Elise DECOURVAL

0150888623

MICS 2

2015/2016

Model Driven Software Development

GMF Project

1. Introduction

The goal of this project is to design from scratch an editor for Fiends. It is a social Network Modelling language using GMF. It's a language allowing the specification of a next generation social network inspired by the following proverb: "The enemy of my enemy is my friend".

In this network, we can create users, relate to other user via an enmity or friendship relation. We can also associate to user publication that only friends can like, share and comment on. A publication can hence be original, or pointing to an existing publication, it consist of text only and is related to a sequence of comments.

For this project we use the **Eclipse** software.

2. Abstract Syntax

First, we have to do the **Abstract Syntax Development** using EMF (Eclipse Modelling Framework). Namely, we create the **metamodel** of our project which allow to define the concepts. All EMF model is an instance of an EMF module with common root for the EMF Ecore model provided by EMF.

Therefore EMF tool allow modelling concepts. We create a new **Ecore** file. Then, we create our root package, Friends, with a unique URI to identify our metamodel. Inside of this package we can create the various entities that allow us to create our class instances. (See *Figure 1: Ecore Diagram Friendz*).

Firstly, we create our **basic entities**, which will allow us to create our class instances. So we need to have *User*, *Publication*, and *Comment*. In the network *Friendz*, we have several users, who can post several publications and comments, that is why the cardinality between each entity and Friends are 0…*.

For each of these classes we add the **attributes** needed. A user is identified by his friends, with his name (*firstName, lastName*). Nevertheless multiple users can have the same name, so we add a primary key, id. Similarly, a publication is a string (*publication*) but several users can have the same idea, so we added it an attribute identifier (*idPub*).

GMF restricted us to create new classes that will use as **link** to our entities.

The entity *Enmity* creates an enmity relationship between two individuals, indeed, in this class we have two eReferences user. The principle is exactly the same for the entity *Friendship*.

Other concepts have also be designed. A user must be connected to his publication then we add an entity *Authorship*. Similarly, a publication containing a reference, must be connected to this reference - *PubliRef*. Also, a comment should be linked to the publication concerned – *CommentTarget*. Each classes contains two eReference, which will connect the two instances concerned.

Regarding the *Like* class. It connects a user to either publication or a comment. To differentiate the two types of likes (comment type and publication type), first I thought add an EEnum, Boolean that enable to differentiate the two (*EEnumLike :* EEnumLiteral : comment, publi). But with this last solution it was too difficult to create the concrete syntax, therefore I crate two type of links, *LikePubli* and *LikeComment*.

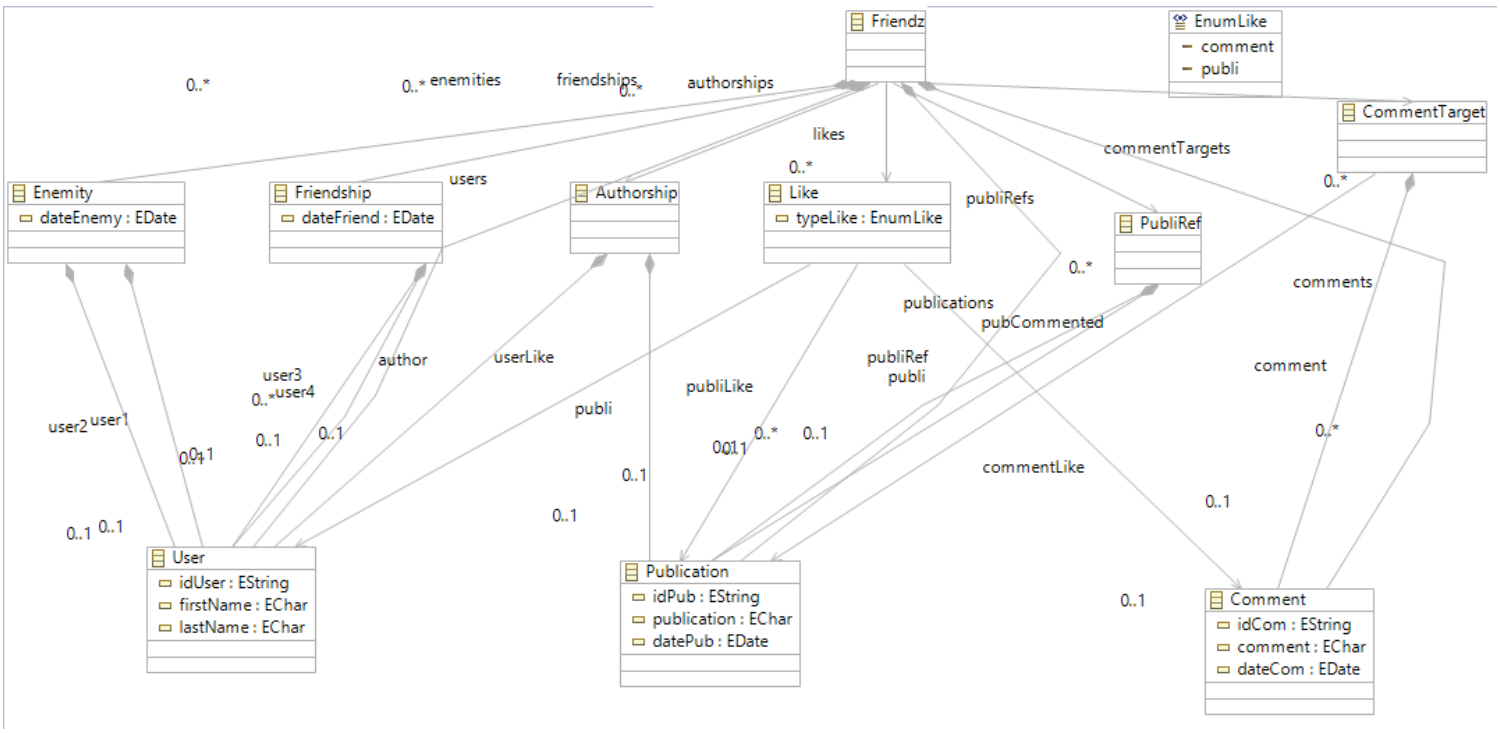


Figure 1: Ecore Diagram Friendz

The relationship between two individuals can change, so we added a *Date* attribute in classes Enmity, Friendship and Publication. I decided to add this attribute, because if a user wants comment or like a publication he must be friend with the author of this publication. Therefore, there should be a checkout about the date of his last relationship. If this individual is included in the author list of friends as well as enmity we consider the last relationship dated.

This constraint must be verified also thanks to **OCL constraint.** This language allows to express constraints, and therefore conduct audits or techniques specifications. With invariants we can create pre- and post-conditions for operation, navigation expressions, Boolean expressions... To create our constraints we use **OCLinEcore** editor.

First we create simple constraints. Each identifier is to be unique, for example, below is an example of this constraints to the User class.

```
invariant
UniqueID ('User must have a unique id'):
User.allInstances () -> forAll (u1, u2 | u1 <> u2 implies u1.idUser <>
u2.idUser);
```

Moreover, with regard to the links of affection, a relationship must consist of two different users.

```
invariant
Relation ('User1 must be different from user2'):
self.user1 <> self.user2 ;
```

More complex constraints are also adding on relations between the users.

For example, I will explain how I check a user who wants to like a publication is a friend of the Author. I do not know if my logic and constraints are correct but I will explain them. First, I create an operation *isAuthor* which recovers the author of a given publication. Secondly, I create an operation *isFriend* which recovers the friends of a given user. Then I can use these two operations to find if a user who wants to like a publication is friends with the author. To do this, I get friends of the author of the publication (*isFriend(isAuthor(self.publication))*), and then with the primitive **include**, I look if my user is in the list of friends of the author of the publication. The result of this operation have to be equal to "true" because the primitive *includes(obj)* returns true if the object is included in the list.

```
operation isAuthor( p : Publication ) : User {
      body: self ->select(u : User | Authorship.publi = p) ;
  }


operation isFriend( u : User ) : User[*] {
      body: self ->select(u2 : User | Friendship.user1 = u) ;
  }

 invariant
  MustBeFriend('The two users must be friends'):
  isFriend(isAuthor(self.publiLike)).includes(self.userLike) = true;
```

The principle is exactly the same to check if a user shares a publication that is friends with the author of the publication. We just need to pick the author of the original publication.

```
invariant
  MustBeFriend('The two users must be friends'):
  isFriend(isAuthor(self.publi)).includes(isAuthor(self.publiRef)) = true;
```

Another part of the constraints are related to the proverb: “The enemy of my enemy is my friend”, but I did not find how to manage this constraint.

3. Concrete Syntax

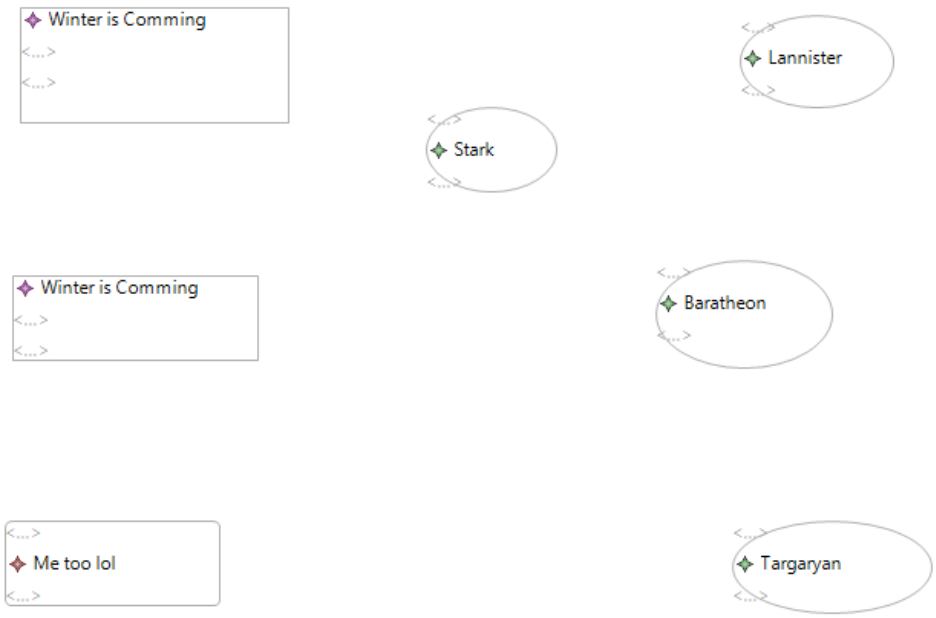
Once the abstract syntax, ecore the model is created, we can start creating our graphical modelling framework. This framework will provide an infrastructure allowing the execution of a graphical editor based on our model Friendz.

For do this first I created the different file with my dashboard: Domain Gen Model, Tooling definition and Graphical definition. During these steps I was careful to set as link my classes that I use as transition: *Authorship*, *Friendship*... For each link I defines the Target and source. For example for *authorship* my source is a user and my target is a publication. Then I generated the diagram editor.

In my mapping model I creates all my node and my links. For each I have added interesting label and change the descriptor figure. I worked without link like because I had an issue with this transition and I didn't find a solution.

I have created new Descriptor figure for my node. For example for a user, I have created a new Ellipse Figure. For each node I have added a label. When I open a new Eclipse Instance for create a new Friendz diagram, my node are ok, but my links don't work. I tried to modify a lot of things, but I don't find the solution.

You can see bellow an example of my Friendz Editor.



4. Conclusion

So we created our Metamodel Friendz then an instance of this metamodel with the editor that we had created.

For improve my project, I would have to add and modify OCL constraints. For example, I have to add the constraint to consider this proverb: “The enemy of my enemy is my friend”. Furthermore, I have to fix my editor Diagram for have correct transition.

This project was very interesting for me, it allowed me to learn to manage the creation of a MetaModel, thanks to an Ecore model but also to handle a concrete syntax GMF framework that generates a graphical model editor.

Nevertheless, I regret having had difficulties creating the OCL constraints and diagram Editor. Having never used such tools and, I found myself quickly blocked on some parts.. I usually learn by myself, but for this project it was difficult because it was hard to find documentation about this topic.