Projet SI4
Serveurs d'Entreprise

PolyTweet

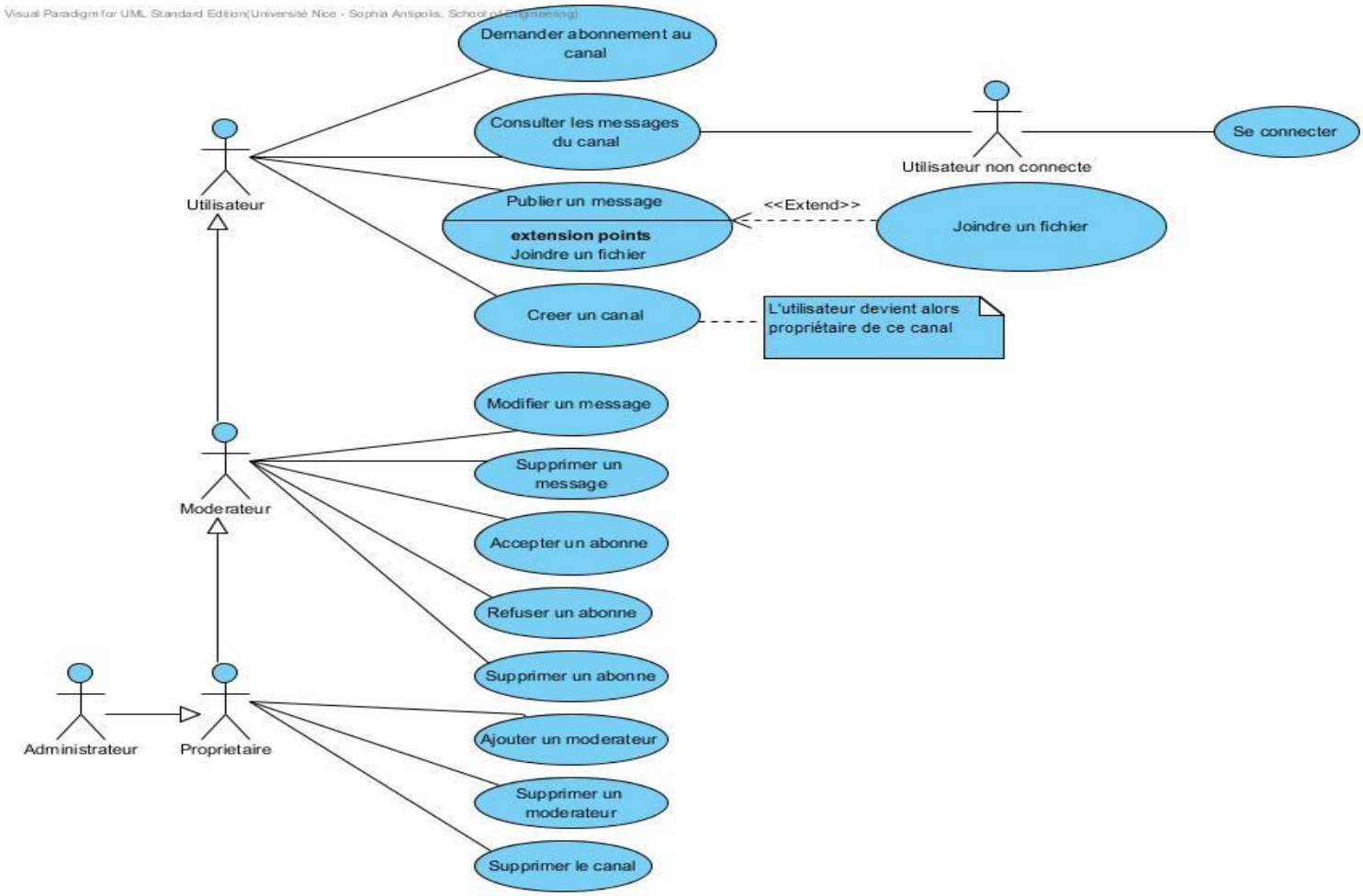
Martin Alfonsi – alfonsi@polytech.unice.fr

Axel Vicard – vicard@polytech.unice.fr

Table des matières

Diagrammes de cas d'utilisations

Diagramme de gestion de canal

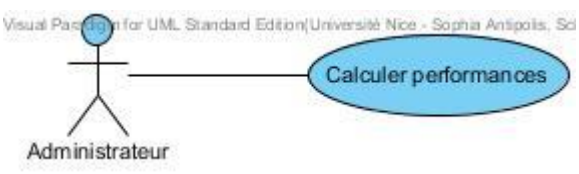


Un utilisateur non connecté n'a que 2 possibilités : il peut soit consulter les messages des canaux publics, soit se connecter et être alors reconnu comme un utilisateur du canal.

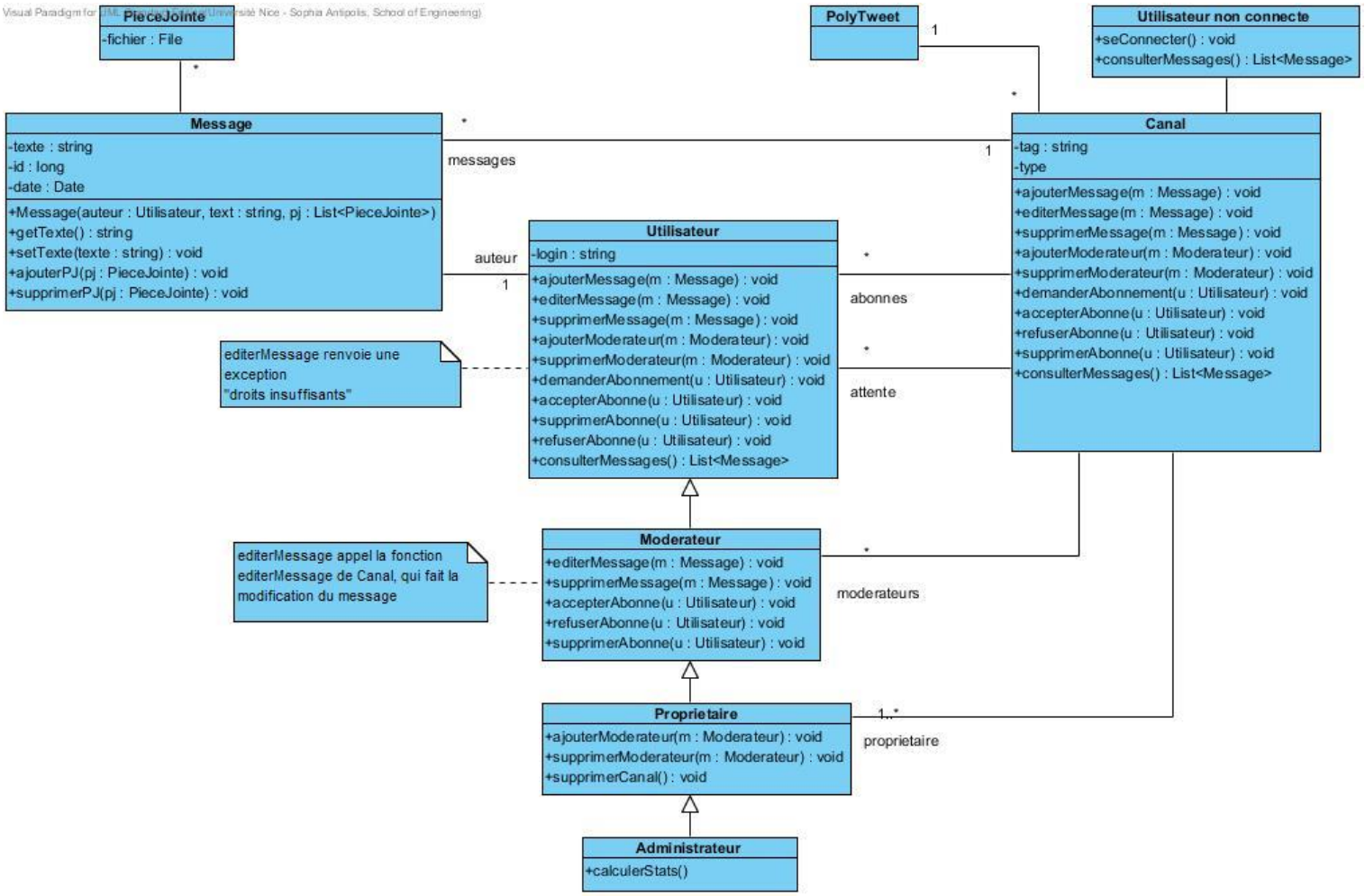
On utilise de l'héritage pour les acteurs car on considère qu'un modérateur est un utilisateur ayant des droits supplémentaires et qu'un propriétaire est un modérateur ayant également plus de droits que ce dernier.

L'administrateur est un utilisateur disposant de tous les droits, il hérite donc de propriétaire.

Diagramme de mesure de performances



Diagrammes de classes



Pour stocker les informations concernant les listes d'abonnés, nous avons retenu 2 possibilités :

- Chaque utilisateur dispose de la liste des canaux auxquels il est abonné
- Chaque canal dispose de la liste des utilisateurs abonné à ce dernier.

La première possibilité permet de voir simplement si l'on est abonné ou non à un canal, vu que l'utilisateur dispose de cette information dans ses attributs. Cependant si jamais on veut supprimer un utilisateur de la liste d'abonné, cela demande à aller modifier le profil de ce dernier.

La deuxième possibilité demande de vérifier les droits de l'utilisateur avant d'effectuer une action, cependant en cas de modification de la liste d'abonné, il suffit simplement de modifier l'information dans le canal.

La récupération des droits étant une manipulation qui n'a besoin de se faire qu'à la connexion au canal, nous avons choisi d'implémenter **la deuxième solution.** Ainsi lorsqu'on veut modifier un canal, il est inutile de parcourir toute la base de données afin de modifier chaque utilisateur.

Nous avons utilisé le même principe pour la gestion des listes de modérateurs et de propriétaires.

Table de la base de données

Table ***« canal »*** :

tag	type
#bde	prive

Table ***« piece_jointe »*** :

fichier	message	date
rapport.pdf	1	15-08-2014 12:00:50

Table ***« message »*** :

id	texte	canal
1	Ceci est un message	#bde

Puisque les relations sont de type 1-n, on utilise des clés étrangères afin de ne pas créer de nouvelles tables dans notre base de données.
Pour relier un message à son canal, on référence la clé étrangère « tag » de canal dans l'attribut « canal ». De la même façon on référence la clé étrangère « id » de message dans l'attribut « message » de piece_jointe.

Pour la gestion de l'héritage, on utilise une seule table ***« utilisateurs »*** car les modérateurs et les propriétaires n'ont pas d'attribut supplémentaire (ils ont seulement un nom.)

login
Alfonsi
Vicard
Soulier
Vella

On utilise ensuite plusieurs tables d'associations qui vont relier la table utilisateur et la table canal :
-Une table ***« abonnement »*** qui associe un utilisateur abonné à un canal

abonne	canal
Soulier	#bde
Vella	#bde
Alfonsi	#bde
Vicard	#bde

-Une table ***« modérateur »*** qui associe un modérateur à un canal

moderateur	canal
Alfonsi	#bde
Vicard	#bde

-Une table ***« propriétaire »*** qui associe un propriétaire à un canal

proprietaire	canal
Vicard	#bde

On utilise des tables d'associations car les utilisateurs peuvent être abonnés à plusieurs canaux et chaque canal possèdent plusieurs utilisateurs.

Pour l'architectural pattern, tout le code SQL va se trouver dans les fonctions de la classe Canal.

Pour les behavioral patterns, on utilise le lazy loading car on charge le statut de l'utilisateur une seule fois lors de sa connexion à un canal. En fonction du statut de l'utilisateur (utilisateur, modérateur ou propriétaire), on utilisera les fonctions de la classe associée.

Diagramme de composants

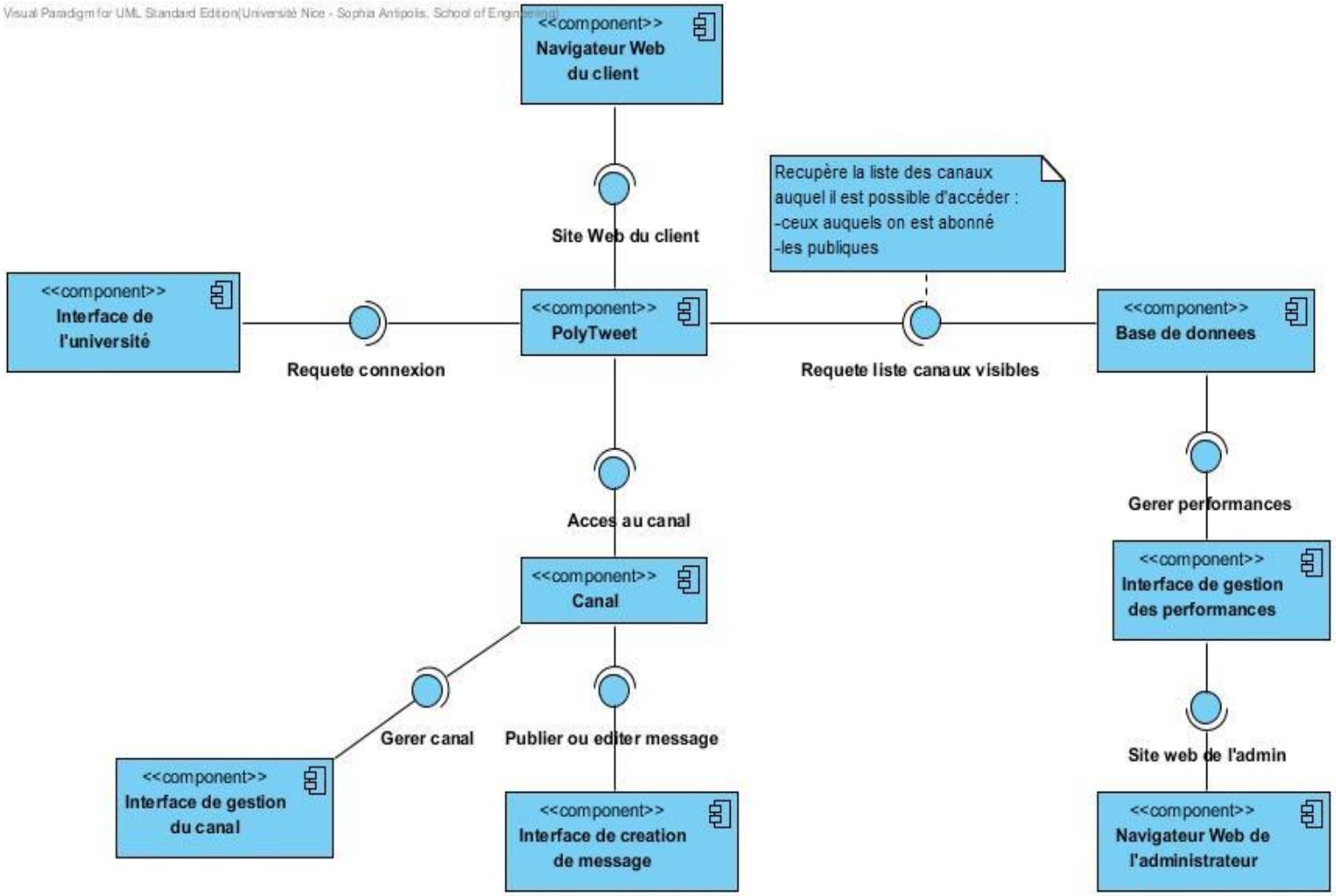


Diagramme de déploiement