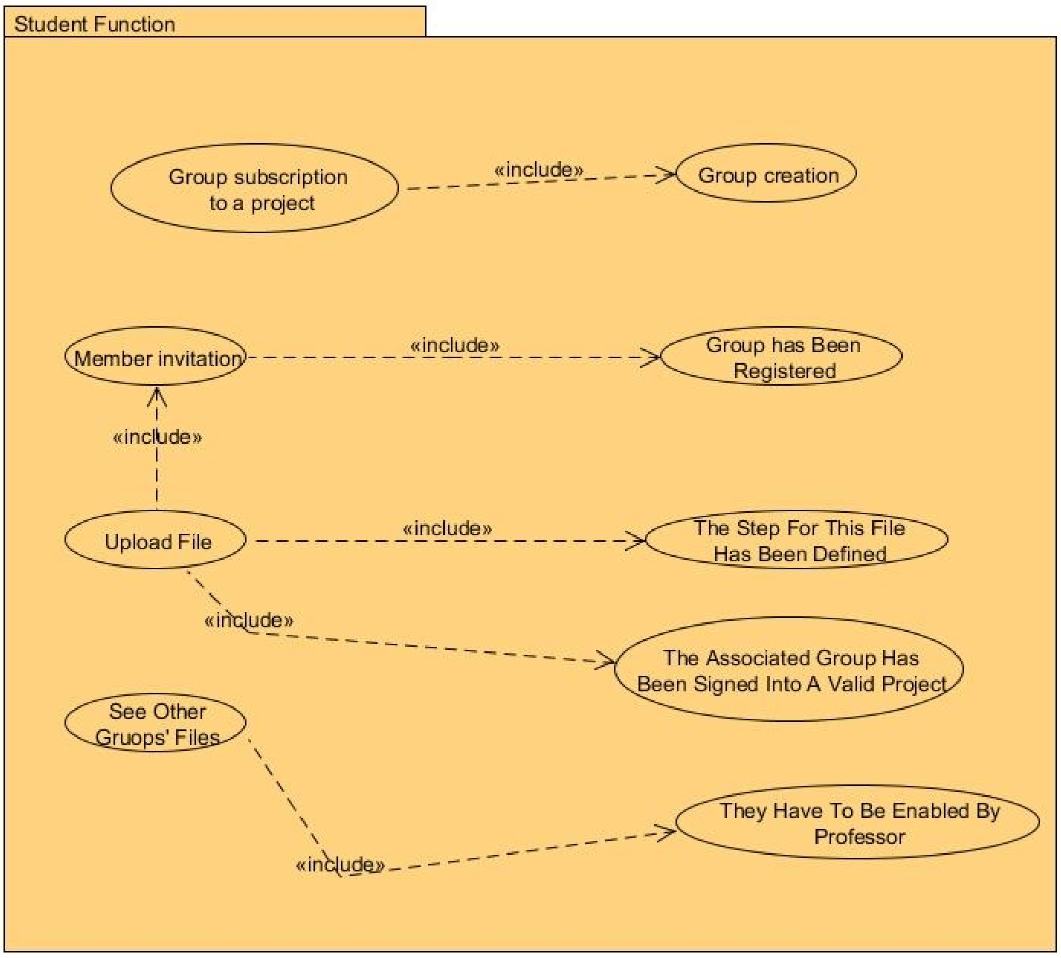

Professor
Student
Administrator
Person
Login
Professor Function
Student Function
Administrator Function


Student
Upload Files
Group Registration
See Other Group file
«include»
Group has been created
# actual people< # max people
Activation by Professor
Professor
Activation
Administrator
Registration
Project Publication
Assign a valuation to a Group
Group has been signed into that project
Group
Registration to a Project

7.2.1 Person

Name	Login
Participating actors	Person
Pre-condition	1. Person visits mph web-site
Events flow	2. The system displays a form with two fields: username and password. 3. Person inputs his username and password. 4. The system checks accuracy of the input.
Post-condition	5. The system displays the personal page.
Exception	- In case at point 4 the system doesn't recognize user, it displays an error message and asks to input user's credential again.

7.2.2 Student

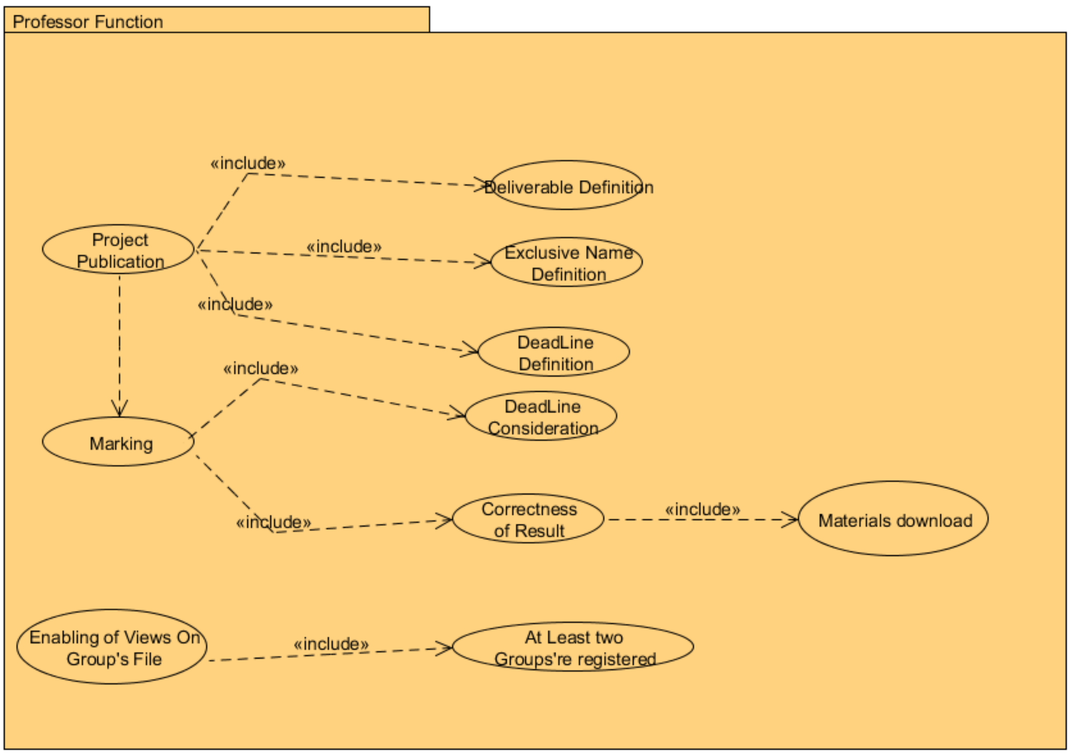

Student Function
Group subscription to a project
«include»
Group creation
Member invitation
«include»
Group has Been Registered
«include»
Upload File
«include»
The Step For This File Has Been Defined
«include»
The Associated Group Has Been Signed Into A Valid Project
See Other Gruops' Files
«include»
They Have To Be Enabled By Professor

Name	Group creation
Participating actors	Student
Pre-condition	1. A student accesses to Group creation web-page.
Events flow	2.The system displays an input form. 3.The student fills it and optionally he can add other group member.
Post-condition	4.The system creates a group page.
Exception	- At point 3 if the student inputs are not valid the system return an error.

Name	Group subscription to a project
Participating actors	Student
Pre-condition	1. A student launches "Group subscription"
Events flow	2. The system asks what project the group wants to subscribe and for the group details displaying a form 3.Student inputs requested informations.
Post-condition	4. The system notifies the group subscription to the desired project
Exception	- In case that the group is already subscribed to another project, the system rejects the request displaying an error page.

Name	File upload
Participating actors	Student
Pre-condition	1. The student is visiting his group page and clicks on upload file link.
Events flow	2.The system displays an input form. 3.The student chooses the files that have to be uploaded.
Post-condition	4.The system notifies the operation success.
Exception	- At point 3 if there is an upload error, the system notifies it.

7.2.3 Professor

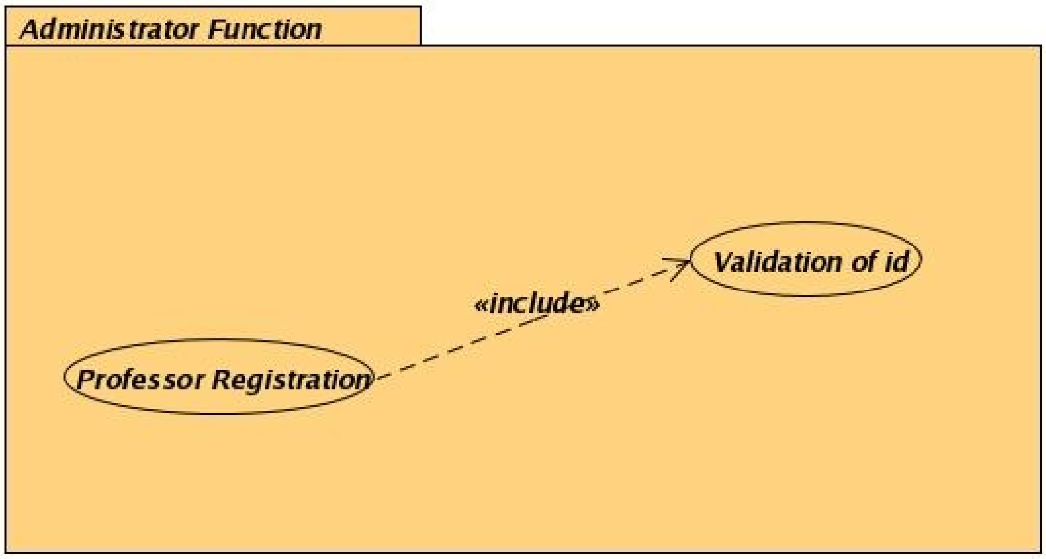


Name	Project publication
Participating actors	Professor
Pre-condition	1.Professor accesses to the project creation page.
Events flow	2. The system displays an input form. 3. He defines deliverables and the deadlines and inputs data.
Post-condition	4. The system displays an input form.
Exception	- At point 3 if the professor inputs are not valid the system return an error.

Name	Project marking
Participating actors	Professor
Pre-condition	1. Professor must download and carefully read group materials.
Events flow	2. After having decided the mark he accesses to group page and fill a form.
Post-condition	3. The system notifies the operation success.
Exception	- At point 2 it there are some mistakes the system display it to user.

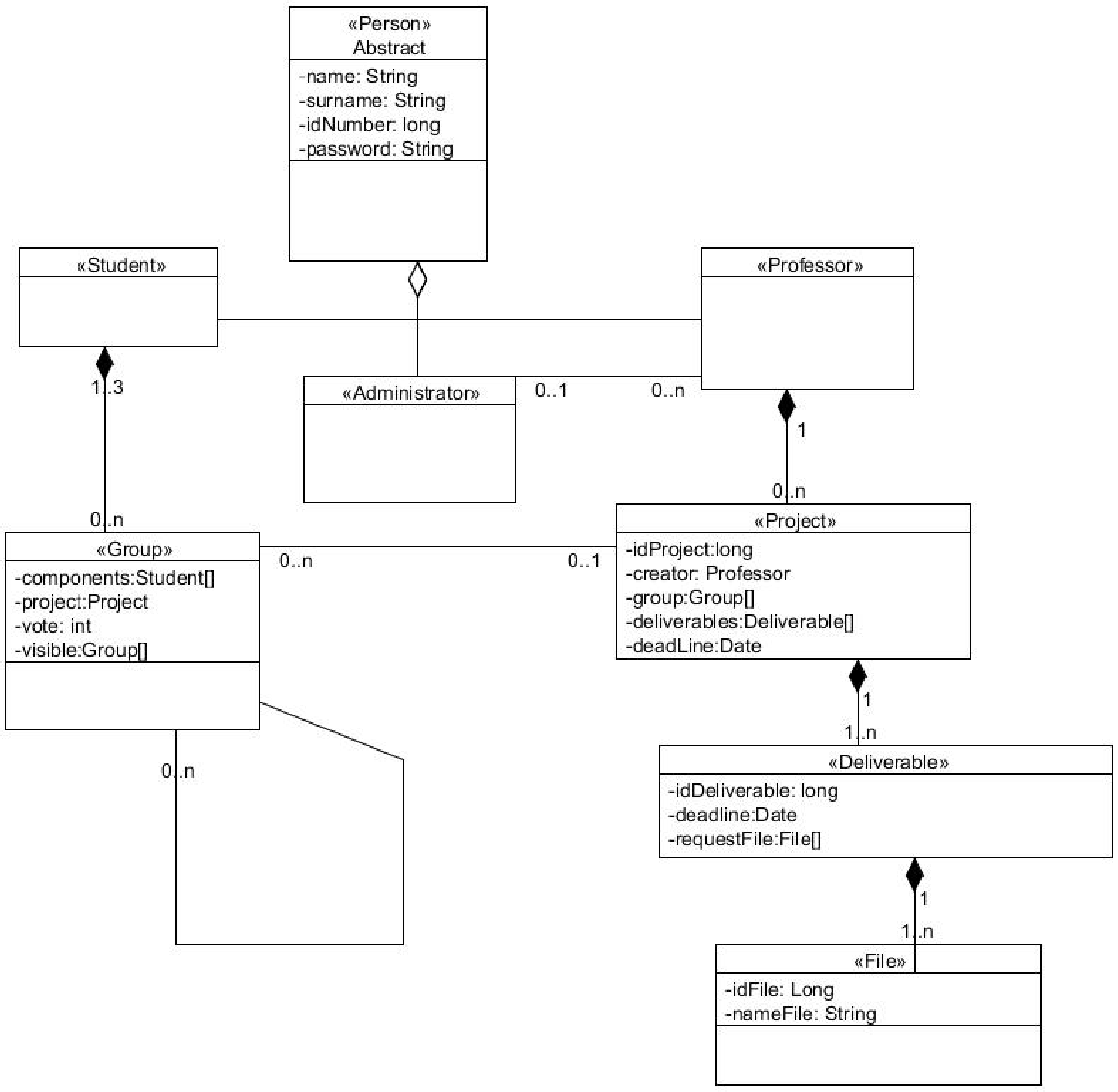
Name	Authorization among groups
Participating actors	Professor
Pre-condition	1. Professor chooses a group and accesses to its web-page.
Events flow	2. Professor selects from the group list which one has to be visible.
Post-condition	3. The system notifies the operation success.
Exception	- At point 2 if the professor inputs are not valid the system return an error.

7.2.4 System Administrator



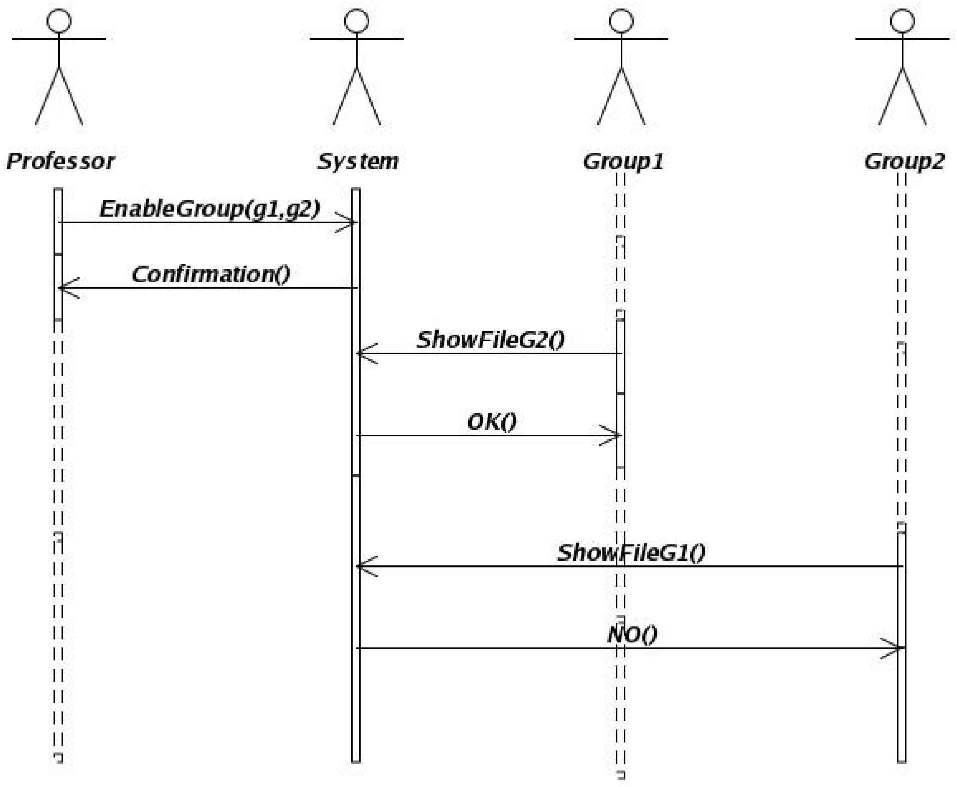
Name	Professor Authentication
Participating actors	System administrator
Pre-condition	1.The administrator receives a registration request by a professor.
Events flow	2.The administrator validates professor id 3.He enables the professor profile.
Post-condition	4.The administrator communicates to the professor his access credentials.
Exception	At point 2 if the administrator considers the request not valid, he sends a error message to the user.

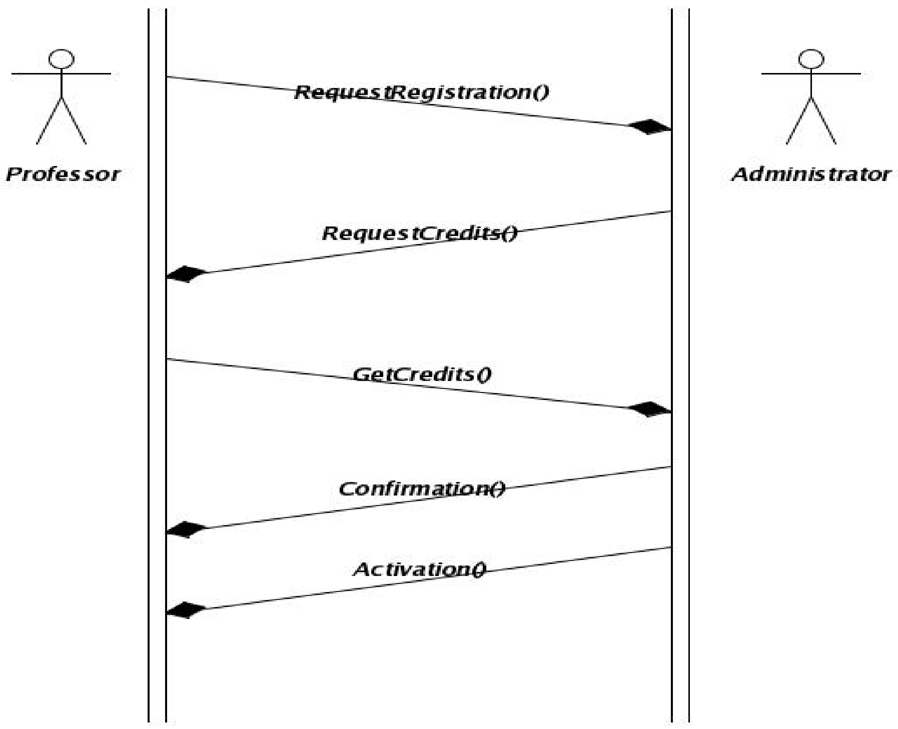
7.3 Class Diagram

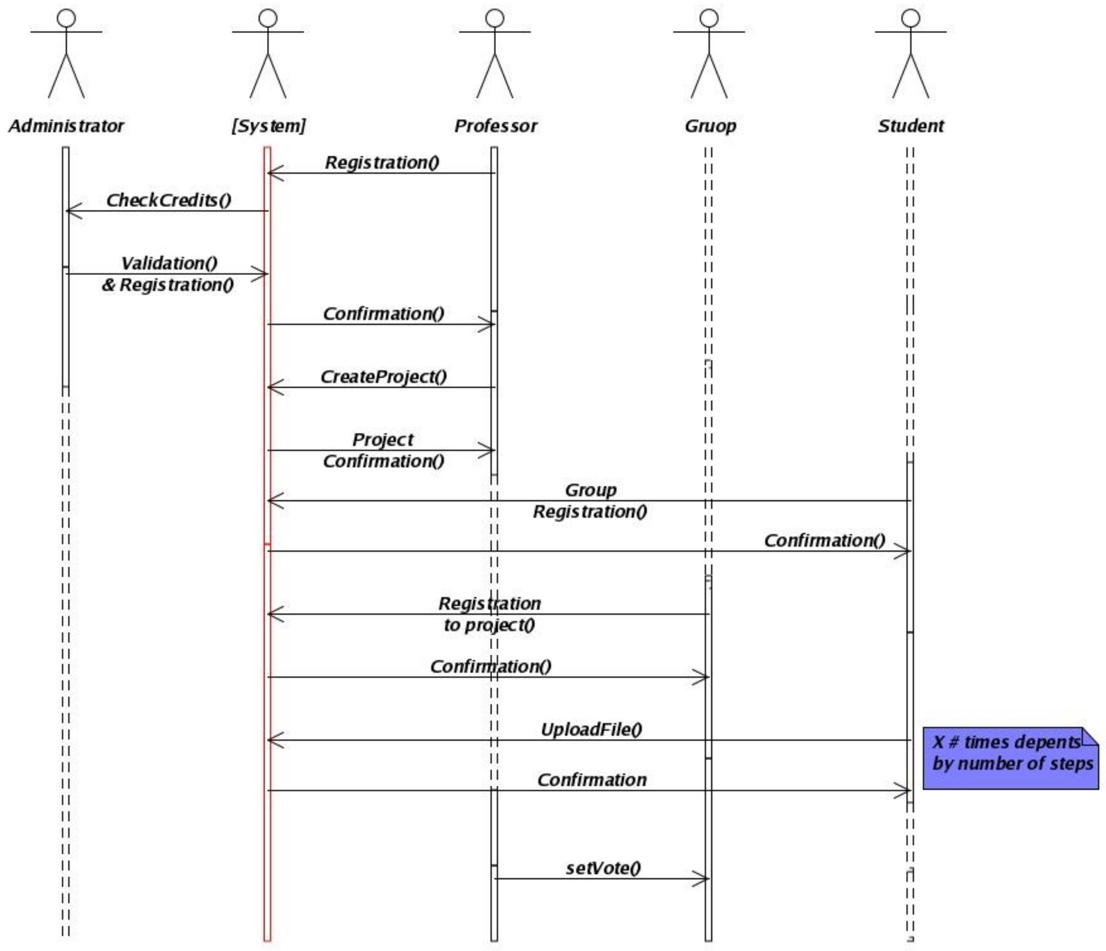


7.4 Dynamic Models

7.4.1 Sequence Diagrams






Administrator
[System]
Professor
Gruop
Student
Registration()
CheckCredits()
Validation()
& Registration()
Confirmation()
CreateProject()
Project
Confirmation()
Group
Registration()
Confirmation()
Registration
to project()
Confirmation()
UploadFile()
X # times depents
by number of steps
Confirmation
setVote()

8 Used tools

To build this document we use the following software tools:

- UMLet
- Alloy Analyzer 4.2
- OpenOffice Writer