**MASTER EN SCIENCES INFORMATIQUES**

# Conception d'une plateforme de publication collaborative pour les réseaux sociaux

Effayong, Wilfried; Leuni, Louis

*Award date:*
2020

*Awarding institution:*
Universite de Namur

Université de Namur
Faculty of Computer Science
Academic Year 2019–2020

# Conception d'une plateforme de publication collaborative pour les réseaux sociaux

LEUNI Louis Dégozard EFFAYONG Wilfried Ludovic


Supervisor : _____ (Signed for Release Approval - Study Rules art. 40)
Pr. Vincent ENGLEBERT

A thesis submitted in the partial fulfillment of the requirements
for the degree of Master of Computer Science at the Université of Namur

# Conception d'une plateforme de publication collaborative pour les réseaux sociaux

**LEUNI Louis Dégozard** et **EFFAYONG Wilfried Ludovic**

## Résumé

Les réseaux sociaux font partie du quotidien de millions de personnes aujourd'hui. L'une des difficultés que rencontrent certaines entreprises ou particuliers est de publier du contenu sur plusieurs comptes en même temps. En fait, si on a des comptes sur Facebook, Twitter, Youtube, Instagram et bien d'autres, comment faire pour y publier simultanément du contenu sur chaque compte tout en respectant les spécificités de chaque réseau social ? C'est à cette question que nous essayons de répondre dans ce mémoire. Nos axes de recherche reposent sur deux aspects : la structure de données et l'architecture de l'application. Concernant la structure de données, nous avons opté pour les ontologies. Quant à l'architecture, nous avons choisi une architecture Spring avec des plugins. Le choix des ontologies et des plugins nous a permis de gérer la particularité de chaque réseau social tout en facilitant l'extensibilité de l'application due au faite que les réseaux sociaux sont toujours en évolution permanente.

# Design of a collaborative publishing platform for social networks

**LEUNI Louis Dégozard** et **EFFAYONG Wilfried Ludovic**

## Abstract

Social media is part of the daily life of millions of people today. One of the difficulties that certain companies or individuals encounter is to publish content on several accounts at the same time. In fact, if you have accounts on Facebook, Twitter, Youtube, Instagram and many others, how do you simultaneously publish content on each account while respecting the specifics of each social network ? It is this question that we try to answer in this brief. Our lines of research are based on two aspects : the data structure and the architecture of the application. Regarding the data structure, we opted for ontologies. As for the architecture, we chose a Spring architecture with plugins. The choice of ontologies and plugins allowed us to manage the particularity of each social network while facilitating the extensibility of the application due to the fact that social networks are always in constant evolution.

# Remerciements

Nous voudrions tout d'abord remercier notre promoteur, le Professeur Vincent Englebert, pour avoir proposé un sujet du mémoire pertinent et pour l'encadrement qu'il nous a apporté tout au long de ce travail. En suite, nous adressons nos reconnaissances à tous nos proches qui ont eu la gentillesse de nous accorder leur temps pour tester le prototype de KWeSSi, ce qui nous a permis d'apporter certaines retouches.

# Table des matières

# Introduction

Aujourd'hui, nous constatons que les réseaux sociaux sont présents presque partout. Ils sont massivement représentés dans des entreprises, chez des particuliers. Nous pouvons y accéder à travers nos ordinateurs portables, nos smartphones, nos tablettes. Bon nombre d'entreprises et de particuliers souhaitent être visible sur les réseaux sociaux. Cette visibilité passe par une publication de contenus sur des réseaux tels que Facebook, Twitter, LinkedIn, Telegram, Instagram, etc. Les raisons pour être présent sur les réseaux sociaux sont différentes selon que l'on soit une entreprise ou un particulier.

Pour une entreprise, il s'agit de faire un marketing virtuel en essayant de cibler de potentiels consommateurs à partir de publicités qu'elles envoient sur leurs différentes plateformes sociales. Par exemple, une entreprise va publier sur son compte Facebook une information sur un nouveau produit qu'elle vient de développer, en y ajoutant une image ou une vidéo, synonyme d'expliquer au mieux le produit pour capter de nouveaux clients ou fidéliser les anciens. Les entreprises sont soucieuses de ce qui se dit sur elles sur les réseaux sociaux, elles essayent de prévoir leurs communications virtuelles, d'amener les internautes du monde entier à avoir une bonne image d'elles : c'est ce qu'on appelle la veille informationnelle des entreprises. Cette veille informationnelle peut permettre à l'entreprise d'anticiper sur les besoins des consommateurs, de lire les intentions des concurrents, d'avoir plus d'informations sur les nouvelles technologies en vigueur, de faire du ciblage promotionnel personnalisé.

Pour un particulier, il s'agit de partager ou de publier de l'information provenant de diverses sources (sites internet de presses, blogs, les quotidiens,. . . ) sur les différents réseaux sociaux. La publication de l'information et le partage d'information sont les plus utilisés par les particuliers. Par exemple, un particulier va publier sur son compte les particularités d'un club de football en y ajoutant une image ou une vidéo dans le but de partager l'un de ses hobbies avec ses amis virtuels. Il peut aussi partager un article, un lien, une photo provenant du « mur » d'un de ses amis. Les particuliers peuvent partager de l'information concernant aussi leurs quotidiens (photos, vidéos, message), certains aiment être au centre de l'attention et d'autres pas. Ils ne sont pas là pour vendre un produit ou un service comme le font les entreprises, ils sont là pour faire circuler de l'information auprès des autres utilisateurs, se divertir ou apprendre. En général, il s'agit d'une mise en scène de soi.

Le partage tel que nous avons expliqué dans les deux paragraphes précédents est une forme de collaboration, mais virtuelle. Pour les entreprises il s'agit de collaborer principalement avec ses clients, parfois avec ses fournisseurs et son personnel. Concernant les particuliers, la collaboration se fait généralement avec les « amis virtuels » qu'ils rencontrent sur chaque réseau social. À titre indicatif, nous constatons que certaines formes de collaborations qui se passaient de façon physique dans la société (conférences, groupes d'échanges et de travail, séminaires, groupes de fans, etc), se sont déplacées virtuellement sur les réseaux sociaux. Nous constatons aussi que des entreprises ou des particuliers sont inscrits sur plusieurs réseaux sociaux. Certaines entreprises créent même plusieurs comptes sur le même réseau social. La forte présence sur les réseaux sociaux des entreprises et des particuliers, la multiplication des comptes qui y sont créés ont suscitées notre attention d'où l'intérêt du sujet de mémoire. Notre question de recherche est donc : **Comment concevoir une plateforme collaborative (PFC) pour la publication de contenus sur les réseaux sociaux en couvrant certains besoins ?** Il s'agit d'avoir une plateforme permettant aux utilisateurs des réseaux sociaux de centraliser les publications qu'ils font sur leurs multiples comptes.

Notre mémoire se divise en trois grandes parties : la description des besoins, la littérature et la conception. La description des besoins nous aide à décrire l'ensemble des spécifications que vont couvrir la plateforme. La littérature représente la partie recherche de ce mémoire. Elle comprend :

- La présentation de quelques réseaux sociaux et leurs spécificités.
- Une description expliquant ce qu'est une plateforme collaborative.
- La description de quelques plateformes collaboratives qui existent sur le marché et qui peuvent déjà faire la publication de contenu sur les réseaux sociaux.
- La présentation des formes d'architectures et modélisation utilisées dans les PFCs.
- Outils pour manipuler les ontologies et outils pour concevoir les plugins en JAVA

La troisième partie s'attarde sur la manière dont nous allons concevoir notre propre PFC. Elle contient :

- La conception d'une base de connaissances pour l'application.
- La conception de l'architecture de notre plateforme collaborative.
- La présentation d'un prototype de l'application.
- Les améliorations qui pourraient être apportées à notre travail.

Ce projet a été réalisé dans le cadre du Master en sciences informatiques à l'Université de Namur. Il a été fait en binôme et ce, par choix et non par contrainte. Il nous ait déjà arrivé de réaliser quelques projets ensemble. Dès que l'opportunité de travailler en binôme nous a été proposée, nous n'avons pas hésité car nous connaissions déjà nos forces et faiblesses. Pour ce qui est du choix du sujet, il a été choisi dans une liste proposée par l'université. Le sujet actuel nous a immédiatement intéressé, nous avons au final été choisis par nos professeurs pour le réaliser.

Notre méthode de travail a commencé par une longue période de recherche qui nous a amené à collecter plusieurs articles scientifiques. La recherche des articles était basée principalement sur trois termes clés : plateforme collaborative, publications de contenus et réseaux sociaux. Nous avons par la suite trié ces articles pour ne retenir que ceux qui cadraient le mieux avec notre thématique, puis nous avons établi notre table des matières.

Les articles nous ont beaucoup aidés dans la partie littérature. Pour cette partie, nous nous sommes répartis les tâches et les articles. Chacun devait lire les articles qui étaient en relation avec ses tâches. Pour les parties description des besoins et conception, le travail n'a pas été repartie. Il a fallu que nous travaillions ensemble sur la rédaction des besoins afin de s'assurer que nous avions tous deux compris l'ensemble des fonctionnalités de la future application. Concernant la conception, nous avons travaillé ensemble mais l'un s'occupait du back-end tandis que l'autre s'occupait du front-end (l'interface utilisateurs). Malgré tout, nous discutions régulièrement pour éclaircir certains points d'ombre.

Pour consolider le travail en un seul document écrit, nous avons utilisé le logiciel en ligne Overleaf. Il s'agit d'un éditeur LaTeX permettant une bonne collaboration grâce à sa synchronisation et sa prévisualisation en temps réel du travail réalisé. Il ne doit pas être installé, seul le lien privé, un ordinateur et une connexion internet sont nécessaires. Overleaf nous permettait de travailler presque n'importe où, sans avoir besoin de transporter des fichiers sur un support physique.

Nous avons opté pour une méthode de travail Agile. Les méthodes agiles sont des méthodes de gestion de projets qui se veulent plus pragmatiques, impliquent le demandeur (client) et permettent une grande réactivité à ses demandes. Les méthodes agiles mettent en avant quatre valeurs fondamentales :

- Les Individus et leurs interactions plus que les processus et les outils.
- Des logiciels opérationnels plus qu'une documentation exhaustive.
- La collaboration avec les clients plus que la négociation contractuelle.
- L'adaptation au changement plus que le suivi d'un plan.

L'application du travail Agile passait par la détermination des itérations. Nous en avons eu trois. Chaque itération correspondant à une partie du travail. Dans chaque itération, nous avons au tout début définit des sprints de deux semaines. Mais ils se sont avérés trop longs, de tel sorte que lors du brainstorming qui avait lieu tous les derniers dimanche du sprint, nous n'arrivions pas à terminer les discussions de clôture du sprint. Suite à ce problème, nous avons ramené la durée des sprints à une semaine. Cette fois ci, chacun devait soumettre son travail sur Google drive au plus tard le dernier lundi du sprint à 12h. Le brainstorming était maintenant prévu le mardi qui suivait à 17h30. Lors des brainstormings, nous discutions des feedbacks du professeur (notre client), nous échangions sur nos différents questionnements et nous déterminions le contenu du prochain sprint.

# Première partie

# Description des besoins issus de la documentation

Avant de commencer ce travail, nous avons reçu plusieurs documents qui expliquent les fonctionnalités des futurs utilisateurs. Cette documentation se trouve dans l'annexe, à la fin de ce mémoire. Il s'agit des annexes suivantes :

- A. Rédiger une news
- B. Consulter une news
- C. Administration
- D. Ajout de réseaux sociaux
- II. Le diagramme entité - relation

L'objectif recherché dans cette partie est d'exploiter ces documents dans le but de dénicher les besoins des utilisateurs. Les besoins que nous avons trouvés en exploitant les documents sont les suivants :

## 0.1 Les besoins fonctionnels

### 0.1.1 Inscription d'un utilisateur

Une personne pourra s'inscrire sur l'application en fournissant son pseudo, nom, prénom, email, mot de passe, date de naissance. Par défaut une personne inscrite à un rôle d'utilisateur.

### 0.1.2 Traiter un utilisateur

Un administrateur sélectionnera les utilisateurs qui se sont inscrits et leur donnera l'autorisation d'utiliser le site ou pas (activé ou désactivé). Il pourra aussi bloquer ou supprimer un utilisateur. Un utilisateur bloqué peut toujours être débloqué par un administrateur. Par contre, lorsqu'un utilisateur est supprimé il ne peut plus utiliser les services que proposent l'application. Si un utilisateur est supprimé et veut réutiliser les services de l'application, il devra se réinscrire.

### 0.1.3 Se connecter

Un utilisateur inscrit pourra se connecter à l'application en fournissant son pseudo (ou son email) et son mot de passe.

### 0.1.4 Ajouter un rôle à un utilisateur

Un administrateur peut donner des rôles à chaque utilisateur. Les rôles possibles sont : utilisateur, membre, rédacteur, censeur, modérateur, administrateur. Une personne peut avoir plusieurs rôles.
Un utilisateur est celui qui utilise les services de l'application sans avoir aucun droit d'ajout, de modification ou de suppression (CRUD). Il n'est pas encore inscrit.
Un rédacteur est un utilisateur qui a le droit de rédiger un article. Il peut aussi supprimer son article du moment où il n'est pas publié.
Un modérateur est celui qui va valider toutes les suggestions de sujet pour un article. C'est lorsqu'il aura validé une suggestion qu'un rédacteur pourra le rédiger.
Le censeur est celui qui décide si un article peut être publié sur un réseau social ou pas. Il maîtrise bien les règles de publication sur ledit réseau social. Les règles peuvent être : le nombre de caractères permis, les termes interdits, la taille maximale des photos et des vidéos, le format des pièces jointes acceptables, les contraintes de connexion, etc. Le censeur du réseau social Twitter publiera les articles sur Twitter tandis que le responsable de LinkedIn décidera des articles qui seront publiés sur LinkedIn. Un censeur de réseau social pourra donc bloquer, supprimer ou publier un contenu sur ses réseaux sociaux. Un membre peut être censeur de plusieurs réseaux sociaux.
Un administrateur est celui qui assure les tâches d'administration de l'application.

### 0.1.5 Rédaction d'un article

Un rédacteur choisira une suggestion d'articles et va le rédiger. Les suggestions d'articles sont proposées par tous les membres. Un article peut avoir les éléments suivants : un titre, une description, une vidéo, un audio, une photo, un lien de référencement. La rédaction d'un article obéit à un workflow.

En fait, lorsqu'un sujet d'article est accepté par un modérateur il passe à l'état « A_REDIGER ». Dès lors, un rédacteur peut choisir un sujet qui a été accepté et le rédiger. À tout moment le rédacteur peut supprimer ou enregistrer (état « ENREGISTRE ») sa rédaction. Si il le supprime, tout ce qu'il avait écrit disparait mais le sujet reste. Une autre personne pourra choisir ce sujet et le rédiger. Dès que la rédaction est terminée, le rédacteur le valide et il passe à l'état « VALIDE ». À ce stade sauf un censeur ou un administrateur peut modifier l'article. Si le censeur ou l'administrateur publie l'article, il passe à l'état « PUBLIE ».

### 0.1.6 Consultation d'un article

Un utilisateur peut consulter les articles qui sont publiés. Il peut voir non seulement le contenu en tant que tel, mais aussi la date de publication et le rédacteur de l'article.

### 0.1.7 Traiter un article

Tant qu'un article n'est pas publié, son rédacteur peut toujours le modifier, le supprimer ou l'attribuer à un autre rédacteur. Un administrateur peut aussi changer le rédacteur d'un article. Un rédacteur a la possibilité de modifier un article mais c'est le censeur qui le publie sur les réseaux sociaux dont il a la responsabilité.

### 0.1.8 Recycler ou customiser un article

Un rédacteur a la possibilité de récupérer un article déjà publié et le modifier. Dès qu'il a terminé un censeur pourra le publié sur un autre réseau social. Par exemple, il peut prendre un article qui était publié sur Facebook, y ajouter une photo et une vidéo, puis un censeur le publie sur Twitter et Instagram.

### 0.1.9 Traiter un réseau social

Un administrateur est la seule personne pouvant ajouter un nouveau réseau social à l'application. Il peut aussi supprimer ou changer le censeur d'un réseau social. Un réseau social est défini par son nom et ses crédentials.
Exemple :

- Nom : Twitter
  Credentials : consumerKey, consumerSecret, accessToken, accessTokenSecret.
- Nom : Facebook
  Crédentials : accessToken, appId.

### 0.1.10 Ajouter un champ à un article

A chaque moment un administrateur pourra ajouter, modifier (les informations sur un champ) ou supprimer un champ au formulaire de rédaction d'un article. Les possibilités sont de : type texte, date, email, image, vidéo, fichier.

### 0.1.11 Envoyer un mail

Lorsqu'une personne s'inscrit, un email lui est envoyé automatiquement pour qu'il valide son adresse email. L'email contiendra un lien sur lequel l'utilisateur cliquera pour activer son compte. Un administrateur peut envoyer un email à un utilisateur, un groupe, ou à tous les utilisateurs. Lorsqu'une inscription est refusée ou validée, un email est envoyé à l'utilisateur. Quand un article est publié, le rédacteur dudit article reçoit également un email. Ces emails ne sont pas des articles dans la mesure où ils ne sont pas faits pour être directement publiés sur les réseaux sociaux.

## 0.2 Les besoins non fonctionnels

### 0.2.1 Fournir une couche de sécurité à l'application

L'application à développer doit avoir une couche de sécurité pour les logins, mot de passe, les crédentials des différents réseaux sociaux. La façon dont les données sont gérées doit tenir compte de l'aspect sécuritaire.

### 0.2.2 Interactivité

L'application sera une plateforme accessible et utilisable sur plusieurs périphériques (ordinateur, smartphone, tablette). L'application demandera sans doute beaucoup d'interaction. En fait, on s'attend à ce que les utilisateurs interagissent beaucoup avec l'application KWeSSi [1] en y publiant du contenu.

### 0.2.3 Extensibilité et flexibilité

À tout moment il sera possible d'ajouter une fonctionnalité à l'application sans modifier le code source. Par exemple, l'ajout d'un réseau social doit se faire sans modification manuelle du code source de l'application ni de sa structure de données. Cela est aussi valable pour la customisation d'un message. Cette customisation doit être facile à réaliser c'est-à-dire qu'un utilisateur ne doit pas mettre trop de temps pour effectuer cette tâche. Pour ce qui est de la publication sur les réseaux sociaux, elle doit automatiquement tenir compte des spécificités de chaque réseau social. En fait, le nombre de caractères, la taille des photos, etc. peut-être différents selon qu'on veut publier sur Facebook, Twitter ou LinkedIn.

1. KWeSSi est composé de deux mots : KWe qui signifie rassembler et SSi qui veut dire un endroit, une place, un lieu. KWeSSi signifie alors rassembler en un lieu, faire une réunion ou meeting, mélanger plusieurs choses en langue Nufi. Le parallèle avec notre travail est que nous mettons en commun différents idées, plusieurs concepts et technologies au sein de notre application.

# Deuxième partie

# Littérature

# Chapitre 1

# Contexte

Aujourd'hui, on serait 7,6 milliards d'habitants sur terre. Il y aurait 4,1 milliards d'internautes qui représentent 54% d'habitants. Parmi les 4,1 milliards d'internautes, on compte 3,3 milliards d'utilisateurs des réseaux sociaux (qui représentent 43% de la population mondiale). Les plus actifs seraient 2,8 milliards, ce qui représente 37% d'habitants. Nous constatons que le nombre d'abonnés sur les réseaux sociaux a explosé depuis les années 2000 et cela continue de nos jours grâce aux succès de nouveaux appareils numériques tels que les Smartphones, tablettes. Les utilisateurs les plus actifs auraient entre 25 et 34 ans. Jusqu'en juillet 2018, la répartition des chiffres serait la suivante[1] :

- 70% des utilisateurs de Snapchat auraient entre 18 et 34 ans.
- 50% des utilisateurs d'Instagram auraient entre 18 et 34 ans.
- 40% des utilisateurs de Twitter auraient entre 18 et 34 ans.
- 38% des utilisateurs de Facebook auraient entre 18 et 34 ans.
- 38% des utilisateurs de LinkedIn auraient entre 18 et 34 ans.

On peut penser que vu le grand nombre de personnes abonnées aux réseaux sociaux, les entreprises peuvent y voir une opportunité afin d'accroître leurs notoriétés, gérer leurs réputations, faire du social selling (vente sur les réseaux sociaux), animer la relation avec leurs clients. Ainsi, ces entreprises vont s'inscrire, par exemple, sur Facebook, Twitter ou LinkedIn dans le but de saisir les opportunités qui peuvent s'y trouver.

À titre d'exemple une entreprise comme la RTBF (Radio Télévision Belge Francophone) aurait au moins 10 comptes sur Facebook. Nous pouvons citer le compte Facebook de RTBF Info, RTBF Sport, Le Grand Cactus - RTBF, La Première - RTBF, Cap 48 – RTBF, etc. Nous constatons aussi que la RTBF a un compte sur LinkedIn. Pour ce qui est de l'Université de Namur, elle aurait trois comptes sur Facebook à savoir : « Université de Namur », « UNamur Alumni » et « AGE - Assemblée Générale des Étudiants de l'UNamur ». En plus elle possède un compte sur LinkedIn. On voit bien, à travers ces deux exemples (non exhaustifs) que les entreprises veulent marquer leurs présences sur les réseaux sociaux.

Étant donné qu'il existe une multitude de réseaux sociaux, les entreprises qui souhaitent saisir les opportunités présentent sur ceux-ci vont devoir créer un compte sur chaque réseau social afin d'y publier du contenu. Le fait de gérer plusieurs comptes peut être une tâche laborieuse. Par exemple, l'administration séparée des comptes : qui publie quoi, sur quel compte (Facebook, Twitter, LinkedIn) ? Il n'est pas évident de contrôler ce genre de tâche si on publie du contenu sur plusieurs réseaux sociaux. Nous constatons donc, qu'il y a un besoin d'avoir une plateforme permettant de centraliser la publication de contenu. C'est dans ce constat que va se placer le besoin d'avoir une plateforme collaborative permettant de faire la publication de contenus sur les réseaux sociaux.

Cette plateforme contiendrait la gestion collaborative des contenus à publier, en permettant par exemple à plusieurs personnes de travailler sur le même contenu. Un contenu étant un texte avec une vidéo, photo, audio, etc. La plateforme peut aussi avoir une gestion des utilisateurs qui publient ces contenus. La plateforme permettrait aussi la gestion des différents réseaux sociaux. Dans ce contexte on peut envisager une possibilité de recherche sur des utilisateurs ou sur

des contenus. Pour être plus complète, la plateforme doit répondre aux besoins décrits dans la partie « Description des besoins », faite plus haut. Dans la partie courante de ce travail, nous allons investiguer pour savoir si une telle plateforme existe. Bien plus, nous allons chercher aussi à savoir comment la concevoir, en mettant en exergue l'aspect architectural et l'aspect modélisation.

Dans le cadre de ce mémoire, la conception d'une plateforme collaborative pour la publication de contenus sur les réseaux sociaux implique la consultation de plusieurs sujets clés de la littérature.

Premièrement, nous allons nous renseigner sur les réseaux sociaux et les spécificités de chacun. En effet, il faudrait comprendre le fonctionnement des réseaux sociaux pour savoir concrètement comment les contenus y sont publiés.

Deuxièmement, nous allons nous informer sur les plateformes collaboratives. Cela va nous permettre d'identifier les fonctionnalités qu'elles proposent et voir les rapports qu'elles ont avec les réseaux sociaux.

Troisièmement, à ce stade nous sommes à mesure de comprendre correctement les plateformes collaboratives. Avec cela, nous allons présenter quelques solutions actuelles, si elles existent, et permettant de publier du contenu sur les réseaux sociaux. Pour chaque application que nous allons trouver, nous allons comparer ses fonctionnalités avec les besoins que nous avons reçus.

Quatrièmement, nous allons étudier en profondeur les plateformes collaboratives en insistant sur quelques architectures qu'elles utilisent. Cela nous aidera à comprendre l'architecture desdites plateformes dans le but de savoir si cette architecture répond à l'exigence d'extensibilité décrit dans la partie «Description des besoins». On peut se demander déjà si cette architecture utilise les technologies comme Spring, OSGI, les plugins, les Framework web.

Cinquièmement, nous allons continuer à faire une étude profonde. Cette fois ci, il est important de savoir comment la modélisation et le stockage peut se faire sous les plateformes collaboratives. Par exemple, utilisent-elles une base de données relationnelle, du NoSQL ou des ontologies ? Cette recherche va nous permettre de savoir plus précisément si la manière de modéliser et de stocker de l'information satisfait bien les besoins décrits précédemment.

Sixièmement, étant donné que nous avons déjà étudié les formes d'architecture et de modélisation utilisées par les PFC, nous allons pouvoir choisir la forme de modélisation et d'architecture adaptée pour développer notre plateforme collaborative KWeSSi. Pour chacun de nos choix, nous allons expliquer les raisons qui nous motivent.

# Chapitre 2

# Les réseaux sociaux

Nous allons devoir faire une application dans laquelle on pourra rédiger plusieurs types de contenus. Un contenu étant un texte avec une vidéo, photo, audio, etc. Ces contenus seront publiés sur différents réseaux sociaux tels que Facebook, Twitter, LinkedIn, etc. Dans ce chapitre nous allons définir ce qu'est un réseau social, quelles sont les particularités de quelques réseaux sociaux existants et comprendre le fonctionnement des réseaux sociaux au niveau de la publication de contenus.

## 2.1 Définition

Un réseau social est d'abord un site internet. Les réseaux sociaux sont comme des services Web permettant aux utilisateurs de créer un profil public ou semi-public dans un système lié, d'énoncer une liste d'autres utilisateurs avec lesquels ils partagent une connexion, consulter et parcourir leur liste de connexions et celles établies par d'autres personnes au sein du système. La nature et la nomenclature de ces connexions peuvent varier d'un site à l'autre[2].

De nos jours, les applications de réseaux sociaux tels que Facebook, Twitter, LinkedIn,... font parties de notre quotidien, nous nous en servons pour partager de l'information, pour communiquer avec d'autres personnes (famille, amis, collègues,...), pour faire du commerce, pour s'entraider, etc. Vers la fin de l'année 2007 on dénombrait une centaine de réseaux sociaux[2]. Pour les années supérieures à 2007, nous n'avons pas trouvé de chiffres mais notre intuition nous laisse croire que le nombre 100 est aujourd'hui considérablement dépassé.

Pour utiliser un réseau social, une personne physique ou morale devra d'abord s'inscrire en créer son profil. Par la suite, la personne pourra paramétrer son compte en y ajoutant, supprimant ou modifiant des données personnelles (photos, vidéos, textes,..). Enfin, la personne pourra effectuer un certain nombre d'actions au sein de son compte : ajouter différents utilisateurs dans son réseau, publier du contenu, diffuser et partager de l'information.

## 2.2 Les différents réseaux sociaux

Nous allons citer, définir et expliquer les particularités des réseaux sociaux tels que Facebook, Twitter, LinkedIn qui feront partie de l'application KWeSSi lors de son lancement.

Afin de publier du contenu sur ces réseaux sociaux, nous avons besoin d'un compte pour chaque réseau social. Grâce à cela, les API des réseaux sociaux nous donnerons des tokens d'accès, qui sont des crédentials permettant d'établir une connexion distant avant de publier du contenu sur les pages des différents réseaux sociaux.

### 2.2.1 Facebook

Facebook est un réseau social en ligne créé le 4 février 2004 par Mark Zuckerberg et ses amis. Le site compte beaucoup d'utilisateurs, ils sont 2,2 milliards inscrits en 2018 [1]. L'objectif premier de Facebook était de créer des

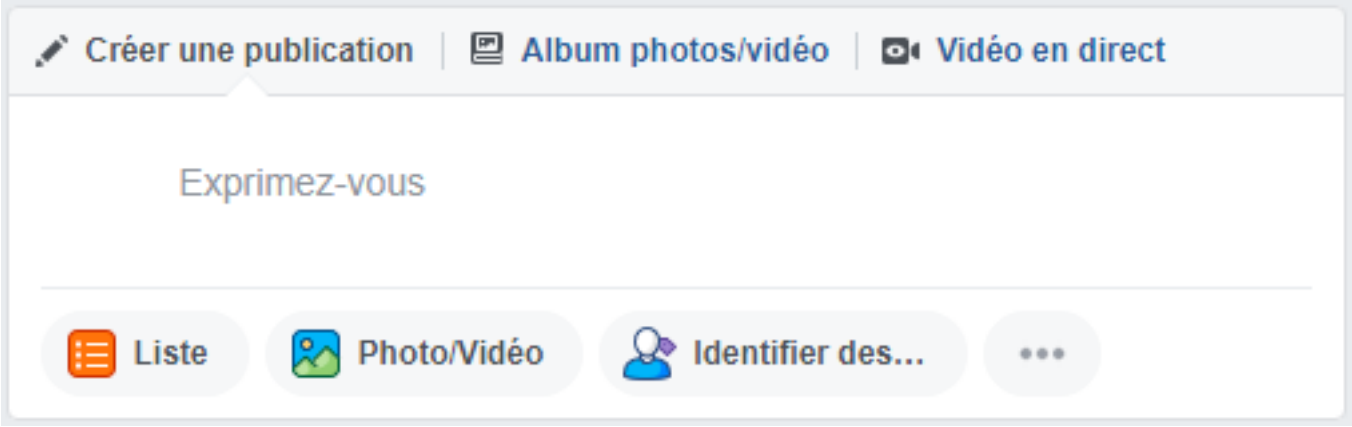
1. http ://www.alexitauzin.com/2013/04/combien-dutilisateurs-de-facebook.html

relations entre des personnes qui sont proches d'où son slogan «*Facebook is a social utility that connects you with the people around you.*».

Facebook permet de créer un profil avec lequel il sera possible d'éditer et partager différents contenus : photos, vidéos, streaming, fichiers, des statuts pour exprimer ses opinions, des évènements (anniversaires, concerts, jobs, meeting, ...) et utiliser une variété d'applications afin de se distraire avec d'autres utilisateurs. Par exemple, UNO (un jeu de carte). Il est aussi possible de lier le réseau social Twitter à Facebook afin de publier différents contenus de manière automatique sur ces réseaux sociaux. Par exemple, lorsque l'utilisateur crée un article sur Facebook, il peut le poster simultanément sur Twitter. Les différentes publications existantes sur Facebook ont leurs niveaux de confidentialité :

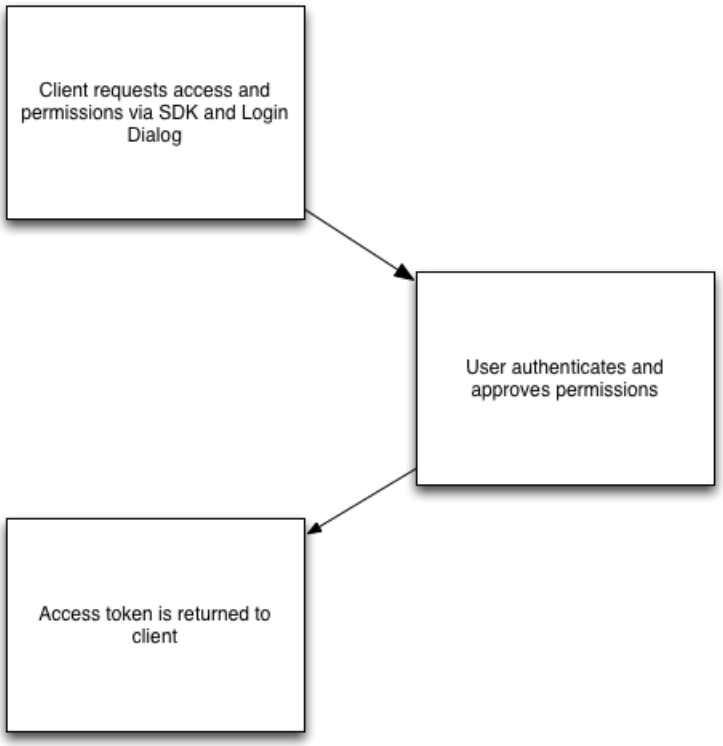
- Public (vue par tous les utilisateurs de Facebook)
- Amis (vue par ses contacts)
- Amis sauf (vue par ses contacts sauf certaines personnes)
- Amis spécifiques (vue par un groupe restreint de ses contacts)
- Moi uniquement (vue seulement par l'utilisateur)

Sur l'image ci-dessous, la publication sur Facebook se fait facilement par son champ "Exprimez-vous" qui met à disposition les différents types de contenus (photo, vidéo,...) qui pourront enrichir la publication. Si notre publication est appréciée, elle recevra un certain nombre de « Like ». Donc elle aura un fort risque d'être partagé par des utilisateurs à d'autres utilisateurs.




Avec notre application KWeSSi, nous allons devoir publier des informations sur notre page Facebook grâce aux différents « token d'accès » [2] (longues chaînes de caractères) qui permettent de s'identifier et d'avoir un accès provisoire et sécuriser aux API Facebook. Ces tokens d'accès ont une date d'expiration et un nom correspondant à l'application qui les appelle. Le schéma ci-dessous repris sur le site developers.facebook.com, montre la requête d'obtention de token d'un utilisateur.

2. https ://developers.facebook.com/docs/facebook-login/access-tokens ?locale=fr_FR




Voici les différents types de tokens d'accès :

**Tokens d'accès utilisateur**

Les tokens d'accès utilisateur sont les plus utilisés car ils permettent de se connecter au compte utilisateur de la personne afin de lire, modifier ou écrire des informations sur ce compte. Il existe plusieurs plateformes différentes développées en JavaScript, Android, etc. et qui proposent des solutions pour obtenir les tokens d'accès utilisateurs et de les gérer.

Malheureusement les tokens ont une certaine durée de vie, court pour certains, long pour d'autres. Quand les tokens ont une durée de vie courte, elles sont généralement d'une heure ou 2 si on les obtient via une connexion web. Par contre, elles peuvent avoir une durée de vie longue de maximum 60 jours si on les obtient par des API côté serveur. Cette limite de durée est une mesure de sécurité, si les tokens sont actifs très longtemps cela donne du temps à un hacker pour les récupérer.

**Tokens d'accès d'app**

Les « tokens d'accès app » servent essentiellement à lire et modifier les paramètres de l'API Facebook. On génère ces tokens en passant un appel à l'API Graph «Il s'agit d'une API http de bas niveau qui permet aux applications d'avoir recours à la programmation pour interroger des données, publier de nouvelles actualités, gérer des publicités, importer des photos et réaliser un large éventail d'autres tâches » [3], sur un compte Facebook.

**Tokens d'accès de Page**

Ces tokens ressemblent aux tokens utilisateurs, cependant ils permettent juste de lire, écrire ou modifier les données d'une Page Facebook. Pour obtenir ces tokens, il faudra recevoir les tokens d'accès utilisateur et demander l'autorisation manage_pages afin d'obtenir le token d'accès de Page via l'API Graph.

**Tokens client**

Le token client est propre à l'API Facebook, il permet d'accéder à une petite partie des API de l'application.

3. https ://developers.facebook.com/docs/graph-api/overview/

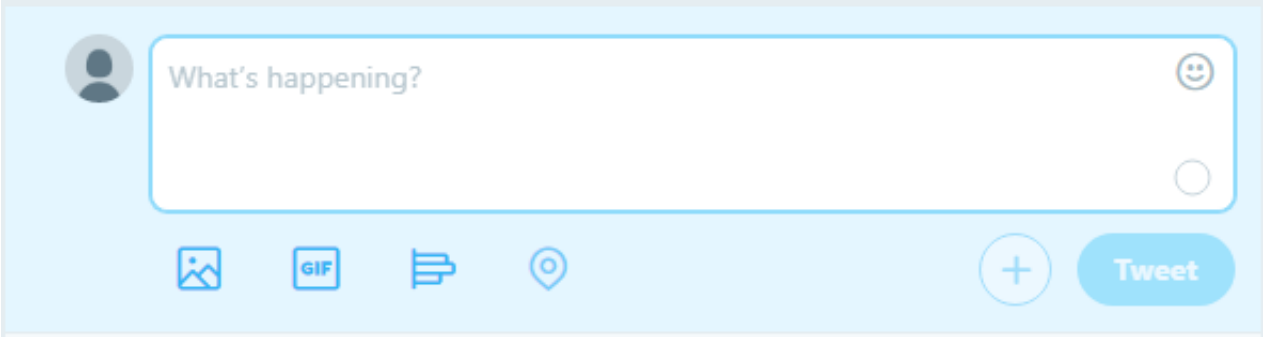
Dès lors que nous recevons les accès tokens afin de se connecter au compte Facebook, nous aurons la possibilité d'effectuer nos publications en respectant les recommandations de Facebook sur certains champs :

| Champs | Dimensions | Type | Format | Durée | Taille |
|---|---|---|---|---|---|
| Image [4] | 1200 x 630 px | JPG ou PNG | | | |
| Vidéo [5] | | | MP4 ou MOV | 120 minutes | 4 GO |

### 2.2.2 Twitter

Twitter est un réseau social en ligne créé le 21 mars 2006 par Jack Dorsey, Noah Glass, Biz Stone et Evran Williams. Il compte 335 millions d'utilisateurs actifs en 2018 [6]. Leur slogan est « *Quoi de neuf ? – Découvrez ce qui se passe en ce moment chez les personnes et dans les organismes qui vous tiennent à cœur – Suivez vos passions.*».

Twitter permet de rédiger des minis blogs que l'on peut publier sous différents contenus : texte (avec quelques contraintes), photo, vidéo. Par exemple, on ne pourra publier du texte qu'avec un maximum de 280 caractères contrairement à Facebook. Cette nouvelle mise à jour est arrivée en septembre 2017. Lors de la publication des contenus, les utilisateurs utilisent beaucoup les hashtag « # » suivi d'un mot ou du nom de profil de l'utilisateur, ce qui lui permettra d'être notifié de toutes informations le concernant.




En se basant sur nos différentes utilisations de twitter, nous observons que les utilisateurs ont la possibilité de suivre différents comptes qui traitent d'un sujet particulier. Plus un sujet est aimé par des personnes, plus la personne concernée aura des « Followers » (suivi) et une grande influence sur d'autres utilisateurs. Ainsi de nombreuses entreprises ou particuliers vont profiter de cela pour faire de la publicité ou pour faire du commerce.

Comme pour Facebook, l'application KWeSSi devra générer les tokens d'accès et les tokens secrets afin de pouvoir publier du contenu sur notre compte Twitter. Pour générer un token d'accès [7] il faut :

1. S'inscrire sur l'application Twitter et avoir demandé ou approuver un compte de développeur.
2. Se connecter aux portails des développeurs
3. Ouvrir l'application Twitter
4. Se rendre à la page clé et jetons
5. Choisir créer dans la section token d'accès et secret du token d'accès

Dès lors que nous recevons les accès tokens afin de se connecter au compte Twitter, nous aurons la possibilité d'effectuer nos publications en respectant les recommandations de Twitter sur certains champs :

| Champs | Dimensions | Tailles | Formats |
|---|---|---|---|
| Image [8] | 440 x 220 px minimum<br>1024 x 512 px maximum | 3 Mo minimum,<br>5 Mo maximale | Gif animé JPG, PNG ou Gif |

| Champs | Résolution | Formats | Durée | Formats |
|---|---|---|---|---|
| Vidéo [9] | entre 32 x 32 et 1920 x 1200 | MP4 ou MOV | 30 secondes | 512 Mo |

6. http ://www.alexitauzin.com/2013/04/combien-dutilisateurs-de-facebook.html
7. https ://developer.twitter.com/en/docs/basics/authentication/guides/access-tokens.html

| Champs | Nombre de caractères |
|---|---|
| Texte | 280 |

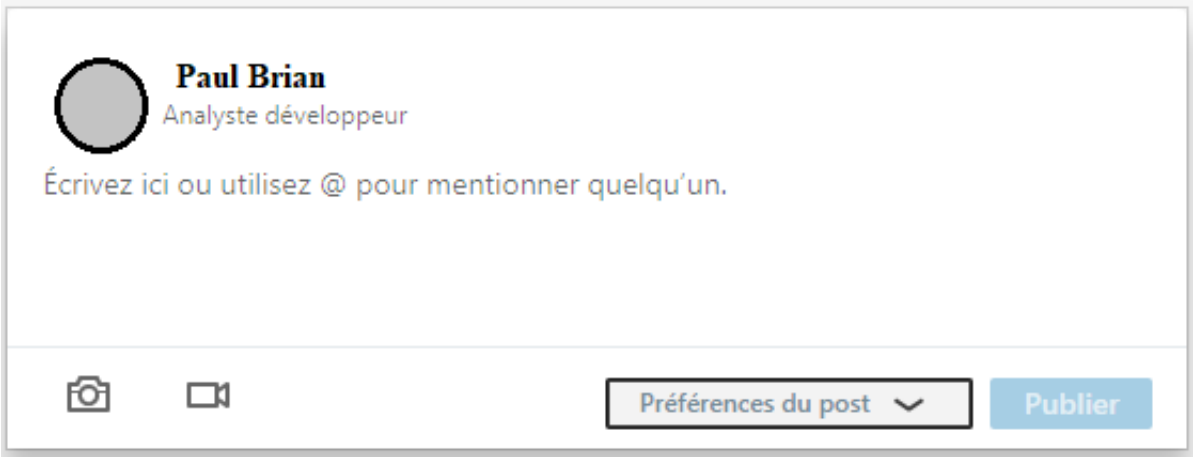
### 2.2.3 LinkedIn

LinkedIn est un réseau social professionnel en ligne créé en 2003 par Reid Hoffman, Allen Blue, Jean-Luc Vaillant et deux autres entrepreneurs.
LinkedIn permet à ses utilisateurs de créer un profil professionnel, comme un CV dans lequel les utilisateurs partagent leurs formations, leurs certifications, leurs expériences, leurs facultés linguistiques, etc. afin de trouver des opportunités de carrière. Des recruteurs, employeurs ou autres auront la possibilité de parcourir notre profile et de nous faire une proposition d'emploi ou pas. Aussi, les utilisateurs pourront être amis avec d'autres personnes afin d'agrandir leurs réseaux professionnels.

Vu que LinkedIn est un réseau social, les utilisateurs auront aussi la possibilité de publier différents contenus sur leurs pages avec une certaine visibilité :

— Public (quiconque sur LinkedIn ou en dehors)
— Public et Twitter (quiconque sur le réseau social LinkedIn et Twitter ou en dehors)
— Relations (nos relations (amis) sur LinkedIn)

Les contenus peuvent être de types photos, vidéo ou texte avec la possibilité de citer un utilisateur faisant parti de nos relations grâce à l'utilisation de « @ ».




Notre application KWeSSi aura la possibilité de publier sur notre profil de LinkedIn via le protocole d'authentification OAuth 2.0 [10] qui garantira la sécurité et l'intégrité des données lors de l'accès à notre compte.

Afin que KWeSSi puisse passer des appels d'API authentifiés à LinkedIn, il devra utiliser OAuth 2.0 [11] et suivre ces différentes étapes :

1. Configurer notre application LinkedIn
2. Demander un code d'autorisation
3. Echanger le code d'autorisation contre un token d'accès
4. Effectuer des demandes autorisées afin d'accéder à LinkedIn
5. Actualiser le token d'accès afin d'éviter une expiration de ce dernier

Enfin, lorsque nous obtenons les accès tokens pour se connecter à notre compte LinkedIn, nous aurons la possibilité de publier du contenu tout en respectant les recommandations de LinkedIn sur certains champs :

10. https ://oauth.net/2/
11. https ://developer.linkedin.com/docs/oauth2#

| **Image** [12] | **Dimensions** | **Formats** |
|---|---|---|
| Image | 360 x 265 px | PNG, JPG ou Gif |

| **Champs** | **Résolution** | **Durée** | **Formats** | **Taille** |
|---|---|---|---|---|
| Vidéo [13] | De 256 x 144 à 4096 x 2304 | De 3 secondes à 600 secondes(10 min) | MP4, AVI, MKV,... | Minimum 75 Ko, Maximum 5 Go |

Nous venons de terminer la présentation de quelques réseaux sociaux et leurs API. Pour résumer, les API fournissent les crédentials pour publier du contenu. Qu'en est-il des autres réseaux sociaux ? Concernant Google plus et Instagram, ils ont aussi leurs propres API. Malheureusement nous n'avons pas trouvé d'articles qui expliquent la manière dont la publication se fait sur d'autres réseaux sociaux. Néanmoins, nous savons que la plupart des réseaux sociaux utilisent les crédentials, au minimum le login et le mot de passe. Selon A. Bilami et al, le système de crédentials utilise une ou plusieurs politiques de sécurité (OAuth 2.0, Secure Socket Layer (SSL)) et un système de certificats afin de permettre le contrôle d'accès. En conséquent, le concept de gestion de la confiance se limite aux règles de politiques définies par chaque application, par exemple la politique de sécurité de LinkedIn lui est propre.

L'étude des réseaux sociaux nous montre que chaque réseau social a ses spécificités. Il peut s'agir de la taille des textes, la taille des images et vidéos, etc. La question à se poser est de savoir comment publier un message sur chaque réseau social de manière à respecter les spécificités de chaque réseau social ? C'est l'une des questions que tentera de répondre notre application nommée KWeSSi.

# Chapitre 3

# Plateforme collaborative (PFC)

Notre mémoire se concentre sur la conception d'une plateforme collaborative pour la publication de contenu sur les réseaux sociaux. Il est donc nécessaire de savoir dans les grandes lignes, ce qu'est une plateforme collaborative. De nos jours plusieurs entreprises mettent l'accent sur la qualité de la communication et de la collaboration en leurs seins et n'hésitent pas à investir pour avoir des logiciels (par exemple les PFCs) capables d'être une solution efficace à ce besoin. Cette observation nous amène à nous demander c'est quoi une PFC ? Pourquoi opter pour une PFC ? Quelle sont les caractéristiques ou fonctionnalités d'une PFC ? On se rend également compte que les entreprises sont soucieuses de leurs visibilités. Avant, cette visibilité se faisait via les médias traditionnels comme la télévision, la radio, la presse. Aujourd'hui, avec l'omniprésence des réseaux sociaux ces entreprises les utilisent de plus en plus pour leurs besoins de visibilités. Dès lors vu la multitude des réseaux sociaux, quel rapport existe-il entre les PFCs et les réseaux sociaux ? Dans ce chapitre nous allons présenter les PFCs en répondant aux questions posées ci-dessus. Il est possible que nous abordions certains points qui ne sont pas liés directement aux dites questions mais qui visent plus à éclaircir la compréhension. Pour ce faire, nous allons nous baser principalement sur l'article de Bruno Chaudet[3], précisément sur les pages allant de 135 à 206.

## 3.1 C'est quoi une PFC ?

Une plateforme collaborative (PFC) est un système informatique qui met à disposition de ses utilisateurs des ressources et des outils pour faciliter le travail collaboratif. La notion de travail collaboratif doit se comprendre comme une forme d'organisation du travail, où des individus concourent ensemble à la réalisation d'objectifs communs, en dehors de toute forme de hiérarchie. La PFC est aussi une sorte de logiciel qui réunit des outils de communications et des outils de collaborations au sein d'un même espace de travail virtuel. Le sens de « virtuel » ici est lié au fait qu'il n'est pas nécessaire que ses utilisateurs se voient physiquement mais peuvent communiquer et collaborer en utilisant les moyens que la plateforme met à leur disposition.

Pour Bruno Chaudet[3], «les plateformes collaboratives sont des dispositifs de coordination de l'action qui, avec l'émergence d'Internet, peuvent en intégrer les attributs. Internet peut en effet être redéfini à travers le préfixe «hyper», c'est-à-dire comme capacité à faire perpétuellement émerger du nouveau (hypermarché, hypermédia, hypertexte et hyperorganisation).». Il ajoute que «les plateformes sont donc des artefacts conçus en vue d'améliorer la collaboration entre les membres d'une même communauté de travaux rassemblés autour d'un but commun.». Nous signalons que notre mémoire est plus orienté vers le côté hyperorganisation des PFCs.

## 3.2 Pourquoi opter pour une PFC ?

Dans de nombreuses entreprises on observe une complexité des processus métier, une multiplication des objectifs, une diversité des outils utilisés par les équipes. De plus, l'emploi de temps de chaque membre du personnel est de plus en plus flexible, notamment dans le secteur informatique où l'on peut commencer ou terminer sa journée à l'heure qu'on veut, l'important étant de bien faire sa tâche et de la soumettre dans les délais fixés.

Il y a aussi dans le cadre de la gestion de projets, une nécessité de faire travailler, de plus en plus, plusieurs personnes sur le même projet ; tout en assurant un suivi harmonieux, communicationnel et collaboratif. Dans ce contexte nait un besoin de simplifier le partage des documents, de l'information, de la connaissance.

L'ensemble des situations que nous venons de citer permet de retrouver une certaine agilité dans le travail mais peut aussi être une source de perte d'efficacité (trop de temps passé en réunion, une trop grande charge de travail, retard sur les projets, dépassement de budgets, etc.) et surtout une perte de cohérence globale au niveau du pilotage (d'un projet, d'une entreprise). La mise en place d'une PFC est une solution à ce problème de cohérence dans la mesure où elle donne à chaque utilisateur la possibilité de travailler dans des conditions idéales en centralisant toutes les informations dont il a besoin en un seul endroit et en simplifiant ses tâches quotidiennes : mises à jour, reporting, l'échange d'information par email, la documentation, la vidéoconférence, etc. Une PFC peut être utile pour le suivi de planning, mais surtout pour faciliter la communication entre les collaborateurs. Elle ajoute au travail collaboratif les dimensions de coproduction, de cocréation et d'innovation. L'image suivante présente une synthèse des atouts d'une PFC.


FIGURE 3.1 – les objectifs des PFCs

## 3.3 Les caractéristiques ou fonctionnalités d'une PFC

Les plateformes collaboratives les plus complètes ont généralement les fonctionnalités suivantes :

- La conduite de projet.
- La gestion des connaissances (méthode, marché d'informations, etc).
- La production de contenus (Wikipédia).
- L'amélioration continue.
- Le développement de l'innovation.
- La gestion de portefeuille de projets.
- Un service de dématérialisation de processus (documentation, qualité, risque, conformité, audit, ...).
- Un service de messagerie (par exemple la messagerie rapide peer to peer).
- Un système de partage de ressources et des fichiers (client/serveur).
- Des outils tels que des forums, chat multi-utilisateurs.
- Un système d'archive collective, et de pages personnelles.
- Des outils de conférence (audio, vidéo, téléphonique).

- Un mur virtuel collaboratif.
- Un annuaire de profil utilisateurs.
- Un calendrier.
- Un outil assurant la gestion des tâches.
- Des blogs, par projet ou par thématique.
- Un système de votes ou de notations (rating) sur les articles.

Parmi ces fonctionnalités nous allons développer brièvement la gestion des connaissances, la production de contenus, l'annuaire de profils utilisateurs et le service de messagerie car se sont-elles qui nous intéressent particulièrement dans le cadre de ce mémoire.

### 3.3.1 La gestion des connaissances

Les PFCs de gestion de connaissances ont principalement les fonctions de capitalisation et de valorisation des connaissances. Pour ce qui est de la capitalisation, une PFC sert à créer, capturer, formaliser, stocker, c'est-à-dire met en place une politique de traçabilité des connaissances. La valorisation quant à elle veut que la plateforme permet d'organiser, de classer, transmettre, diffuser, distribuer, retrouver, partager, enrichir les connaissances. «Dans cette perspective, il s'agit de transformer de la connaissance implicite en une connaissance explicite, c'est-à-dire une connaissance investie dans une situation sous la forme d'une compétence.»[3]. La notion de connaissance sera abordé plus en détail dans le chapitre «Présentation générale d'OWL et des BDRs».

### 3.3.2 La production de contenus

La production de contenus donne la possibilité aux utilisateurs de travailler sur le même espace de production et de partage de fichiers. Dans le contexte de la rédaction d'un article ou d'un message, il est possible que plusieurs personnes puissent modifier le contenu du message. La production et la diffusion du message se fait alors de manière collaborative.

### 3.3.3 Un service de messagerie

Le service de messagerie permet aux utilisateurs de communiquer via une messagerie instantanée comme un «chat». Le système peut intégrer aussi des envois de mails via un serveur de messagerie interne ou externe. Le service de messagerie peut avoir pour rôle de limiter les pertes de temps dues aux déplacements d'un bureau à l'autre. Toutefois, il est parfois préférable de se déplacer ou téléphoner que d'écrire un long message qui pourrait prendre bien plus de temps.

## 3.4 Les Groupes de PFCs

Il existe plusieurs groupes de PFCs. Certains proposent des outils standards qui peuvent s'adapter à de différentes situations (extensibles), donnant la possibilité de les personnaliser en fonction de nos objectifs ou de nos besoins. Nous pouvons citer les groupes suivants :

- Les CMS : Le CMS (Content Management System) ou système de gestion de contenu en français est une famille de logiciels destinés à la conception et à la mise à jour dynamique des sites web ou des applications multimédia. Les CMS les plus connus sont Drupal, Joomla, WordPress, Magento, PrestaShop, MediaWiki, etc.
- Les ERPs : Un ERP (Entreprise Ressource Planning) ou Progiciel de Gestion Intégré en français, est un progiciel qui permet de gérer l'ensemble des processus opérationnels d'une entreprise en intégrant plusieurs fonctions de gestion telles que la gestion commercial, la gestion comptable, la gestion de la trésorerie, la gestion immobilière, la gestion de la facturation, la gestion du personnel, etc. Nous pouvons citer comme exemples : SAP, SAGE SAARI, WinBooks.

- Les Clouds : Le mot « cloud » signifie littéralement « nuage » en français. Dans le milieu informatique le mot est lié régulièrement à la notion du « Cloud computing » qui signifie d'après le NIST (National Institute of Standards and Technology), « Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that an be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.». Comme exemple de cloud nous pouvons citer Dropbox, Google Drive, OwnCloud, Mega, SharePoint, etc. Les principaux acteurs du domaine sont Amazon, Citrix, Google, Microsoft.
- Les outils pour le développement et la gestion de projets : Dans ce groupe de plateformes collaboratives, on trouve les outils pour le partage des sources d'un projet informatique, des outils pour la gestion des tâches et du temps de travail, des outils pour la gestion des versions d'un projet. On peut avoir comme exemple Git, SVN, Jira, Toggl.

## 3.5 Les PFCs et les réseaux sociaux

Il est important de ne pas confondre les plateformes collaboratives aux réseaux sociaux. Les raisons sont multiples. Par exemple, «là où les plateformes collaboratives mettent l'accent sur les situations, les réseaux sociaux mettent plus volontiers l'accent sur l'individu.» [3]. Pour être plus claire, concernant les réseaux sociaux «nous ne sommes pas dans une logique de résolution des problèmes en situation mais dans le partage d'idées, dans la présentation de soi, dans la construction d'une identité plus que dans la construction d'une situation.» [3]. Nous voyons bien que la problématique des réseaux sociaux et celles des plateformes collaboratives ne sont pas les mêmes. Mais alors, comment expliquer que certaines entreprises disposant d'une plateforme collaborative s'intéressent fortement aux réseaux sociaux ? Dans [3] on y trouve quelques pistes de réponses qui sont :

- Beaucoup d'innovations émergent dans la sphère publique ou privée, plus que dans la sphère professionnelle. Les entreprises se trouvent obliger de capter ces innovations publiques ou privées et de les intégrer dans le contexte professionnel dans le but de créer de la valeur ajoutée.
- « Il y a aussi la captation des digitales natives. Si les entreprises veulent séduire les nouveaux recrutés, il faut qu'elles disposent des outils les plus avancés en matière de nouvelles technologies. ».

Les nouvelles technologies sont de plus en plus présentes dans notre société, les réseaux sociaux en font partie et sont utilisés par le personnel des entreprises d'abord sur leurs appareils privés comme les smartphones, tablettes, puis en entreprise. Nous constatons qu'il y a aussi de la collaboration sur les réseaux sociaux. Mais est-ce une collaboration au sens professionnel ? Pas nécessairement, mais il serait bien pour chaque entreprise d'exploiter cette forme de collaboration comme un vecteur d'accroissement de sa valeur ajoutée. Certaines entreprises utilisent les réseaux sociaux plus comme un moyen de mise en visibilité qu'un moyen de mise en relation ou de collaboration.

## 3.6 Recommandations pour une PFC

Les littératures que nous avons consultées nous citent les étapes suivantes qui aident à concevoir ou à choisir une PFC.

- **Définir les besoins à satisfaire et les objectifs à atteindre :** cette première phase va contribuer à identifier clairement les besoins de l'entreprise pour formaliser les résultats attendus et les indicateurs pour mesurer lesdits résultats.
- **Préciser quel sera le périmètre de la plateforme :** servira-t-elle à la gestion de contenus ? de projet ? à mettre en place une base de connaissances ? à favoriser l'innovation collaborative ? sera-t-elle accessible à l'extérieur de l'entreprise ? la définition du périmètre contribue à se focaliser sur les objectifs.
- **Anticiper sur la conduite des changements :** cela permet de définir les avantages et gains de temps apportés aux utilisateurs dans le but de s'assurer de leurs adhésions et de leurs implications futures.

- **Identifier les animateurs et les sponsors :** il est important de choisir les personnes qui vont servir de référents et développer l'envie et la motivation nécessaires à la réussite de la plateforme.
- **Enfin, s'assurer que l'utilisation de la plateforme corresponde bien à la culture de l'entreprise :** il faut veiller à ce que la culture de l'entreprise soit en droite ligne avec l'ouverture et l'agilité.

## 3.7 Avantages et limites des PFCs

D'après tout ce qui précède, il ressort que les PFCs ont 4 principaux avantages : remettre l'énergie des équipes au service de la création de valeur, améliorer la communication autour des projets, favoriser la réutilisation des connaissances, libérer les idées et favoriser le développement de l'innovation. Ces avantages font face à certaines limites qui ne nuisent pas systématiquement à la popularité des PFCs.

Comme première limite, on peut noter les contraintes liées au modèle collaboratif [4]. En fait la plupart de temps les gens ne collaborent pas de façon naturelle. Cela est dû au poids des habitudes et des méthodes transmissives de leurs passés. Il faut donc trouver une stratégie pour créer les conditions pour qu'il y ait une réelle activité collective entre les utilisateurs de la PFC afin de les inciter à interagir. Leur donner des outils techniques de communication est insuffisant pour les amener à interagir entre eux. On note aussi que la collaboration fonctionne mieux avec de petits effectifs. Une équipe composée de trois co-équipiers semble être la bonne taille pour les projets collaboratifs, où le troisième équipier est amené à jouer le rôle d'arbitre quand un désaccord se produit entre les deux premiers. Une équipe collaborative à quatre ou cinq équipiers peut également fonctionner à condition que personne ne cache son inaction derrière les autres. Les contraintes liées à la taille des groupes se combinent à celles relatives au rythme de travail demandé. En fait, certains de ces groupes arrivent à former des collectifs intelligents d'apprentissage qui présentent un fonctionnement en mode autogéré reposant sur la présence de leaders de groupe.

La deuxième limite est liée aux procédures d'évaluation [4]. La multiplicité des éléments à prendre en compte dans le cadre d'une évaluation d'une PFC complexifie cette évaluation. Ici il faudrait évaluer non seulement les capacités d'assimilation et de synthèse mais aussi la propension à participer au travail d'équipe, à entrainer les autres et à mettre à leurs dispositions les connaissances reçues. Mais alors comment évaluer par exemple l'implication dans une activité collaborative ? Comment évaluer avec exactitude l'apport du travail collaboratif dans le chiffre d'affaire de l'entreprise ? Ces quelques questions restent sans réponses formelles et indiscutables.

# Chapitre 4

# Solutions actuelles

Pour ce mémoire nous nous sommes demandé s'il existait déjà des plateformes (ou applications) servant à faire de la publication de manière collaborative sur les réseaux sociaux. Après nos différentes lectures et recherches, nous avons constaté qu'il existe plusieurs plateformes. Nous pouvons citer les CMS, eXo Plateforme, Sprout Social, HootSuite, Agorapulse, Khoros, Sprinklr, Later, Sked Social, Airtable, Tweetdeck, Penable, Post Planner, HumHub.

Notre intérêt pour ces plateformes est de savoir si au moins une pourrait couvrir l'ensemble de nos besoins. Pour cela, lorsqu'une de ces applications mentionne une utilisation de réseaux sociaux, nous faisons particulièrement attention à ces fonctionnalités. On peut facilement se rendre compte si quelques-unes des spécifications relevées dans notre description des besoins ont étés respectées et/ou implémentées. Après une première sélection, nous analysons les applications les plus intéressantes en termes de fonctionnalités, d'extensibilités et de flexibilité.

Certaines de ces applications notamment Later, Sked Social, permettent de publier du contenu uniquement sur Instagram, tandis que Tweetdeck le fait seulement sur Twitter.

D'autres comme Post Planner, Airtable se contentent seulement de faire une analyse statistique des messages qui sont déjà publiés sur les réseaux sociaux. Ils sont capables de faire une analyse croisée en intégrant plusieurs comptes d'un même réseau social ou de réseaux sociaux différents.

eXo Plateforme, Penable, HumHub, sont quant à elles, des plateformes collaboratives (PFC) orientées principalement vers les RS (réseau social d'entreprise). Sprout Social, HootSuite, Agorapulse, Khoros, Sprinklr et les CMS sont des PFC qui se rapprochent de nos besoins. Après une étude de ces dernières applications, ne pouvant pas toutes les présentées, nous avons choisi celles qui couvrent le mieux nos aspirations. Dans la suite de ce travail nous présenterons Sprout Social, Agora pulse et le CMS Joomla.

## 4.1 Sprout Social

### 4.1.1 Historique

Tout au long de cette section nous allons utiliser [5] comme référence. Sprout Social a été fondé en 2010 par Justyn Howard, Aaron Rankin, Gil Lara et Peter Soung. À la base, Justyn Howard avait besoin d'un outil pour communiquer avec les clients de l'entreprise dans laquelle il travaillait. Cette communication devait se faire sur les différents réseaux sociaux sur lesquelles l'entreprise était inscrite. Après avoir étudié le marché sans trouver un outil adapté à son besoin, Justyn Howard contacte ses collègues pour mettre sur pieds Sprout Social. C'est une plateforme collaborative qui offre la possibilité aux entreprises et aux particuliers de gérer de manière centralisée leurs présences sur plusieurs réseaux sociaux.

En fin décembre 2017, l'entreprise comptait environ 400 employés. En mars 2015, l'entreprise Sprout Social Inc intègre sur sa plateforme une autre application nommé BAMBU, qui donne aux employés un moyen simple d'amplifier

la portée de leurs marques en partageant des contenus sélectionnés, sur leurs réseaux sociaux. Quelques mois plus tard, les concepteurs acquièrent Simply Measured, une application d'analyses sociales. Cette acquisition est aussi intégrée dans Sprout Social et a pour but de fournir une analyse intelligente des informations se trouvant sur les réseaux sociaux. Cette analyse vise à atteindre les bonnes personnes, une meilleure connaissance du marketing de contenus pour créer des supports plus pertinents. Elle vise aussi à conquérir de nouvelles opportunités et fidéliser la clientèle soit à travers des moyens publicitaires, soit par une communication de plus en plus personnalisée.

De nos jours, Sprout Social est une plateforme web qui ne gère que 6 réseaux sociaux à savoir Twitter, Facebook, Instagram, LinkedIn, Pinterest, et Google Analytics. Ces réseaux sont prédéfinis et il n'y a pas la possibilité, pour l'instant, d'en ajouter d'autres. Nous pouvons y ajouter au maximaux 10 comptes. Chaque compte que vous créez sur un réseau social est appelé « profil » dans Sprout Social. La plateforme a été développée en Java, la modélisation et le stockage de données se fait avec la base de données relationnelles MySQL. Elle a 3 versions. La version «Standard» coûte 99 dollars par mois pour 5 profils, la «Professional» 149 dollars par mois pour 10 profils et la «Advanced» 249 dollars par mois pour 10 profils. Pour chaque version nous avons une période de 30 jours d'essai. Nous avons essayé la version «Advanced» dans le cadre de ce mémoire.

### 4.1.2 Description des fonctionnalités

L'interface principale de Sprout Social se présente ainsi.

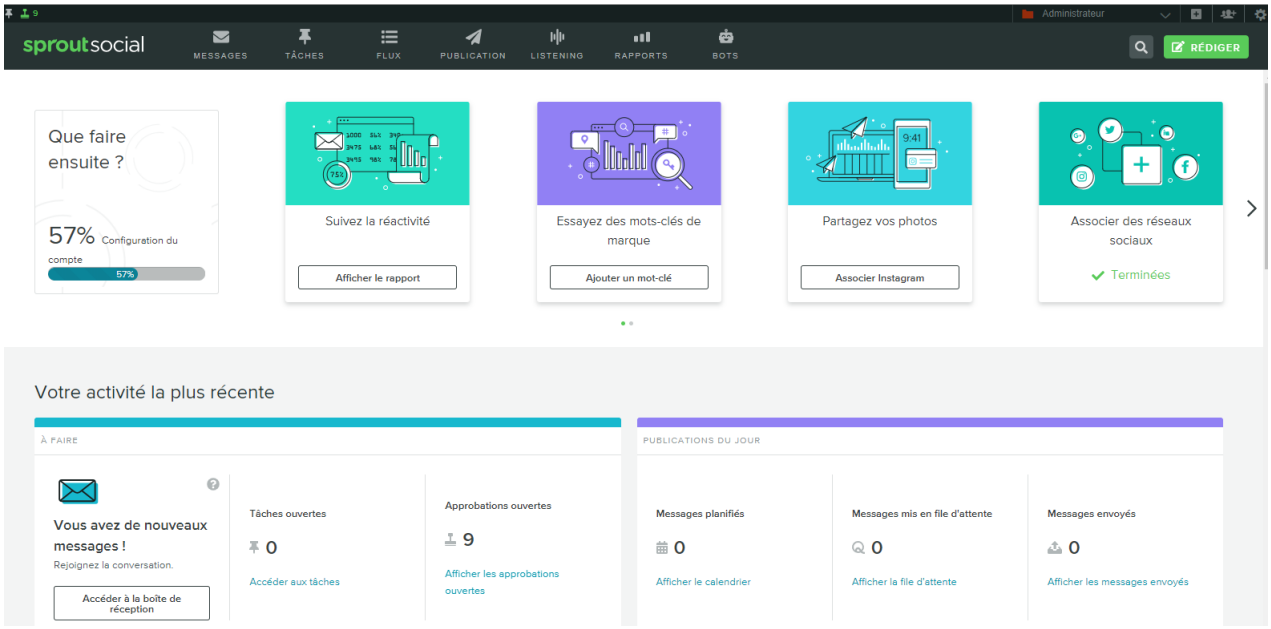



FIGURE 4.1 – Sprout Social, interface principale.

Sur l'image précédente, nous voyons en haut de l'image, la fonctionnalité « MESSAGES ». Cette fonctionnalité permet de voir tous les contenus qu'on a déjà publiés. Pour cela, on doit cliquer sur le sous-menu «Smart Inbox». Sprout Social utilise le vocabulaire «message» pour désigner un contenu. On a la possibilité de trier les messages par réseau social ou par d'autres critères que nous avons nous-même ajoutés. Les messages enregistrés pour une publication future sont aussi visibles.

La fonctionnalité «TÂCHES» montre toutes les tâches qui ont été attribuées. Normalement nous ne voyons que les tâches qui nous sont attribuées. Seul un administrateur de la plateforme voit toutes les tâches. Il peut décider qu'une personne peut voir les tâches d'autres personnes. Les tâches sont principalement les rédactions de messages. Chaque tâche est liée à un ou plusieurs réseaux sociaux et nous ne pouvons supprimer ou ajouter un réseau social à une tâche que si nous avons ces droits.

La fonctionnalité «FLUX» montre le flux d'actualités sur chaque profil. Nous pouvons décider de ne voir que des actualités d'un profil bien précis, et pour certains amis (amis sur les réseaux sociaux) bien déterminés. Avec le peu d'expérience que nous avons eu en manipulant Sprout Social, nous pensons que cette fonctionnalité est très intéressante. Pour illustration, considérons une entreprise qui veux recruter et qui dispose d'un profil Facebook, un profil Twitter et un profil LinkedIn. En temps normal, Sprout Social va afficher dans son FLUX toutes les actualités venant des trois profils. Mais comme l'entreprise est dans une urgence de recrutement, elle peut décider d'afficher d'abord dans son FLUX les réponses à l'offre d'emploi qu'elle a publiée sur son profil LinkedIn. Bien plus, elle peut trier automatiquement ces réponses selon certains critères comme le nombre d'années d'expériences, les domaines de compétences. Ainsi, elle gagnerait en temps dans le processus de recrutement.

Après le «FLUX», nous avons la fonctionnalité «PUBLICATION». Elle propose un sous-menu « Calendrier » qui définit un calendrier de publications comme nous le voyons sur l'image suivante.

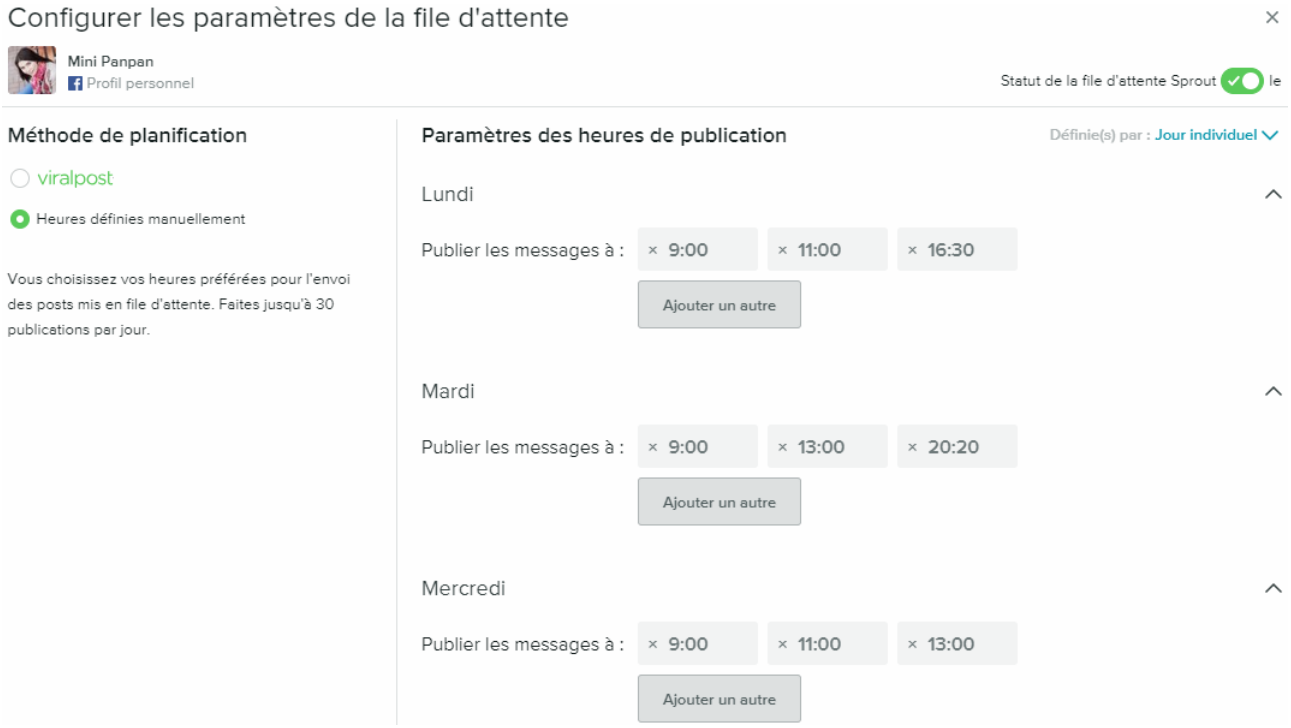



FIGURE 4.2 – Sprout Social, exemple de calendrier de publication de messages.

Pour chaque profil, Sprout Social va publier automatiquement nos messages, aux heures que nous avons enregistrées dans le calendrier. Pour le cas présent, sur le profil Facebook ayant pour nom «Mini Panpan», tous les lundis, il y aura une publication automatique à 9h, 11h et 16h30. Les mardis, cela sera à 9h, 13h, 20h20 et les Mercredi à 9h, 11h et 13h. Les heures ont été définies manuellement. Si nous souhaitons faire un calendrier «intelligent» basé sur la manière dont les informations se passent sur un profil précis, il faudrait cocher l'option «viralpost». Sprout Social va faire une analyse automatique, croisée des messages du profil concerné et nous proposer un calendrier de publication pour chaque jour de la semaine. Nous apprécions également cette fonctionnalité malgré qu'elle ne soit pas définie dans notre description de besoins.

La fonctionnalité «PUBLICATION» propose aussi un sous-menu qui nous suggère des contenus. Cette suggestion est basée sur l'ensemble des actualités sur Internet. Nous pouvons faire nos propres choix par pays, par thème (sport, politique, culture, cinéma, etc), par période (4 heures, 1 jour, 2 semaines, 1 mois, etc). Ces suggestions nous servent comme base pour rédiger nos propres messages et les publier sur nos différents profils. Il y a aussi une file d'attente qui représente l'ensemble des messages qui devront être publiés automatiquement sur chaque profil. Par défaut la file est classée par heure de publication la plus imminente. Un autre sous-menu «Brouillons» recense tous les messages qui sont considérés comme des idées de prochaines rédactions. Il y a aussi la possibilité de voir tous les messages qui

doivent être encore approuvés ainsi que les différents niveaux d'approbations. La notion de brouillon et de niveau d'approbation sont pertinentes pour nous dans la mesure où elles correspondent à l'un de nos besoins.

En ce qui concerne la fonctionnalité «REDIGER», en haut à l'extrême droit, elle permet à un utilisateur de rédiger un message. Lors de cette rédaction il doit indiquer les profils sur lesquels il veut publier, le workflow de validation dudit message. Lors de notre recherche pratique en utilisant l'application, nous avons pu créer un workflow qui valide un message en passant successivement par les étapes REDIGE, ENREGISTRE, VALIDE, et PUBLIE. Le workflow de rédaction de messages est explicité dans notre description des besoins et il est respecté dans le cadre de Sprout Social.

Dans Sprout Social, il est possible de créer des groupes d'utilisateurs et d'affecter des rôles à chaque utilisateur ou à un groupe. Ainsi on peut attribuer des rôles Administrateur, Modérateur Censeur ou Rédacteur à un utilisateur. Ces rôles, qui sont définis dans nos besoins, sont bien implémentés dans l'application de telle sorte que seul un Administrateur ou un Censeur peuvent publier un message.

L'inquiétude que nous avons eue en testant Sprout Social se trouve principalement au niveau de la vie privée. Nous avons créé un compte de test sur Facebook, Twitter et LinkedIn. En fait, l'application a accès à tous nos comptes sociaux sans que nous ne sachions exactement quels sont tous les traitements qui sont faits de nos données à caractère privées comme : nom et prénom, login et mot passe, numéro de téléphone, photos, vidéo, adresse IP. Lors d'une inscription à l'application on doit accepter les conditions qu'on nous propose et c'est cela qui donne l'autorisation à Sprout Social de manipuler nos différents données, mais ces conditions ne pas toujours claires.

## 4.2 Agorapulse

### 4.2.1 Historique

Agorapulse est créée en 2011 par Emeric Ernoult et Ben Hediard à New York [1]. Elle est une plateforme web très similaire à Sprout Social. La première version (la 1.0) comportait juste Facebook mais en 2014 il y a eu l'ajout de Twitter, Instagram, LinkedIn. Au courant de l'année 2017 YouTube et Google plus ont été intégrés. Pour chaque réseau social on peut créer plusieurs comptes. Comme dans Sprout Social, un compte est appelé «profil» mais un contenu est appelé «contenu».

Actuellement, la plateforme peut supporter jusqu'à 40 profils sociaux et compte plus de 4500 clients à travers le monde[6]. Elle est développée en utilisant les langages de programmations JAVA et Groovy (Spring, Grails, Gradle, Spock ...) pour le back-end et le Framework AngularJS pour le front-end. Au départ, Agorapulse utilisait une base de données AWS RDS/MySql (AWS RDS signifie Amazon Web Service Relation Database Service). Depuis quelques temps elle a migré vers une AWS DynamoDB (une base de données NoSQL).

Agorapulse a actuellement 4 versions. La version «Samall» peut accepter seulement trois profils, un utilisateur et est gratuite. La version «Meduin» peut supporter 10 profils, trois utilisateurs et coûte 99 euros par mois. Les versions «Large» coûte 199 euros par mois, accepte 25 profils et 6 utilisateurs. Enfin, la version «Entreprise» coûte 299 euros par mois, gère 40 profils et 12 utilisateurs. Pour chaque version, si on veut ajouter un profil ou un utilisateur, il faudrait payer en plus.

### 4.2.2 Description des fonctionnalités

L'image ci-dessous expose l'interface principale d'Agorapulse.

Comme nous l'avons dit plus haut, Agorapluse a presque les mêmes fonctionnalités que Sprout Social. Pour cela, nous n'allons pas développer toutes les fonctionnalités vu que nous l'avons déjà fait avant. Nous signalons qu'Agorapulse n'a pas la suggestion de contenu que propose Sprout Social. À l'inverse, Agorapulse propose d'importer des contenus depuis un fichier Excel, un site web ou une vidéo, ce que ne fait pas Sprout Social. Il y a aussi une différence

1. Plus d'information sur : https ://www.agorapulse.com/fr/notre-histoire

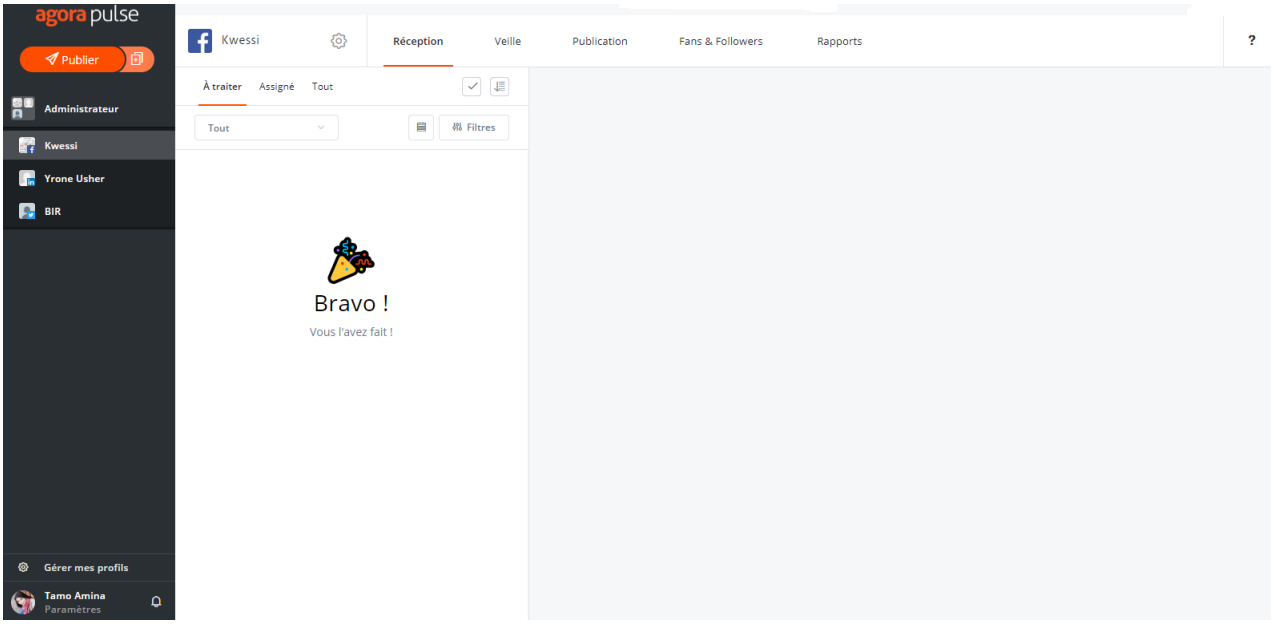



FIGURE 4.3 – Agorapulse, interface principale.

sur la manière dont chacun gère le calendrier de publication de contenus.

Dans la version d'essai que nous avons utilisée, nous avons ajouté une page Facebook nommée «KWeSSi», un profil LinkedIn nommé «Yrone Usher» et un profil Twitter nommé «BIR». Sur l'image ci-dessus on voit, sur le coin inférieur à gauche, que l'utilisateur connecté s'appelle «Tamo Amina». Tout en haut, on remarque qu'il a un rôle administrateur et qu'il gère les trois profils disponibles (KWeSSi, Yrone Usher, BIR). Au centre de l'image on constate qu'il n'a aucune tâche à traiter. Le point d'interrogation en haut, à l'extrême droite, est réservé pour l'aide. La fonctionnalité liée au calendrier de publication de contenus se présente ainsi.

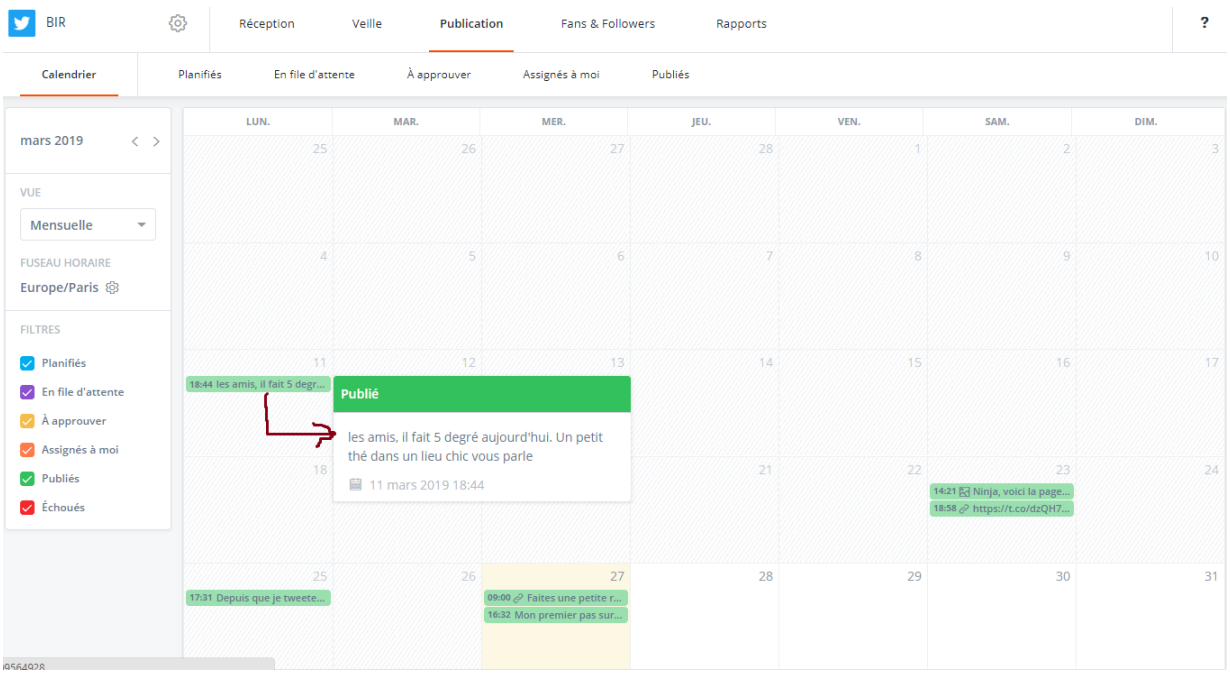



FIGURE 4.4 – Agorapulse, exemple de calendrier de publication de contenus.

Dans ce calendrier du mois de mars 2019, on voit les contenus qui ont été publiés (en verts) sur le profil Twitter (BIR). Si on met le curseur de la souris sur un contenu, il s'affiche avec plus de détails. À gauche de l'image se trouve d'autres informations avec des codes couleurs. Normalement, les contenus planifiés s'affichent en bleu, ceux de la file d'attente en violet, ceux approuvés en jaune, ceux assignés à l'utilisateur connecté en orange, ceux publiés en vert et ceux qui ont subi une erreur en rouge. Avec tous ces détails nous trouvons que le calendrier de publication dans Agorapulse est nettement mieux structuré que celui de Sprout Social.

## 4.3 Le CMS Joomla

Un CMS (Content Management System) est une plateforme collaborative permettant de faire principalement la gestion de contenus. Ils permettent de réaliser des sites internet statiques ou dynamiques, de façon rapide. Généralement, les fonctionnalités principales d'un CMS sont :

- La gestion de contenu : elle permet par exemple, à plusieurs utilisateurs de travailler sur le même contenu. Elle permet aussi de structurer et publier du contenu (texte, photo, vidéo, fichier, etc) en local ou à l'extérieur du site.
- La gestion des utilisateurs : elle se fait à travers la hiérarchisation des utilisateurs, les rôles et les permissions.
- La séparation des opérations de gestion de la forme et du contenu.
- Le ciblage promotionnel : permet de faire de la publicité.
- La gestion des extensions : elle se fait via des composants tels que des modules, des plugins.

Joomla a vu le jour le 17 août 2005. Elle est une application développée avec le langage de programmation PHP et utilise une base de données MySQL. Il est gratuit et dispose d'une forte communauté de développeurs et d'utilisateurs. Certaines de ses extensions sont gratuites et d'autres payantes. Le but recherché ici est de savoir si Joomla peut répondre à la description des besoins que nous avons faite au début de ce travail. Par exemple nous allons chercher à savoir si elle peut faire la customisation des messages et les publiés sur les réseaux sociaux comme Facebook, Twitter, LinkedIn, etc. Il est aussi question de savoir si on peut ajouter aisément un nouveau champ ou un nouveau réseau social sur Joomla. D'après nos lectures et nos séances pratiques, Joomla permet globalement de faire :

- La gestion des rôles et des permissions utilisateurs.
- La gestion du contenu : il s'agit de la rédaction, l'enregistrement, la correction, la publication d'un message (texte, photo, vidéo, fichier, etc). Les messages peuvent être publiés sur le site internet qu'on est en train de développer ou sur les réseaux sociaux.
- La gestion de la messagerie : il s'agit d'un système d'envoi de mail soit à un groupe d'utilisateurs soit à un utilisateur spécifique. Le mail peut contenir un lien permettant d'interagir directement avec l'application.
- Le ciblage promotionnel : Joomla prévoit dans sa fonction de base une possibilité de faire de la publicité. Avec WordPress vous devez installer un plugin permettant de faire cette tâche.
- La gestion des statiques : elle vous permet de voir par exemple le contenu le plus ou le moins consulter.
- Le e-Commerce : avec Joomla, vous devez installer une extension, par exemple VirtueMark. C'est une extension adaptée aux grandes entreprises. Pour des entreprises de tailles moyennes ou petites vous pouvez utiliser l'extension HikaShop. Avec WordPress, pour faire du e-Commerce vous devez installer un plugin comme WooCommerce, WP eCommerce, ou Jigoshop.
- Une gestion personnalisable du design via les Templates.
- Selon le Template utilisé, il est possible de faire un ajout ou une suppression de champs sur un article, sur un réseau social, etc.
- La gestion du multilinguisme. On y trouve du français, l'anglais, le néerlandais, l'allemand, etc.
- Le responsive design : s'adapte bien à plusieurs appareils (PC, tablettes, smartphone).

### 4.3.1 Interface principale

Cette interface se présente ainsi :

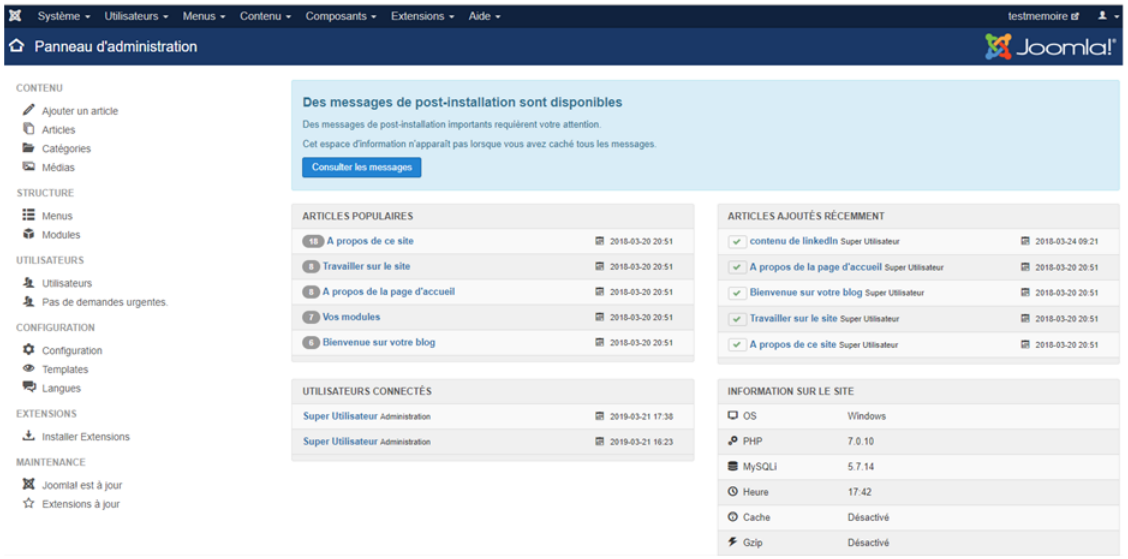



FIGURE 4.5 – Interface principale de Joomla

L'interface principale (Figure 4.5) de l'application Joomla telle qu'on le voit ci-dessus, a sept fonctionnalités qui sont : Système, Utilisateurs, Menus, Contenu, Composants, Extensions, Aide. Au milieu de l'image se trouvent un ensemble d'informations sur les articles (ou contenus), les utilisateurs de l'application, et les paramètres systèmes (OS, version de PHP, type de SGBD (MySQL), etc). Le développement des 7 fonctionnalités se présente ainsi :

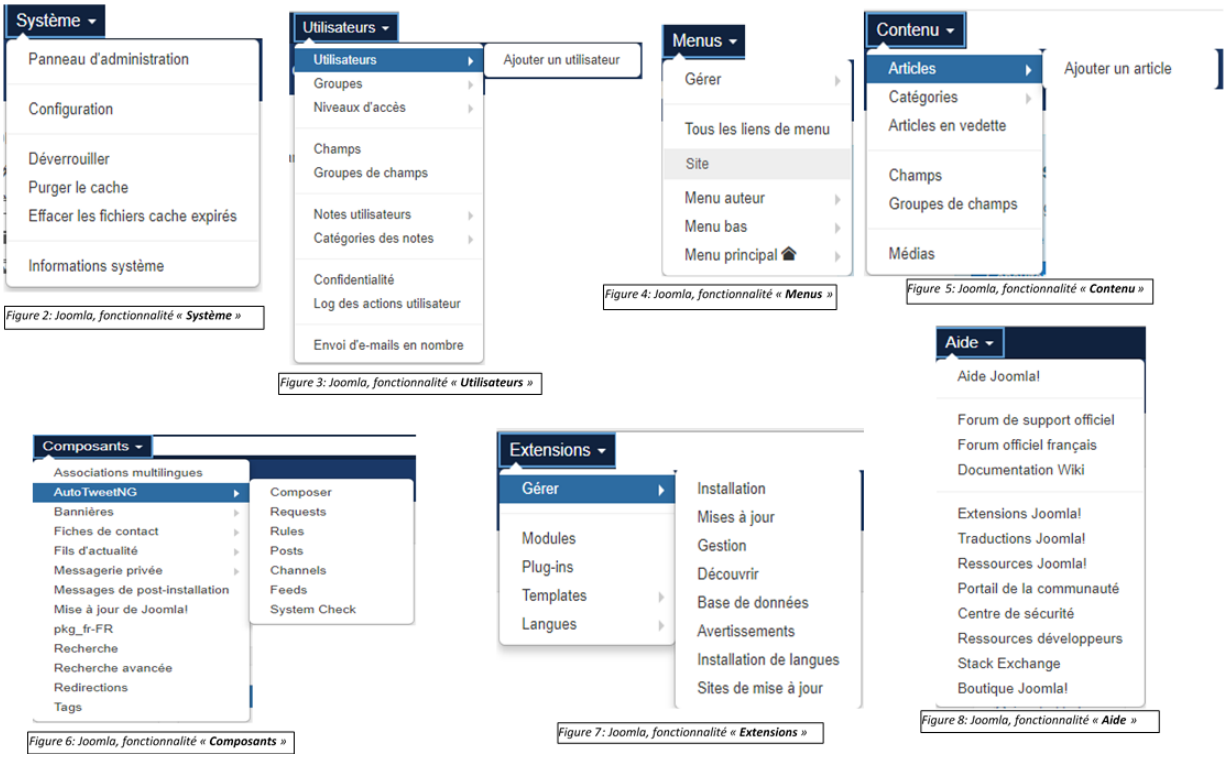



FIGURE 4.6 – Détail des fonctionnalités de Joomla

### 4.3.2 Description des fonctionnalités

La fonctionnalité «Système» (Figure 2) permet de voir toutes les informations statistiques sur le système. Pour cela, il faut aller dans le sous-menu «Panneau d'administration». Il est aussi possible de faire des configurations en précisant, par exemple, la méthode d'envoi des mails (PHP Mail, Sendmail, ou SMTP), la méthode pour exécuter les requêtes en PHP (MySQLi ou PDO), les paramètres pour le transfert FTP (File Transfert Protocol), l'éditeur pour les articles (TinyMCE, WYSIWYG ou CodeMirror), etc. Sur cette figure les autres fonctionnalités servent à gérer le

cache de l'application.

La gestion des utilisateurs est faite via la Figure 3. Il est possible d'ajouter des utilisateurs en utilisant le sous-menu «Ajouter un utilisateur». Un utilisateur peut être affecté à un groupe, on peut aussi lui donner des rôles et des droits via le sous-menu «Niveaux d'accès». Les actions des utilisateurs sont logées dans un fichier qu'on peut consulter en se servant de «Log des actions utilisateur». Si on le souhaite, on peut envoyer des e-mails à un ou plusieurs utilisateurs en utilisant «Envoi d'e-mails en nombre». Bien plus, dès qu'un nouvel utilisateur est ajouté nous pouvons décider de lui envoyer automatiquement un e-mail pour l'inviter à confirmer son inscription à la plateforme. En comparant avec ce qui se trouve dans notre description des besoins, cette fonctionnalité correspond bien.

Joomla est d'abord un site internet paramétrable. Pour cela on a la possibilité d'ajouter, modifier ou supprimer des menus (Figure 4). On peut y ajouter des menus verticaux ou horizontaux. Les menus sont des éléments qui nous permettent d'exposer les fonctionnalités que nous avons développées. Dans un menu principal, nous avons la possibilité de créer autant de sous-menu que nous voulons.

La fonctionnalité « Contenu » (Figure 5) sert à faire la gestion de contenus. Les contenus sont appelés «Article» dans Joomla. Chaque article ajouté doit être obligatoirement dans une catégorie que nous avons nous-même définie. Lors de la rédaction d'un article on peut y ajouter un titre, une description, une vidéo, un audio, une photo, un fichier et un lien de référencement. Dans Joomla nous pouvons attribuer un rôle «Rédacteur» à ceux qui rédigent des articles, un rôle «Modérateur» à des utilisateurs qui vont valider les articles rédigés avant leurs publications. On peut aussi avoir un rôle «Administrateur» ou «Super Utilisateur» pour coordonner toutes les actions de la plateforme. Les trois rôles que nous venons de citer sont dans nos besoins et sont rencontrés par Joomla. Par contre le rôle Censeur, décrit aussi dans nos besoins n'est pas prévu par Joomla. En clair, il est impossible de définir un utilisateur comme censeur d'un réseau social. Dans notre cas, le workflow de rédaction d'un article se trouve handicaper par l'absence de cette censure. La customisation d'articles (une spécification se trouvant dans nos besoins) est possible dans Joomla. Pour le faire, nous devons créer un nouvel article et y copier manuellement le message à customiser, puis le publier sur le réseau social de notre choix. Cette technique est très éloignée de ce qui nous est demandé dans la description des besoins. Le sous-menu « Medias » sert à gérer les images, audios et vidéos liés à un article, à une extension ou à un profil utilisateur.

La fonctionnalité «Composants» (Figure 6) permet d'étendre la plateforme Joomla en y installant un nouveau composant, module, plugin, ou template. Ces éléments seront développés dans le chapitre suivant de cette partie. Sur la figure, le composant AutoTweetNG nous intéresse plus particulièrement. L'installation de ce composant permet d'étendre le Framework de base et de permettre qu'on puisse faire des publications de contenus de manière collaborative sur les réseaux sociaux. Ce composant prévoit 22 réseaux sociaux sur lesquels nous pouvons publier. Pour un réseau social, il admet l'utilisation de plusieurs comptes. La version gratuite ne contient que trois réseaux actifs, à savoir Facebook, Twitter, LinkedIn. Si on utilisait le CMS WordPress, la publication sur les réseaux sociaux se fait aussi en installant un plugin, tel que : NextScripts : Social Networks Auto-Poster (SNAP). Ce plugin est payant, il permet de publier des messages sur plusieurs réseaux sociaux notamment Facebook, Twitter, LinkedIn et bien d'autres. Avec WordPress, SNAP n'est pas le seul plugin, il existe aussi CoSchedule, Jetpack, AccessPress Social AutoPost, HYPESocial Buffer, MailChimp Social... Les différences entre eux sont souvent au niveau du nombre de réseaux sociaux et du fait qu'ils sont payants ou pas.

Le composant AutoTweetNG rencontre le besoin de publication sur les réseaux sociaux tel que décrit dans nos besoins. Mais ce composant ne prévoit ni l'ajout de nouveaux réseaux sociaux, ni l'ajout d'un champ à un réseau social. Cela est regrettable car ce sont des besoins importants pour nous. Néanmoins, il est possible de contourner ce manquement en customisant le code source du composant ou en développant notre propre composant qui répond à nos exigences. Quand nous installons une extension ou singulièrement un composant, Joomla ajoute dans le répertoire «media» un dossier avec le nom de ce composant. Dans le cas d'AutoTweetNG, il créera le dossier «com_autotweet». Si c'est un module il créera un dossier dont le nom commence par « mod_» suivi du nom exact du module. Pour un plugin le nom du dossier commencera par «plg_» suivit du nom exact du plugin. Ce système de fichier est représenté

ainsi :

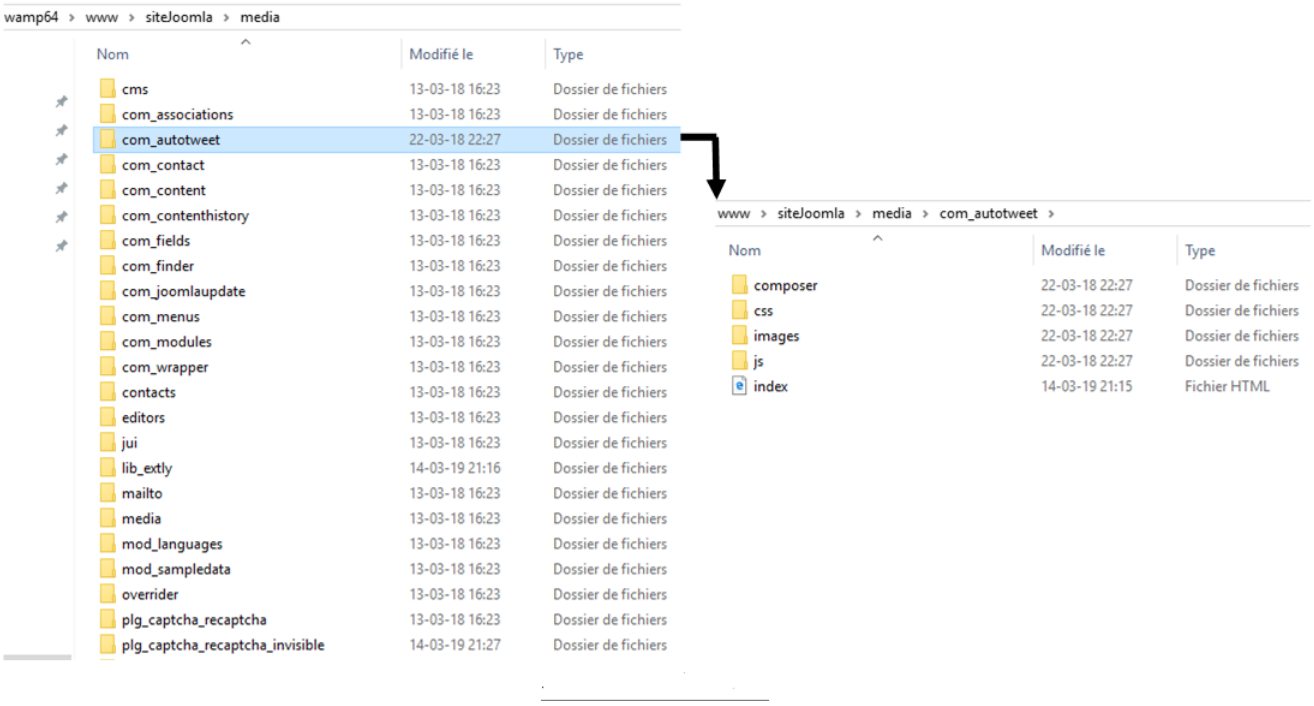



FIGURE 4.7 – Système de fichier Joomla pour les médias

C'est en utilisant la fonctionnalité «Extensions» que nous pouvons installer un composant, un module, un plugin ou un template (Figure 7). Joomla est une plateforme multilingue. On Utilise le sous-menu «Langues» pour activer de nouvelles langues. Si nous cliquons sur le sous-menu «Plug-ins» nous pouvons activer, désactiver, ou désinstaller un plugin. Cela est valable aussi pour les modules et les templates. Dans le sous-menu «Mise à jour» on a la possibilité de mettre à jour toutes nos extensions.

La Figure 8 donne la possibilité d'avoir de l'aide. Par exemple, sur le forum officiel de Joomla il y a une communauté pour répondre à nos différentes questions. Ce forum existe en plusieurs langues. La boutique Joomla expose de multitudes extensions, gratuites ou payantes, que nous pouvons télécharger et utiliser selon nos besoins.

Globalement, Joomla remplit les exigences que nous avons décrites au début de ce travail lorsque nous faisions la description des besoins. Au vu de cette description, certains besoins ne sont pas couverts ou couverts en parti. Pour apporter plus de clairvoyance, nous allons résumer les limites de Joomla dans le point suivant.

### 4.3.3 Les limites le Joomla

Les limites de Joomla sont particulièrement les suivantes :

- Il faut payer pour avoir certaines extensions.
- Pour customiser un message on doit créer un nouvel article et y copier manuellement le message à customiser, puis le publier sur le réseau social de votre choix. Cela n'est pas très pratique.
- Impossible pour un utilisateur d'ajouter un nouveau réseau social, sauf s'il peut customiser le code du plugin, ce qui demande une compétence spécifique et pointue.
- Si on customise le code du plugin en modifiant le code source, il est possible de perdre ces modifications lors de l'installation d'une mise à jour de Joomla.
- On est un peu limité par ce qui est prévu par les templates. Mais si on maîtrise PHP, JavaScript, HTML, MySQL et surtout la philosophie des CMS, il est possible de développer soi-même des extensions ou des plugins qui correspondent à nos propres besoins, mais cela peut prendre beaucoup de temps.
- L'installation de Joomla propose par défaut une base de données relationnelle contenant au moins 75 tables. Il faut du temps pour comprendre les relations entre ces tables et les données qui s'y trouvent.

## 4.4 Tableau récapitulatif des fonctionnalités par solution.

Les fonctionnalités se trouvant dans le tableau sont celles qui se trouvent dans notre description des besoins. Ce tableau donne une vue globale des fonctionnalités que supporte chaque solution.

| Solutions / Fonctionnalités | Sprout Social | Agorapulse | Joomla |
|---|---|---|---|
| **Spécifications de base** | | | |
| Inscription d'un utilisateur | Oui | Oui | Oui |
| **Traiter un utilisateur** | Oui | Oui | Oui et plus élaborée |
| Se connecter | Oui | Oui | Oui |
| **Ajouter un rôle à un utilisateur** | Oui | Oui | Oui |
| Rédaction d'un article | Oui | Oui | Oui |
| **Consultation d'un article** | Oui | Oui | Oui |
| Traiter un article | Oui | Oui | Oui et plus élaborée |
| **Customiser un article** | Oui | Oui | Oui mais pas adapté. |
| Traiter un réseau social | Pas d'ajout de réseaux | Pas d'ajout de réseaux | Oui, si on développe sa propre extension (component, module ou plugin). On ne pas définir un utilisateur comme censeur d'un réseau social. |
| **Ajouter un champ à un article** | Nom | Nom | Oui |
| Envoyer un mail | Oui mais insuffisant | Oui mais insuffisant | Oui et bien élaborée |
| **Fournir une couche de sécurité à l'application** | Quid de la vie privée | Quid de la vie privée | Oui<br>- avec un certificat TLS (Transport Layer Security)<br>- possibilité de définir des backups automatiques |
| Interactivité | 6 réseaux<br>10 profils<br>Illimité | 6 réseaux<br>40 profils<br>12 utilisateurs | 22 réseaux<br>Illimité<br>Illimité |
| **Extensibilité** | Non | Non | Oui avec les extensions |
| **Spécifications annexes** | | | |
| **Open source** | Non | Non | Oui |
| Gratuité | Non | Non | Oui, sauf certaines extensions sont payantes |
| **Utilisabilité** | Oui | Oui | Fait pour des spécialistes |
| Responsive | Oui | Oui | Oui |
| **Documentation** | Peu | Peu | Bien fournie |

FIGURE 4.8 – Fonctionnalités par solution

En comparant les trois solutions, nous constatons qu'aucune solution actuelle ne répond complètement à nos besoins. Notre critère de comparaison est le nombre de fonctionnalités non couvert par les différentes solutions. Ce sont les zones en rouges dans le tableau ci-dessus. La solution qui s'approche le mieux est le CMS Joomla avec seulement trois fonctionnalités non couvertes. Elle est la seule à couvrir presque tous les besoins. Les manquements de Joomla sont liés au fait qu'il ne permet pas de définir un utilisateur comme censeur d'un réseau social et le fait que l'ajout d'un nouveau réseau social passe non seulement par la maitrise de la philosophie des CMS, mais aussi qu'il faut savoir comment développer un plugin pour Joomla.

Joomla tout comme d'autre CMS, est une plateforme collaborative qui satisfait en grande partie nos exigences. Les trois fonctionnalités qu'elle ne couvre pas peuvent être développées en modifiant le code source de son Framework

ou en développant des plugins spécifiques à Joomla. Malgré que cette solution soit satisfaisante, ne pouvons-nous pas trouver une autre alternative à Joomla qui répond au mieux à nos besoins ? Pour répondre à cette question, nous allons étudier quelques formes d'architectures et de modélisation qui sont utilisées par les PFCs. Cela va nous permettre d'avoir une idée sur comment nous pouvons concevoir une plateforme collaborative personnalisée. À la fin de cette partie nous serons capables de dire avec précision, entre une solution existante et une solution personnalisée, en fonction des situations, laquelle est la mieux adaptée.

# Chapitre 5

# Quelques architectures ou outils pour PFC

Comme nous l'avons déjà dit plus haut, les plateformes collaboratives sont l'une des notions importantes dans notre travail, parce que nous allons faire des publications de contenus de manière collaborative sur les réseaux sociaux. Pour cela, nous cherchons à savoir quelle est l'architecture pouvant nous aider dans la conception de notre plateforme ? Cette question nous aidera à faire un état de l'art sur le sujet afin de dénicher l'architecture d'application qui pourrait être adaptée à notre situation. Dans la littérature, plusieurs architectures sont citées ou sont utilisées dans la conception des PFCs. Celles qui reviennent le plus souvent sont l'architecture à base de Plugins, le Framework Spring et l'architecture à base du web sémantique.

## 5.1 L'architecture à base de Plugins

### 5.1.1 Généralités

Le terme *plugin* provient de la métaphore de la prise électrique en ce sens qu'en branchant une prise sur un réseau électrique, la prise doit immédiatement fonctionner. Les architectures de plugins constituent une solution attrayante pour les développeurs qui cherchent à créer des applications modulaires, personnalisables et facilement extensibles. Lorsqu'un plugin est installé, il peut immédiatement fonctionner (à la manière d'une prise branchée sur un réseau électrique) et exécuter les actions qui lui sont dédiées. C'est un moyen astucieux d'autoriser des tiers à ajouter des fonctionnalités à une application sans accès au code source. Cela permet pour de nombreux développeurs, d'évoluer vers une méthodologie à part entière pour le développement d'applications.

La structure d'une application en tant que structure de base est conçue, et un ensemble de plugins pour l'étendre, offrent de nombreux avantages pour des développeurs d'applications :

- Ils peuvent implémenter et intégrer des fonctionnalités d'application très rapidement.
- Les plugins étant des modules distincts dotés d'interfaces bien définies, on peut rapidement isoler et résoudre les problèmes.
- Ils peuvent créer des versions personnalisées d'une application sans modifier le code source.
- Les tiers peuvent développer des fonctionnalités supplémentaires sans aucune intervention des développeurs d'origine.
- Les interfaces de plugin peuvent être utilisées pour envelopper du code hérité, écrit dans différentes langues.

Les utilisateurs finaux tirent également parti des applications utilisant une architecture de plugin :

- Ils peuvent personnaliser des ensembles de fonctionnalités pour des flux de travail particuliers.
- Ils peuvent désactiver les fonctionnalités indésirables, simplifiant potentiellement l'interface utilisateur de l'application, réduisant l'encombrement de la mémoire tout en améliorant les performances.

En général, les plugins sont des ensembles chargeables, des packages de code exécutable et de ressources connexes pouvant être chargés au moment de l'exécution. Cette flexibilité permet de concevoir des applications hautement modulaires, personnalisables et extensibles. Les modules d'économiseurs d'écrans, les panneaux de préférences et les palettes Interface Builder, les filtres graphiques Adobe Photoshop et les visualiseurs de musique iTunes sont tous des exemples de plugins. Nous les utilisons chaque fois que nous souhaitons ajouter plusieurs instances d'un type de module particulier fournissant une unité de fonctionnalité bien définie, telle qu'un nouveau filtre d'exportation dans un programme graphique, un nouveau style de transition dans un programme de montage vidéo ou un autre type de fonctionnalité.

Dans une application acceptant les plugins, il existe une partie hôte, dite de base. On peut penser à une application hôte comme une sorte de puzzle avec un nombre infini de places pour mettre de nouvelles pièces. Les plugins sont des pièces supplémentaires à attacher au puzzle et leurs architectures déterminent la forme des pièces de puzzle autorisées. Si un plugin présente une forme incorrecte, il ne peut pas rejoindre le reste du puzzle.

L'architecture des plugins peut s'avérer très utile pour le développement d'applications complexes, nécessitant une exigence en termes d'extensibilité. Nous l'expliquerons à travers un exemple dans la section 2) qui suit. La décision d'utiliser cette architecture doit être prise au début du projet et doit être bien murie. Cette architecture a tout de même quelques inconvénients. Elle nécessite une ouverture à d'autres développeurs dans le souci de permettre une continuité dans la création de plugin. Cette ouverture pose un problème de sécurité, de conformité et de permission. Il faut aussi tenir compte du temps à mettre pour constituer une communauté de développeurs capables de bien comprendre l'architecture de base afin de développer des fonctionnalités additionnelles (ou plugins).

### 5.1.2 Exemple de PFC utilisant des Plugins

Dans nos recherches et discussions nous avons trouvé un groupe de plateformes collaboratives qui utilise particulièrement les plugins. Il s'agit des CMS. Nous vous présentons ici l'architecture du CMS nommé Joomla qui est celle que nous avons étudiée. Vu le grand nombre de CMS (Jomla, Wordpress, Drupal, etc.) il est impossible pour nous de tous vous les présenter. À travers cet exemple, nous montrons comment une architecture plugin fonctionne avec une base de données relationnelle et faisons un rapprochement avec notre situation.

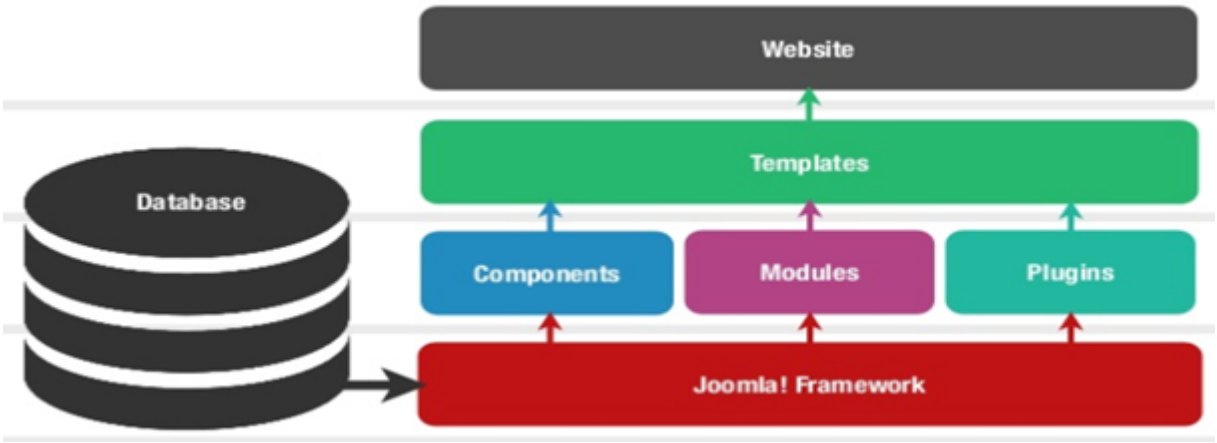



FIGURE 5.1 – Exemple d'architecture Plugins : Cas du CMS Joomla.

**Database :** tous les contenus, en dehors des images et les documents, sont stockés dans une base de données relationnelle. Ici, la base de données utilisée est MySQL.

**Joomla ! Framework :** c'est une collection d'applications open source. La couche Framework de l'architecture Joomla est développée selon le modèle MVC (Model Vue Contrôleur).

**Components, Modules, Plugins, Templates :** ce sont les éléments qui permettent d'étendre le Framework Joomla. Étant donné que Joomla est open source, nous pouvons y développer nos propres Components, Modules, Plugins ou Templates et les installer sur le Framework de base. Cette ouverture permet que le Framework de base soit adaptable et extensible. La description de chaque extension se présente comme suit :

- Components : ils peuvent être considérés comme des mini-applications conçues pour faciliter les interactions dans le Framework Joomla. La plupart des composants sont composés de deux éléments principaux : une interface d'administration (back-end) et une partie site (front-end). La partie site est utilisée pour le rendu des pages quand le composant est appelé pendant le fonctionnement normal du site. L'interface d'administration permet de configurer et de gérer différents aspects du composant. Elle est accessible depuis l'espace d'administration de Joomla. VirtueMart, AutoTweetNG sont des exemples de composant Joomla. VirtueMart sert à faire de l'e-Commerce. Il est gratuit. On peut le télécharger, l'installer et le customiser selon nos besoins.
- Modules : les modules sont des extensions, légères et flexibles, utilisées dans le rendu de page. Ils peuvent être autonomes ou être utilisés pour afficher les données d'un composant. Les modules sont gérés à partir du «Gestionnaire de modules» (qui est lui-même un composant). Les Modules sont assignés à des éléments de menu. Ainsi, nous pouvons par exemple décider de montrer ou de cacher le module de connexion selon la page (élément de menu) que l'utilisateur est entrain de consulter. Certains modules sont liés à des composants : le module « derniers articles », par exemple, lie le contenu du composant de contenu (com_content) et affichera les éléments de contenu les plus récents.
- Plugins : les Plugins proposent des fonctionnalités associées à des événements déclencheurs tels que les «trigueurs». Joomla propose nativement un ensemble d'événements pour plugins, mais toute extension peut créer ou personnaliser des événements. Lorsqu'un événement particulier intervient, toutes les fonctions du plugin associées à l'événement sont exécutées en suivant une séquence. Cette méthode est très efficace pour étendre les fonctionnalités du Framework de base. Les plugins offrent aux développeurs d'extensions la possibilité de voir d'autres extensions interagir avec leurs actions. Ainsi, les extensions peuvent fonctionner de manière étendue. Les plugins pour Joomla sont conçus en respectant l'architecture Observer. Dans le Framework Joomla, la classe «JPlugin» permet de lier les actions du plugin avec des événements natifs ou personnalisés. La classe «JEventDispatcher (dans Joomla ! 3.x) est un gestionnaire d'événements qui appelle tous les plugins enregistrés pour un événement particulier, lorsque cet événement est déclenché.
- Templates : ils déterminent le look et les «sensations» que peuvent avoir un site. Ce sont des extensions qui modifient l'apparence du site web. Il existe deux types de template : les templates de site (Front-end) et les templates d'administration (Back-end). Les templates de site modifient l'apparence de votre site pour l'utilisateur final. La plupart des templates que vous verrez seront des templates de Site. Les templates d'administration permettent de modifier l'apparence de l'interface d'administration. Ils sont plus rares. Ils concernent les fonctionnalités telles que : utilisateurs, menu, article, catégorie, module, composant, plugin et gestion des templates.

**Website :** enfin, le site web est ce qui permet aux utilisateurs et vous d'interagir.

### 5.1.3 Rapprochement avec notre travail

Tel que vous l'avez vu sur l'architecture CMS précédente, il y existe un Framework de base. Sur ce Framework on peut greffer plusieurs autres fonctionnalités en utilisant soit un composant, soit un module, soit un plugin, soit un template. Le cas des plugins nous intéresse particulièrement car après avoir fait l'étude des CMS, nous avons constaté que les plugins sont particulièrement utilisés pour étendre les fonctionnalités de ces CMS. De plus, dans notre description des besoins, l'extensibilité est une spécification très importante.

Joomla stocke ses données via une base de données MySQL. On peut étendre le CMS Joomla en y installant, par exemple, un plugin pour le planning. Un planning ayant : un ou plusieurs utilisateurs, une date, une ou plusieurs tâches. Nommons ce plugin «Task». L'installation de Task aura comme conséquence :

- L'extension dynamique du code source en ajoutant les contrôleurs, les services, les DAO (Data Access Object) lié au planning. Étant donné que l'objet «utilisateurs» et «date» existent déjà avant l'installation du plugin, il

y aura la création automatique dans le code source de l'objet « planning » avec pour attributs : ID, la liste des utilisateurs (ou participant), la liste des tâches, la date du planning. Il y aura aussi la création automatique de l'objet «tache» avec pour attributs : ID, description.

- Dans la base de données MySQL, la table «Planning» est créée automatiquement et contient les propriétés id_utilisateur, date, id_tache. La table « Tache » est aussi créée automatiquement avec pour propriété ID et description.
- Au niveau du template, un menu «planning» est aussi ajouter automatiquement pour permettre de créer un planning en y mettant des utilisateurs et des tâches.
- Le système de fichier est automatiquement complété avec un répertoire « plg_Task » tel que nous l'avons expliqué avant.

Nous venons d'expliquer comment les plugins sont utilisés dans les CMS. Cette technique peut être mise en pratique dans notre cas pour développer un plugin soit pour ajouter un nouveau champ, soit pour ajouter un nouveau réseau social, soit pour customiser un contenu et le publier sur un autre réseau social. On peut aussi imaginer une application de base où il n'y a pas l'envoi de mail. Pour un utilisateur qui aura besoin de cette fonctionnalité, il lui faudra juste installer le plugin pour l'envoi des mails. Tel que cela se passe dans le cas des CMS, l'installation de nouveaux plugins aura pour effet d'étendre dynamiquement le code source et la base de données.

## 5.2 Le Framework Spring

Nous nous intéressons à l'architecture Spring parce que dans nos recherches nous avons vu des applications collaboratives basées sur cette architecture, à l'exemple d'eXo Platform. Cette plateforme collaborative met en avant une certaine extensibilité avec la possibilité d'intégrer n'importe quelle application tierce faite avec PHP ou même ASP.net. Notre intérêt sur Spring commence par les généralités.

### 5.2.1 Généralités

Spring est un Framework libre pour construire et définir une infrastructure d'application Java. La première version a été écrite par Rod Johnson en 2002. Il a été publié pour la première fois en juin 2003 sous la licence Apache version 2.0. En rupture avec les conteneurs J2EE, (disqualifiés par de nombreux experts pour leurs lourdeurs [7]), Spring voit le jour et est basé sur la notion «de conteneurs légers». Le cœur de Spring entre dans cette catégorie de solutions [7].

Rod Johnson, dans l'introduction de [8], explique le fait que Spring est un conteneur léger en ces termes : «Spring est effectivement un conteneur dit «léger», c'est-à-dire une infrastructure similaire à un serveur d'applications J2EE. Il prend donc en charge la création d'objets et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendances entre ces objets. Le gros avantage par rapport aux serveurs d'application est qu'avec Spring, les classes n'ont pas besoin d'implémenter une quelconque interface pour être prises en charge par le Framework (au contraire des serveurs d'applications J2EE et des EJBs). C'est en ce sens que Spring est qualifié de conteneur «léger».».

Pour nous faire une idée plus précise de la nature d'un conteneur léger, «nous pouvons établir un parallèle avec les Framework gérant les interfaces graphiques, comme Swing ou SWT (Standard Widget Toolkit). Avec ces outils, nous définissons les composants graphiques à utiliser et nous nous connectons aux événements qu'ils sont susceptibles de générer, un clic sur un bouton, par exemple. Ces Framework prennent en charge le cycle de vie des composants graphiques de manière complètement transparente pour l'application concernée. Un conteneur léger peut offrir des services similaires, mais avec des objets de toute nature.» [7].

Les conteneurs légers fournissent une infrastructure de gestion de l'ensemble des composants de l'application. Via des mécanismes de configuration (tels les fichiers XML permettant d'initialiser les composants), de gestion du cycle de vie des composants et de gestion des dépendances entre composants. Ils sont indépendants de la technologie J2EE et

peuvent fonctionner sur n'importe quel type de serveurs d'applications. Ils peuvent fonctionner également sans serveurs d'applications en utilisant une simple machine virtuelle Java. Les conteneurs légers rendent les applications qui les utilisent à la fois plus flexibles et mieux testables, car le couplage entre les composants est géré par le conteneur, et non plus directement à l'intérieur du code [7]. La notion de flexibilité telle que décrite dans notre description de besoin est rencontré ici via les conteneurs légers. Le Framework Spring s'appuie aussi sur trois concepts clés : l'inversion de contrôle, la programmation orientée aspect (POA), une couche d'abstraction (les couches MVC).

L'inversion de contrôle est gérée de deux manières : la recherche de dépendances et l'injection de dépendances. La recherche de dépendances pour un objet consiste à interroger le conteneur afin de trouver ses dépendances avec les autres objets. L'injection de dépendances peut être de trois façons : via le constructeur, via les modificateurs (setters), via une interface. Grâce à l'inversion de contrôle, les conteneurs légers apportent une solution élégante à la gestion des dépendances et à la création des objets. Ils encouragent la séparation claire des différentes couches de l'application, aidant ainsi à la séparation des préoccupations[7].

La POA comporte des classes et des aspects. Un aspect se différencie d'une classe par le fait qu'il implémente une fonctionnalité transversale à l'application (une fonctionnalité qui, en programmation orientée objet ou procédurale, serait dispersée dans le code de l'application : comme la sécurité, la persistance). La présence de classes et d'aspects dans une même application de type Spring introduit deux dimensions de modularité : celle des fonctionnalités implémentées par les classes et celle des fonctionnalités transversales, implémentées par les aspects [8]. La POA introduit donc de nouvelles notions qui viennent compléter celles de l'approche objet.

Le Framework Spring via sa couche d'abstraction ne concurrence pas d'autres Frameworks. Avec son modèle architecturale MVC (Modèle Vue Contrôleur), Spring s'avère être un Framework multi-couches. Il est facile de faire des insertions au niveau de toutes les couches ; modèle, vue et contrôleur. Pour cela, Spring peut permettre l'intégration d'autres Frameworks comme Hibernate ou iBATIS pour la couche de persistance, Struts et JavaSever Face (JSF) pour la couche de présentation.

### 5.2.2 Exemple de PFC utilisant Spring

L'exemple d'architecture que nous avons choisi de vous présenter est l'architecture de la plateforme collaborative eXo Platforme. Cette architecture se présente ainsi :

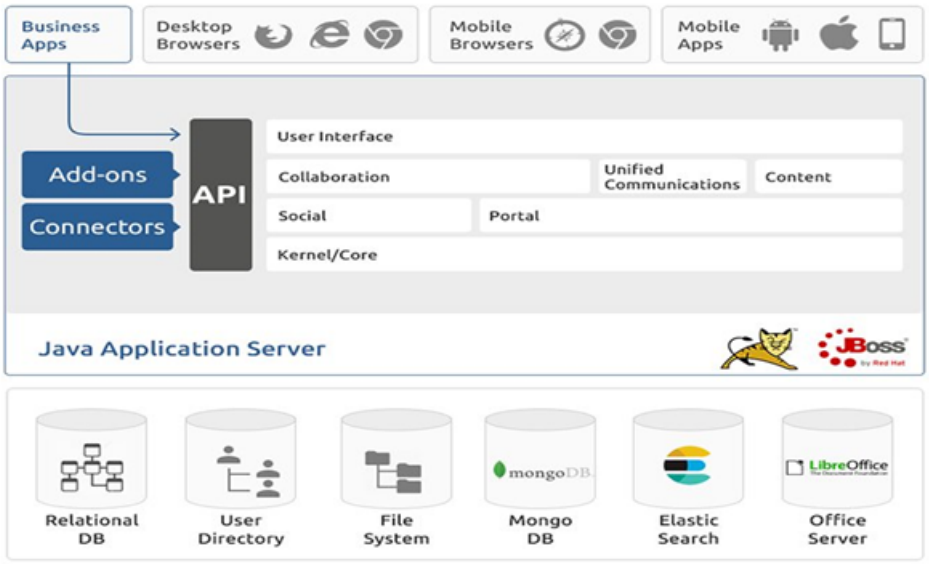



FIGURE 5.2 – Exemple d'architecture utilisant Spring : cas de eXo Platforme

eXo gère l'authentification, les autorisations, la composition d'interface de différents portails tels que le portail par défaut «intranet». La couche sociale amène les composants de base pour la collaboration des utilisateurs : profils

riches, flux d'activités et espaces, wikis, forums, calendrier, chats. Ces fonctionnalités permettent de publier des activités, d'envoyer des notifications.

Il y a également une couche relative au contenu web grâce à Java Content Repository embarqué. De plus, les contenus sont exposables à travers CMIS [1] (Content Management Interoperability Services) et WebDav [2] (Web-based Distributed Authoring and Versioning). La plateforme est open-source. Elle embarque un serveur Tomcat mais peut aussi être déployée sur d'autres serveurs, par exemple JBoss EAP.

Elle stocke les données sur un système de fichiers ainsi que dans une base de données relationnelle au travers des APIs JCR (Java Content Repository) et JPA (Java Persistence Application : Hibernate). En plus il existe une fonctionnalité de chat qui utilise MongoDB [3]. L'indexation et la recherche de données sont faites par ElasticSearch [4].

La plateforme a un microkernel qui orchestre le cycle de vie des services de base tels que la gestion des utilisateurs, l'ordonnanceur de jobs, l'envoi de notifications et le stockage des données. Ces services du microkernel peuvent facilement être surchargés, entendus ou intégrés par des services et plugins personnalisés.

Pour les développeurs, ils peuvent écrire leurs propres applications. Ces applications peuvent s'exécuter au sein de la plateforme, sous forme de gadgets OpenSocial ou de portlets [5], et ce, en utilisant leurs propres Framework web (JSF, Spring, etc.).

### 5.2.3 L'apport de Spring dans notre cas

Notre description de besoins a une exigence en termes de sécurité. En fait, il s'agit d'une couche de sécurité pour les logins, les mots de passe, les crédentials des différents réseaux sociaux. Malgré que cette exigence ne soit pas capitale, il est possible de toujours utiliser la technique des aspects présente dans l'architecture Spring pour isoler et centraliser la sécurité pour éviter qu'elle se trouve dispersé dans le code source de l'application. Comme nous l'avions déjà dit, Spring est une architecture en couche qui donne la possibilité d'y intégrer d'autres Frameworks comme Hibernate, Strust, et bien d'autres. On peut aussi y intégrer des plugins Spring. Dans notre travail, nous devons publier du contenu sur les réseaux sociaux. Pour cela, il existe un plugin Spring nommé «spring-social» que nous pouvons intégrer sur une architecture de base. L'intégration est simple car il faut seulement ajouter dans le fichier de dépendances, le code se trouvant sur l'image suivante pour avoir les fonctions principales du plugin.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.social</groupId>
        <artifactId>spring-social-core</artifactId>
        <version>1.1.6.RELEASE</version>
    </dependency>
</dependencies>
```

FIGURE 5.3 – Dépendance à spring-social core dans un fichier pom.xml (MAVEN)

Si maintenant nous souhaitons ajouter des réseaux sociaux sur lesquels nous voulons publier, il suffit d'ajouter

1. Est un standard ouvert géré par OASIS. Son but est d'augmenter l'interopérabilité entre les systèmes de gestion de contenu.
2. WebDAV est un protocole défini par le groupe de travail IETF du même nom. Décrit dans la RFC 4918, WebDAV permet de simplifier la gestion de fichiers avec des serveurs distants. Il permet de récupérer, déposer, synchroniser et publier des fichiers rapidement et facilement.
3. MongoDB est un serveur utilisant Lucene pour l'indexation et la recherche des données. Il fournit un moteur de recherche distribué et multi-entité à travers une interface REST. C'est un logiciel libre écrit en Java et publié en open source sous licence Apache.
4. ElasticSearch, est un système de gestion de base de données orientée documents, répartissable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données.
5. Un portlet est une application informatique que l'on peut placer dans un portail web, qui sert alors de conteneur. C'est un objet qui affiche un bloc sur une page web.

dans le même fichier de dépendances une dépendance pour chaque réseau social.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.social</groupId>
        <artifactId>spring-social-core</artifactId>
        <version>1.1.6.RELEASE</version>
    </dependency>
     <dependency>
        <artifactId>spring-social-facebook</artifactId>
        <version>2.0.3.RELEASE</version>
    </dependency>
    <dependency>
        <artifactId>spring-social-linkedin</artifactId>
        <version>2.0.3.RELEASE</version>
    </dependency>
    <dependency>
        <artifactId>spring-social-twitter</artifactId>
        <version>2.0.3.RELEASE</version>
    </dependency>
</dependencies>
```

FIGURE 5.4 – Dépendance spring-social pour quelques réseaux sociaux

Sur l'image on a ajouté 3 dépendances. Une pour Facebook, une pour LinkedIn et une pour Twitter. L'ajout de ces dépendances va permettre que les méthodes pour publier sur les réseaux sociaux ci-dessus mentionnés soient disponibles dans notre code source. Ainsi nous pouvons tout simplement utiliser ces méthodes sans perdre du temps à les écrire. C'est l'une des forces de Spring qui gère très bien la notion d'abstraction. Le problème avec spring-social est que, pour l'instant, il permet la publication seulement sur 3 réseaux sociaux, le plugin n'est pas open source. Le code source n'étant pas ouvert il est presque impossible de customiser le plugin. Sur Spring il est possible d'ajouter d'autres réseaux sociaux. Pour Google plus, il faut passer par l'API de Google. Instagram et WhatsApp ont chacun leur propre API que nous pouvons utiliser.

Pour ce qui est de l'envoi de mails aux utilisateurs tel que défini dans notre description de besoins, Spring a déjà une fonctionnalité d'envoi de mail. Cela nous faciliterait la tâche si nous voulons utiliser une architecture Spring. En plus, il existe un Framework développé avec Spring dont le nom est « Spring Boot ». Nous pouvons utiliser ce Framework pour la conception de notre plateforme. Son apport est lié au fait qu'il comporte déjà les différentes couches de Spring (MVC) qu'il faudrait adapter à nos besoins.

## 5.3 L'architecture à base du Web Sémantique

Dans nos différentes recherches, nous avons trouvé plusieurs plateformes collaboratives qui utilisent une architecture liée au web sémantique, telle que BoWiki. Ces plateformes utilisent particulièrement l'OWL (Ontologie Web Language).

### 5.3.1 Généralité

Le web sémantique (expression venant de Tim Berners-Lee[9]) cherche à faciliter les recherches sur le web. Pour cela, il veut mettre des règles sémantiques compréhensibles pour les utilisateurs et les machines. Les utilisateurs vont exécuter différentes tâches sur le web afin d'obtenir des réponses qui satisferont leurs recherches. Et les machines vont essayer d'accéder aux contenus disponibles à travers un raisonnement automatique. Le web sémantique [10] est la base d'une structure qui donne forme à un ensemble de règles. Cette base doit effectuer différentes tâches permettant de

solidifier l'esprit d'ouverture du Web pour tous ses utilisateurs.

Pour effectuer ses différentes tâches, cette base doit trouver des langages de représentation ou des ontologies utilisées afin d'essayer d'automatiser l'interopérabilité et les transformations entre les différentes ontologies et les différents formalismes. Pour permettre ses transformations, elle va mettre en place des calculs et des raisonnements complexes pour arriver à une automatisation. Avec cette automatisation, nous ne devons pas oublier qu'il faudra garantir la sécurité des utilisateurs avec un ensemble de règles déterminées qui augmentera le niveau de confiance de ces derniers.

Nous ne pouvons pas limiter le web sémantique seulement à sa façon de structurer de l'information. Dans le chapitre suivant nous irons un peu plus en détail en insistant sur la manière dont le web sémantique modélise et stock de l'information. Présentement nous parlerons seulement de l'architecture à base du web sémantique. Cette architecture se présente comme suit[10].

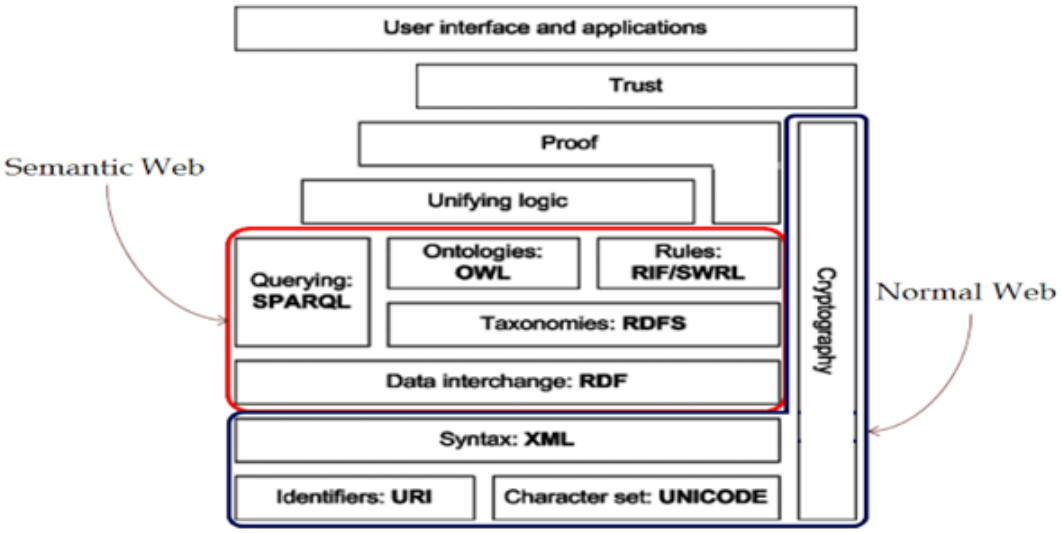



FIGURE 5.5 – Achitecture à base du web sémantique.

L'architecture à base du Web sémantique est illustrée dans la figure ci-dessus. La première couche, URI et Unicode, suit les caractéristiques importantes du WWW existant. Unicode est un standard de codage des jeux de caractères internationaux. Il permet d'utiliser toutes les langues humaines (écrites et lues) sur le Web, à l'aide d'un seul formulaire normalisé. Uniform Resource Identifier (URI) est une chaîne de forme normalisée qui permet d'identifier de manière unique les ressources (par exemple, des documents). Un sous-ensemble d'URI est Uniform Resource Locator (URL), qui contient un mécanisme d'accès et un emplacement (réseau) d'un document, tel que http ://www.example.org/. Un autre sous-ensemble d'URI est URN, qui permet d'identifier une ressource sans impliquer son emplacement ni les moyens de le déréférencer. Un exemple est URN : isbn : 0-123-45678-9. L'utilisation de l'adresse URI est importante pour un système Internet distribué, car elle fournit une identification compréhensible de toutes les ressources. Une variante internationale de l'URI est Internationalized Resource Identifier (IRI), qui permet d'utiliser des caractères Unicode dans un identifiant et pour laquelle un mappage vers l'URI est défini. Dans la suite de ce texte, chaque fois que l'URI est utilisé, l'IRI peut être utilisé [10].

La couche XML (Extensible Markup Language) avec les espaces de noms XML et les définitions de schéma XML permet de s'assurer qu'une syntaxe commune est utilisée dans le Web sémantique. XML est un langage de balisage à usage général pour les documents contenant des informations structurées [10]. Un document XML contient des éléments qui peuvent être imbriqués et qui peuvent avoir des attributs et un contenu. Les espaces de noms XML permettent de spécifier différents vocabulaires de balisage dans un document XML. Le schéma XML sert à exprimer le schéma d'un ensemble particulier de documents XML.

Un format de représentation de données de base pour le Web sémantique est Resource Description Framework

(RDF). RDF est un cadre pour représenter des informations sur les ressources sous forme de graphique. Il était principalement destiné à la représentation de métadonnées sur les ressources WWW, telles que le titre, l'auteur et la date de modification d'une page Web, mais il peut être utilisé pour stocker d'autres données. Il est basé sur des triplets sujet-prédicat-objet formant un graphe de données. Toutes les données du Web sémantique utilisent RDF comme langage de représentation principal [10]. La syntaxe normative pour la sérialisation de RDF est XML au format RDF / XML. La sémantique formelle de RDF est également définie.

RDF lui-même sert à décrire un graphe formé de triples. Tout le monde peut définir le vocabulaire des termes utilisés pour une description plus détaillée [10]. Pour permettre une description normalisée des taxonomies et d'autres constructions ontologiques, un schéma RDF (RDFS) a été créé avec sa sémantique formelle au sein de RDF. RDFS peut être utilisé pour décrire les taxonomies de classes et de propriétés afin de les utiliser pour créer des ontologies légères.

Des ontologies plus détaillées peuvent être créées avec le langage OWL (Web Ontology Language). Le langage OWL est un langage dérivé de la logique de description et offre davantage de constructions que RDFS. Il est syntaxiquement intégré dans RDF. Ainsi, comme RDFS, il fournit un vocabulaire normalisé supplémentaire. OWL existe en trois espèces : OWL Lite pour les taxonomies et les contraintes simples, OWL DL pour la prise en charge de la logique de description complète et OWL Full pour une expressivité maximale et la liberté syntaxique de RDF. Comme OWL est basé sur la logique de description, il n'est pas surprenant qu'une sémantique formelle soit définie pour ce langage [10].

RDFS et OWL ont une sémantique définie et cette sémantique peut être utilisée pour raisonner au sein d'ontologies et de bases de connaissances décrites à l'aide de ces langages. Pour fournir des règles allant au-delà des constructions disponibles à partir de ces langues, les langages de règles sont également standardisés pour le Web sémantique. Deux normes émergent - RIF et SWRL (Semantic Web Rule Language).

Pour interroger les données RDF ainsi que les ontologies RDFS et OWL avec des bases de connaissances, un langage SPARQL (Simple Protocol) et RDF (Query Language) est utilsé. SPARQL est un langage semblable à SQL, mais utilise des triplets et des ressources RDF pour la partie correspondante de la requête et pour le retour des résultats de cette requête. Comme RDFS et OWL sont tous deux basés sur RDF, SPARQL peut également être utilisé pour interroger directement des ontologies et des bases de connaissances. SPARQL n'est pas seulement un langage de requête, c'est aussi un protocole d'accès aux données RDF [10].

Toute la sémantique et les règles sont exécutées au-dessous de la couche Proof et le résultat utilisé pour prouver les déductions de règles OWL [10]. La couche Proof et Trust sont utilisées pour vérifier que les «imputs» (données utilisateurs) viennent d'une source fiable. Pour ces «imputs», il convient d'utiliser des moyens de cryptographie, tels que des signatures numériques, pour vérifier l'origine des sources. Au-dessus de ces couches, une interface utilisateur peut être construite pour faire par exemple des requêtes de recherche, d'ajout, de modification ou de suppression de données.

### 5.3.2 L'apport du web sémantique dans notre travail

Le web sémantique (notamment OWL) pourrait nous être très utile si au final nous le choisissons pour la conception de notre plateforme collaborative. Tout d'abord, l'OWL permet de structurer des informations plus complexes qui peuvent évoluer dans le temps. Par exemple, on peut être intéressé par des rédacteurs qui ont écrit et publié un contenu ayant au moins une photo «libre de droit».

Via des mécanismes propres à OWL il est possible de filtrer de l'information d'une manière très précise en utilisant les mécanismes d'inférence. On peut par exemple décider qu'avant de publier un contenu sur les réseaux, pour des besoins d'éthique, on filtre les contenus ayants certains mots clefs ou certaines significations. On peut aussi filtrer des contenus ayants des images ou des vidéos choquantes. Dans notre description des besoins, il y a une spécification qui décrit le workflow de rédaction d'un contenu. En fait, un contenu peut être dans un état REDIGE, ENREGISTRE,

VALIDE, ou PUBLIE. Pour satisfaire ce genre de besoin, une architecture à base du web sémantique accepte d'autres extensions comme Semantic MediaWiki. Cette extension peut nous aider à ajouter des annotions sémantiques pour REDIGE, ENREGISTRE, VALIDE, ou PUBLIE sur nos contenus afin de matérialiser l'état exact où se trouve lesdits contenus. À titre d'exemple, Robert Hoehndorf et al, dans leur projet BoWiki [12], qui est un système web, utilise Semantic MediaWiki pour annoter et classer des données biologiques.

Dans une architecture à base du web sémantique il est aussi possible d'intégrer des éditeurs d'ontologies comme : Collaborative Protégé, OntoWiki ou Hozo. Si on utilise par exemple Collaborative Protégé, on peut y associer Jena, qui est une API java donnant la possibilité d'interroger OWL ou RDF depuis une autre application Java. Cette technique apport une certaine flexibilité qui peut être utile dans la conception de l'architecture de notre propre plateforme collaborative.

Une architecture à base du web sémantique est extensible [11]. Pour cela, si on veut ajouter de nouvelles informations on peut utiliser son extension OWL-plugin pour la création de contenu web sémantique[12]. Cette extension pourrait être utilisée pour ajouter un nouveau réseau social ou un nouveau champ. Le chapitre suivant explique plus en détail comment on peut ajouter de nouvelles informations dans le but de densifier une base de connaissances OWL.

# Chapitre 6

# Présentation générale d'OWL et des BDRs

Nous avons vu dans le chapitre 5 que certaines plateformes collaboratives utilisent une base de données relationnelle (BDR), d'autres une OWL et/ou un système de gestion des documents comme MongoDB. C'est pourquoi nous nous sommes intéressés à ces différentes formes de modélisations de données. Nous allons parler uniquement de la modélisation avec des BDRs et l'OWL, la gestion des documents ne faisant pas partie de nos besoins.

## 6.1 Base de données relationnelle (BDR)

En informatique, une base de données relationnelle est une base de données où l'information est organisée dans des tableaux à deux dimensions appelés des relations ou tables. Les systèmes de gestion des bases de données relationnelles (SGBDRs) sont des logiciels permettant de concevoir, de stocker et de structurer ces informations. Certaines plateformes collaboratives utilisent les BDRs notamment Joomla, Agorapulse. Ce modèle relationnel a été proposé par Cold[13] et a révolutionné la gestion des référentiels des données de masses. Le fonctionnement des BDRs obéit à plusieurs mécanismes : la normalisation, les contraintes d'intégrité et d'unicité, l'index, les procédures stockées, les trigger, les vues, le commit, le rollback, etc. Dans ce travail nous ne développerons pas ces mécanismes, nous les considérons comme des acquis, vous pouvez consulter[14] pour plus d'informations.

## 6.2 Ontology Web Language (OWL)

Dans le chapitre précèdent nous avons abordé plusieurs concepts liés à OWL. Le but de notre travail n'est pas de détailler en profondeur OWL mais d'avoir des notions de base nécessaires à la structuration de l'information. Dans cette section nous allons aborder : les domaines de connaissances, la description logique, les raisonneurs et les mécanismes d'inférences. Pour apprendre plus sur OWL, nous conseillons de lire[15].

### 6.2.1 domaine de connaissance (DC)

Le domaine de connaissance est un ensemble de termes ou de vocabulaires (propres à un domaine) contenant des règles ou des relations qui constituent l'ontologie. Cette dernière est généralement constituée d'attributs, de relations, de classes. Elle permet de structurer, de décrire de l'information dans un contexte déterminé tout en utilisant un vocabulaire précis et compréhensible par les utilisateurs dudit contexte[16]. Un DC peut être représenté à travers un graphe conceptuel. On peut voir un DC comme un champ sémantique, qui est un groupe de mots qui partagent les aspects sémantiques et qui sont utilisés dans le même contexte pour décrire un sujet spécifique. Notre sujet spécifique est : la publication sur les réseaux sociaux. Un ensemble de mot décrivant cette publication peut être : compte, profil, like, mur, photo, vidéo, live, Messenger, post, tweet, selfi, commentaire, amis, publique, privé, gif, emotion, emoji, partager, etc. Un domaine de connaissance pour la publication sur les réseaux sociaux devrait contenir tous ces mots ainsi que les relations qui les lient.

### 6.2.2 La description logique

Dans [17] les logiques de description aussi appelé logiques descriptives (LD) sont une famille de langages de représentation de connaissance. Elles peuvent être utilisées pour représenter la connaissance terminologique d'un domaine d'application d'une manière formelle et structurée. Le nom de logique de description se rapporte, d'une part à la description de concepts utilisée pour représenter un domaine et d'autre part à la sémantique basée sur la logique qui peut être donnée par une transcription en logique des prédicats du premier ordre [6]. La logique de description a été développée comme une extension des langages orientés cadre, une famille de langage de programmation pour l'intelligence artificielle, et des réseaux sémantiques, qui ne possédaient pas de sémantique formelle basée sur la logique. La description logique [18] permet de représenter un domaine de connaissance qui est modélisé avec OWL. Dans ce sens, 2 termes importants émergent : la TBox (terminological box) et l'ABox (assertional box). Les éléments importants d'une description logique sont :

- les concepts : catégories générales d'individus
- les relations entre les concepts et leurs propriétés
- les individus : éléments réels appartenant au domaine.

### 6.2.3 ABox et TBox

La TBox contient les axiomes définissant les classes, les propriétés et les types de données du domaine. Ces classes du domaine doivent être uniques, donc elles doivent apparaître qu'une seule fois dans le domaine de connaissance. L'ABox contient les assertions sur les individus, c'est-à-dire les instances de classes du domaine et les relations entre ces mêmes individus.

Les termes "ABox" et "TBox" sont utilisés pour décrire deux types d'énoncés différents dans une base de connaissances. Les déclarations de TBox décrivent la conceptualisation d'un domaine d'intérêt en définissant différents ensembles d'individus décrits en fonction de leurs caractéristiques (propriétés). ABox sont des déclarations compatibles TBox sur des individus appartenant à ces ensembles. Par exemple, un étudiant spécifique est un individu du groupe appelé "Étudiant". Cet ensemble peut être défini comme un sous-ensemble de toutes les personnes fréquentant un établissement d'enseignement, ce qui permet d'indiquer l'établissement d'enseignement spécifiquement fréquenté par chaque personne. Ensemble, les déclarations ABox et TBox constituent une base de connaissances.

### 6.2.4 Raisonneur et mécanisme des moteurs d'inférences

Avant d'expliquer le mécanisme des moteurs d'inférence, nous nous sommes questionnés sur qu'est-ce que l'inférence ? Selon W3C [1], «Inference means that automatic procedures can generate new relationships based on the data and based on some additional information in the form of a vocabulary, e.g., a set of rules. Whether the new relationships are explicitly added to the set of data, or are returned at query time, is an implementation issue.». Cela signifie que les moteurs d'inférences sont des procédures automatiques pouvant générer de nouvelles relations basées sur les données et sur des informations supplémentaires sous la forme d'un vocabulaire, par exemple un ensemble de règles. Ces moteurs sont des raisonneurs. Les principales tâches d'un raisonneur sont :

- De déterminer si une description est satisfaisable, une description étant satisfaisable s'il n'y a pas de contradiction dans le modèle.
- De déterminer si une description est la subsomption d'une autre, c'est-à-dire de déterminer si la description est plus générale qu'une autre.
- De vérifier que les assertions de la ABox sont consistantes. Une ABox est consistante s'il existe un modèle, c'est-à-dire s'il existe une interprétation satisfaisant tous les axiomes de la TBox.
- De s'assurer que les assertions de la ABox permettent de préciser à quel(s) concept(s) est attaché un individu.
- De lister les individus qui instancient un concept particulier.

1. Source : https ://www.w3.org/standards/semanticweb/inference ?fbclid=IwAR1YRTXWIHkmXTXc5dBaQDYfhQD_DwVEP0XaBrjB6QoncmheYXSqECkujI4, 04/09/2019

L'utilisateur d'un raisonneur peut se baser sur ces propriétés pour lui poser des questions propres au domaine modélisé. Le raisonneur aura donc la responsabilité de répondre aux questions en utilisant différents algorithmes. En logique descriptive, il y a quatre propriétés intéressantes à prouver pour une TBox T [18] :

- **Satisfaisabilité** : Un concept C est satisfaisable relativement à T s'il existe un modèle I de T tel que I(C) $\neq \phi$. En d'autres mots, un concept est satisfaisable s'il existe au moins une entité du monde décrit qui peut appartenir à l'ensemble décrit par ce concept. Par exemple, le concept Homme $\sqcap$ ¬Homme est insatisfaisable, puisque l'intersection d'un ensemble avec son complément est toujours l'ensemble vide. Il est donc impossible qu'une entité du monde appartienne à cette classe.
- **Subsomption** : Un concept C est subsumé par un concept D relativement à T si I(C) $\subseteq$ I(D) pour tout modèle I de T. Dans ce cas, on écrira T $\models$ C $\sqsubseteq$ D. Par exemple, si une terminologie T contient l'axiome Mère $\equiv$ Femme $\sqcap$ $\exists$aEnfant.Personne, il est relativement aisé de démontrer la subsomption suivante : Mère $\sqsubseteq$ Femme. Autrement dit, si une entité appartient à l'ensemble des mères, elle appartient aussi à l'ensemble des femmes.
- **Équivalence** : Deux concepts C et D sont équivalents relativement à T si I(C) = I(D) pour tout modèle I de T. Dans ce cas, on écrira T $\models$ C $\equiv$ D.
- **Disjonction** : Deux concepts C et D sont disjoints relativement à T si I(C) $\cap$ I(D) = $\phi$ pour tout modèle I de T. On désignera ce fait par l'énoncé suivant : T $\models$ C $\sqcap$ D $\sqsubseteq$ $\bot$. Par exemple, on peut démontrer que les concepts Père et Mère sont disjoints. En effet, selon la définition du concept Homme, un homme fait partie du complément de l'ensemble des femmes. Ainsi, une entité ne peut être à la fois un homme et une femme, ce qui implique qu'elle ne peut pas être à la fois un père et une mère, puisque par définition un père est aussi un homme, et une mère est aussi une femme.

### 6.2.5 Ensembles de règles

Les règles appliquées sur les moteurs d'inférences sont des fonctions qui prennent un à plusieurs formules en paramètre pour en obtenir qu'une seule à la sortie. Les formules qui viennent en paramètre sont appelées : prémisses et celle en sortie est une conclusion. Si nous faisons un parallèle avec la logique des prédicats, nous remarquons l'existence des méta-règles (expressions logiques) qui permettent de représenter les prémisses et la conclusion. Par exemple dans, le modus ponens, le modus tolens et le chaînage [2].

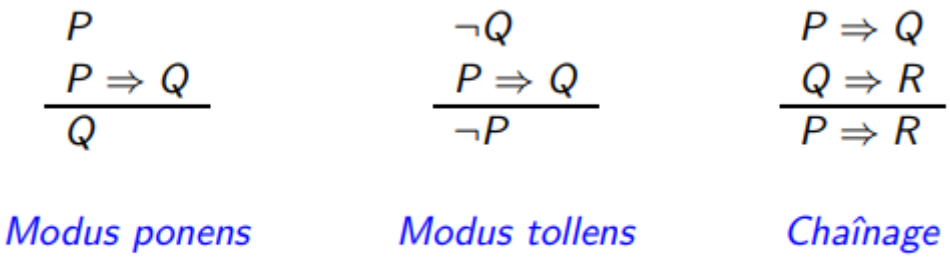



FIGURE 6.1 – méta-règles

Dans ces différentes formules, nous observons différentes expressions logiques représenter par des axiomes (proposition considérée comme vraie). Prenons le cas du chaînage qui pourrait représenter les relations entre un administrateur, un rédacteur et un article. Admettons que :
P = Administrateur
Q = Rédacteur
R = Article
D'après la règle du chaînage, un administrateur est un rédacteur, un rédacteur peut rédiger un article, on en déduit

2. Cours de Technique d'Intelligence Artificielle – Jean-Marie Jacquet 2015, Université de Namur

qu'un administrateur peut aussi rédiger un article. Dans cet exemple, nous avons montré un des modes de fonctionnement du moteur d'inférence mais il en existe d'autres :

- Celui que nous avons représenté dans l'exemple est le chaînage avant. Cela consiste à arriver à des conclusions en considérant les différentes prémisses comme vrai.
- Le chaînage arrière part de la conclusion afin de trouver les prémisses pouvant correspondre au résultat. On se rapproche d'un raisonnement déductif.

Le chaînage mixte est un mélange des 2 modes, considérés comme étant efficace, voir le plus efficace parce que l'on pourra trouver immédiatement les prémisses ou la conclusion. Pour exprimer ces règles de raisonnement, nous allons utiliser les requêtes SPARQL. Nous rappelons que SPARQL est un langage de requête pour traiter les données contenues dans les documents RDF. Avec SPARQL, nous allons montrer comment l'OWL fait face à l'extensibilité des données.

### 6.2.6 Représentation d'un document OWL

Un document OWL peut être représenté sur plusieurs formats notamment le format graphe, Turtle, JSON-LD ou le format RDF/XML Syntax. Dans cette partie nous allons utiliser principalement la représentation RDF/XML Syntax. Les spécifications sur cette représentation peuvent être trouvées sur le site web https ://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ (consulter le 07 avril 2020).

### 6.2.7 L'en-tête d'une ontologie

L'en-tête d'une ontologie est particulièrement constitué des espaces de nom. Pour pouvoir utiliser les termes dans une ontologie, il est important d'indiquer avec rigueur de quels vocabulaires ces termes découlent. Ainsi, comme pour tout document XML, une ontologie débute par une définition d'espace de nom qui est inclus dans la balise rdf :RDF.

```
<!DOCTYPE rdf:RDF [
  <!ENTITY sn "http://www.semanticweb.org/will/ontologies/2019/7/kwessi#" >
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
]>

<rdf:RDF xmlns      ="&sn;"
     xmlns:SocialNetwork="&sn;"
      xml:base="&sn;"
      xmlns:owl="http://www.w3.org/2002/07/owl#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:xml="http://www.w3.org/XML/1998/namespace"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:swrl="http://www.w3.org/2003/11/swrl#"
      xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#"
      xmlns:swrlb="http://www.w3.org/2003/11/swrlb#">
```

FIGURE 6.2 – exemple d'en-tête d'ontologie OWL

Dans l'exemple ci-dessus, l'ontologie (KWeSSi) que nous décrivons a comme espace de nom « sn ». On voit aussi des espaces de nom pour owl, rdf, xml, xsd, rdfs, etc

### 6.2.8 Les classes

Une classe fournit un mécanisme d'abstraction permettant de regrouper des ressources avec des caractéristiques semblables. Chaque classe OWL est associée à un groupe d'individus appelé « extension de classe ». Une classe a une signification intensionnelle mais non égale à son extension de classe. Pour cela, deux classes peuvent avoir la même extension de classe et être des classes différentes. Par exemple l'individu de nom « Dégozard » peut être en même temps de la classe « Enseignant » et « Étudiant », et pourtant les classes « Enseignant » et « Étudiant » sont bien différentes. Une subClassOf est un mécanisme qui permet d'hériter d'une autre classe. Pour définir les classes OWL, on utilise des descriptions de classe, qui peuvent se combiner à des axiomes de classe.

**I. Les descriptions de classe**

Une description de classe définit une classe OWL en nommant la classe, ou bien en définissant l'extension de classe d'une classe anonyme. OWL contient 6 types de description de classe :
c1- L'indicateur de classe (l'adresse URI),
c2- L'énumération : liste les individus formant les instances de classe,
c3- La restriction de propriétés : permet d'appliquer certaines contraintes sur les propriétés
c4- L'intersection
c5- L'union
c6- Le complémentaire

«Les descriptions de classe des types c2 à c6 décrivent respectivement une classe contenant exactement les individus énumérés (type c2), une classe de tous les individus satisfaisant à une restriction de propriété particulière (type c3), ou des classes satisfaisant à des combinaisons booléennes de descriptions de classe (type c4, c5 et c6). On peut assimiler l'intersection, l'union et la complémentarité aux opérateurs logiques ET, OU et NON respectivement. Les quatre derniers types conduisent à des descriptions de classe imbriquées qui peuvent donc théoriquement produire des descriptions de classe de complexité arbitraire. En pratique, le niveau d'imbrication est habituellement limité.» [15]. Un exemple de la syntaxe de la description de classe de type c1 est : <owl :Class rdf :ID="SocialNetwork"/>

«Il y a deux identificateurs de classe OWL prédéfinis, à savoir les classes owl :Thing et owl :Nothing. L'extension de classe de owl :Thing est l'ensemble de tous les individus. L'extension de classe de owl :Nothing est l'ensemble vide. Par conséquent, chaque classe OWL est une sous-classe de owl :Thing, et inversement owl :Nothing est une sous-classe de chaque classe.» [17].

**A) L'énumération**

Elle se définit en utilisant la propriété **owl :oneOf**. Elle est une liste d'individus constituant une instance de la classe. Ainsi, on peut définir une classe en faisant une énumération exhaustive de ses instances. L'extension de classe d'une classe décrite avec owl :oneOf doit contenir exactement les individus énumérés, ni plus ni moins. Dans une énumération, la liste des individus est représentée par une structure RDF comme ceci «rdf :parseType='Collection'». Ci-dessous un exemple de la syntaxe RDF/XML définissant la liste des personnes qui travaillent sur ce mémoire.

```
<owl:Class>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="kwessi#Leuni"/>
    <owl:Thing rdf:about="kwessi#Effayong"/>
  </owl:oneOf>
</owl:Class>
```

FIGURE 6.3 – énumération en owl.

Par défaut tous les individus sont des instances de la classe owl :Thing. Notons que dans le langage OWL Lite, il n'y a pas d'énumération.

**B) Les restrictions de propriétés**

Dans [16], «une restriction de propriété est un type particulier de description de classe. Elle décrit une classe anonyme, c'est-à-dire la classe de tous les individus satisfaisant à la restriction. Le langage OWL distingue deux types de restrictions de propriété : celles contraignant sa valeur et celles contraignant sa cardinalité.
Une contrainte de valeur exerce une limitation sur l'image de la propriété lorsqu'elle s'applique à cette description de classe particulière.». Par exemple, en nous inspirant de l'exemple se trouvant sur l'image suivante, nous pouvons nous baser sur des individus dont la valeur de la propriété hasAccount serait un compte (Account) pour l'utiliser ensuite dans un axiome de classe, peut-être même dans un axiome de classe de Account. Ce n'est pas la même chose qu'avec rdfs :range qui s'utilise à toutes les situations où la propriété est utilisée.

«Une contrainte de cardinalité exerce une limitation sur le nombre des valeurs prises par une propriété dans le contexte de cette description de classe particulière.» [16]. Par exemple, on peut indiquer que la propriété hasCaracter appliquée à un message de Twitter n'accepte que 280 caractères. La même propriété, appliquée à un message de Facebook acceptera, cette fois ci, que 63206 caractères.
Les contraintes de valeurs sont multiples et sont les suivantes :

- owl :allValuesFrom,
- owl :someValuesFrom
- owl :hasValue.

Dans le langage OWL Lite, La contrainte de valeur owl :hasValue n'est pas permise. Un exemple avec owl :allValuesFrom est le suivant :

```
<owl:Class rdf:about="kwessi#FacebookAccount">
    <rdfs:subClassOf rdf:resource="kwessi#Account"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource=" kwessi#hasFacebookPublication"/>
            <owl:allValuesFrom rdf:resource="kwessi#FacebookPublication"/>
        </owl:Restriction>
    </rdfs:subClassOf>
<owl:disjointWith rdf:resource="kwessi#TwitterAccount"/>
```

FIGURE 6.4 – exemple de owl :hasValue.

Dans l'exemple ci-dessus, FacebookAccount est défini comme une sous-classe d'Account. Une classe FacebookAccount est reliée normalement à une classe FacebookPublication via la propriété hasFacebookPublication. La restriction de valeur owl :allValuesFrom appliquée à une propriété hasFacebookPublication permet de préciser que cette propriété ne peut recevoir que les publications de la classe FacebookPublication. Par conséquent, elle ne pourra pas accepter les publications de la classe TwitterPublication.

D'autres contraintes de cardinalité sont :

- owl :maxCardinality
- owl :minCardinality
- owl :cardinality

Un exemple avec owl :cardinality peut être :

```
<owl:Class rdf:about="kwessi#Account">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="kwessi#hasAccount"/>
            <owl:cardinality rdf:datatype="&xsd; nonNegativeInteger">1</owl:cardinality>
            <owl:onClass rdf:resource="kwessi#SocialNetwork"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

FIGURE 6.5 – exemple de owl :cardinality.

L'exemple précédent montre qu'un individu de la classe Account est lié (via la propriété hasAccount) a exactement un seul individu de la classe SocialNetwork.

### C) L'intersection, l'union et la complémentarité

Ces 3 descriptions de classe sont les plus évoluées de la logique de description [16]. L'exemple ci-dessus montre que la valeur de la propriété owl :intersectionOf est une liste de deux descriptions de classe. On a une énumération

décrivant des individus Message1, Message2 et une autre décrivant les individus Message1 et Message3. L'intersection est dans ce cas une classe d'un seul individu, c'est-à-dire Message1, parce que c'est le seul individu qui est présent dans les deux énumérations.

- L'intersection

```
<owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class>
          <owl:oneOf rdf:parseType="Collection">
            <owl:Thing rdf:about="kwessi#Message1" />
            <owl:Thing rdf:about="kwessi#Message2" />
          </owl:oneOf>
        </owl:Class>
        <owl:Class>
          <owl:oneOf rdf:parseType="Collection">
            <owl:Thing rdf:about="kwessi#Message1" />
            <owl:Thing rdf:about="kwessi#Message3" />
          </owl:oneOf>
        </owl:Class>
    </owl:intersectionOf>
</owl:Class>
```

FIGURE 6.6 – exemple de owl :intersection.

En fait, les trois individus doivent être tous différents. Par définition, ce n'est pas vrai dans OWL [16]. Des appels différents à des adresses URIs peuvent se rapporter aux mêmes individus, car le langage OWL ne présuppose pas de «noms uniques». La propriété owl :intersectionOf est comme une conjonction logique.

- L'union
  Si nous prenons l'exemple précédent et que nous remplaçons intersectionOf par unionOf, l'union sera une classe contenant 3 individus : Message1, Message2, Message3. Dans le langage OWL Lite, la propriété owl :unionOf n'existe pas. Cette propriété est comme une disjonction logique.

- La complémentarité
  Considérons l'exemple suivant qui présentent les réseaux sociaux «non Twitter» :

```
<owl:Class>
  <owl:complementOf>
    <owl:Class rdf:about="kwessi#Twitter"/>
  </owl:complementOf>
</owl:Class>
```

FIGURE 6.7 – exemple de owl :complementOf.

Une extension de la description de classe de l'exemple précédent contiendra tous les individus n'étant pas de la classe « Twitter ». Nous notons que owl :complementOf ne fait pas partie du langage OWL Lite.

### II. Les axiomes de classe

«Les descriptions de classe constituent les blocs d'assemblage pour définir les classes au travers d'axiomes de classe. L'axiome de classe le plus élémentaire prend la forme d'une description de classe de type c1 : il déclare juste l'existence de la classe au moyen de l'élément owl :Class et d'un identificateur de classe» [16].
L'exemple <owl :Class rdf :ID="SocialNetwork"/>, est l'axiome de classe dont URI est «#SocialNetwork» et le nom de la classe est « SocialNetwork ».

L'exemple d'axiome de classe précédant ne nous donne pas grand-chose sur la classe SocialNetwork. Il est donc nécessaire de combiner les axiomes de classe aux descriptions de classe pour mieux représenter une classe. En langage OWL, il existe 3 techniques pour combiner les descriptions de classe aux axiomes de classe :

- La relation rdfs :subClassOf : permet d'étendre une description de classe par héritage. Comme exemple les classes Facebook, Instagram peuvent toutes hériter de la classe SocialNetwork.

- La relation owl :disjointWith : permet de matérialiser le fait que deux ou plusieurs descriptions de classes n'ont aucune extension de classe en commun.

- La relation owl :equivalentClass : permet de dire qu'une description de classe a absolument la même extension de classe qu'une autre description de classe.

Pour les deux dernières techniques, un exemple peut être :
Facebook owl :equivalentClass SocialNetwork,
Instagram owl :equivalentClass SocialNetwork,
Facebook owl :disjointWith Instagram.

Littéralement cet exemple exprime que le réseau social Facebook est exactement un SocialNetwork, de même que le réseau social Instagram, mais Facebook et Instagram sont complétement différents.

### 6.2.9 Les propriétés

Dans le langage OWL il existe 2 formes de propriétés servant à concevoir une ontologie en y introduisant des relations entre les éléments de cette ontologie. Il s'agit de :

- Les propriétés d'objets : représentées par l'instruction clé, owl :ObjectProperty.

- Les propriétés de types d'objets : représentées par l'instruction clé, owl :DatatypeProperty.

«Une propriété d'objet est définie comme une instance de la classe OWL intégrée owl :ObjectProperty. Une propriété de type de donnée est définie comme une instance de la classe OWL intégrée owl :DatatypeProperty. Toutes les deux sont des sous-classes de la classe RDF rdf :Property.
Dans le langage OWL Full, les propriétés d'objets et de types de données ne sont pas disjointes. Puisqu'on peut traiter les valeurs de données comme des individus, les propriétés de types de données sont effectivement des sous-classes de propriétés d'objets. Dans OWL Full, la classe owl :ObjectProperty équivaut à la classe rdf :Property. En pratique, cela entraîne principalement des conséquences sur l'utilisation de la propriété owl :InverseFunctionalProperty» [16].

```
<!-- Les propriétés d'objet -->
<owl:ObjectProperty rdf:ID="hasAccount">
    <rdfs:domain rdf:resource="kwessi#SocialNetwork"/>
    <rdfs:range rdf:resource="kwessi#Account"/>
</owl:ObjectProperty>
```

FIGURE 6.8 – exemple de propriété d'objets.

Dans l'exemple ci-dessus, la propriété d'objets hasAcount permet de lier une classe SocialNetwork à tous ses comptes (Account).

```
    <!-- Propriétés de type de données -->
<owl:DatatypeProperty rdf:ID="nameNetwork">
    <rdfs:domain rdf:resource="kwessi#SocialNetwork"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

FIGURE 6.9 – exemple de propriété de type de données.

L'exemple précédent permet d'ajouter sur la classe SocialNetwork une propriété de type de données «nameNetwork» qui est de type «string . En clair, un réseau social a un nom dont la valeur est une chaine de caractères.

Les axiomes de propriétés contiennent d'autres caractéristiques de propriétés. Le langage OWL a aussi les structures ci-dessous dans ses axiomes de propriétés :

- owl :equivalentProperty et owl :inverseOf : permettent de définir les relations avec d'autres propriétés
- rdfs :subPropertyOf, rdfs :domain et rdfs :range : ce sont des structures du schéma RDF
- owl :symetricProperty et owl :transitiveProperty : définissent des caractéristiques de propriétés logiques sur des axiomes de propriétés.
- owl :functionalProperty et owl :inverseFunctionalProperty : définissent un ensemble de contraintes de cardinalité globales.

De la même manière que les classes sont étendues, les propriétés peuvent aussi l'être. Une extension d'une propriété est l'ensemble d'instances associé à cette propriété. Les instances de propriétés ne sont pas des éléments isolés mais des couples de déclarations de propriétés sujet-objet. En comparaison avec les bases de données relationnelles, on appellerait les instances d'une propriété les tuples d'une relation binaire (la propriété) [16].

### 6.2.10 Les instances ou individus

Les individus sont des éléments les plus concrets dans une structure de données OWL en ce sens que l'ensemble des informations réelles sont stockées à travers les individus (encore appelés instances ou « faits »). Dans l'extension de classe, les individus sont appelés des instances de la classe. Globalement on distingue deux types de faits qui sont les suivants :

- Les faits concernant les valeurs de propriétés des individus et l'appartenance de classe,
- Les faits concernant l'identité des individus.

#### I. Les faits concernant les valeurs de propriétés des individus et l'appartenance de classe

Ce sont des déclarations permettant de préciser une appartenance à une classe ou à une valeur de leurs propriétés.

```
<SocialNetwork rdf:ID="kwessi#facebook">
    <nameNetwork>Facebook</nameNetwork>
    <urlNetwork>https://www.facebook.com</urlNetwork>
    <hasAccount rdf:ID="kwessi#account1">
        <nameAccount>dipanda</nameAccount>
        <emailAccount>dipanda@kwessi.cm</emailAccount>
        <password>dipanda1234=</password>
    </hasAccount>
</SocialNetwork>
```

FIGURE 6.10 – exemple concernant les valeurs de propriétés des individus et l'appartenance de classe.

L'exemple ci-dessus définit un individu de la classe SocialNetwork et un autre de la classe Account. L'individu «facebook» de type SocialNetwork a pour nom dont la valeur est «Facebook» et pour URL dont la valeur est «https ://www.facebook.com». L'individu «account1» de type hasAccount a pour nom dont la valeur est «dipanda», pour email de valeur «dipanda@kwessi.cm» et pour mot de passe de valeur «dipanda1234=». De plus, l'individu «account1» est un compte de l'individu «facebook». On peut constater par cet exemple que les individus manipulent des informations concrètes.

#### II. Les faits concernant l'identité des individus

Selon [16], «beaucoup de langages admettent un postulat de soi-disant noms uniques : des noms différents désignent des choses différentes dans la réalité. Sur le Web, ce postulat est impossible. Par exemple, la même personne pourrait être désignée de plusieurs façons différentes (c'est-à-dire avec des appels d'adresse URI différents). Le langage OWL

ne retient pas cette hypothèse pour cette raison. Sauf déclaration explicite selon laquelle deux adresses URIs se rapportent au même individu ou sinon à des individus différents, les outils OWL devrait en principe supposer les deux éventualités.». Pour exprimer les faits concernant l'identité des individus, le langage OWL possède 3 structures :

- owl :AllDifferent : c'est une relation qui permet de déclarer que les individus d'une liste sont tous différents,

- owl :differentFrom : c'est une relation qui est utilisée pour définir que deux adresses URIs pointent sur des individus différents,

- owl :sameAs : cette relation permet de déclarer que deux adresses URIs pointent sur un même individu.

### 6.2.11 Les types de données

«Le langage OWL utilise le système de typage de données RDF, qui fournit un mécanisme permettant une référence aux types de données du schéma XML (XML Schema Datatypes). Les valeurs de données sont des instances de la classe rdfs :Literal du schéma RDF. Les littéraux peuvent se présenter soit sous une forme brute (pas de type de donnée), soit sous une forme typée. Les types de données sont des instances de la classe rdfs :Datatype. Dans RDF/XML, le type d'un littéral est indiqué par un attribut rdf :datatype» [16].

#### I. Les types de données recommandés

Pour un document RDF on recommande d'utiliser les types de données intégrés du schéma XML suivants :

- Le type de données primitif xsd :boolean,

- Le type de données primitif xsd :string, plus ses types dérivés suivants : xsd :normalizedString, xsd :token, xsd :language, xsd :NMTOKEN, xsd :Name et xsd :NCName,

- Les types de données primitifs relatifs au temps xsd :dateTime, xsd :time, xsd :date, xsd :gYearMonth, xsd :gYear, xsd :gMonthDay, xsd :gDay et xsd :gMonth,

- Les types de données primitifs xsd :hexBinary, xsd :base64Binary et xsd :anyURI,

- Les types de données primitifs numériques xsd :decimal, xsd :float et xsd :double, plus ses types dérivés suivant : xsd :integer, xsd :positiveInteger, xsd :nonPositiveInteger, xsd :negativeInteger, xsd :nonNegativeInteger, xsd :long, xsd :int, xsd :short, xsd :byte, xsd :unsignedLong, xsd :unsignedInt, xsd :unsignedShort, xsd :unsignedByte.

Il est possible pour une application utilisant le langage OWL de définir ses propres types de données. Pour cela, il faudrait s'appuyer sur une instance de la classe rdfs :Datatype. Ces types de données ne sont pas reconnus par la syntaxe RDF mais ils sont traités de la même manière que les types de données non pris en charge par le raisonneur (données non consistantes).

#### II. Le raisonnement des types de données

« Les outils peuvent varier en matière de gestion du raisonnement des types de données. Au minimum, les outils doivent prendre en charge le raisonnement des types de données xsd :string et xsd :integer du schéma XML. Les outils OWL Full doivent également prendre en charge le type rdf :XMLLiteral. En ce qui concerne les types de données non pris en charge, les littéraux lexicalement identiques devraient être considérés égaux, tandis que ceux lexicalement différents ne sauraient être décidés égaux ou inégaux. Les types de données non reconnus devraient recevoir le même traitement que pour les types de données non pris en charge. » [16].

### 6.2.12 Les restrictions sur les données

La mise en commun de l'ensemble des éléments que nous avons expliqués plus haut permet d'appliquer des contraintes sur les données. Par exemple, dans le cadre de ce mémoire, d'après la description des besoins, chaque réseau social à un nombre de résolution spécifique pour ses vidéos. Twitter n'admet que les résolutions 1200x1900, 1920x1200 et 32x32. L'exemple suivant permet de gérer ce genre de contraintes.

```
<owl:DatatypeProperty rdf:about="kwessi#resolutionVideoTwitter">
    <rdf:type rdf:resource="owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="kwessi#TwitterVideoResolution"/>
    <rdfs:range>
        <rdfs:Datatype>
            <owl:oneOf>
                <rdf:Description>
                    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
                    <rdf:first rdf:datatype="&xsd;string">1200x1900</rdf:first>
                    <rdf:rest>
                        <rdf:Description>
                            <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
                            <rdf:first rdf:datatype="&xsd;">1920x1200</rdf:first>
                            <rdf:rest>
                                <rdf:Description>
                                    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
                                    <rdf:first rdf:datatype="&xsd;string">32x32</rdf:first>
                                    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                                </rdf:Description>
                            </rdf:rest>
                        </rdf:Description>
                    </rdf:rest>
                </rdf:Description>
            </owl:oneOf>
        </rdfs:Datatype>
    </rdfs:range>
</owl:DatatypeProperty>
```

FIGURE 6.11 – restriction sur la résolution des vidéos de Twitter.

# Chapitre 7

# Extensibilité des structures de données (BDR et OWL)

Les besoins que nous avons décrits dans la partie 1 de ce travail mettent un accent sur la flexibilité et l'extensibilité de la plateforme future que nous sommes censés développer. Du point de vu de la modélisation, la question à se poser est : une BDRs et une OWL peuvent-elles être extensible en recevant une intégration automatique de nouvelles données ?

Dans la suite nous allons répondre à cette question en rappelant d'abord les fonctionnalités pouvant être extensibles. En parcourant quelques littératures, nous observons que, de façon générale, pour étendre une structure de données il faut prendre en compte deux possibilités : «une modification de données» et/ou «une modification de la structure». En y ajoutant nos expériences personnelles sur la manipulation de données, nous avons reformulé ces deux possibilités pour qu'elles expriment le mieux que possible notre contexte de travail. Cette reformulation nous a amené à distinguer deux situations, à savoir :

- Situation 1 : modification de données sans modification de la structure du schéma de base.
- Situation 2 : modification de données avec modification de la structure du schéma de base.

Par la suite, pour chacune des formes de modélisation (BDR ou OWL), et pour chacune des situations, nous allons voir si une BDR et OWL peuvent accepter facilement de nouvelles informations. Cette démarche va se faire en s'appuyant sur des exemples précis. Ces exemples peuvent être appliqués dans un contexte de PFCs existant ou personnalisés.

## 7.1 Les besoins pouvant être extensibles.

Nous rappelons que l'objectif général de notre travail est d'étudier comment concevoir une plateforme permettant de publier du contenu sur les réseaux sociaux de manière collaborative. Nous avons commencé notre travail en décrivant l'ensemble des besoins que doivent satisfaire une telle plateforme. Parmi ces besoins il y a «l'extensibilité et la flexibilité» qui se décrit concrètement en ces points :

- L'ajout d'une nouvelle information à un réseau social : admettons que le réseau social Facebook a maintenant trois champs : un champ TITRE qui contient le titre du message a publié, un champ CONTENU qui contient le message lui-même et un champ PHOTO qui permet d'ajouter une photo au message avant de publier. Si dans le futur on souhaiterait ajouter à Facebook un champ REFERENCE qui contient un lien de référence, notre plateforme devra l'intégrer automatiquement avec les traitements qui lui sont propres (est-ce un texte, une image, une vidéo, un audio, etc.).
- L'ajout d'un nouveau réseau social : imaginons que nous avons une plateforme avec Facebook et nous souhaitons y ajouter Instagram avec deux comptes. On aimerait que notre plateforme collaborative puisse être étendue automatiquement en intégrant ce nouveau réseau social, même s'il viendrait avec de nouveaux champs comme

QRCode (authentification via une image).

Notre application se veut extensible au niveau de la modélisation parce qu'elle peut recevoir de nouveaux réseaux sociaux ou de nouveaux champs. Ces champs doivent s'adapter facilement avec la structure de données existante. Pour savoir comment modéliser notre application, nous allons étudier les différentes formes de modélisations (BDR et OWL) qui ont été relevé dans le chapitre 4. En fait, notre démarche consiste à analyser chaque forme de modélisation que nous avons trouvée dans la littérature et qui sont liées au PFC en nous appuyant sur les deux situations que nous avons énumérées plus haut.

## 7.2 Extensibilité d'une BDR

Un schéma BDR est automatiquement extensible s'il est capable de recevoir dynamiquement de nouvelles relations.

D'après [13] et selon nos expériences personnelles (plus de 3 ans) en tant que développeur d'applications tournant sur des schémas BDR, nous pouvons dire, sans risque de nous tromper, que les BDRs sont extensibles. Cette extensibilité est simple dans certains cas, notamment si c'est l'ajout de quelques lignes dans une ou plusieurs tables, car cela ne modifie pas la structure initiale de la base de données : on dit qui y a **extension de données**. Par contre, quand la structure change, on dit qu'il y a **extension de structure**. Dans ce cas l'extension est plus complexe à réaliser de façon automatique. En fait, il faut vérifier, parfois, beaucoup de contraintes notamment les contraintes de référencement, d'unicité, de « null » et respecter une certaine cohérence dans la structure de base [13]. Toutes ces vérifications sont souvent fastidieuses, crée aussi des pertes de temps. Il peut également avoir des actions qui provoquent une **extension hybride**, en modifiant en même temps les données et la structure du schéma. Par l'exemple, si on souhaite ajouter une nouvelle table à un schéma de BDR initiale. Les BDRs utilisent plusieurs mécanismes, notamment les procédures stockées (PS), les «trigger». Ces mécanismes sont très pratiques en ce sens qu'elles permettent de faciliter l'automatisation de plusieurs sortes de vérifications de données et de rendre dynamique l'extensibilité de la structure de données. Elles peuvent permettre par exemple de s'assurer que certaines relations n'existaient pas avant de les créer, empêchant ainsi des exceptions inutiles. Les PS peuvent être exécutées automatiquement par des applications externes, écrites en JAVA, C#, PHP ou d'autres.

## 7.3 Extensibilité d'OWL

Dans toute cette section on va se référer sur[15] pour la compréhension des «requêtes SPARQL». Considérons un graphe d'OWL avec ses classes, ses relations et ses types de données. Ce graphe est extensible s'il est capable de recevoir dynamiquement de nouvelles classes, relations ou types de données. C'est ce que nous allons montrer dans cette section en explorant les deux situations que nous avons énumérées plus haut. Notre domaine de connaissance de base est le suivant :

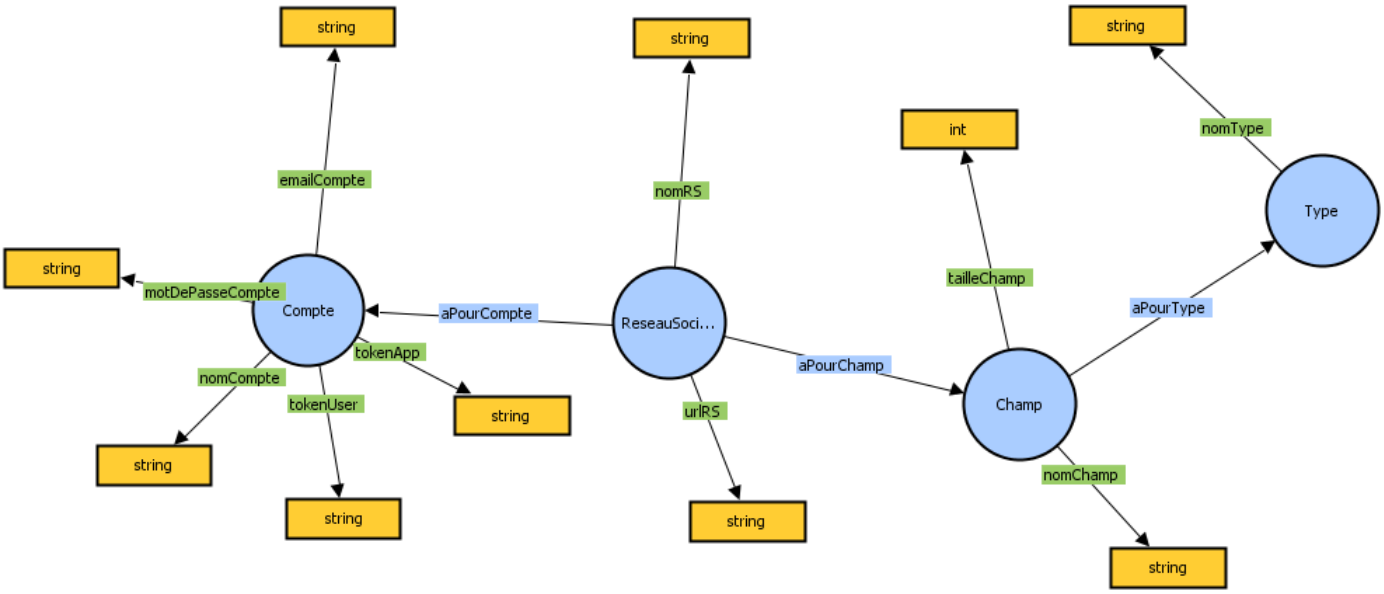



FIGURE 7.1 – domaine de connaissance de base.

Notre domaine de connaissance ci-dessus qui va servir à modéliser les 2 situations est constitué de :

- 4 classes : ReseauSocial, Champ, Compte et Type.
- 3 relations : « aPourchamp », « aPourCompte » et « aPourType ».
- 10 propriétés : « nomCompte », « emailCompte », « motDePasseCompte », « tokenUser », « tokenApp », « nomRS », « urlRS », « nomChamp » et « nomType » qui sont des chaînes de caractères. Et « tailleChamp » de type entier.

Ce domaine de connaissance se lit de la manière suivante : un réseau social, composé de 2 propriétés, a un champ ou plusieurs champs, composés eux aussi de 2 propriétés et possède un compte, composé de 5 propriétés. Le champ possède un et un seul type composé d'une seule propriété. Les informations de base, relatives au domaine précédant, sont représentées ainsi dans notre document OWL :

```
:Bolani  a                owl:NamedIndividual , :Compte ;
        :emailCompte      "bolani@unamur.be" ;
        :motDePasseCompte "mmmmmm" ;
        :nomCompte        "bolani" ;
        :tokenApp         "pppp" ;
        :tokenUser        "kkkk" .

:Facebook  a          :ReseauSocial , owl:NamedIndividual ;
        :aPourChamp   :TitreFacebook ;
        :aPourCompte  :Bolani ;
        :nomRS        "facebook" ;
        :urlRS        "www.facebook.com" .

:TitreFacebook  a     owl:NamedIndividual , :Champ ;
        :aPourType    :Texte ;
        :nomChamp     "titre" ;
        :tailleChamp  200 .


:Texte  a         :Type , owl:NamedIndividual ;
        :nomType  "texte" .
```

FIGURE 7.2 – triplets se trouvant dans le document OWL

Les données se trouvant dans le document OWL nous montrent différents triplets et dans ces triplets nous trouvons :

- des relations : « Facebook » a pour champ « TitreFacebook », « TitreFacebook » a pour type « Texte », « Facebook » a pour compte « Bolani ».
- des instances de classes : « Facebook » est un Réseau Social, « TitreFacebook » est un Champ, « Texte » est un type, « Bolani » est un compte.
- des attributs d'instances : « Facebook » a un nom et une url, « TitreFacebook » a un nom et une taille, « Texte » a un nom, « Bolani » a un email, un nom, un mot de passe, un tokenUser et un tokenApp.

Nous avons créé notre ontologie avec le logiciel «Protégé». Nous avons également utilisé ce logiciel afin de vérifier la consistance de notre domaine. C'est-à-dire que le logiciel nous a permis de savoir si les relations entre nos classes et nos instances sont cohérentes. Dans la suite nous utilisons des fonctions pour l'insertion des données. Il s'agit des fonctions paramétrées dont les valeurs des variables sont connues lors de l'utilisation desdites fonctions. Maintenant nous allons voir comment réagit OWL face à l'extension du domaine de connaissance dans les deux situations suivantes :

**I. Situation 1 : modification de données sans modification de la structure du schéma de base.**

Dans cette situation il ne devrait pas avoir d'ajout de nouvelles classes, relations ou type de données. On s'attend à ce qu'il ait ajout de nouvelles données dans le domaine de connaissance existant. Cette situation correspond aux deux besoins que nous avons cités au début de ce chapitre : L'ajout d'un nouveau réseau social et l'ajout d'un nouveau champ à un réseau social.

**A) L'ajout d'un nouveau champ à un réseau social.**

Admettons que nous voulons ajouter un champ « photo » de type image au réseau social Facebook. Une fonction va se lancer du côté applicatif JAVA [1]. Il s'agit d'une fonction qui va utiliser les librairies JENA. Ces librairies intègrent les requêtes SPARQL afin de manipuler dynamiquement les données RDF. La fonction va exécuter les requêtes SPARQL qui suivent.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ns: <http://www.semanticweb.org/will/ontologies/2019/4/kwessi#>

INSERT {
    ns:Image rdf:type ns:Type, owl:NamedIndividual .
    ns:Image ns:nomType "image" .
} WHERE {
    FILTER NOT EXISTS {
        ns:Image rdf:type ns:Type .
  }
}
```

FIGURE 7.3 – requête SPARQL d'ajout d'un type de champ

1. Est un Framework Java gratuit et open source pour la construction de Web sémantique et d'applications de données liées

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ns: <http://www.semanticweb.org/will/ontologies/2019/4/kwessi#>

INSERT {
    ns:PhotoFacebook rdf:type ns:Champ, owl:NamedIndividual .
    ns:PhotoFacebook ns:aPourType ns:Image .
    ns:PhotoFacebook ns:nomChamp "photo" .
    ns:PhotoFacebook ns:tailleChamp 5 .
    ns:Facebook ns:aPourChamp ns:PhotoFacebook .
} WHERE {
    FILTER NOT EXISTS {
        ns:PhotoFacebook rdf:type ns:Champ .
  }
  FILTER EXISTS {
        ns:Facebook a ns:ReseauSocial .
        ns:Image a ns:Type .
  }
}
```

FIGURE 7.4 – requête SPARQL d'ajout d'un champ photo

D'abord la fonction va ajouter l'instance « Image » qui sera propre à différents types de champs. Durant cette opération d'ajout, il y aura une condition dans la clause WHERE qui va vérifier l'existence du type. Dans notre cas, l'instance « Image » n'existe pas donc il va être ajouté. Enfin, après avoir ajouté le nouveau type « Image » au document, nous allons pouvoir ajouter la nouvelle instance « PhotoFacebook » en vérifiant son existence grâce aux conditions qui se trouvent dans « FILTER NOT EXISTS » et « FILTER EXISTS ». La première condition vérifie que l'instance « PhotoFacebook » n'existe pas déjà. La deuxième condition vérifie l'existence de l'instance « Facebook ». Lors de l'ajout, 5 opérations d'insertions vont être exécutées :

- La première opération va créer l'instance « PhotoFacebook » de type Champ.
- La deuxième opération va ajouter la relation « PhotoFacebook aPourType Image».
- La troisième opération va initialiser l'attribut « nomChamp » de « PhotoFacebook ».
- La quatrième opération va initialiser l'attribut « tailleChamp » de « PhotoFacebook ».
- La cinquième opération va ajouter la relation « Facebook aPourChamp PhotoFacebook ».

Après avoir effectué toutes ces différentes requêtes, nous allons avoir ces nouveaux triplets dans le document OWL :

```
:Facebook  a          :ReseauSocial , owl:NamedIndividual ;
        :aPourChamp   :TitreFacebook , :PhotoFacebook ;
        :aPourCompte  :Bolani ;
        :nomRS        "facebook" ;
        :urlRS        "www.facebook.com" .

:TitreFacebook  a     :Champ , owl:NamedIndividual ;
        :aPourType    :Texte ;
        :nomChamp     "titre" ;
        :tailleChamp  200 .

:Texte  a         :Type , owl:NamedIndividual ;
        :nomType  "texte" .

:Image  a         :Type , owl:NamedIndividual ;
        :nomType  "image" .

:PhotoFacebook  a     :Champ , owl:NamedIndividual ;
        :aPourType    :Image ;
        :nomChamp     "photo" ;
        :tailleChamp  5 .
```

FIGURE 7.5 – ajout d'un nouveau triplet de champs dans le document OWL

Nous constatons l'ajout du champ « PhotoFacebook » avec ses propriétés, l'ajout du type « Image » et sa propriété, l'ajout de la relation « Facebook aPourChamp PhotoFacebook » dans le document.

**B) L'ajout d'un nouveau réseau social.**

Considérons les données obtenues dans la section précédente. Nous voulons ajouter automatiquement un nouveau réseau social nommé « Twitter » avec ses 2 comptes :« kankan » et « dipanda ».
Un compte a comme propriétés : « nomCompte », « emailCompte », « motDePasseCompte », « tokenUser » et « tokenApp ».
Le réseau social twitter a comme propriété : « nomRS » et « urlRS » et il vient avec les champs suivants :

- titre de type « texte » et de taille maximale 200 caractères.
- photo de type « image » et de taille maximale 4 Mo.
- profil de type « image » et taille maximale 3 Mo.

Dans un premier temps, une autre fonction JAVA va faire appel à la fonction « ajout champ » afin d'insérer les nouveaux champs de Twitter : «PhotoTwitter», «TitreTwitter» et «ProfilTwitter». Comme pour les champs de Facebook, nous avons décidé d'ajouter le nom du réseau social devant chaque nom de champ parce que les réseaux sociaux n'ont pas forcément les mêmes contraintes aux niveaux de leurs champs. Par exemple, Facebook accepte une photo de taille maximale égale à 5Mo et Twitter accepte une photo de taille maximale égale à 4Mo. Nous avons le champ titre qui possède la même taille chez Twitter et Facebook mais cette taille peut varier en fonction des mises à jour de chaque réseau social.
Ensuite, la fonction va vérifier l'existence du réseau social Twitter grâce aux conditions se trouvant dans «FILTER NOT EXISTS» et «FILTER EXISTS». Puis la fonction va pouvoir exécuter la requête pour l'ajout du réseau social Twitter avec ses champs «titre» de type texte, « photo» de type image et « profil » de type image. Enfin, la fonction va faire appel à la fonction ajout compte, vu que le réseau social Twitter vient avec 2 comptes. La requête d'ajout d'un nouveau réseau social se présente ainsi :

```
INSERT {
    ns:Twitter rdf:type ns:ReseauSocial, owl:NamedIndividual .
    ns:Twitter ns:aPourchamp ns:PhotoTwitter .
    ns:Twitter ns:aPourchamp ns:TitreTwitter .
    ns:Twitter ns:aPourchamp ns:ProfilTwitter .
    ns:Twitter ns:nomRS "twitter" .
    ns:Twitter ns:urlRS "www.twitter.com" .
} WHERE {
  FILTER NOT EXISTS {
        ns:Twitter rdf:type ns:ReseauSociaux
  }
  FILTER EXISTS {
        ns:PhotoTwitter a ns:Champ .
        ns:TitreTwitter a ns:Champ .
        ns:ProfilTwitter a ns:Champ .
  }
}
```

FIGURE 7.6 – ajout du Réseau Social twitter

Grâce à la fonction d'ajout d'un champ, Les instances «TitreTwitter», «PhotoTwitter» et «ProfilTwitter» vont être créées et ajoutées. Ensuite, nous allons créer le réseau social «Twitter» et l'associer à la classe «ReseauSocial». Puis nous allons ajouter les relations entre Twitter et ses différents champs. Enfin nous allons créer les valeurs nomRS «twitter» et urlRS «www.twitter.com». Après avoir insérer le réseau social Twitter, la fonction JAVA va continuer son traitement et ajouter les 2 comptes liés à Twitter : «kankan» et «dipanda». La requête d'ajout d'un compte se présente ainsi :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ns: <http://www.semanticweb.org/will/ontologies/2019/4/kwessi#>

INSERT {
    ns:Kankan rdf:type ns:Compte, owl:NamedIndividual .
    ns:Kankan ns:nomCompte "kankan" .
    ns:Kankan ns:emailCompte "kankan@unamur.be" .
    ns:Kankan ns:motDePasseCompte "aaaaaa" .
    ns:Kankan ns:tokenUser "yyyy" .
    ns:Kankan ns:tokenApp "xxxx" .
    ns:Twitter ns:aPourCompte ns:Kankan
} WHERE {
  FILTER NOT EXISTS {
        ns:Kankan a ns:Compte .
  }
  FILTER EXISTS {
        ns:Twitter a ns:ReseauSocial .
  }
}
```

FIGURE 7.7 – ajout d'un compte

Nous avons montré l'insertion du compte «Kankan» qui va ressembler à celui de l'insertion du compte «Dipanda». Comme dans les insertions précédentes, l'opération va vérifier l'existence du réseau social « Twitter » et du compte « Kankan ». Les données concernant le mot de passe et les tokens vont être cryptés et non visible. Au final, nous allons avoir de nouvelles données dans le document OWL :

```
:Twitter  a           :ReseauSocial , owl:NamedIndividual ;
        :aPourCompte  :Kankan , :Dipanda ;
        :aPourchamp   :PhotoTwitter , :ProfilTwitter , :TitreTwitter ;
        :nomRS        "twitter" ;
        :urlRS        "www.twitter.com" .

:PhotoTwitter  a      :Champ , owl:NamedIndividual ;
        :aPourType    :Image ;
        :nomChamp     "photo" ;
        :tailleChamp  4 .

:ProfilTwitter  a     :Champ , owl:NamedIndividual ;
        :aPourType    :Image ;
        :nomChamp     "profil" ;
        :tailleChamp  3 .

:TitreTwitter  a      :Champ , owl:NamedIndividual ;
        :aPourType    :Texte ;
        :nomChamp     "titre" ;
        :tailleChamp  200 .

:Kankan  a                :Compte , owl:NamedIndividual ;
        :emailCompte      "kankan@unamur.be" ;
        :motDePasseCompte "aaaaaa" ;
        :nomCompte        "kankan" ;
        :tokenApp         "xxxx" ;
        :tokenUser        "yyyy" .

:Dipanda  a               :Compte , owl:NamedIndividual ;
        :emailCompte      "dipanda@unamur.be" ;
        :motDePasseCompte "bbbbbb" ;
        :nomCompte        "dipanda" ;
        :tokenApp         "vvvv" ;
        :tokenUser        "zzzz" .
```

FIGURE 7.8 – ajout des nouveaux triplets dans le document (Twitter, PhotoTwitter, ProfilTwitter, TitreTwitter, Kankan, Dipanda)

**II. Situation 2 : modification de données avec modification de la structure du schéma de base.**

L'extension d'OWL peut se faire en ajoutant une nouvelle classe, une nouvelle relation ou des nouvelles propriétés. Nous allons voir comment OWL réagit lorsqu'on introduit de nouvelles relations. Comment on peut en tirer profit en utilisant le mécanisme d'inférence. Pour cela, nous allons prendre l'exemple de la relation qui peut avoir entre le champ photo de Facebook et la résolution. La nouvelle relation signifie qu'un champ peut avoir plusieurs résolutions.

Dans un premier temps nous allons construire une nouvelle relation entre un champ et un réseau social avec des règles d'inférences. Avant il y avait une relation « un réseau social a des champs ». Maintenant nous voulons préciser, par déduction inférentielle, qu'un « champ est lié à un réseau social » (relation inverse). Nous allons utiliser l'opération « CONSTRUCT » de SPARQL qui permet de construire de nouveaux triplets non encore insérés dans le document OWL existant. À quoi cela sert-il ? Tout simplement pour réutiliser l'opération dans un autre contexte. L'opération se présente ainsi :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ns: <http://www.semanticweb.org/will/ontologies/2019/4/kwessi#>

CONSTRUCT {
  ns:estUnChamp  a   owl:ObjectProperty .
  ns:estUnChamp rdfs:domain  ns:Champ .
  ns:estUnChamp rdfs:range   ns:ReseauSocial .
  ?x ns:estUnChamp ?y .
} WHERE {
  ?x a ns:Champ .
  ?y a ns:ReseauSocial .
  ?y ns:nomRS "facebook" .
  ?y ns:aPourChamp ?x .
  FILTER EXISTS {
     ns:Champ a owl:Class .
     ns:ReseauSocial a owl:Class .
  }
}
```

QUERY RESULTS

Table | Raw Response

```
1  @prefix :      <http://www.semanticweb.org/will/ontologies/2019/4/kwessi#> .
2  @prefix ns:    <http://www.semanticweb.org/will/ontologies/2019/4/kwessi#> .
3  @prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix owl:   <http://www.w3.org/2002/07/owl#> .
5  @prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
6  @prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
7
8  ns:PhotoFacebook  ns:estUnChamp  ns:Facebook .
9
10 ns:estUnChamp  a     owl:ObjectProperty ;
11         rdfs:domain  ns:Champ ;
12         rdfs:range   ns:ReseauSocial .
13
14 ns:TitreFacebook  ns:estUnChamp  ns:Facebook .
15
```

FIGURE 7.9 – opération pour construire une nouvelle relation estUnChamp

Comme on peut le voir sur la figure précédente, nous allons d'abord définir la nouvelle relation que nous voulons construire « estUnChamp »(ObjectProperties), ensuite nous allons la relier avec 2 inconnus « ?x » et « ?y ». Si la classe « Champ » et la classe « ReseauSocial » existe, nous allons obtenir les inputs pour les insérer. N'oublions pas que le but va être d'ajouter de nouvelles propriétés au champ photo de Facebook. C'est pour cela que nous précisons le nom du réseau social sinon l'opération allait rajouter des triplets entre toutes les instances de champs et de réseaux sociaux. Par exemple, « ns :PhotoFacebook ns :estUnChamp ns :Twitter », ce qui signifierait une incohérence de données. Nous rappelons que les instructions dans la clause WHERE sont les conditions (propositions d'entrées) et les instructions dans le CONSTRUCT sont la conclusion (proposition de sortie). Après avoir obtenu le résultat ci-dessus, notre fonction JAVA va pouvoir l'insérer dans le document avec l'opération INSERT. Cela va ressembler aux différents cas montrés plus haut. Notre document OWL va avoir ceci comme nouvelles données :

```
:estUnChamp   a          owl:ObjectProperty ;
          rdfs:domain    :Champ ;
          rdfs:range     :ReseauSocial .

:TitreFacebook   a       :Champ , owl:NamedIndividual ;
          :aPourType     :Texte ;
          :estUnChamp    :Facebook ;
          :nomChamp      "titre" ;
          :tailleChamp   200 .

:PhotoFacebook   a       :Champ , owl:NamedIndividual ;
          :aPourType     :Image ;
          :estUnChamp    :Facebook ;
          :nomChamp      "photo" ;
          :tailleChamp   5 .
```

FIGURE 7.10 – ajout de la nouvelle relation "estUnChamp"

Ensuite, nous allons construire notre 2ème règle d'inférence avec l'opération « CONSTRUCT », cette opération va nous permettre de construire la nouvelle relation (qui va être une DataProperties). Cette opération va pouvoir être réutilisée si l'on veut ajouter une nouvelle « DataProperties » à un champ. Dans notre cas, c'est le champ photo de Facebook. Voici l'opération :

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ns: <http://www.semanticweb.org/will/ontologies/2019/4/kwessi#>

CONSTRUCT {
  ns:resolution  a  owl:DatatypeProperty .
  ns:resolution     rdfs:domain  ns:Champ .
  ns:resolution      rdfs:range   xsd:string .
  ?x ns:resolution "700x1024" .
} WHERE {
  ?x ns:nomChamp "photo" .
  ?x ns:estUnChamp ns:Facebook .
  FILTER NOT EXISTS {
       ns:resolution a  owl:DatatypeProperty .
  }
  FILTER EXISTS {
       ?x a ns:Champ .
  }
}
```

QUERY RESULTS

Table | Raw Response

```
1  @prefix :      <http://www.semanticweb.org/will/ontologies/2019/4/kwessi#> .
2  @prefix ns:    <http://www.semanticweb.org/will/ontologies/2019/4/kwessi#> .
3  @prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix owl:   <http://www.w3.org/2002/07/owl#> .
5  @prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
6  @prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
7
8  ns:PhotoFacebook  ns:resolution  "700x1024" .
9
10 ns:resolution  a     owl:DatatypeProperty ;
11         rdfs:domain  ns:Champ ;
12         rdfs:range   xsd:string .
13
```

FIGURE 7.11 – opération pour construire la nouvelle relation (DataProperties) resolution.

Dans cette opération, nous avons créé la nouvelle DataProperties « resolution ». Nous allons dire que c'est une relation propre à un Champ avec la valeur « 700x1024 ». Mais avant cette insertion, des conditions vont être définies :

- Le nom du champ concerné est photo.
- La relation « estUnChamp » concerne PhotoFacebook et Facebook.
- Le filtre va vérifier si résolution existe déjà. Si oui, les nouveaux triplets ne vont pas être construits.
- Le filtre va vérifier l'existence de la classe Champ.

Avec le résultat obtenu, la fonction JAVA va insérer ces nouveaux triplets dans le document OWL. Voici le résultat final :

```
:resolution  a        owl:DatatypeProperty ;
        rdfs:domain   :Champ ;
        rdfs:range    xsd:string .

:PhotoFacebook  a     :Champ , owl:NamedIndividual ;
        :aPourType    :Image ;
        :estUnChamp   :Facebook ;
        :nomChamp     "photo" ;
        :resolution   "700x1024" ;
        :tailleChamp  5 .
```

FIGURE 7.12 – ajout du DataProperties "resolution"

Nous n'oublions pas que le but de cette situation est de montrer l'extension de la structure du domaine de connaissance. Cette structure se présenter ainsi :

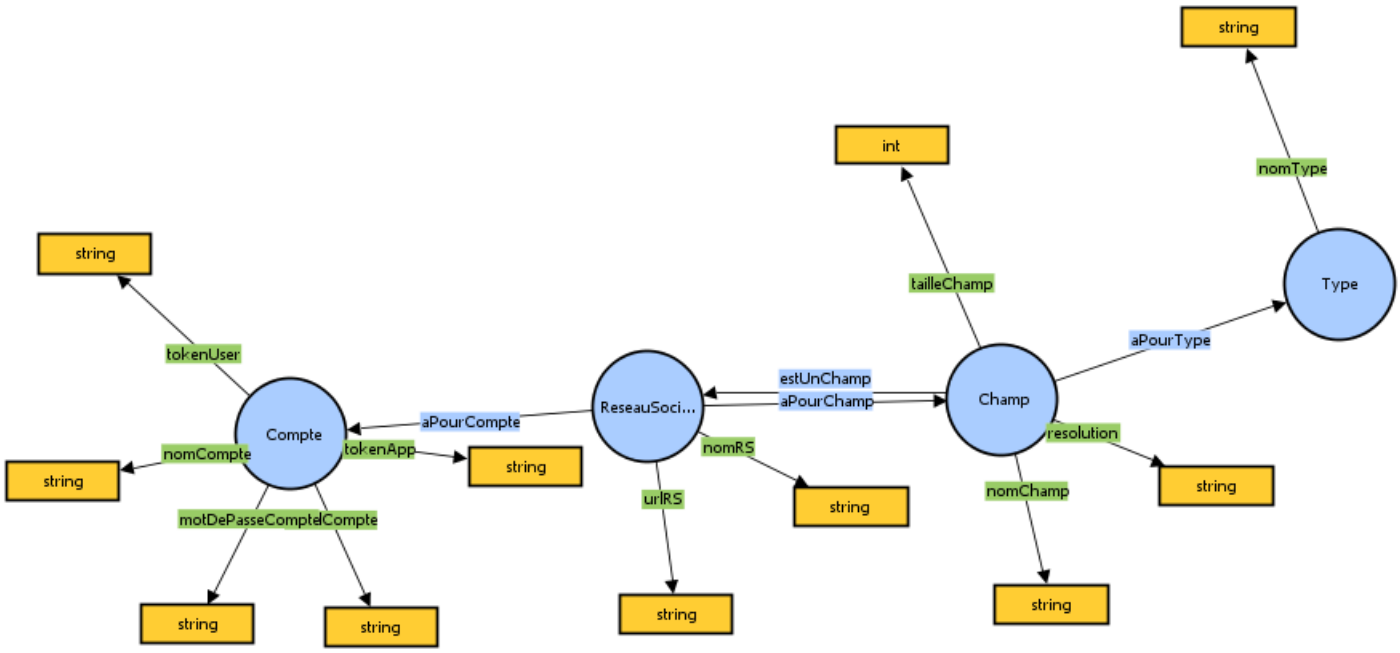



FIGURE 7.13 – nouvelle structure du graphe avec estUnChamp et resolution.

L'extension de structure s'est fait par l'ajout de nouvelles relations :

- estUnChamp
- resolution

### 7.3.1 Discussion sur les possibilités d'extension d'OWL

Nous constatons après des études que nous venons de présenter que l'OWL est extensible. Cette extensibilité se passe très bien pour les 2 situations. Dans la situation 1 «modification de données sans modification de la structure du schéma de base», il s'agit d'une extension de données lors de l'ajout d'un champ ou d'un réseau social. Dans la situation 2 «modification de données avec modification de la structure du schéma de base», Il s'agit d'une extension de structure et de données avec l'ajout de la relation "estUnChamp" et de la propriété de donnée "resolution". OWL utilise des URIs pour identifier ses ressources contrairement aux bases de données qui utilisent les clés primaires. Avec les différentes opérations comme «FILTER», OWL peut facilement filtrer ses informations soit pour insérer de nouvelles données, soit pour rechercher des informations. Enfin, nous avons utilisé l'opération «CONSTRUCT» qui nous permet

de construire de nouveaux triplets dans le domaine de connaissance en appliquant quelques règles d'inférences propre à notre domaine. Ces opérations vont pouvoir être réutilisées dans d'autres situations ce qui va être fort pratique. Nous avons vu la capacité d'OWL à insérer de nouvelles propriétés de manière efficace et nul doute qu'il va pouvoir répondre aux exigences citées dans le chapitre de description des besoins. Pour écrire nos exemples, nous avons utilisé le serveur apache FUSEKI qui tourne avec les librairies Jena (à voir plus loin).

## 7.4 Appréciation BDR et OWL

Notre critère d'appréciation est : ***la flexibilité et l'extensibilité***. La structure de données qui semble s'adapter le mieux avec ces deux critères est OWL. Notre plateforme collaborative doit pouvoir intégrer de nouveaux réseaux sociaux dans le souci d'apporter plus de possibilités de communications. Sachant qu'un réseau social vient avec ses propres champs et exigences, OWL va mieux l'intégrer dans la structure de données de base que les BDRs. Avec les BDRs il faut tenir compte parfois de beaucoup de contraintes notamment les contraintes de référencement, d'unicité, de «null» et respecter une certaine cohérence dans la structure de base. Il y existe régulièrement un couplage très fort entre deux ou plusieurs tables, ce couplage est matérialisé par les clés secondaires. Si ce couplage n'est pas fait ou mal fait, on pourrait ne pas avoir accès à certaines informations. OWL est très souple, les relations entre les objets sont basées la plupart de temps sur des descriptions logiques. Pas besoin de gérer soi-même les cardinalités telle qu'on le fait dans les BDRs. OWL peut même déduire certaine relation en utilisant les moteurs d'inférence. Par exemple, si on dit qu'un homme est une personne et qu'un père est un homme, OWL peut déduire qu'un père est aussi un homme alors que dans les BDRs il faut définir soi-même les différentes relations.

Avec les BDRs, certaines situations peuvent être difficiles à normaliser et très compliquer à maintenir. Par exemple, les relations d'amitié sur Facebook. Une petite analyse de ces relations montre que l'amitié sur Facebook obéit à un schéma de type graphe (avec la réciprocité, la transitivité, etc.). Il est possible de modéliser cela avec les BDRs. Mais la maintenance deviendra de plus en plus compliquer au fur et à mesure qu'une personne aura d'autres personnes comme ami, et si plusieurs personnes ont de plus en plus des amis en commun. OWL accepte des situations beaucoup plus complexes que peuvent faire les BDRs. Les propriétés de l'OWL ont plus de sémantiques que les types de données des bases de données relationnelles. Les ontologies n'ont pas besoins d'être normalisés comme les BDRs.[19].

Vu les imperfections des BDRs dans la réalisation des besoins que notre mémoire doit couvrir, dans la suite de ce travail nous parlerons uniquement de OWL, qui nous servira pour structurer nos données.

# Chapitre 8

# Techniques pour étendre notre plateforme collaborative (PFC)

La question que nous voulons répondre dans ce chapitre est : ***comment étendre dynamiquement le code source et le schéma OWL du Framework de base de notre PFC ?*** Il faudrait voir cette question sur deux angles : le code source du Framework de base et le schéma OWL du Framework de base. Cela est dû au fait que nous pouvons étendre le schéma sans étendre le code source, vice-versa. Ainsi, trois cas s'offrent à nous avec chacun ses avantages et ses inconvénients. Ces cas sont les suivants :

- Extension par fonction,
- Extension par Plugins,
- Extension par importation ou merge.

## 8.1 Extension par fonction

Il est question d'ajouter un RS avec ses champs via une interface utilisateur intégrer dans l'application globale. D'après un prototype que nous avons fait, cela est possible, l'ajout du RS et ses champs se fait très bien. Par contre si l'on veut prévoir l'ajout des restrictions sur des champs, «l'ajout par fonctionnalités» devient de plus en plus limiter au fur et à mesure que les restrictions augmentent et sont variées. Il faudrait prévoir toutes les possibilités, ce qui est quasi impossible car nous ne savons pas ce qu'un RS pourrait ajouter comme restrictions à un moment donné. De plus, il faudrait lier chaque restriction à son champ de telle sorte qu'une restriction sur un champ de Facebook ne se trouve pas par hasard sur un champ de Twitter. Nous dénombrons, parfois, plus d'une cinquantaine de restrictions par champ et les mettre sur notre interface la rendrait superflue, touffue, incompressible, pas du tout ergonomique. Cette méthode laisse également peu de place à la flexibilité. Avec cette technique la possibilité de modifier manuellement le code source de l'application ou le schéma de la structure de données est très grande, ce que nous voulons éviter.

Dans notre prototype, nous avons commencé par ajouter un réseau social. Après on clique sur « suivant » pour ajouter ses champs. À partir de ce moment les choses se compliquent car pour chaque champ, il faudrait ajouter l'ensemble de ses restrictions. Certain champ peuvent avoir au minimum 10 restrictions.

## 8.2 Extension par Plugins

Ici nous distinguons 2 sous cas.

**Sous cas 1** : le plugin vient avec un réseau social (RS) et une interface pour la gérer (installer, stopper, désinstaller).
Imaginons que notre application contient déjà le réseau social "TWITTER" et "FACEBOOK". Nous voulons rajouter

un nouveau réseau social "INSTAGRAM". Le plugin d'ajout de réseau social doit venir avec les informations concernant "INSTAGRAM", notamment ses champs et les restrictions sur ses champs. Cette solution présente les mêmes avantages que l'importation (expliquée plus bas) à la seule différence qu'elle n'accepte qu'un réseau social par plugin. Si l'on veut avoir 20 réseaux sociaux (par exemple) dans la plateforme collaborative, cela demande d'installer 20 plugins à raison d'un plugin par RS, ce qui peut être une fastidieuse tâche. Le fait que chaque plugin vient avec son interface est encombrant pour l'application englobante.

**Sous cas 2** : le plugin vient avec une interface permettant d'ajouter n'importe quel type de réseau social.
On se retrouve presque dans le cas d'ajout par fonctionnalité, à la seule différence que le plugin fonction de manière dynamique sans modification manuelle du code source. La conception d'un tel plugin s'avère compliqué car au moment de sa conception, il est difficile d'anticiper sur les champs ou les restrictions qui seront créées dans le futur.

Dans tous les sous cas, l'architecture du Framework de base de l'application doit être conçue de manière à accepter automatiquement l'installation, l'arrêt, et la désinstallation des plugins. Cette tâche n'est pas simple et requière des compétences pointues en architecture d'application. Très souvent une architecture d'application qui accepte les plugins est faite pour des applications qui nécessite beaucoup de points de variations (ou d'extension) dans le but d'enrichir le Framework de base par de nouvelles fonctionnalités. Elle peut alors accepter un plugin pour la gestion du planning, un autre pour faire du e-commerce, ou pour le tchat, etc. Dans notre cas, nous avons un seul point d'extension (ajout de nouveaux réseaux sociaux), cela nous semble peu pour mettre sur pied une telle architecture. Opter pour cette possibilité s'apparenterait comme «utiliser un fusil pour tuer une mouche». C'est-à-dire qu'il s'agit d'une solution faisable mais très peu adapter au besoin. La responsabilité de la rédaction du code source du plugin qui viendra étendre le code source de l'application de base est laissée à celui qui écrira le plugin, ce qui retire une tâche importante à celui qui conçoit le Framework de base. De plus, le plugin qui vient avec son propre code pourra intégrer dynamiquement le Framework de base sans modifier son code source, ce qui nous convient.

## 8.3 Extension par importation (ou merge ou fusion)

Dans le chapitre 7, nous avons expliqué comment on peut étendre une base de connaissances (BDC) en utilisant principalement les requêtes SPARQL. À ce niveau, le but recherché était simplement de montrer qu'une ontologie est extensible. Nous l'avons fait en prouvant qu'il est possible d'ajouter un réseau social (RS) ou un champ à une BDC via les requêtes SPARQL.

Il est aussi possible d'ajouter un RS avec ses champs (y compris les contraintes sur ces champs) par l'importation d'un fichier OWL pour étendre le schéma OWL du Framework de base. Pour cela, le développeur devra avoir une connaissance technique dans la conception d'un fichier OWL. Il devra aussi, pour le réseau social qu'il veut ajouter, connaitre non seulement l'ensemble des champs qu'il possède dans le cadre de la publication de contenus, mais aussi, répertorier les restrictions qu'il y a sur chaque champ. Le fichier OWL à importer peut contenir plusieurs réseaux sociaux. Ce fichier devra respecter certaines règles de validation pour bien s'intégrer dans le fichier de base (par exemple les règles de nommage, de doublon). Un exemple pour un fichier qui ajoute Twitter est :

**Schéma de base**
SocialNework -> hasAcount -> Account

**Schéma à importer**
Twitter -> hasTwitterAccount -> TwitterAccount
Twitter -> subclassOf -> SocialNework
TwitterAccount -> subclassOf -> Account

**Schéma final**
SocialNework -> hasAcount -> Account

Twitter ->subclassOf -> SocialNework
hasTwitterAccount -> subPropertyOf -> hasAcount
TwitterAccount -> subclassOf -> Account
Twitter -> hasTwitterAccount -> TwitterAccount

Cette solution s'avère meilleur que celle par "fonctionnalité" ou par plugin. En fait, on n'a plus besoin d'intégrer dans l'application une interface pour l'ajout d'un réseau social, ses champs et ses restrictions. Cette responsabilité est délocalisée à celui qui élaborera le fichier à importer. L'import se fait via un traitement automatique sans modification manuelle du code source pour la partie schéma de la structure de données. Les difficultés principales à gérer ici sont : la détermination de toutes les règles de validation du fichier à importer, l'intégration automatique des fonctionnalités (la rédaction, la publication de messages) nécessaires à la manipulation du nouveau RS ajouté dans le code source du Framework de base.

Dans le cadre de ce mémoire, nous avons pu élaborer toutes les règles de validation du fichier à importer et les quelques tests que nous avons faits se sont bien passés. Le schéma OWL importé s'intègre parfaitement dans le schéma OWL du Framework de base sans aucune inconsistance de données. Par contre, l'intégration automatique des fonctionnalités nécessaires à la manipulation du nouveau RS dans le code source du Framework de base n'est presque pas possible. Pour ajouter de nouveaux RS on doit absolument modifier manuellement le code source puis redéployer tout l'application pour qu'elle prenne en compte ces ajouts.

La technique par importation (ou merge) correspond en partie à notre travail. Elle nous apporte au moins une extension automatique de schéma de base.

## 8.4 Technique retenue

Prise individuellement, aucune des techniques présentées plus haut ne correspond à nos attentes. Néanmoins il y a moyen de les combiner pour obtenir un résultat plus intéressant. Ainsi nous allons combiner la technique d'importation (ou merge) pour étendre le schéma de base et la technique de plugin pour étendre le code source. En plus claire, nous mettrons dans le dossier qui contient le code source du plugin le fichier OWL contenant les champs et les caractéristiques du RS à ajouter. Dès que le plugin va démarrer, il se chargera d'importer le fichier que nous avons mis pour étendre le schéma OWL du Framework de base. Cette tâche se fait sans une intervention manuelle. Le plugin va être installé dynamiquement sur le Framework de base sans que le code source du Framework soit modifié manuellement. La combinaison de ces deux techniques nous permet de répondre à notre question de départ. Étant donné qu'il existe des Frameworks aidant à développer les plugins, cela va nous faciliter la tâche pour la conception d'une architecture d'application acceptant les plugins.

# Chapitre 9

# Outils pour manipuler les ontologies et outils pour créer les plugins en JAVA

## 9.1 Outils pour faire le «merge» des ontologies

Il existe plusieurs outils permettant de «merger» les ontologies. Nous citons SAMBO, OWLDiff, PROMPT Suite, FCA-OntMerge [1], FCA-Merge [2] , FFCA [3]. Nous présenterons seulement les 3 premiers.

### 9.1.1 SAMBO

C'est régulièrement un système d'alignement et de fusion des ontologies biomédicales. Il est un outil s'appuyant particulièrement sur un Framework développé grâce à des stratégies applicables sur les ontologies du milieu biomédical. Il prend en entrée deux ontologies de type OWL. Le merge passe par une phase d'alignement comportant deux étapes : l'alignement des relations et l'alignement des concepts. Il «est semi-automatique puisqu'il possède un système générateur de suggestions qui nécessite l'intervention humaine pour sélectionner le comparateur désiré (comme WordNet, terminologies lexicales, hiérarchie). Pour chacune des propositions, l'utilisateur doit juger si les termes sont équivalents. Dans ce cas, un nouveau nom sera créé ; sinon, la suggestion sera rejetée...

Le processus de fusion de SAMBO présente cependant quelques limites. La première limite concerne l'intervention humaine pour la vérification des conflits. Cela existe également dans l'alignement avec la présence d'une liste de suggestions destinées à l'utilisateur (comme dans PROMPT). La deuxième limite concerne l'absence de traitement des super/sous-concepts c'est-à-dire que l'algorithme ne traite pas la notion des super/sous-concepts (en d'autres termes, l'abstraction des concepts). De plus, à la fin de l'algorithme de fusion, il existe une forme de copier-coller des termes, dans l'ontologie finale, qui n'ont pas été alignés, et cela se réalise sans aucune vérification et sans aucun traitement sémantique. Donc, cela peut influer sur le domaine cible de l'ontologie fusionnée et créer une incohérence au sein de cette ontologie en termes de concept et de relation. Par exemple, si SAMBO copie un concept non pertinent dans l'ontologie fusionnée, il y aura alors une incohérence dans l'ensemble des concepts.» [20].

### 9.1.2 OWLDiff

Il s'agit d'un outil intéressant pour la comparaison et la fusion des ontologies OWL. Il a été ajouté comme plug-in à Protégé et à NeOn Toolkit pour les enrichir en technologies de fusion et decomparaison. Le but de cet outil est d'enrichir la gestion des ontologies et surtout d'effectuer leurs mises à jour. Il est intéressant pour les utilisateurs qui modifient régulièrement les ontologies.

«OWLDiff utilise essentiellement un utilitaire appelé Diff qui permet de vérifier les changements syntaxiques et sémantiques des deux ontologies OWL entrées comme arguments. L'une de ses ontologies est nommée par exemple

1. Référence : https ://ieeexplore.ieee.org/abstract/document/5564899 (consulter 13/04/2020)
2. Référence : https ://dl.acm.org/doi/abs/10.5555/1642090.1642121 (consulter 13/04/2020)
3. Référence : https ://www.sciencedirect.com/science/article/abs/pii/S1568494610001432 (consulter 13/04/2020)

originale et l'autre mise à jour. Par la suite, l'outil OWLDiff exploite un raisonneur pour vérifier si les deux ontologies d'entrées sont similaires. Aussi, OWLDiff offre une interface graphique pour visualiser les différences dans le cas où les ontologies ne sont pas similaires. Ainsi, ces différences sont bien celles qui doivent être mises à jour dans l'ontologie fusionnée ... OWLDiff est un outil intéressant en termes de comparaison des ontologies et de détection des similarités entre les différents éléments d'une ontologie. Cependant, OWLDiff présente quelques limites en tant qu'outil de fusion des ontologies. Avec OWLDiff, la fusion se réalise avec une intervention majeure de l'utilisateur, ce qui remet en cause l'automatisation et l'intelligence de l'approche. En outre, comme avec PROMPT, il y a absence totale de traitement des abstractions des concepts (le cas échéant) et de traitement des concepts pertinents, ces deux notions étant très importantes pour garantir la pertinence de l'ontologie fusionnée. Aussi, OWLDiff n'exploite pas les relations interconcepts, ce qui diminue le niveau de l'analyse sémantique des ontologies.» [20].

### 9.1.3 PROMPT Suite

Il est constitué d'un ensemble de modules qui sont très importants dans les services d'alignement et de la fusion. «PROMPT Suite contient un outil de fusion des ontologies appelé iPROMPT, un outil appelé Anchor-PROMPT pour trouver les similarités entre les ontologies, un outil de comparaison des versions des ontologies appelées PROMPTDiff et un outil appelé PROMPTFactor qui permet de créer une ontologie factorisée ... La fusion avec PROMPT Suite est très cohérente en termes d'interaction entre les sous-modules faisant partie intégrante de cet outil, et eu égard à la richesse de l'algorithme (incluant la détection des incohérences et la proposition des solutions). Cependant, le module iPROMPT présente quelques limites : (1) la semi-automatisation de l'algorithme de fusion, puisqu'il inclut une étape de sélection des opérations exécutées par l'utilisateur, une étape clé du processus de fusion, (2) on retrouve le même défaut pour certaines confirmations des tâches, et par conséquent, iPROMPT ne possède pas l'intelligence suffisante pour exécuter certaines tâches, (3) iPROMPT prend en considération la structure de l'ontologie, mais ne prend pas en compte le traitement des relations entre les concepts ainsi que la pertinence des concepts ou la génération de nouvelles abstractions essentielles pour une meilleure structure ontologique.» [20].

Compte tenu des imperfections des outils de merge existants (trop d'interaction avec l'utilisateur, non gestion des super/sous-concepts, copie des concepts non pertinents dans l'ontologie fusionnée), nous allons développer notre propre algorithme de merge, en nous appuyant sur l'API Jena.

## 9.2 Outil pour la visualisation des ontologies

Il existe plusieurs outils de visualisation des ontologies [21]. Nous allons citer ici quelques-uns :

- WebVOWL (Web-based Visualization of Ontologies) [4]
- yWorks [5]
- OWLGrEd [6]
- OntoGraf [7]
- MEMO GRAPH [8][22]
- 7wData [9]
- OWLViz [10]

Nous allons nous limiter à l'énumération de ces outils. Pour avoir plus d'explications nous vous conseillons de lire les liens de référence. Dans ce travail nous avons utilisé principalement 2 outils : WebVOWL et OntoGraf. WebVOWL est adapté pour la visualisation des ontologies de petite taille (une cinquantaine de classes), nous l'avons utilisé pour un

4. Référence : http ://vowl.visualdataweb.org/webvowl.html (consulter le 09/05/2020)
5. Site internet : https ://www.yworks.com/use-case/visualizing-an-ontology (consulter le 09/05/2020)
6. Référence : http ://owlgred.lumii.lv/ (consulté le 09/05/2020)
7. Référence : https ://protegewiki.stanford.edu/wiki/OntoGraf (consulté le 09/05/2020)
8. Référence : https ://www.sciencedirect.com/science/article/pii/S1877050916319408 (consulté le 09/05/2020)
9. Référence https ://www.7wdata.be/data-science/web-based-visualization-of-ontologies/ (consulté le 09/05/2020)
10. Site internet : http ://www.co-ode.org/downloads/plugins-3.x.php (consulté 09/05/2020)

début. Il fait un affichage de tous les éléments de l'ontologie et c'est pour nous son point fort. Mais au fur et à mesure que la taille de notre ontologie augmentait, WebVOWL n'affichait plus tous nos objets, raisons pour laquelle nous avons changé vers OntoGraf. L'un des inconvénients avec OntoGraf est qu'il n'affiche pas les propriétés de données, ce que fait WebVOWL. Notre base de connaissances étant un peu grande, nous avons parfois utilisé OWLViz pour ne visualiser que les classes. La classification de certains outils se trouve dans [22] à la page 266.

## 9.3 Le système de gestion des ontologies

Comme les BDRs qui disposent des Systèmes de Gestion des Bases de Données Relationnelles (SGBDR), tels que : Oracle, MySQL, PostgreSQL ; les ontologies ont aussi leurs systèmes de gestion. Nous pouvons citer plusieurs notamment NeOn ToolKit [11] , SWOOP [12] , OWLGrEd [13] , Hozo, Protégé, etc. Ne pouvant pas tous les présenté nous nous attarderons sur les deux principaux.

### 9.3.1 Hozo

Hozo [14] est un outil pour construire des ontologies dans un environnement distribué. Il est développé en langage java. Il est un éditeur graphique d'ontologies spécialement créé pour produire des ontologies lourdes. Il a été développé au Japon grâce à un partenariat entre le Département des systèmes de connaissances (Laboratoire Mizoguchi), l'Université ISIR-Osaka et Enegate Co, Ltd. La dernière version, la 5.76 beta, est sortie le 07 octobre 2019 et est disponible en anglais et en japonais. Il contient trois principaux composants : éditeur d'ontologie, serveur d'ontologie et gestionnaire d'ontologie. Le gestionnaire d'ontologies gère des projets dans lesquels plusieurs ontologies sont construites en collaboration et dans un environnement distribué via Internet. Le serveur d'ontologie stocke les ontologies et les instances et fournit des API pour les clients. Les principales caractéristiques de Hozo sont résumées comme suit [20] :

- Il aide les utilisateurs à créer des ontologies avec des rôles d'une manière naturelle soutenue par la théorie avancée des rôles.
- Les informations sur l'héritage sont explicites et sont toujours accessibles. Deux modes d'héritage : l'une de super classes via « is-a » lien et l'autre de la contrainte de classe.
- Une interface graphique conviviale est disponible.
- La gestion des versions est disponible avec une fonction pour afficher les modifications.
- La construction d'ontologies dans un environnement distribué sur Internet est prise en charge.
- Des API sont disponibles pour accéder aux ontologies et aux instances de l'extérieur.
- En septembre 2011, il ne fonctionnait que sous les systèmes Mac, Windows.
- Il fonctionne avec la version 1.5.0_10 et plus du JVM java.

Le problème avec Hozo est le manque de documentation. Quand bien même on en trouve, elle est peu riche. La seule documentation que nous avons trouvée est [20].

### 9.3.2 Protégé

Protégé [15] est un logiciel développé en langage Java par l'université de Stanford. Il est gratuit et son code source est publié sous une licence libre, la Mozilla Public License ( BSD 2-clauses). Il tourne sur la machine virtuelle Java et est multiplateformes (linux, Windows, MacOS, ...). Protégé est capable de lire et sauvegarder des ontologies sous de nombreux formats d'ontologies notamment RDF, RDFS, OWL, Turtle, JSON-LD, etc. Sa dernière version, la 5.5.0, est disponible depuis le 14 mars 2019.

11. Site internet http ://neon-toolkit.org/wiki/Main_Page.html (consulté le 04/04/2020)
12. Site internet http ://www.mindswap.org/2004/SWOOP/ (consulté le 04/04/2020)
13. Site internet http ://owlgred.lumii.lv/ (consulté le 04/04/2020)
14. Site internet : http ://www.hozo.jp/ (consulté le 04/04/2020)
15. Site internet : https ://protege.stanford.edu/

Le logiciel permet de créer, modifier, supprimer ou «merger» les ontologies. Il possède des possibilités de «raisonner» une ontologie via des raisonneurs tels que HermiT, Ontop ou Pellet.

«Le logiciel Protégé figure parmi les meilleurs logiciels de gestion des ontologies existantes. L'efficacité de ce logiciel repose sur l'efficacité des outils qu'il intègre (comme PROMPT Suite présenté précédemment).» [23].

Protégé est le système que nous allons utiliser dans la suite de notre travail. Il est un logiciel open source, les plus utilisés et stable. Contrairement aux autres systèmes de gestion des ontologies, Il dispose d'une forte communauté très active, assez réactive aux questions qu'on peut se poser, ce qui fournit une aide importante à la prise en main de l'outil. En plus, avec Protégé on peut importer une ontologie sous forme de fichier XML, installer des plugins pour les ontologies. Protégé est fournie avec l'API Jena.

## 9.4 L' API OWL et l'API Jena

Dans nos recherches nous nous sommes intéressés à savoir comment manipuler un domaine de connaissance de l'extérieur. Pour manipuler un domaine de connaissance de l'extérieur nous avons trouvé l'API OWL et l'API Jena, qui sont deux librairies Java.

Nous n'avons pas trouvé dans nos recherches une comparaison entre les deux API. En utilisant l'API OWL, nous avons constaté que sa documentation est limitée, peu claire. Nous étions parfois contraints de poser nos questions sur les forums [16] et certaines réponses nous venaient deux à trois jours après. Nous y avons eu du mal à faire la relation intersectionOf (via les requêtes SPARQL). Avec l'API Jena nous avons trouvé plus de documentations. De plus, Jena nous a paru plus simple à utiliser, contrairement à l'API OWL. Dans la suite de ce mémoire nous utiliserons donc l'API Jena.

## 9.5 Jena

Pour toute cette section la source est : https ://jena.apache.org/documentation/ontology/ (lu le 10/04/2020). En fait, Jena vise à fournir une interface de programmation cohérente pour le développement d'applications d'ontologies, quel que soit le langage d'ontologie (OWL, RDF, RDFS, SWRL, etc.) qu'on utilise dans le programme. Jena est un ensemble d'outils dédiés à la construction d'applications orientées Web sémantique. Parmi ces outils, on trouve notamment une API Java open-source permettant de manipuler de nombreux langages tels que OWL, RDF/RDFS, SPARQL ou encore N3 et de raisonner sur des modèles ontologiques à l'aide de moteurs d'inférences inclus dans Jena ou externes.
En outre, Jena propose également des systèmes permettant d'assurer la persistance des modèles. On distingue deux systèmes :

- Jena SDB, un magasin de triplets utilisant une base de données relationnelle pour fonctionner ;
- Jena TDB, un système de stockage natif (c'est-à-dire qu'il utilise son propre système de stockage).

L'architecture permettant à Jena de raisonner sur une base de connaissances (BDC) se présente ainsi :

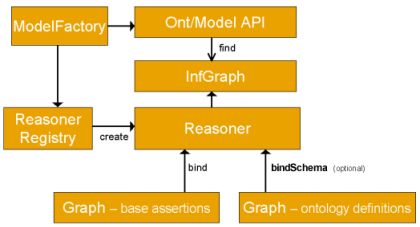



FIGURE 9.1 – architecture permettant à Jena de raisonner.

16. Exemple de forum pour l'API OWL : https ://github.com/owlcs/owlapi/wiki/Tutorial :-A-starter's-starter

Les applications accèdent normalement à la machine d'inférence en utilisant ModelFactory pour associer un ensemble de données à un groupe de raisonneurs afin de créer un nouveau modèle. Les requêtes adressées au modèle créé renverront non seulement les instructions présentes dans les données d'origine, mais également des instructions supplémentaires pouvant être déduites des données à l'aide des règles ou d'autres mécanismes d'inférence implémentés par le raisonneur.

Il existe plusieurs raisonneurs open sources notamment FaCT++, Pellet ou HermiT. Pellet est celui que nous allons utiliser parce qu'il serait l'un des plus rapide. En plus, Il est parmi les plus sollicités et les plus documentés.

## 9.6 Enrichir le domaine de connaissance avec l'API Jena

À titre de rappel, dans le chapitre 7, nous avons expliqué comment une base de connaissances (BDC) peut être étendue. Nous y avons également exposé les requêtes SPARQL utilisée pour ajouter de l'information dans une BDC, il s'agissait de la requête INSERT pour l'extension de données et la requête CONSTRUCT pour l'extension de structure.

Jena propose des services (ou méthodes) qu'on peut directement utiliser pour ajouter, modifier ou supprimer des données dans une ontologie OWL. Parmi ces services nous pouvons citer UNION, INTERSECTION, ADD. Par exemple, si l'ajout d'un réseau social vient avec la création de nouvelle classe dans la BDC, c'est une méthode ADD qui est appelée.

## 9.7 Outils pour concevoir les plugins en JAVA

Notre plateforme collaborative est une application «open source» développée en langage Java. Pour cette raison nous nous sommes demandé quels sont les outils Java pouvant nous aider à concevoir une architecture d'application capable d'accepter les plugins. Dans nos recherches nous avons trouvé (liste non exhaustive) :

- OSGI(Open Services Gateway initiative) [17]
- JSPF (Java Simple Plugin Framework) [18]
- JPF (Java Plug-in Framework) [19]
- API bukkit [20]
- PF4J (Plugin Framework for Java) [21]

Dans ce mémoire nous n'allons pas nous attarder sur tous ces outils. Si vous aimerez avoir plus d'explications nous vous conseillons de lire les liens de références. Compte tenu du temps très court qui nous restait, nous n'avons pas pu tester tous ces outils dans le but de les comparer et de choisir exactement celui qui correspond le mieux à nos besoins. En nous basant sur la documentation que nous avons lue sur chaque outil, nous avons choisi d'utiliser le Plugin Framework for Java (PF4J). La raison est simple. Nous avons trouvé dans sa documentation un exemple explicatif sur la manière de concevoir une architecture plugin à base de PF4J. En testant cet exemple cela correspondait à ce que nous voulons. Ainsi, nous n'avons plus poussé notre recherche plus loin en testant d'autres outils.

Avec PF4J, on peut facilement transformer une application java monolithique en une application modulaire. PF4J est un Framework de plug-in open source (licence Apache), léger (environ 100 Ko) pour java, avec des dépendances minimales (uniquement slf4j-api et java-server) et très extensible (voir PluginDescriptorFinder et ExtensionFinder). Il est un micro-Framework et son but est de garder le noyau simple mais extensible. Pour l'instant le Framework a les extensions suivantes :

- pf4j-update (mécanisme de mise à jour pour PF4J)

17. Site internet : https ://www.osgi.org/ (consulté le 09/05/2020)
18. Référence : https ://code.google.com/archive/p/jspf/ (consulté le 09/05/2020)
19. Référence : http ://jpf.sourceforge.net/ (consulté le 09/05/2020)
20. Référence : : https ://bukkit.gamepedia.com/Plugin_Tutorial/fr (consulté le 09/05/2020)
21. Référence : https ://pf4j.org/ (consulté le 09/05/2020)

- pf4j-spring (PF4J - intégration de Spring Framework)

- pf4j-web (PF4J dans les applications web)

- pf4j-wicket (fenêtrage de Plugin Framework basé sur PF4J)

On peut marquer n'importe quelle interface ou classe abstraite comme point d'extension (avec l'interface marqueur ExtensionPoint) et on peut spécifier qu'une classe est une extension avec l'annotation Extension. Les composants principaux de PF4J sont :

- Plugin : c'est la classe de base pour tous les types de plugins. Chaque plugin est chargé dans un chargeur de classe distinct pour éviter les conflits.

- PluginManager : c'est utilisé pour tous les aspects de la gestion des plugins (installation, démarrage, arrêt). Vous pouvez utiliser une implémentation intégrée en tant que DefaultPluginManager ou vous pouvez implémenter un gestionnaire de plug-ins personnalisé à partir de AbstractPluginManager (implémenter uniquement les méthodes de création d'objets).

- PluginLoader charge toutes les informations (classes) nécessaires à un plugin.

- ExtensionPoint est un point de l'application où le code personnalisé peut être invoqué. C'est un marqueur d'interface java. Toute interface java ou classe abstraite peut être marquée comme point d'extension (implémente l'interface ExtensionPoint).

- Extension est une implémentation d'un point d'extension. C'est une annotation java sur une classe.

# Chapitre 10

# Choix de solutions

Notre question de recherche est : «Comment concevoir une plateforme collaborative (PFC) pour la publication de contenus sur les réseaux sociaux en couvrant certains besoins ?». Nos investigations nous montrent que deux solutions peuvent être envisagées :

- La première est de choisir ***une solution existante***, c'est-à-dire utiliser une application présente sur le marché et qui permet de publier du contenu sur les réseaux sociaux de manière collaborative. Dans le cadre de ce mémoire, nous avons présenté l'application Sprout Social, Agorapulse et Joomla.
- La deuxième est ***une solution personnalisée***. Il s'agit de développer soi-même une application («from scratch»).

Dans les chapitres précédents, nous avons présenté, dans un premier temps, des solutions existantes. Puis dans un second temps, nous avons fait des recherches sur des outils (OWL, PF4J, Spring, Plugin, etc.) pouvant servir à développer une solution personnalisée. Mais alors, comment faire pour opérer un choix entre ces deux solutions ?

Parmi ces solutions il est parfois difficile de faire un choix. Cette préoccupation revient régulièrement au début de chaque projet informatique. La section courante va donner quelques directives pouvant aider à mieux se positionner. Certaines littératures que nous avons consultées donnent des recommandations pour aider à concevoir ou choisir une plateforme collaborative. Ces recommandations ont été développées dans le chapitre 3, il s'agissait des points ci-dessous :

- Définir les besoins à satisfaire et les objectifs à atteindre,
- Préciser quel sera le périmètre de la plateforme,
- Anticiper sur la conduite des changements,
- Identifier les animateurs et les sponsors,
- Enfin, s'assurer que l'utilisation de la plateforme corresponde bien à la culture de l'entreprise.

Ces recommandations sont très générales et expliquent simplement comment on organise les idées pour aborder le problème afin de faire un choix. Elles n'informent pas concrètement sur les éléments déterminants sur lesquels un choix peut se faire. Après avoir étudié les solutions existantes, elles sont plus explicites et particulièrement axées sur :

- Le nombre de réseaux sociaux supporté.
- Le nombre de comptes par réseau social.
- Le nombre d'utilisateurs.
- L'ajout de nouveaux réseaux sociaux ou d'autres sites internet de publication de contenus.

Si on veut une plateforme avec des réseaux connus tels que Facebook, Twitter, LinkedIn, Instagram, avec environ 10 comptes par réseau social, il est préférable d'opter pour une solution existante. Une solution comme Agorapulse peut accepter jusqu'à 40 profils, 12 utilisateurs et fonctionne avec bon nombres des réseaux sociaux les plus utilisés.

Par contre si on souhaite ajouter autant de réseaux sociaux, de comptes, d'utilisateurs que l'on veut, il est préférable de se pencher vers une solution personnalisée. Dans ce cas, les outils que nous avons présentés précédemment seront d'une grande utilité.

En fait, bon nombre des solutions existantes ne sont pas extensibles. On est régulièrement limité aux fonctionnalités qu'elles proposent. Si on souhaite également que la plateforme fasse des publications sur un site internet quelconque, ou alors on veut avoir la main mise sur la manipulation des crédentials (login, mot de passe, les tokens d'accès), nous conseillons une solution personnalisée. En fait, les solutions existantes ne donnent pas d'informations sur la manière dont elles gèrent les crédentials, pour cela nous avons des réserves sur la sécurisation de ces crédentials.

Dans la suite de ce travail, pour notre cas, nous expliquerons comment nous allons concevoir notre solution personnalisée de PFC nommée KWeSSi en nous appuyant sur les outils présentés plus haut.

# Troisième partie

# Conception

# Chapitre 11

# Contexte

## 11.1 Délimitation de l'existant.

Les plateformes collaboratives sont utilisées non seulement par les entreprises mais aussi par les particuliers. En ce qui concerne les plateformes collaboratives pour la publication de contenus sur les RS, elles existent mais ne sont pas beaucoup vulgarisées (en consultant, par exemple Google Play Store (le 20/06/2019), nous constatons que Sprout Social a plus ou moins cent mille téléchargements, tandis qu'Agorapulse en a plus moins dix mille). Cette situation joue un peu en notre faveur quant à notre insertion dans le secteur. Ces plateformes sont souvent destinées aux entreprises et aux particuliers. Notre application KWeSSi ne doit pas aller en marge de ces deux débouchés pour une raison simple : si nous nous focalisons uniquement sur les entreprises ou sur les particuliers, nous sommes déjà en retard par rapport aux plateformes qui existent sur le marché. Certaines de ces plateformes offrent d'autres fonctionnalités attractives, non décrites dans nos besoins, que nous nous trouverons obliger d'intégrer dans notre propre application dans le future. Il s'agit par exemple de la fonctionnalité «statistique des messages publiés» ou «du journal de publication». L'intégration de ces fonctionnalités s'inscrit dans une logique de veille concurrentielle.

Notre plateforme collaborative est une application « open source ». L'ouverture du code source de notre application va permettre qu'elle acquière plus rapidement de nouvelles intelligences ou technologies, ce qui pourrait permettre une extension rapide. Dans ce contexte, il sera donc possible, pour autant qu'on ait les compétences, de développer son propre schéma OWL, qui répond à son propre besoin, et l'importer sur notre application pour ajouter de nouveau réseaux sociaux. Nous avons décidé que le téléchargement du Framework de base de notre application soit gratuit, mais les plugins peuvent être payants. Ce Framework contient toutes les fonctionnalités qui ont été présentées dans le chapitre « description des besoins ».

## 11.2 Les clients types

Au début de ce travail nos premières analyses nous ont permis de cibler les entreprises comme les clients potentiels pouvant être intéressés à utiliser une application comme la nôtre. Mais en étudiant les solutions actuelles et en regardant de plus près la communauté des personnes fréquemment présentes sur les réseaux sociaux, nous avons inclus les particuliers, notamment les influenceurs et toutes les personnes désireuses de publier du contenu sur plusieurs comptes ou sur plusieurs RS simultanément.

Malgré que nous ayons choisi de proposer notre plateforme en même temps aux entreprises et aux particuliers, il convient encore de choisir «une niche» de test et de lancement. Ce marché de «niche» est l'université de Namur. Le choix de cette université est basé sur plusieurs critères :

- L'université, les étudiants, les professeurs, le personnel administratif, ont sans doute plusieurs comptes sur les réseaux sociaux d'où la présence d'un besoin de centralisation des publications de chacun.

- L'université est une entreprise mais est fréquenté par des particuliers, ce qui nous permet de faire «d'une pierre deux coups» pour les tests. En fait, cela nous donne l'occasion de faire les tests sur une entreprise (Université

de Namur) et sur les particuliers (étudiants, professeur, etc)

- Nous connaissons bien cette université, nous y avons développé de bonnes relations, donc en tant que notre premier client, cela peut nous faciliter la compréhension des feedbacks.

- De manière générale, l'université est un centre de partage et de collaboration. Nous souhaitons profiter de cette situation naturelle pour y promouvoir notre plateforme collaborative.

## 11.3 Les acteurs de l'application.

Durant le fonctionnement de l'application KWeSSi, différents acteurs vont interagir avec le système. Chaque utilisateur effectue des actions correspondant à des rôles qui lui sont attribués. Les acteurs présentés ici sont extraits de la description des besoins que nous avons fait plus haut. Voici donc les acteurs qui vont faire vivre notre application :

- Un utilisateur : il est une personne inscrite mais non active. Pour accéder aux fonctionnalités du système, elle doit être activée.

- Un membre : un membre est utilisateur inscrit et actif. Il a pour rôle de consulter les contenus publiés.

Un rédacteur : le rédacteur a les mêmes droits qu'un simple membre et a la possibilité de suggérer et de rédiger un contenu.

- Un rédacteur : le rédacteur a les mêmes droits qu'un simple membre et a la possibilité de suggérer et de rédiger un contenu.

- Un censeur : le censeur a tous les droits d'un membre, d'un rédacteur et en plus, il peut gérer les différents contenus (modifier ou supprimer) et publier ces contenus sur les réseaux sociaux dont il a la charge.

- Un modérateur : le modérateur a tous les droits d'un membre, rédacteur et d'un censeur, son rôle est de valider la suggestion de contenu faite par le rédacteur ou le censeur.

- Un administrateur : l'administrateur est un utilisateur qui a tous les droits dans l'application. Il peut contrôler les accès, gérer les utilisateurs, leurs rôles et les différents canaux de publications.

Après avoir vu les différents acteurs du système, nous allons voir en détails les actions que chacun d'eux peut effectuer. Le diagramme de «cas d'utilisations» (ou diagramme d'actions) ci-dessous, définit le rôle de chaque utilisateur.

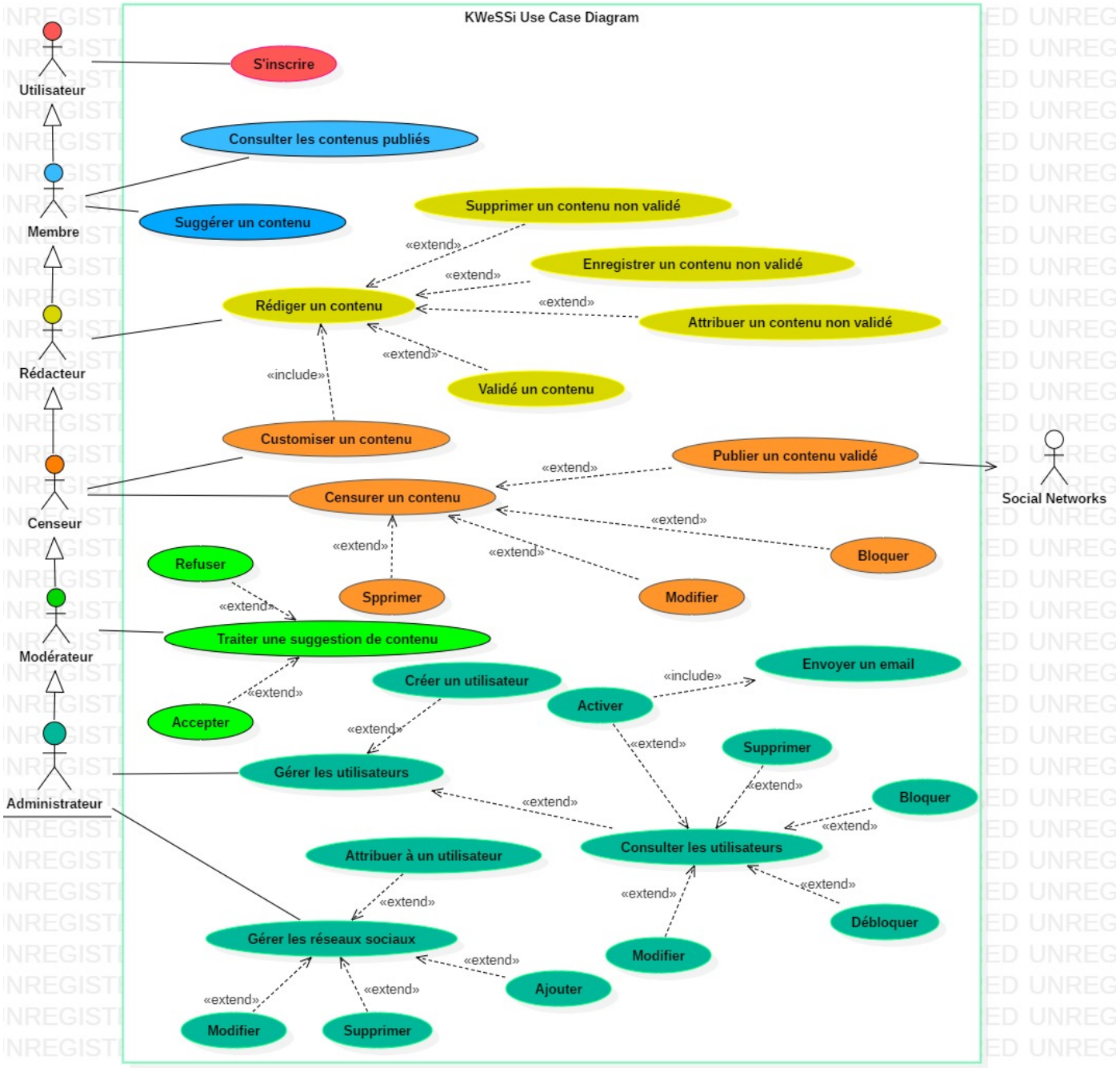



FIGURE 11.1 – diagramme de cas d'utilisations ou d'actions.

# Chapitre 12

# Conception d'une base de connaissances (BDC) pour l'application KWeSSi

## 12.1 Problème et méthode

Dans le chapitre 2, où nous avons étudié les réseaux sociaux, nous avons constaté que chaque réseau social avait ses spécificités. Par exemple, pour la publication de contenus, Facebook accepte un nombre de caractères maximal égale 64206, alors que Twitter accepte seulement 280 caractères. Certains champs sont présents sur Facebook mais ne le sont ni sur Twitter, ni sur LinkedIn. À la fin du chapitre 7, nous avons opté pour OWL pour modéliser nos données. Ce choix a été motivé non seulement par la souplesse d'OWL mais aussi parce qu'elle s'adapte le mieux à nos besoins d'extension. Notre préoccupation dans le présent chapitre est : comment concevoir la base de connaissances de notre application pour la publication de contenu sur tous les réseaux sociaux, de manière à prendre en compte les spécificités de chacun ? Notre mémoire se concentre sur la publication de contenu. Ainsi nous n'allons pas étudier d'autres aspects des réseaux sociaux tels que la communication entre amis, les « like », etc. Pour Atteindre notre objectif nous allons utiliser la méthodologie suivante :

- Préciser ce que nous entendons par «publication de contenu sur les réseaux sociaux »,
- Choisir un ensemble de réseaux sociaux représentatif en s'appuyant sur des critères objectifs,
- Pour chaque réseau social, nous allons répertorier les données qui lui sont propres et concernant la publication de contenus,
- Concevoir un schéma OWL pour chaque réseau social,
- Intégrer les schémas OWL précédents pour obtenir un schéma OWL global contenant les données de tous les réseaux sociaux,
- Ajouter au schéma global les données ne venant pas directement des réseaux sociaux (comme le rôle utilisateur) pour obtenir un schéma final pour l'application KWeSSi.

## 12.2 Publication de contenus sur les réseaux sociaux.

Nous entendons par contenu, toute forme d'informations que l'on peut diffuser sur les réseaux sociaux. Il ne s'agit pas seulement d'un simple message textuel, mais nous prenons en compte aussi les images, les audios, les vidéos, les fichiers de formats très variés (.exe, .docx, .xlsx, .txt .pdf, .rar, etc.). La liste que nous avons donnée n'est pas exhaustive car il est presque impossible de citer toutes les formes d'informations que les réseaux sociaux peuvent accepter. Certains réseaux sociaux comme Facebook, Twitter ou LinkedIn acceptent tout genre de contenus tandis que d'autres comme YouTube, Instagram sont plus spécialisés. En fait, YouTube est spécialisé pour les contenus de type vidéo alors qu'Instagram, dès son lancement était spécialisé pour les contenus de type photo. Maintenant, Instagram a beaucoup évolué et prend aussi en compte les vidéos. Beaucoup d'autres réseaux sociaux ont aussi évolué. Facebook, YouTube, Instagram, pour ne citer que ceux-là, permettent maintenant des «lives» (vidéos en direct). Le but de

recenser les différentes formes de contenus qui sont acceptées par les réseaux sociaux nous permet de savoir quels genres d'informations nous voulons garder dans notre base de connaissances (BDC).

## 12.3 Sélection d'un ensemble représentatif de réseaux sociaux

De nos jours il existe plus d'une centaine de réseaux sociaux. Ne pouvant tous les étudiés un par un, ce qui prendrait trop de temps sans garantir un bon résultat, nous avons élaboré des critères qui vont nous permettre de choisir les plus pertinents. Ceux que nous allons choisir doivent nous donner l'ensemble des données que les réseaux sociaux manipulent lors de la publication de contenus. Il est possible de prendre d'autres critères mais le résultat ne sera pas nécessairement identique au nôtre. Les critères que nous avons retenus sont les suivants :

- La popularité : C'est le nombre d'utilisateurs abonnés à un réseau social. Dans le classement publié par statista et contenu dans le lien https ://fr.statista.com/statistiques/570930/reseaux-sociaux-mondiaux- classes-par-nombre-d-utilisateurs/, au 8 avril 2019, Facebook dépassait les 2 milliard d'utilisateurs, suivit de YouTube qui avoisinait les 2 Milliard, puis WhatsApp avec environ 1500 Millions d'utilisateurs. Dans cette catégorie nous choisissons Facebook et YouTube car leur nombre d'utilisateurs cumulé nous semble suffisamment grand.

- La thématique : Ici nous avons les réseaux sociaux généralistes et les réseaux spécialisés. Les réseaux généralistes acceptent toutes formes de contenus. On peut citer Facebook, Twitter, LinkedIn. Les réseaux sociaux spécialisés se concentrent sur la publication d'un contenu spécifique. Nous avons YouTube ou Snapchat pour les vidéos, Instagram pour les images. Ici nous gardons Instagram.

- L'accessibilité (public et privé) : Nous entendons par réseau public un réseau social sur lequel on a accès aux contenus qui se trouve sur un compte qui n'est pas le nôtre. Dans cette catégorie il y a Facebook, Twitter, LinkedIn, Instagram. Les réseaux sociaux privés ne laissent pas la possibilité à des uns de voir ce qui est publié sur le compte des autres. Dans ce cas nous pouvons citer Skype, WhatsApp, Viber, Snapchat. Dans ce mémoire nous allons considérer seulement les réseaux publics pour la seule raison que les réseaux privés n'ont pas d'intérêt pour notre étude.

Au final nous retenons les RS suivants : Facebook, YouTube, Twitter, LinkedIn, Instagram. En collectant les informations sur ces 5 RS nous sommes presque sûrs d'avoir ce qu'il faut pour concevoir une base de connaissances pour les réseaux sociaux dans le cadre de la publication de contenus.

## 12.4 Collecte de données et conception d'une BDC des RS pour la publication de contenus

Étant donné que nous n'avons pas accès aux bases de données des réseaux sociaux, nous nous sommes rendus sur la page web de chaque réseau social afin de collecter les informations sur la publication de contenu (on parle du rétro-ingénierie). Généralement, sur chaque RS la publication se fait via un formulaire. Nous avons analysé ces formulaires et nous avons énuméré toutes les données brutes qui y sont manipulées. Par la suite, en utilisant le logiciel Protégé, nous avons conçu la base de connaissances propre à chaque réseau social. Puis, nous avons intégré les 5 bases de connaissance réalisées (Facebook, Twitter, Instagram, YouTube, LinkedIn) pour obtenir une base de connaissances des réseaux sociaux dans le cadre de la publication de contenus.

### 12.4.1 Facebook

Le formulaire de publication de Facebook se présente comme suit :

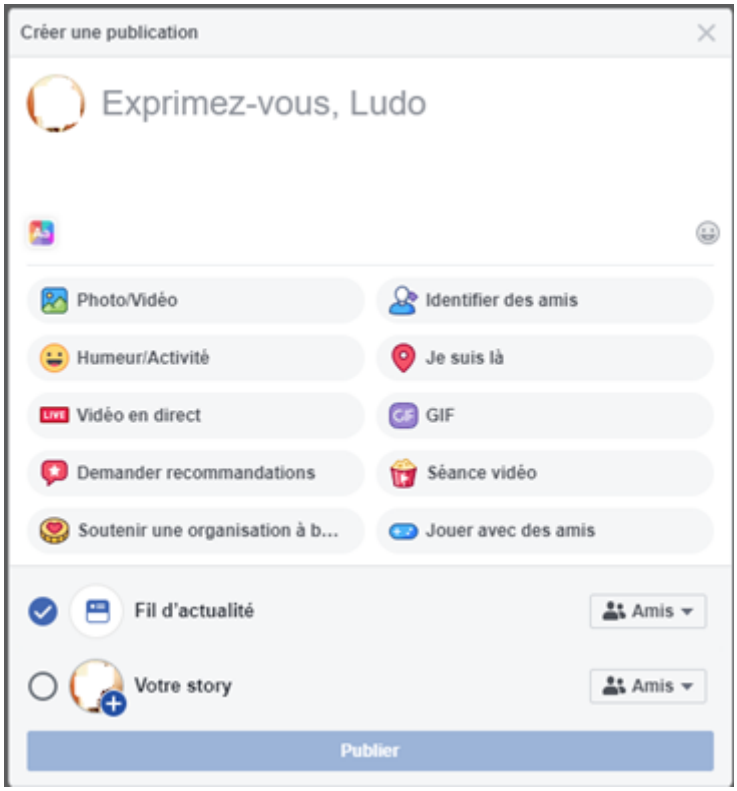



FIGURE 12.1 – formulaire de publication sur Facebook.

Les données collectées sur ce formulaire sont récapitulées dans le tableau :

| Nom du champ | Type | Taille Maximale | Taille Minimale | Résolution |
|---|---|---|---|---|
| « Exprimez-vous» | Texte | 63 206 caractères | 1 caractère | - |
| Photo | PNG / JPEG | Ko | - | - |
| Video | MOV / MP4 | Mo | - | - |
| Gif | Gif | - | - | - |
| Vidéo en direct | MOV / MP4 | - | - | - |
| je suis là | Texte | - | - | - |
| Emoji | Emoticône | - | - | - |
| Fichier | Tout type de fichier | 100 MB | - | - |
| Identifier des amis | | | | |
| Arrière-plan (thème) | | | | |
| Humeur/Activité | | | | |
| Demande de recommandations | | | | |
| Séance vidéo | | | | |
| Jouez avec des amis | | | | |

Un champ est égal à une donnée manipulée. Certains de ces champs sont pertinents pour réaliser notre base de connaissances («Exprimez-vous Ludo», photo, vidéo, Gif. . . ). D'autres sont plutôt des fonctionnalités («Jouez avec des amis», «Fil d'actualité», Votre story, «Amis») et n'entrerons pas dans nos données de travail car ils ne sont pas directement liés à la publication de contenus. Par exemple, le champ «Fil d'actualité» permet de voir l'historique des contenus qui ont été publié sur un compte Facebook, cela n'est pas définit dans notre description de besoins. Le champ

«je suis là» permet à Facebook de géolocaliser la position où le contenu a été publié, il n'est pas non plus définir dans nos besoins. Pour tous les champs choisis, nous avons récupéré son « nom », son type et sa taille. Par exemple le champ de nom « Exprimez-vous, Ludo » est de type texte et à un maximum de 63206 caractères. Quand on dépasse cette taille, Facebook renvoie le message suivant :

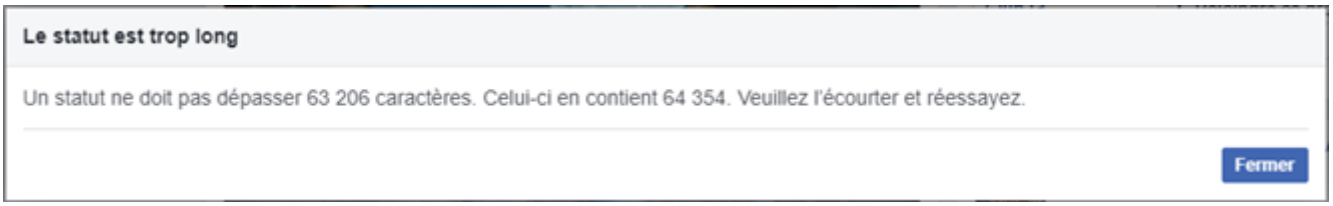



FIGURE 12.2 – nombre de caractères maximal sur Facebook.

Durant nos différentes recherches, nous n'avons pas pu obtenir des données précises sur la taille et la résolution des photos et des vidéos. Nous avons tenté d'ajouter une photo de résolution 50x50 et de 7216x5412. Facebook a adapté la taille de la photo par rapport à son champ photo. Après avoir collecté les données, nous avons étudié les relations qui peuvent exister entre ces données. Il en ressort par exemple qu'une publication sur un compte Facebook a plusieurs champs et qu'un champ a au moins un type (texte, photo, vidéo, etc). Cette étude nous a permis de concevoir le schéma OWL (base de connaissances) de Facebook qui se trouve sur le schéma suivant. Pour des raisons techniques nous n'avons pas pu prendre une image qui contient les propriétés de données parce que le schéma était trop grand. Pour cette raison, dans toute la suite nous vous montrerons des schémas OWL ne contenant que des classes. Ces schémas ont été réalisés avec OWLViz. Nous allons joindre en annexes de ce mémoire les fichiers OWL complets. Pour voir tous les éléments du schéma que nous allons mettre en annexe, vous pouvez utiliser WebVOWL.

Sur le schéma les zones en couleur jaune sont appelée « classe ». L'inscription « is-a » représente une «dataObject».

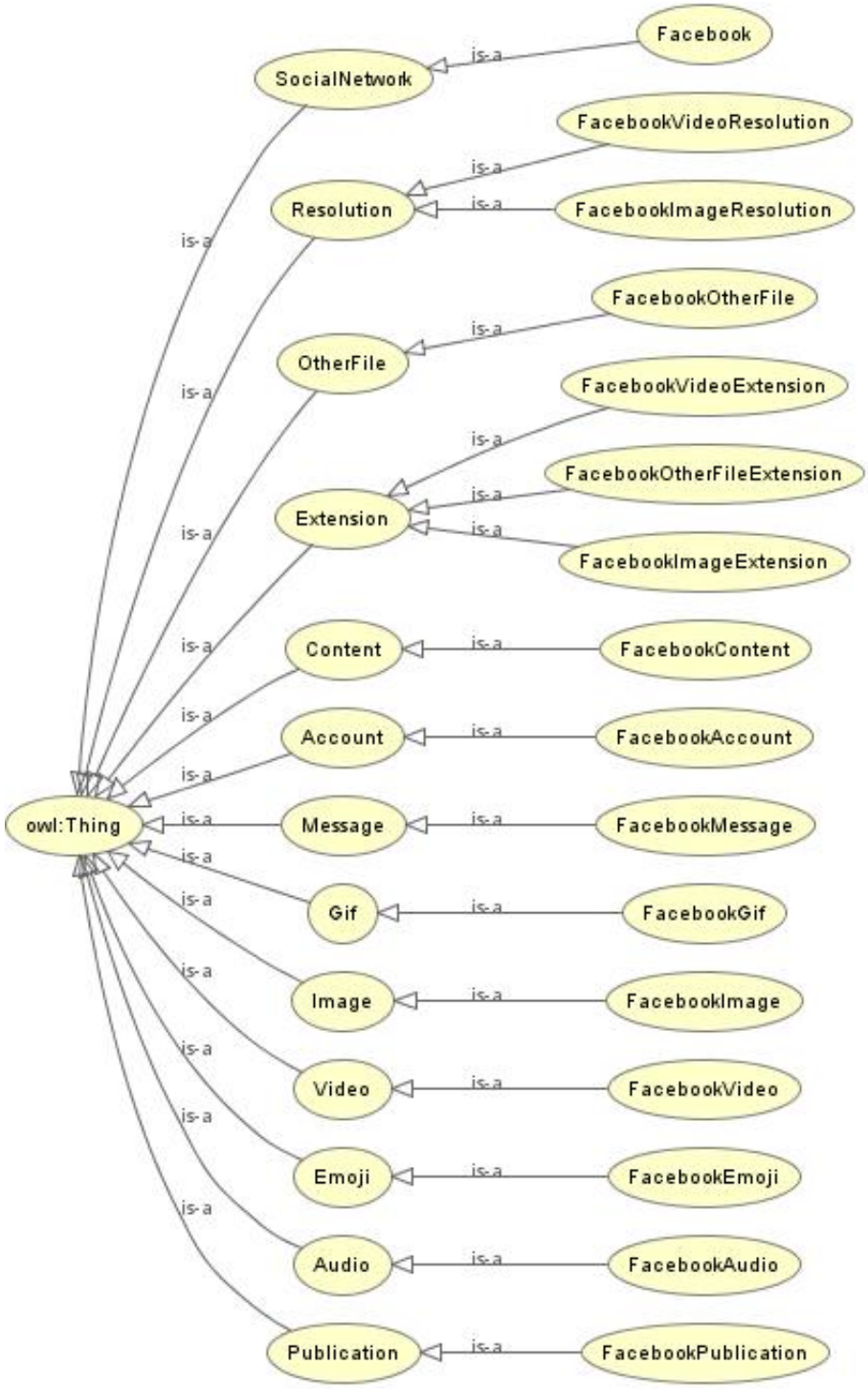



FIGURE 12.3 – schéma OWL pour Facebook.

### 12.4.2 Twitter

Sur Twitter le formulaire de publication [1] est :

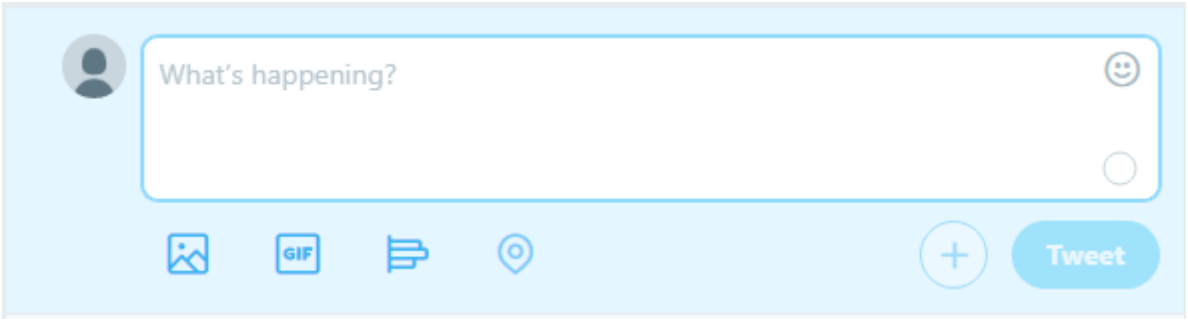



FIGURE 12.4 – formulaire de publication sur Twitter.

Les données récoltées en observant le formulaire ci-dessus sont dans le tableau suivant :

| Nom du champ | Type | Audio | Taille Maximale | Taille Minimale | Temps de diffusion | Résolution Maximale | Résolution Minimale |
|---|---|---|---|---|---|---|---|
| « Quoi de neuf » | Texte | - | 280 caractères | 1 caractère | - | - | - |
| Photo | PNG / JPEG Ne sont pas accepté (BMP et TIFF) | - | 5 Mo | - | - | - | - |
| Video | MOV / MP4 | audio AAC | 512Mo | - | 2 minutes et 20 seconds. | 1 920 x 1 200 (et 1 200 x 1 900) | 32 x 32 |
| Gif | Gif | - | 5 Mo (Mobile) et 15 Mo (Web) | - | - | - | - |
| Sondage | QCM | - | - | - | - | - | - |
| Localisation | Texte | - | - | - | - | - | - |
| Emoji | Emôticone | - | - | - | - | - | - |

Nous remarquons que les champs de Twitter ne possèdent pas de noms leur permettant de directement les identifier comme cela était sur Facebook. Grâce aux symboles, aux différents tests que nous avons effectués et aux informations que nous avons trouvées sur le site internet de Twitter, nous avons pu, pour certains trouver leurs dénominations et pour d'autres trouver des termes adéquats. Comme nous pouvons le voir sur le formulaire, Twitter a juste un nom de champ qui apparait. Il s'agit de « Sondage » qui permet d'établir des votes pour différents types de sujets. Les autres champs ressemblent de près aux champs de Facebook. Ces champs vont être repris afin de constituer notre base de connaissances de publication de contenus pour Twitter. Le champ « What's happening ?», comme celui de Facebook « Exprimez-vous, Ludo », doit contenir le texte du message que l'on veut rédiger. Nous avons recueilli de nouvelles caractéristiques pour certains champs de Twitter : Temps de diffusion, Audio et la Résolution.

Dans la suite nous n'allons plus faire le schéma de base de connaissances pour chaque RS parce que la technique pour les faire ne change pas par rapport à ce que nous avons réalisé pour Twitter et Facebook. Nous allons continuer

1. Plus d'informations sur : https ://help.twitter.com/fr/using-twitter/tweeting-gifs-and-pictures et https ://help.twitter.com/fr/using-twitter/twitter-videos

à collecter les informations sur chaque RS puis présenter la base de connaissances de tous les RS et la base de connaissances de notre plateforme collaborative. Le schéma qui représente la base de connaissances de Twitter est :

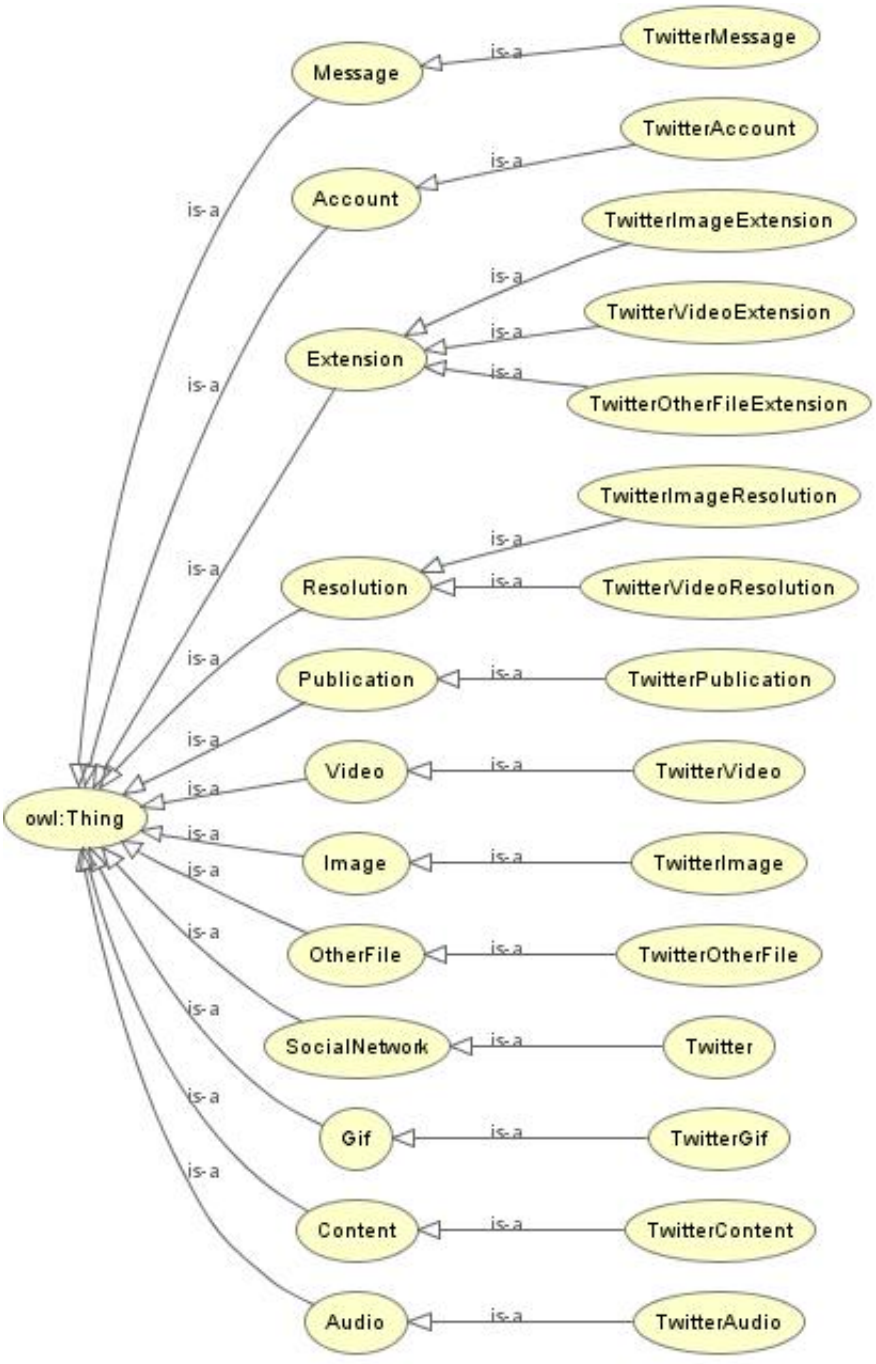



FIGURE 12.5 – schéma OWL pour Twitter.

### 12.4.3 Instagram

Avant de citer les différents champs de ce réseau social, nous allons brièvement le présenter car nous ne l'avons pas fait dans le chapitre RS. Instagram est un réseau social apparu le 6 octobre 2010 et créé par Kevin Systrom et Mike Krieger [2]. . Il s'agit d'un réseau social servant à partager des photos et vidéos publiques ou privées. Actuellement, il appartient à Facebook. Donc si l'utilisateur a un compte Facebook et un compte Instagram, il aura la possibilité de publier ses photos instantanément dans les 2 réseaux sociaux, de Facebook à Instagram ou inversement. Ci-dessous, se trouve le formulaire de publication sur Instagram.

2. Page d'aide d'Instagram : https ://help.instagram.com/488619974671134/ ?helpref=hc_fnav&bc[0]=Aide%20sur%20Instagram&bc[1]=Utilisation%20d%E2%80%99Instagram&bc[2]=Partage20de%20photos%20et%20de%20vid%C3%A9os

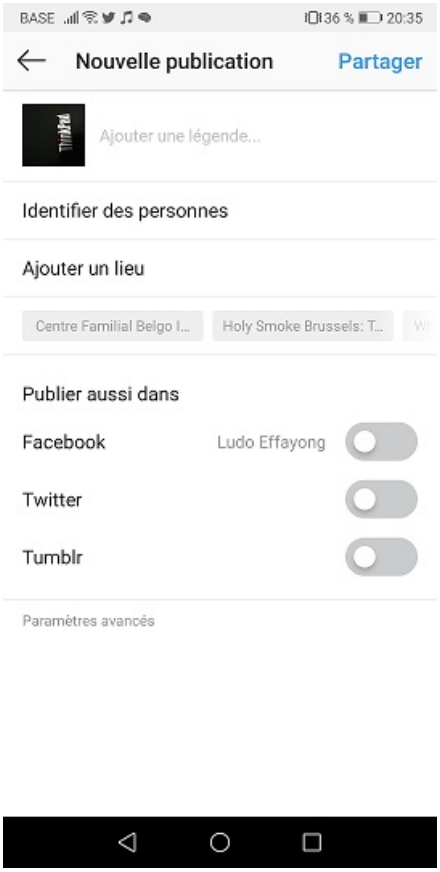



FIGURE 12.6 – formulaire de publication sur Instagram.

Les données récupérées sur le formulaire d'Instagram sont consignées dans le tableau ci-dessous :

| Nom du champ | Type | Taille Maximale | Taille Minimale | Durée | Résolution |
|---|---|---|---|---|---|
| « Ajouter une légende» | Texte | - | 1 caractère | - | - |
| Photo | - | - | - | - | - |
| Video | - | - | - | Entre 3 et 60 secondes | - |
| « Publier aussi dans Facebook / Twitter / Tumblr» | - | - | - | - | - |

Caractéristiques de la photo :

| Caractéristique | Type |
|---|---|
| Identifier des personnes | Texte |
| Ajouter un lieu | Texte |

Sur Instagram, nous allons prendre les champs « Ajouter une légende », photo, vidéo, « Identifier des personnes » et « Ajouter un lieu ». Les autres champs sont plus des fonctionnalités d'Instagram. Sur le formulaire de publication d'Instagram, nous ne voyons pas le champ photo et vidéo, cela est normal. Après avoir pris la photo, nous nous sommes redirigés vers la page de rédaction d'une publication. La photo se trouve sur le coin supérieur gauche, près de « Ajouter une légende ».

### 12.4.4 YouTube

Nous allons aussi présenter brièvement le réseau social YouTube. C'est un réseau social qui permet aux utilisateurs de regarder, commenter, évaluer, partager et surtout poster des vidéos de tout genre : sportif, vlog, critique, trailer de films ou séries, court-métrage,... Il a été créé le 14 Février 2005 par Jawed Karim, Chad Hurley et Steve Chen. Ensuite il a été racheté par Google le 9 Octobre 2006. Ci-dessous, nous avons le formulaire de publication sur YouTube. [3].

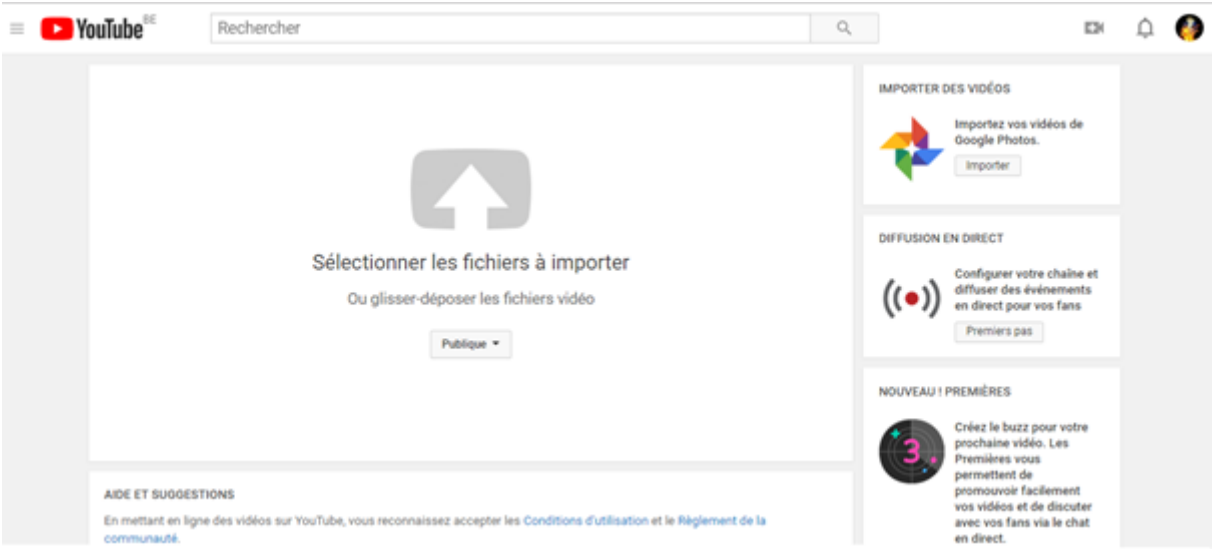



FIGURE 12.7 – formulaire de publication sur YouTube

Les données collectées via le formulaire sont consignées dans le tableau suivant :

| Nom du champ | Type | Taille | Durée | Résolution |
|---|---|---|---|---|
| **Vidéo** | .MOV<br>.MPEG4<br>.MP4<br>.AVI<br>.WMV<br>.MPEGPS<br>.FLV<br>3GPP<br>WebM<br>DNxHR<br>ProRes<br>CineForm<br>HEVC (h265) | - | - | 2160p : 3 840 x 2 160<br>1440p : 2 560 x 1 440<br>1080p : 1 920 x 1 080<br>720p : 1 280 x 720<br>480p : 854 x 480<br>360p : 640 x 360<br>240p : 426 x 240 |
| **Vidéo en direct** | **Similaire** | - | - | **Similaire** |
| **Titre** | **Texte** | - | - | - |
| **Description** | **Texte** | - | - | - |
| **Tags** | **Texte** | - | - | - |

En parcourant la fonctionnalité d'aide du site YouTube on peut lire « YouTube pour ordinateur utilise un format standard de 16 :9. Si votre vidéo est à un autre format, la taille du lecteur s'adaptera automatiquement pour correspondre à votre vidéo et à l'appareil du spectateur ». YouTube possède moins de champs de publication contrairement aux autres réseaux sociaux que nous avons étudiés.

3. Page d'aide de YouTube : https ://support.google.com/youtube/answer/6375112 ?co=GENIE.Platform%3DDesktop&hl=fr, https ://support.google.com/youtube/troubleshooter/2888402 ?hl=fr&ref_topic=9257782

### 12.4.5 LinkedIn

Le formulaire de rédaction d'une publication sur LinkedIn se présente ainsi :

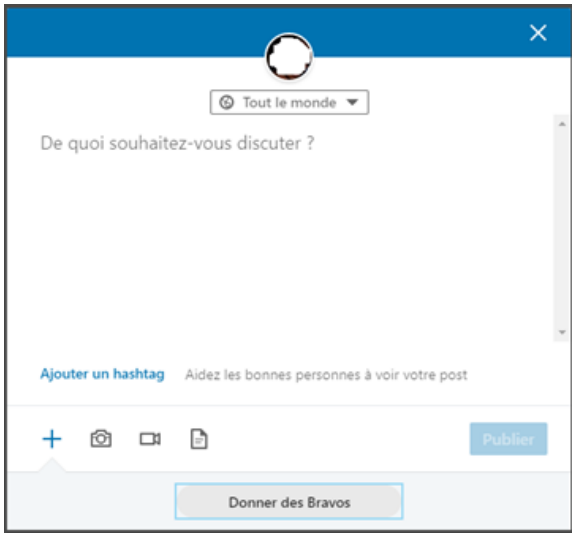



FIGURE 12.8 – formulaire de publication sur LinkedIn.

Les informations obtenues en analysant le formulaire sont : [4]

| Nom du champ | Type | Taille Maximale | Taille Minimale | Durée minimale | Durée maximale | Résolution Minimale (Largeur x hauteur) | Résolution maximale (Largeur x hauteur) |
|---|---|---|---|---|---|---|---|
| « De quoi souhaitez-vous discuter » | Texte | 1300 caractères | 0 caractère | - | - | - | - |
| Photo | PNG / JPEG | - | - | - | - | 200x - | - |
| Video | ASF/AVI/FLV/MPEG-1/MPEG-4/MKV/QuickTime/WebM H264/AVC MP4/VP8/VP9/WMV2/ WMV3/AAC, MP3 et Vorbis. | 5 Go | 75 Ko | 3 secondes | 10 min | 256x144 | 4096x2304 |
| Document | PPT/PPTX/DOC /DOCX/PDF | 100 Mo et 300 Pages | - | - | - | - | - |

Le champ vidéo sur LinkedIn présente des spécificités suivantes :

- Proportions : 1 :2.4 - 2.4 :1
- Fréquence : de 10 à 60 images par seconde
- Vitesse : de 192 kbit/s à 30 Mbit/s

Ce champ permet de charger des fichiers de différents types, par exemple : PowerPoint(PPT), PDF, docx.

4. Autres sources https ://www.linkedin.com/help/recruiter/answer/73101/specifications-d-images-de-vos-pages-linkedin-et-carrieres ?lang=fr, https ://www.linkedin.com/help/linkedin/answer/97789/partage-de-documents-sur-linkedin ?lang=fr

### 12.4.6 Base de connaissances des réseaux sociaux pour la publication de contenus

Pour obtenir la base de connaissances pour la publication sur l'ensemble des réseaux sociaux, nous avons intégré, manuellement, les 5 bases de connaissances ci-dessus. Nous avons constaté que la base de connaissances de Facebook avait plus d'informations que les autres, donc nous l'avons utilisé comme base de notre intégration. Puis nous avons élaboré des critères d'intégration qui sont :

- Classes ou propriétés spécifiques : il s'agit des informations qui sont propres à un réseau social. Par exemple, YouTube a un champ « Tags » que Facebook n'a pas. Nous devons ajouter cela dans le schéma de Facebook.

- Classes ou propriétés semblables : Ce sont des classes ou propriétés qui ont la même signification dans Facebook. Par exemple le champ « What's happening ?» de Twitter contient les mêmes informations que le champ « Exprimez-vous, Ludo » de Facebook. Dans ce cas nous gardons l'un des champs ou nous trouvons un nouveau nom qui résume mieux les champs semblable. Pour l'exemple que nous venons de citer, nous avons résumé les deux champs sous le nom « Message ».

En respectant ces critères, nous avons d'abord intégré le schéma de Facebook et Twitter. Puis nous avons intégré Instagram sur le résultat obtenu. Enfin, en suivant la même procédure, nous avons respectivement intégré le schéma de YouTube et LinkedIn. Normalement, l'ordre de l'intégration n'est pas fixe. On peut choisir n'importe quel ordre et aboutir au même résultat, qui se trouve sur l'image qui suit. Pour des raisons de lisibilité, nous avons délibérément enlevé LinkedIn, Youtube, Intagram sur l'image pour qu'elle entre sur une page.

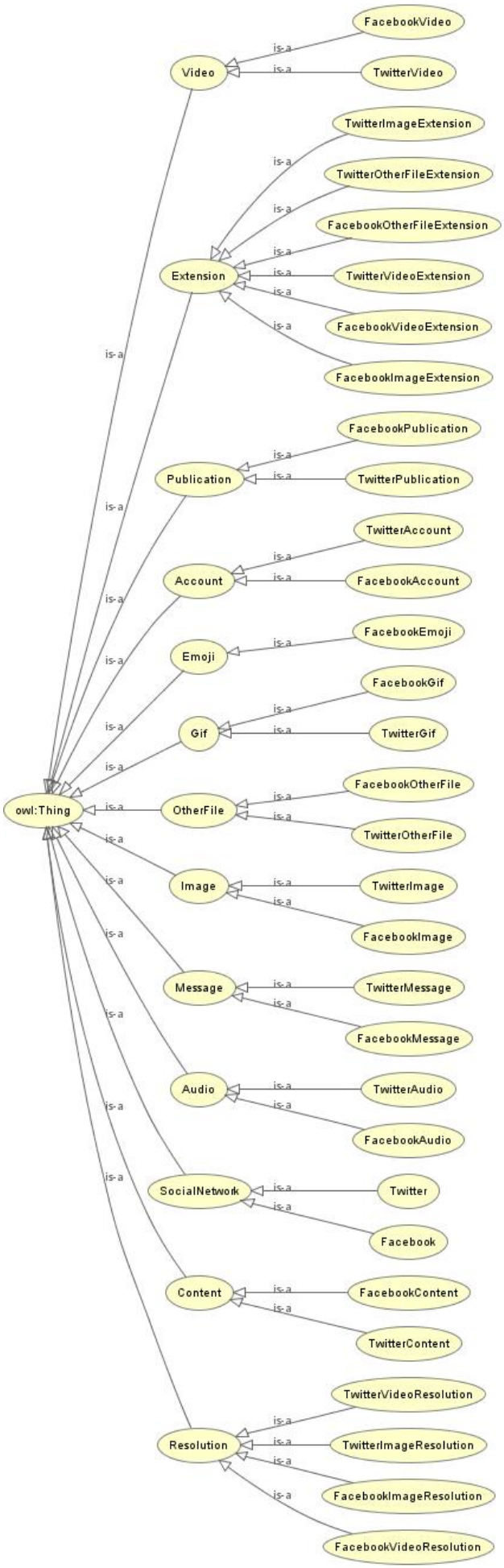



FIGURE 12.9 – schéma OWL global pour tous les réseaux sociaux.

### 12.4.7 base de connaissances de l'application KWeSSi

Après avoir obtenu le schéma qui contient la base de connaissances des réseaux sociaux concernant la publication de contenus, nous avons regardé dans nos besoins les spécifications qui manquaient. Il ne restait plus que la gestion des utilisateurs et le rôle des utilisateurs. La relation entre ces spécifications et les autres informations déjà traitées plus haut est : un utilisateur peut avoir plusieurs rôles sur un ou plusieurs réseaux sociaux. Les rôles possibles pour un utilisateur sont : utilisateur, membre, rédacteur, censeur, modérateur, administrateur. En nous basant sur la relation que nous venons de donner nous avons ajouté les 2 spécifications au schéma des réseaux sociaux pour obtenir la base de connaissances de l'application KWeSSi. Encore pour des raisons de lisibilité, nous avons délibérément enlevé LinkedIn, Youtube, Intagram sur l'image pour qu'elle entre sur une page.

Sur le schéma un utilisateur est remplacé par la classe « Personne » pour éviter une ambiguïté avec le rôle «utilisateur». Cela permet aussi de gérer le fait qu'une base de connaissances n'accepte pas les doublons entre les classes. Sur l'image ci-desous, de gauche à droite, toutes les classes se trouvant sur la même verticale que la classe «OtherFile» constituent le schéma OWL du Framework de base de notre PFC. Pour ajouter un nouveau réseau social il faut toujours utiliser la méthode que nous avons décrite au début de cette section.

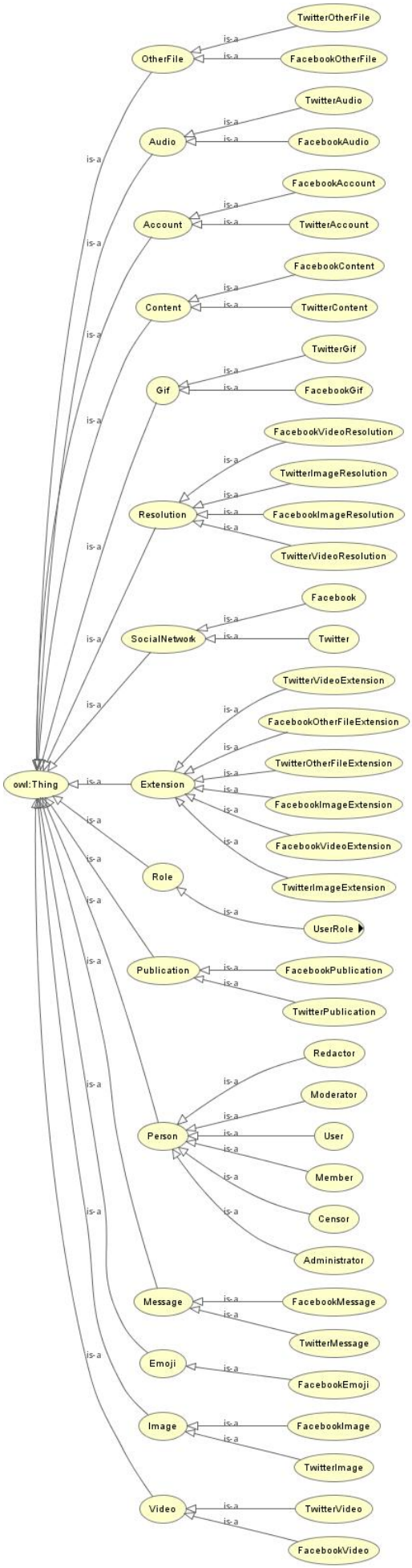



FIGURE 12.10 – schéma OWL final de l'application KWeSSi.

### 12.4.8 Appréciation de notre méthode

Notre méthode est adaptée à un contexte ne manipulant pas une grande quantité de données. Sur notre schéma final on peut compter facilement le nombre de classes qui s'y trouvent. Dans le cas où les données sont très grandes on peut procéder comme dans[23], particulièrement dans son chapitre 7, où on établit d'abord une base de données relationnelle pour chaque réseau social avant de la convertir en base de connaissances.

# Chapitre 13

# Architecture de l'application KWeSSi

## 13.1 Description de l'architecture KWeSSi

Après avoir étudié les trois architectures mentionnées dans la «partie littérature» nous constatons qu'en pratique, elles peuvent se combiner. Nous allons donc utiliser cette technique dans le cadre de notre application. Nous avons décidé d'utiliser OWL/RDF pour la modélisation et le stockage de données. Ce choix nous oriente un peu vers du web sémantique (OWL). L'architecture à base du web sémantique, tel que nous l'avons décrite plus haut, a une couche «User interface et applications». Cette couche laisse une certaine liberté au niveau de sa conception. Ainsi, on peut y mettre une API Jena pour faire des requêtes vers OWL. On peut aussi y mettre du Spring. Avec Spring nous allons définir le cœur de notre architecture, c'est-à-dire le Framework de base. Sur ce Framework nous allons ajouter dynamiquement de nouvelles fonctionnalités en utilisation des plugins faits à base du Framework PF4J. L'utilisation des plugins est un bon moyen pour étendre une application. En fait, les plugins laissent une bonne marge de flexibilité aux développeurs, aux utilisateurs dans la mesure où chacun peut développer ses propres plugins. À tout moment, ils peuvent être installés, stoppés ou désinstallés. Spring nous facilite également la publication sur les réseaux sociaux comme Facebook, Twitter ou LinkedIn. Cela se fait en utilisant la librairie « spring-social ». L'architecture à base de web sémantique offre beaucoup plus de flexibilité. Par exemple, on peut utiliser son extension OWL-plugin pour étendre le domaine de connaissance de base. Au final nous allons combiner les trois architectures (Spring, plugins et Web sémantique), cela nous permet de bénéficier des atouts de chacun. Nous nous sommes demandé, dans la littérature, comment ces trois architectures sont combinées.

Dans nos recherches, nous avons constaté que l'application BoWiki [11] n'utilise pas Spring. Son interface utilisateur est développée en PHP, sa modélisation est faite avec une base de données relationnelle combinée avec du OWL. Pour évaluer les nouvelles données saisies ou les requêtes sémantiques, le serveur BOWiki requiert une ontologie au format OWL-DL (fournie lors de l'installation dudit serveur). Des données sémantiques cohérentes y sont stockées. Si une incohérence est détectée, la page modifiée est rejetée avec une explication de l'incohérence. Le serveur BOWiki utilise le Framework Web sémantique Jena 2. Après vérification, les données sémantiques sont stockées dans une partie distincte, une base de donnée relationnelles (MySQL). L'application eXo plateforme (expliquer plus haut) quant à elle utilise Spring mais sa modélisation est faite uniquement avec MySQL. Le fait qu'elle n'utilise pas OWL peut s'expliquer par le fait qu'elle ne manipulerait pas de relations complexes. Le CMS Jomla n'utilise ni Spring, ni du Web sémantique. Développé en PHP, il a une architecture qui lui est propre. Cette architecture a déjà été expliquée plus haut. Joomla modélise ses données aussi avec MySQL. BoWiki, eXo plateforme et Joomla utilisent aussi les plugins.

Comme vous le constatez, nous n'avons pas trouvé une application existante (ou plateforme collaborative) utilisant en même temps le Spring, les plugins, et OWL. Dans la suite, en nous appuyant sur nos expériences, nous allons expliquer comment concevoir une telle combinaison.

## 13.2 Conception de l'architecture KWeSSi

Le schéma de cette architecture se présente ainsi :

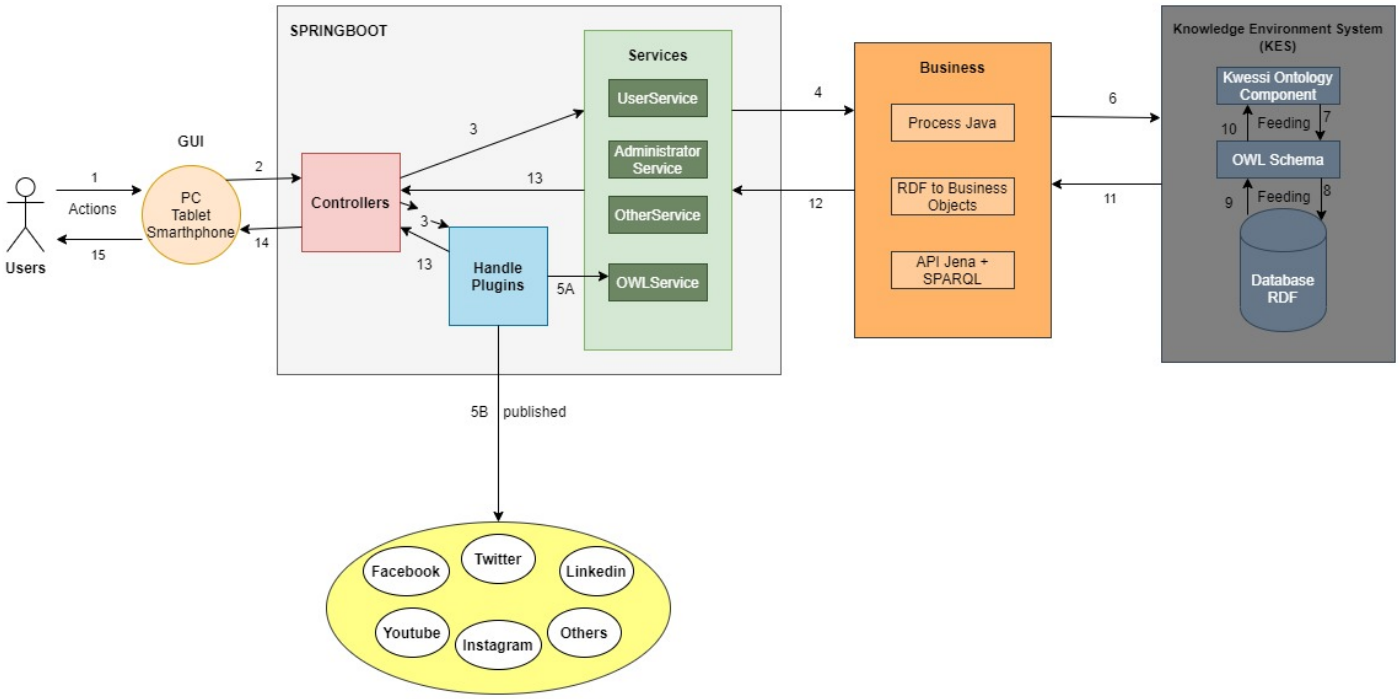



FIGURE 13.1 – Architecture du PFC KWeSSi.

Une personne souhaite effectuer une action (1) sur l'application KWeSSi. Pour cela il peut utiliser soit une tablette, soit un smartphone, soit un PC (personal computer) pour envoyer (2) une requête au contrôleur (**Controllers**), qui se charge de valider ladite requête, puis l'envoyer soit à un service (Services) (3) soit à un plugin (3) qui pourra faire appel à un service (5A). Si la requête concerne la publication d'un message, la couche **Handle Plugins** se charge de publier (5B) le message sur le bon réseau social. Dans ce cas l'action (5A) et (5B) peuvent se passer simultanément. La couche Services choisit le bon service pour interpréter et emballer ladite requête sous un objet facilement compréhensible par la couche business (**Business**). Par la suite, elle transmet (4) la requête à la couche Business. La couche **Business** doit à son tour transformer les objets reçus en des objets RDF, puis l'API Jena va prendre ces objets pour les envoyer (6) à la couche **Knowledge Environnement System (KES)**. Cette couche va choisir (7) le bon composant pour lire(8) les données se trouvant dans le Graphe RDF. Dès que les données sont lues, la réponse à la requête est envoyée à l'utilisateur via le chemin inverse (de 9 à 15).

## 13.3 Réalisation de l'architecture KWeSSi avec le Framework PF4J

Nous rappelons que notre application est « open source » et est développé en langage Java. Pour la suite nous conseillons d'avoir un prérequis en Java et en programmation orientation objet. Dans cette section nous allons expliquer la structure Java de l'architecture que nous venons de présenter. Nous allons insister particulièrement sur le Framework de cette architecture et comment y intégrer un plugin. La structure complète de l'architecture se présente ainsi :

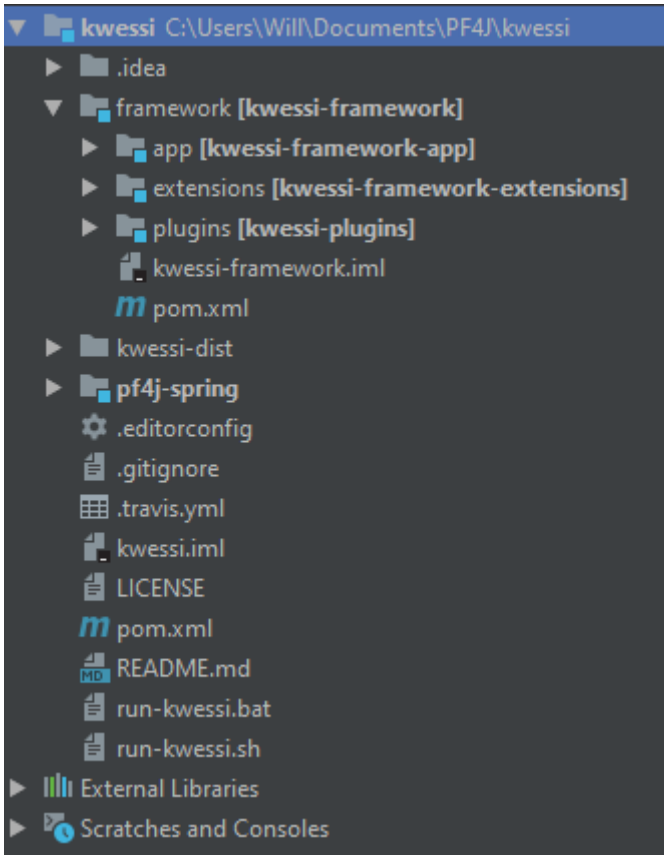



FIGURE 13.2 – structure du PFC KWeSSi.

La structure de l'application comprend 7 niveaux qui sont généralement des modules indépendants :

- **kwessi** : c'est le module principal qui contient tous les autres composants de l'application tels que : les configurations globales, les librairies englobantes, les dépendances vers les autres modules, etc.

- **framework** : c'est le module dans lequel nous définissons les services, les contrôleurs, l'interface graphique, le schéma OWL du Framework de base de notre application, etc.

- **app** : c'est un module du «framework» où on définit concrètement la logique business, les services génériques, les contrôleurs, l'interface graphique, etc.

- **extensions** : c'est aussi un sous-module de « framework ». on doit définir à ce niveau les points d'extension. Ce sont les endroits de l'application qui seront étendu par les plugins pour ajouter dynamiquement de nouvelles fonctionnalités à l'application. Dans notre cas, nous avons un seul point d'extension qui consiste à l'ajout de nouveaux réseaux social. Généralement un point d'extension est concrètement définit par une interface Java qui expose des méthodes qui seront appelées ou implémentées de manière personnalisée par des plugins.

- **plugins** : c'est un sous-module de «framework» dont sa particularité est qu'il peut accepter dynamiquement d'autres modules. Même quand l'application est entrain de « tourner » on peut mettre à cette endroit de nouveaux plugins et l'application va le détecter automatiquement et va démarrer le fichier zip correspondant qui se trouve dans le dossier kwessi-dist . Il s'agit là du « cœur » de PF4J.

- **kwessi-dist** : C'est le répertoire des « builds ». Quand l'application trouve des plugins il les «build» et les met dans ce répertoire pour permettre au module principal de voir lesdits plugins. C'est ici que nous allons mettre le schéma de OWL de base de l'application (base de connaissances génériques). Une illustration est la suivante :

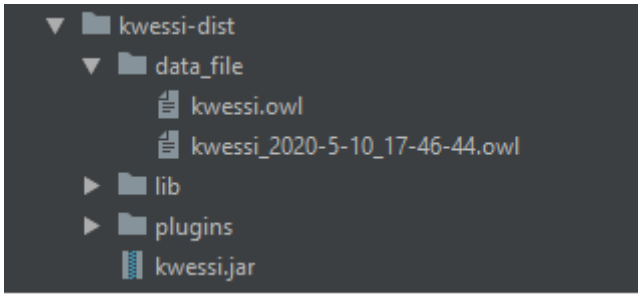



FIGURE 13.3 – position du schéma OWL générique de KWeSSi.

En fait le schéma initial est « kwessi.owl ». Mais quand ont ajout un nouveau réseau social il vient avec son propre fichier OWL qui définit les champs, les contraintes sur les champs de ce RS. Le service de merge que nous avons développé va fusionner automatiquement le fichier «kwessi.owl» avec celui du RS qu'on vient d'ajouter. Cette fusion crée un nouveau fichier au nom de « kwessi.owl » et l'ancien fichier prend le nom « kwessi_Année-mois-jour_Heure-seconde-tierce.owl ». Cette technique est une mesure de sécurité pour prévoir des cas de corruption de données.

- **pf4j-spring** : contient du code source intégrer au Framework PF4J. Par exemple le code pour la construction des injections de dépendances pour détecter et charger automatiquement de nouveaux plugins, un exemple de code pour vous expliquer comment créer votre propre plugin dans le module plugins.

### 13.3.1 Vu détaillée de la couche framework de KWeSSI

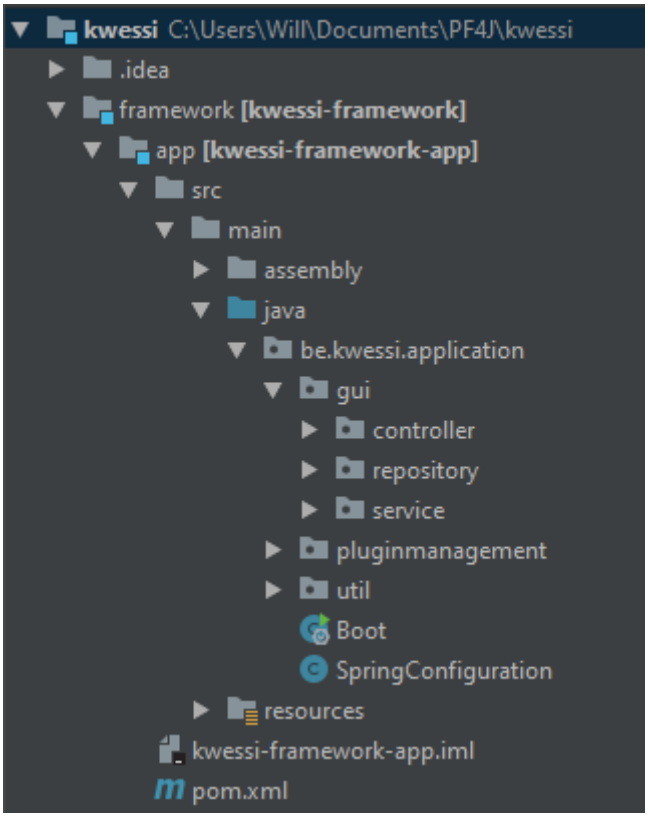



FIGURE 13.4 – couche « framework » de KWeSSI détaillée.

Dans la couche ci-dessus nous écrivons tout ce qui est générique à notre plateforme collaborative, en particulier les services. Ces services peuvent être utilisés par chaque plugin en passant les définitions que nous allons mettre dans le module extension. Un cas concret va être expliqué plus bas.

### 13.3.2 La couche extensions de KWeSSi

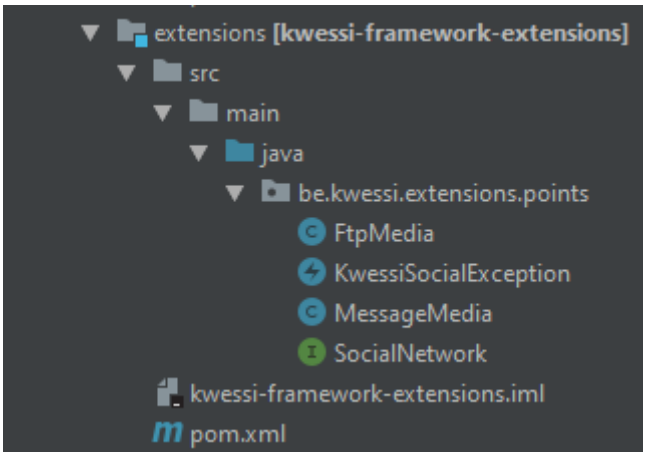



FIGURE 13.5 – les points d'extension.

Cette couche est l'endroit névralgique pour la mise en place d'une architecture plugin dans le Framework PF4J. Il est primordial de réfléchir minutieusement aux services qu'on doit exposer au plugin. Si on donne accès à tous et à n'importe quoi au plugin, vu que ce sont des applications qui sont développées par des personnes externes dont nous ne pouvons pas anticiper sur leurs réelles intentions, elles peuvent via de nouveaux codes qu'elles introduisent créer de graves problèmes de sécurité à l'application. Pour notre situation nous respectons deux recommandations du Framework PF4J qui sont :

- **Toujours utiliser des objets intermédiaires entre les plugins et la couche « framework »**. L'objet de notre business le plus important est le contenu et nous le manipulons le plus. Nous avons créé une classe java « MessageMedia » pour emballer les contenus que les utilisateurs envoient avant de le transmettre aux plugins pour qu'ils les publient sur les réseaux sociaux. Les attributs de la classe « MessageMedia » sont : le titre du contenu, le message du contenu, la photo, la vidéo. Ce sont les champs qui se trouvent sur presque tous les formulaires de publication des réseaux sociaux. Avec la classe « MessageMedia » nous définissons un objet générique que chaque plugin peut hériter et compléter par les champs qui lui sont propres.

- **Toujours définir les points d'extension via les interfaces** : notre point d'extension est représenté par l'interface « SocialNetwork ». Son rôle est d'exposer les services qui seront personnalisés par chaque plugin. Cela permet au plugin d'être indépendant et peut réécrire les services selon ses propres besoins sans être obligé de modifier le code source du Framework de base de l'application. Nous avons ici quatre services : getNameSocialNetwork, publish, merge(), merge(String path).

Sur l'image suivante, le service getNameSocialNetwork sert à récupérer le nom d'un RS dans le fichier OWL du RS qu'on veut ajouter. Le service publish permet de publier un contenu sur les réseaux sociaux. consumerKey, consumerSecret, accessToken, accessTokenSecret sont des crédentials de la plupart de RS pour la publication de contenus à partir d'une plateforme externe comme notre application. Si un RS n'utilise pas une partie ou tous ces crédentials, le plugin de ce RS va redéfinir le service sans les préciser. Les services merge() et merge(String path) sont utilisés pour fusionner deux schémas OWL comme nous l'avons expliqué plus haut. Si le schéma du RS à ajouter est placé manuellement dans l'endroit réserver, on utilise le service merge() et si par contre ledit fichier est importer depuis la GUI de l'application, on utilise merge(string path).

Il est indispensable que le point d'extension hérite de la classe EntensionPoint sinon les plugins ne pourront pas utiliser correctement les services.

```java
package be.kwessi.extensions.points;

import org.pf4j.ExtensionPoint;

/**
 * @author Effayong et Leuni
 */
public interface SocialNetwork extends ExtensionPoint {

    String getNameSocialNetwork();

    void publish(MessageMedia message, String consumerKey,
                 String consumerSecret,
                 String accessToken,
                 String accessTokenSecret);
    boolean merge();
    boolean merge(String path);
}
```

FIGURE 13.6 – services du point d'extension.

### 13.3.3 La couche plugins

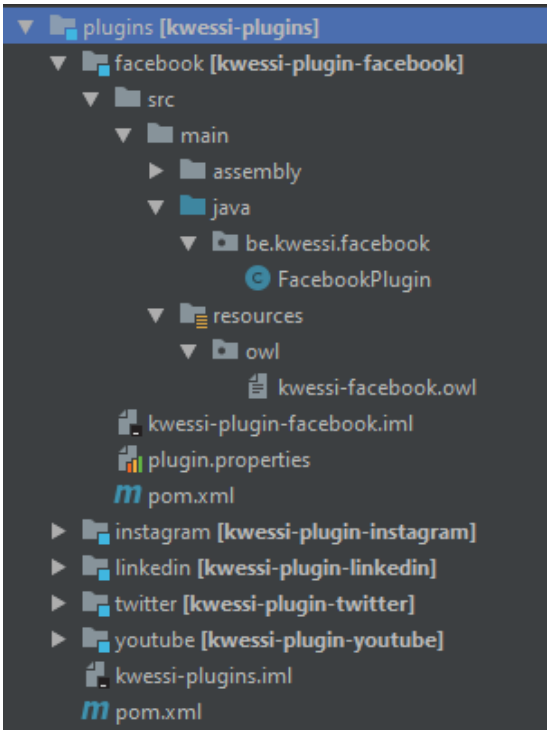



FIGURE 13.7 – les différents plugins.

Sur l'image ci-dessus, chaque plugin correspond à un RS ajouté. Pour l'instant on a les plugins avec pour nom facebook, instagram, linkedin, twitter et youtube. Chacun de ces plugins correspond à un module indépendant de tous les autres composants de l'application. Il est absolument important de mettre dans chaque plugin une classe qui hérite de classe Plugin du Framework PF4J comme ceci. « public class FacebookPlugin extends Plugin ». La classe « FacebookPlugin » qui représente le plugin pour facebook peut réécrire les méthodes start() et stop() qu'elle a héritées de la classe « Plugin ». Elle pourrait aussi réécrire les services que nous avons exposés dans « SocialNework ». Pour cela, elle peut définir une classe interne qui implémente le service « SocialNework ». Cette classe interne doit obligatoirement avoir l'annotation @Extension pour signaler qu'elle veut étendre un point d'extension, qui est « SocialNework » dans notre cas.

Le dossier OWL contient le schéma de base de connaissances pour Facebook. Dans l'architecture de notre application ce répertoire doit être créé par celui qui développe un plugin et le fichier à merger doit se trouver exactement à cet endroit. Pour plus d'explications sur comment concevoir un plugin PF4J, nous conseillons de lire son site internet [1] . PF4J utilise la classe AbstractPluginManager pour gérer les différents états (start, stop, delete) d'un plugin.

### 13.3.4 Mécanisme d'intégration dynamique des plugins à l'application principale

Dans la JVM (Java Virtual Machine), les chargeurs de classe sont responsables de charger dynamiquement les classes Java pendant l'exécution d'une application [2]. Ces chargeurs de classe font partie de l'environnement d'exécution Java. Lorsque la machine virtuelle Java demande une classe, le chargeur de classe essaie de localiser la classe et de charger la définition de classe dans le « runtime » (file d'exécution) en utilisant le nom de la classe complet. La méthode java.lang.ClassLoader.loadClass est responsable du chargement de la définition de classe dans le runtime. Il essaie de charger la classe en fonction d'un nom complet. Si la classe n'est pas déjà chargée, elle délègue la demande au chargeur de classe parent. Ce processus se produit récursivement.

PF4J utilise la classe PluginClassLoader pour charger des classes à partir de plugins. Ainsi, chaque plugin disponible est chargé à l'aide d'un PluginClassLoader différent. Une instance de PluginClassLoader doit être créée par le gestionnaire de plug-ins, pour chaque plug-in disponible. Par défaut, ce chargeur de classe est l'un des derniers parent de ClassLoader - il charge les classes à partir des jars (fichier build) du plugin avant de les déléguer au chargeur de classe parent. Par défaut aussi, PluginClassLoader utilise la stratégie ci-dessous lorsqu'une demande de classe de chargement est reçue via la méthode loadClass (String nomClasse) :

- Si la classe est une classe système (nom de la classe commençant par java.), il délègue le processus au chargeur système.

- Si la classe fait partie du moteur du plugin (nom de la classe commençant par org.pf4j), il utilise le chargeur de la classe parent (ApplicationClassLoader en général)

- Ou il va essayer de charger les plugins en utilisant l'instance actuelle de PluginClassLoader.

- Si le PluginClassLoader actuel ne peut pas charger la classe, il va essayer de déléguer à PluginClassLoaders des dépendances du plugin.

- Ou il va déléguer le chargement de classe au chargeur de classe parent.

```
new DefaultPluginManager() {
    @Override
    protected PluginClassLoader createPluginClassLoader(Path pluginPath, PluginDescriptor pluginDescriptor) {
        ;
        return new PluginClassLoader(pluginManager, pluginDescriptor, getClass().getClassLoader(),  parentFirst: true);
    }
};
```

FIGURE 13.8 – exemple de PluginClassLoader.

Si nous voulons savoir quel plugin a chargé une classe spécifique, nous pouvons utiliser : « pluginManager.whichPlugin (MyClass.class) ».

PF4J utilise par défaut un chargeur de classe distinct pour chaque plugin mais cela ne signifie pas que nous ne pouvons pas utiliser le même chargeur de classe (probablement le chargeur de classe d'application) pour tous les plugins. Si notre application nécessite ce cas d'utilisation, ce que nous devons faire est de renvoyer le même chargeur de classe à partir de PluginLoader.loadPlugin :

1. Création d'un plugin avec PF4J : https ://pf4j.org/doc/plugins.html (consulté le 04/03/2020)
2. Chargement dynamique des classes : https ://pf4j.org/doc/class-loading.html (consulté le 04/03/2020)

```java
public interface PluginLoader {

    boolean isApplicable(Path pluginPath);

    ClassLoader loadPlugin(Path pluginPath, PluginDescriptor pluginDescriptor);

}
```

FIGURE 13.9 – interface PluginLoader.

Si on utilise DefaultPluginManager, on peut choisir de remplacer DefaultPluginManager.createPluginLoader et / ou DefaultPluginLoader.createClassLoader.

### 13.3.5 Algorithme de merge ou de fusion des schémas OWL.

Dans les sections précédentes nous avons commencé à expliquer comment fonctionne notre algorithme de merge. Les explications complémentaires et suffisantes se trouvent sur la figure suivante. Cette méthode s'appuie sur l'API Jena. Par exemple la classe ModelFactory est une classe de la libraire de cette API.

```java
/**
 *  This method receive a file to merge with the main app file (kwessi.owl)
 *  First time, we read file (parameter) in the model (modelFileParameter).
 *  Second time, we retrieve the main file and reading in the model (modelMain).
 *  Third time, we create a variable "Writer" to write in the main file.
 *  Fourth time, we use the method "add" of the modelMain to merge with the modelFileParameter.
 *  Last time, we call the method "verifyModelOntology" to verify the consistency of our file.
 * @param file file to merge
 * @throws IOException
 */
public static void mergeOWLFile(File file) throws IOException {
    Model modelFileParameter = ModelFactory.createDefaultModel();
    InputStream inputStream = new FileInputStream(file);
    if (inputStream == null) {
        throw new IllegalArgumentException("File: "
            + " not found");
    }
    modelFileParameter.read(inputStream, s: "", s1: "RDF/XML");
    Model modelMain = ModelFactory.createDefaultModel();
    String inputFileMain = HandleFileOWL.getMainFile();
    InputStream inputStreamMain = FileManager.get().open(inputFileMain);
    if (inputStreamMain == null) {
        throw new IllegalArgumentException("File: "
            + " not found");
    }
    modelMain.read(inputStreamMain, s: "", s1: "RDF/XML");
    HandleFileOWL.saveFile(modelMain);
    try {
        Writer w = new FileWriter(HandleFileOWL.getMainFile());
        modelMain.add(modelFileParameter).write(w);
        verifyModelOntology(HandleFileOWL.getMainFile());
        w.close();
        logger.info("Merge done !");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

FIGURE 13.10 – algorithme de merge de 2 schémas OWL.

### 13.3.6 Algorithme de publication de contenu sur twitter

L'algorithme commence par vérifier que le contenu (ou message) est présent dans le schéma OWL du Framework de notre plateforme collaboratrice KWeSSi. Puis il va créer une connexion à Twitter avec les tokens passés en paramètre. Si la connexion se passe bien, on charge le contenu dans le statut et on appelle la méthode updateStatus(statusUpdate) de l'API twitter pour publier.

```
public void publish(MessageMedia message,
                    String consumerKey,
                    String consumerSecret,
                    String accessToken,
                    String accessTokenSecret) {
    if(message.getIdMessage().toLowerCase().equals(getNameSocialNetwork().toLowerCase())) {
        Twitter twitter = twitterCreator.getTwitterTemplate(consumerKey,
            consumerSecret, accessToken, accessTokenSecret);
        try {
            StatusUpdate statusUpdate = new StatusUpdate(message.getContent());
            if (message.getImage() != null && !message.getImage().isEmpty()) {
                File file = new File(message.getImage());
                statusUpdate.setMedia(file);
            }
            twitter.updateStatus(statusUpdate);
        } catch (TwitterException ex) {
            logger.info("Unable to tweet" + message.getContent());
            ex.printStackTrace();
        }
    }
}
```

FIGURE 13.11 – algorithme de publication sur Twitter.

Cet algorithme est propre à twitter. Pour chaque réseau social la stratégie à utiliser lui est propre. C'est la raison pour laquelle nous laissons la possibilité à chaque réseau social d'implémenter sa logique de publication.

## 13.4 Quelques règles de validation du fichier OWL à merger.

| **Règle 1: l'URI** | | | |
|---|---|---|---|
| URI du schéma à importer | même que le schéma de base | | |
| **Règle 2: les classes génériques** | | | |
| **Schéma de base** | **Schéma à importer** | **Obligatoire** | **Signification** |
| SocialNetwork | SocialNetwork | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| Account | Account | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| Publication | Publication | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| Content | Content | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| Video | Video | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Resolution | Resolution | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| OtherFile | OtherFile | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Message | Message | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Image | Image | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Gif | Gif | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Extension | Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Emoji | Emoji | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |

L'ensemble des règles de validation se trouve en pièce jointe. Si au moment de merger (ou importer) le fichier OWL pour ajouter le réseau social avec le fichier OWL du Framework de base, l'une de ces règles n'est pas respectée, un message d'erreur est renvoyé à l'utilisateur et le merge est refusé.

# Chapitre 14

# Prototype graphical user interface (GUI)

## 14.1 Écran de connexion à la PFC KWeSSi

Chaque utilisateur pourra se connecter à l'application en fournissant son email et son mot de passe. S'il l'a oublié il peut toujours créer un nouveau mot de passe en cliquant sur « Mot de passe oublié ? ».

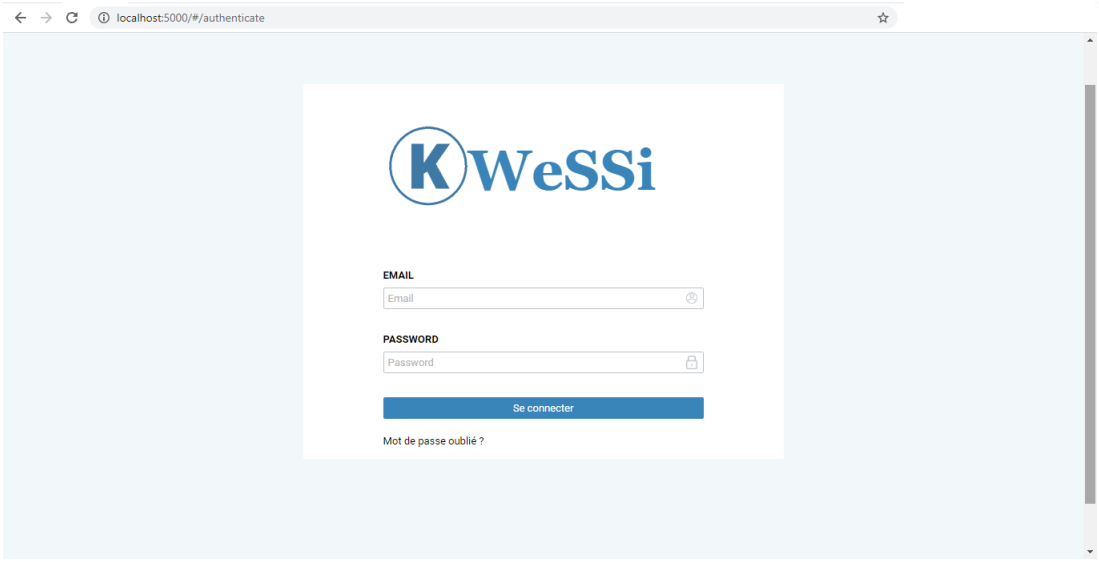



FIGURE 14.1 – écran de login de KWeSSi.

## 14.2 Écrans de gestion des utilisateurs et des rôles

Cet écran montre un administrateur qui est entrain d'ajouter un utilisateur de nom «Dipanda» avec un rôle «Membre». Un mail est envoyé à l'utilisateur avec un lien pour activer son compte. S'il ne le fait pas, il ne pourra pas se connecter.

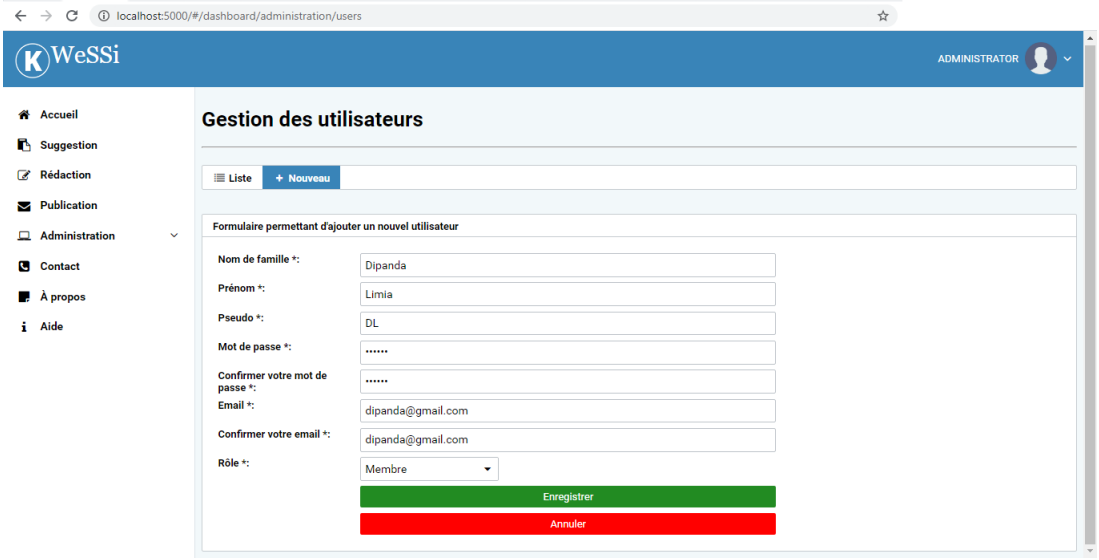



FIGURE 14.2 – écran d'ajout d'un utilisateur.

Dès que l'utilisateur est ajouté, l'administrateur peut encore changer son rôle en cliquant sur l'icône de l'œil qui se trouve devant son nom.

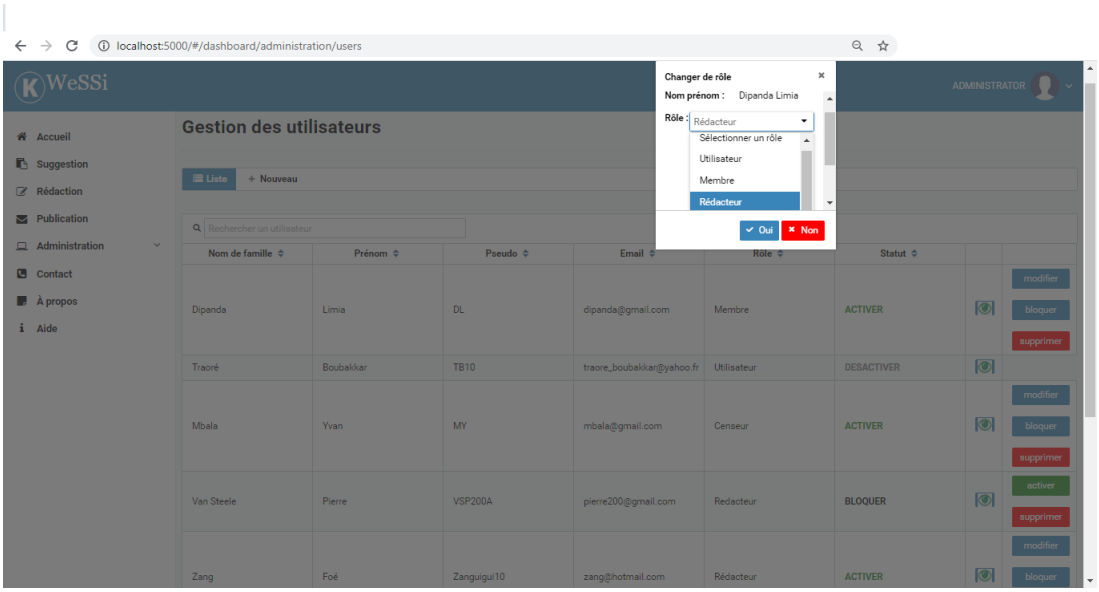



FIGURE 14.3 – écran de gestion des rôles.

## 14.3 Écrans de rédaction et de publication de contenus

### 14.3.1 Digramme de workflow

Ce diagramme montre les différentes actions qui sont permises lorsqu'un contenu se trouve dans un état bien précis. Au point d'entrée, le modérateur prend un contenu et veut le modérer. S'il refuse le workflow se termine. S'il la proposition est acceptée, elle passe à l'état **A Rédiger**. À ce niveau un rédacteur peut prendre le contenu pour le rédiger. Tant qu'il n'a pas fini, il peut enregistrer son travail et dans ce cas le contenu passe à l'état **Enregistré**. Dans cet état, le rédacteur a le choix entre modifier, supprimer, attribuer le contenu à un autre rédacteur ou valider le contenu. S'il supprime ou attribue le contenu qu'il avait écrit, le sujet du contenu reste et prend l'état **A Rédiger**. Quand il finit de rédiger le contenu, il le valide et le contenu passe à l'état **Validé**. À ce stade un censeur peut récupérer le contenu pour le censurer. Au moment de la censure il peut bloquer, modifier, supprimer, ou accepter le contenu que le rédacteur a rédigé. Si le contenu est bloqué, il passe à l'état **Bloqué**, si le contenu est supprimé le workflow prend

fin, si le contenu est accepté il passe à l'état **Publié**. Dès que le contenu est publié il y a la possibilité de le récupérer, le customiser. Dans ce cas le contenu est renvoyé à la rédaction où on peut y ajouter d'autres informations avant de le publier sur un autre RS. Le diagramme de workflow se présente alors comme suit.

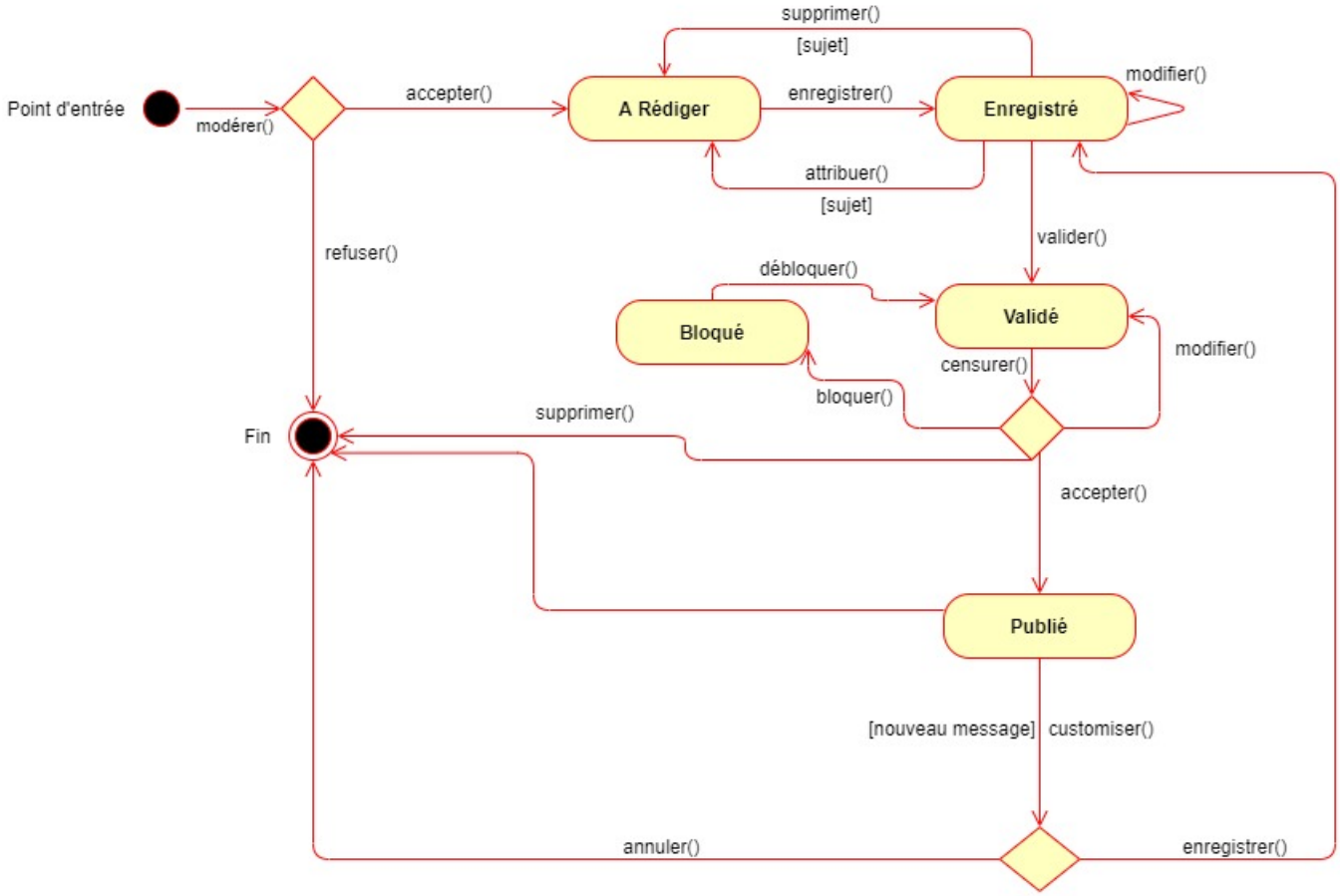



FIGURE 14.4 – Diagramme de workflow de publication d'un contenu.

### 14.3.2 Les écrans

#### A) Suggestion de contenu

L'utilisateur choisit les réseaux sociaux sur lesquels il souhaite qu'on y publie sa suggestion. Dans l'exemple qui est sur l'écran, le titre de sa suggestion est « La Belgique face au COVID-19 » et il aimerait que le contenu qui sera rédigé à propos de ce titre soit publié sur « Twitter » et « LinkedIn».

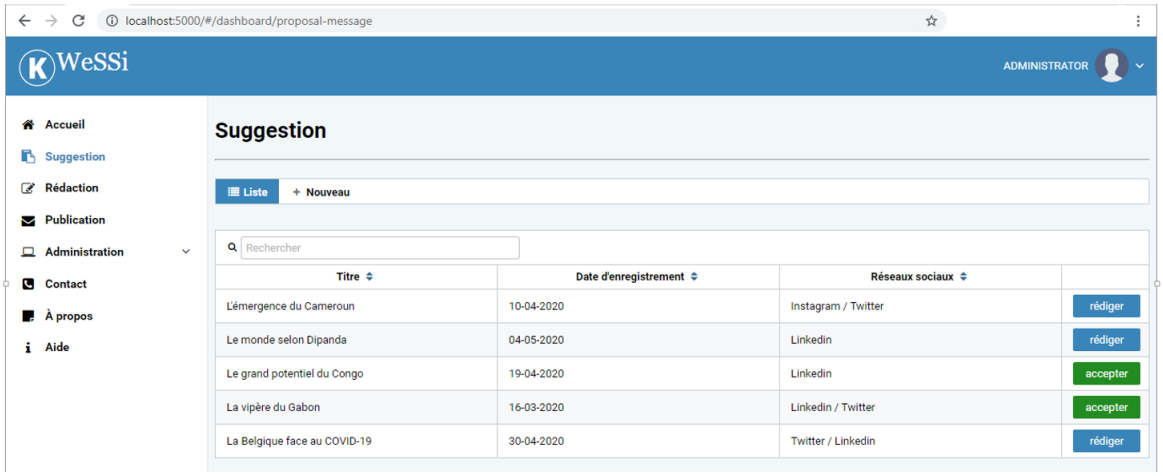



FIGURE 14.5 – écran de suggestions de contenus enregistré

Dès qu'il enregistre sa suggestion, un modérateur pourra alors décider si la suggestion peut être rédigé ou pas.

### B) Écrans de rédaction de contenus

Un rédacteur choisit le message qui vient d'être suggéré et le rédige. Dans une première page il met un texte.

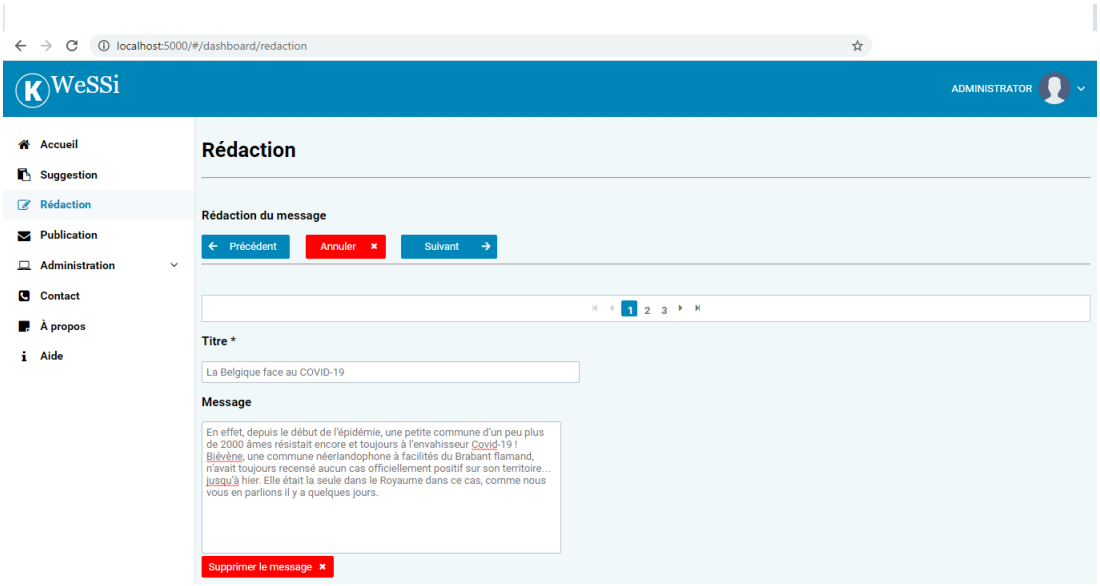



FIGURE 14.6 – écran de rédaction d'un contenu (page 1).

Dans la page 2 le rédacteur est invité à fournier d'autres informations comme la photo, la vidéo, l'audio, ou un autre fichier.

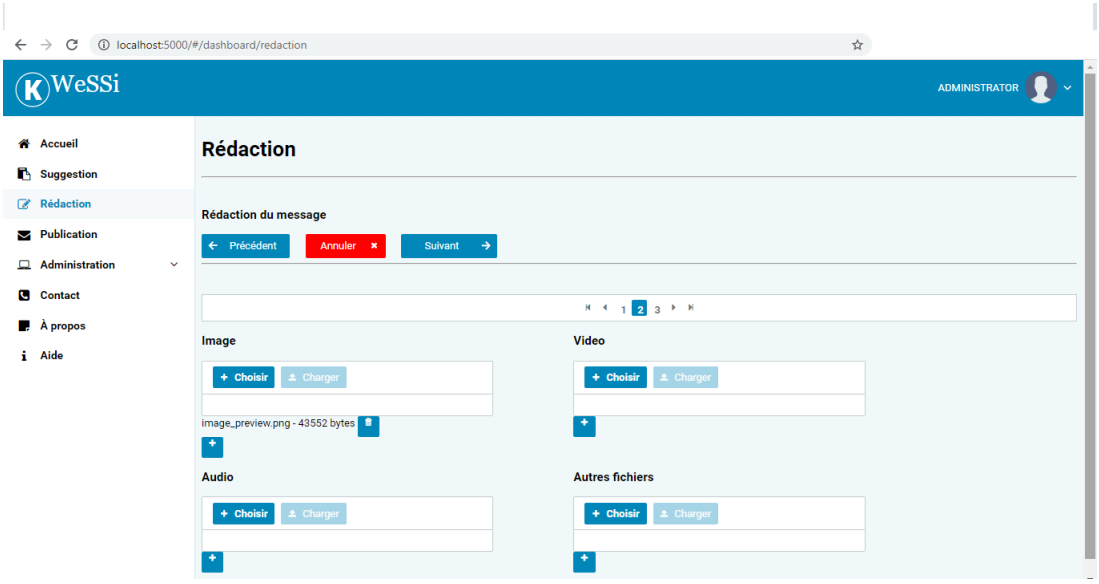



FIGURE 14.7 – écran de rédaction d'un contenu (page 2).

Sur la page 3 le rédacteur peut ajouter d'autres donnés sur son contenu tel qu'un gif, emoji, etc.

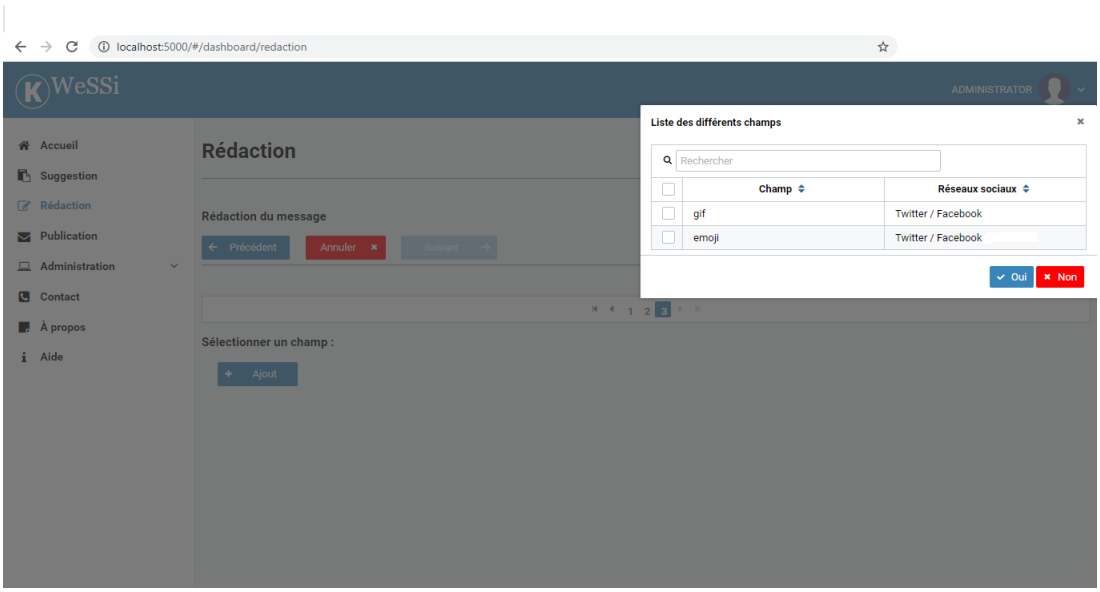



FIGURE 14.8 – écran de rédaction d'un contenu (page 3).

La dernière page montre le récapitulatif du contenu rédigé. Le rédacteur peut enregistrer son contenu ou cliquer sur « précédent » pour le modifier.

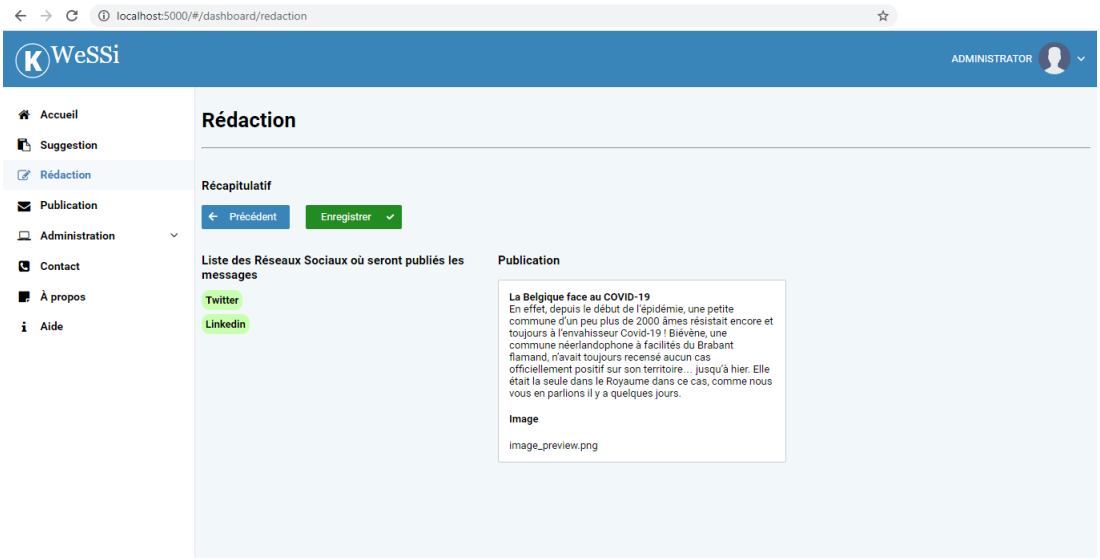



FIGURE 14.9 – écran de rédaction d'un contenu (page 4).

Quand le contenu est enregistré et validé par un rédacteur, un censeur peut alors le publier et on a l'écran suivant. Lors de l'enregistrement, si la taille du texte, de l'image, etc. est trop grand pour n'importe quel RS, le raisonneur des ontologies va renvoyer un message d'erreur et le rédacteur devra ajuster son contenu.

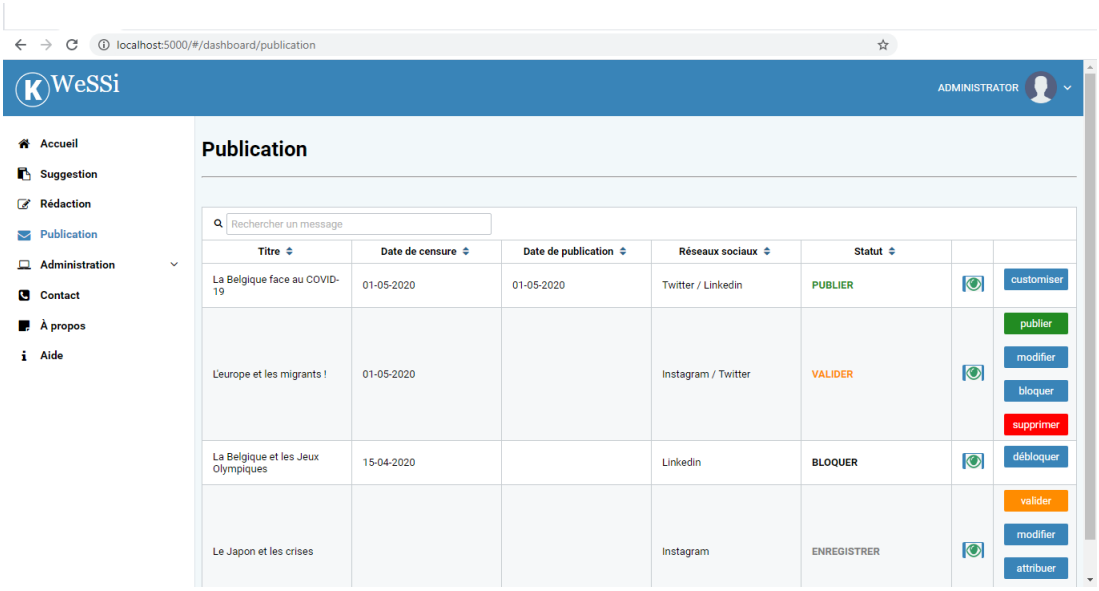



FIGURE 14.10 – écran de rédaction d'un contenu (page 5).

Le contenu a été publié simultanément sur Twitter et LinkedIn. Sur Twitter il se présente ainsi :


FIGURE 14.11 – contenu publié sur Twitter.

Dès que le contenu est publié par un censeur on peut le customiser. Pour cela il faudrait cliquer sur le bouton «customiser» qu'il y a devant le contenu publié. Cette action a pour effet de récupérer le contenu et de le renvoyer vers l'écran de rédaction.

Si on souhaite voir les détails d'un contenu, il suffit de cliquer sur l'œil qu'il y a devant ledit contenu et on obtient l'écran ci-dessous.

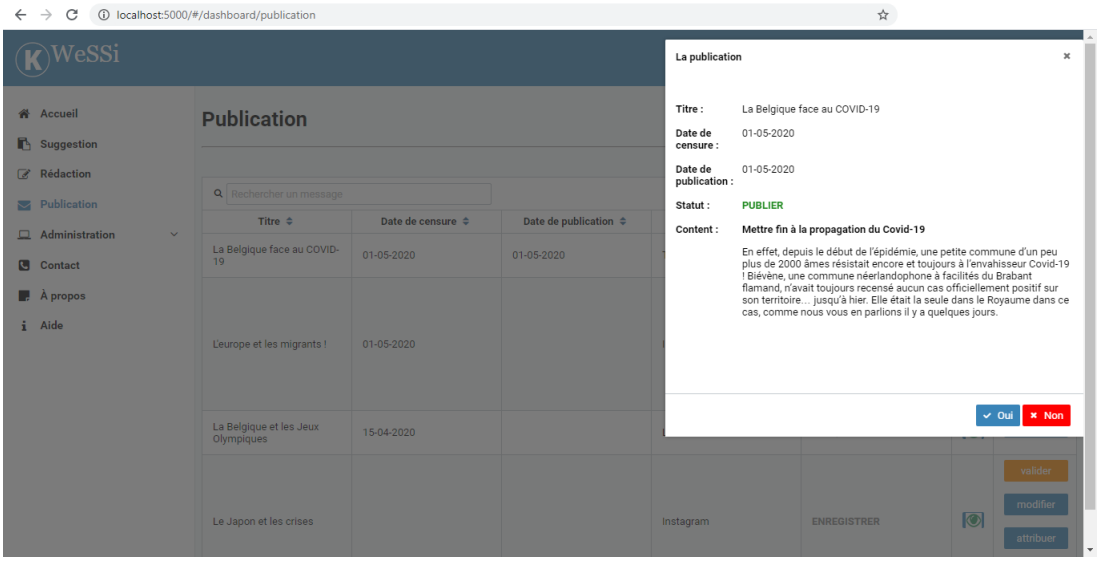



FIGURE 14.12 – écran de détails d'un contenu.

## 14.4 Écrans d'ajout d'un réseau social

Dans cet écran l'administrateur doit d'abord développer le plugin pour ajouter le réseau social tel que nous l'avons expliqué plus haut. S'il n'a pas encore intégré le fichier OWL contenant le réseau social, ses champs et ses contraintes, il peut le faire via l'interface ci-dessous.

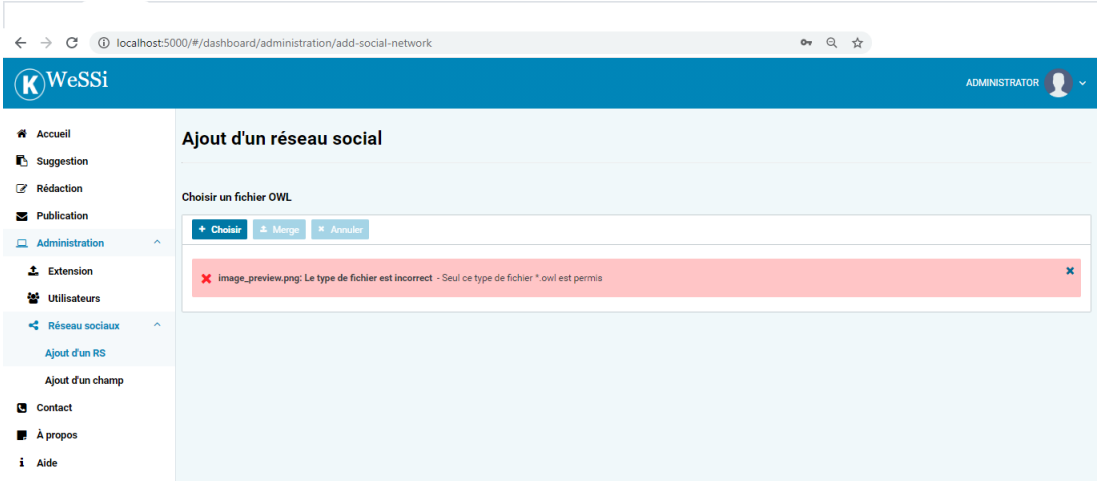



FIGURE 14.13 – écran d'ajout d'un RS.

Dès que le fichier est choisi, l'administrateur peut cliquer sur «Merge» pour «merger» le fichier choisi avec celui du Framework de base. Lors de cette opération le raisonneur de OWL va vérifier toutes les règles de validation que nous avons présentées. On va également s'assurer que le fichier est bien un fichier OWL sinon une erreur est renvoyée comme vous le voyez sur l'image précédente. On va aussi récupérer le nom du réseau social dans le fichier choisi, puis vérifier que le plugin avec ce nom existe dans l'application avant de faire le «merge». Dans le cas contraire, on va renvoyer un message d'erreur. L'écran suivant présente les plugins qu'il y a dans l'application.

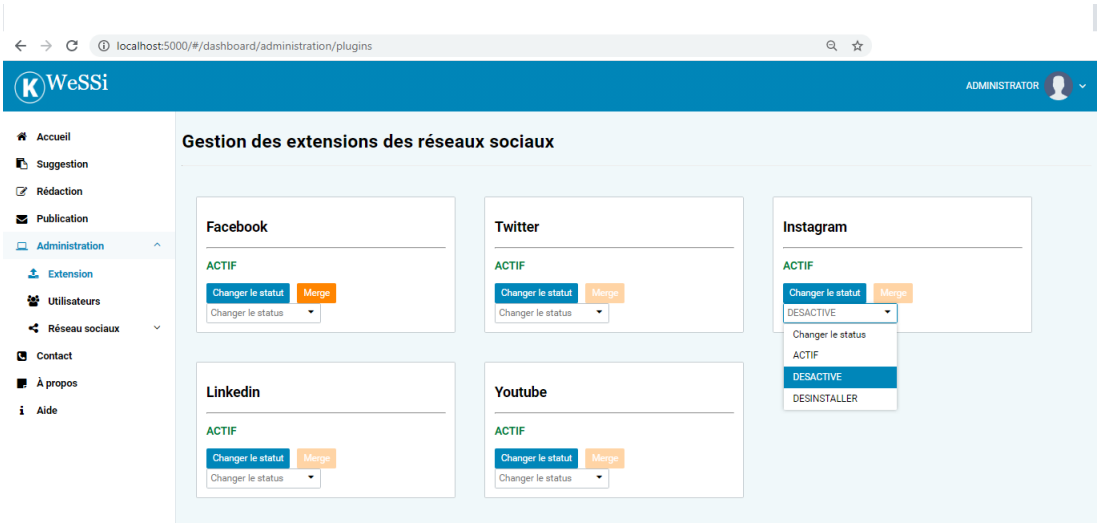



FIGURE 14.14 – liste des plugins de l'application.

Dans une autre situation, celui qui a développé le plugin pour ajouter un RS peut placer le fichier OWL du RS à ajouter directement dans le répertoire OWL de son plugin. C'est valable pour le plugin «Facebook» dans le dernier écran. Dans ce cas il doit encore cliquer sur «Merge» pour «merger» le fichier OWL de Facebook avec celui du Framework de base. Ici on ne vérifie plus que le plugin existe vu que c'est le cas, mais le raisonneur devra tout de même vérifier les règles de validations sur le fichier OWL de Facebook avant de le «merger».

## 14.5 Écrans de réseaux sociaux et leurs comptes

Sur l'écran on voit la liste de quelques réseaux sociaux avec chacun son nombre de comptes et son nombre de messages.

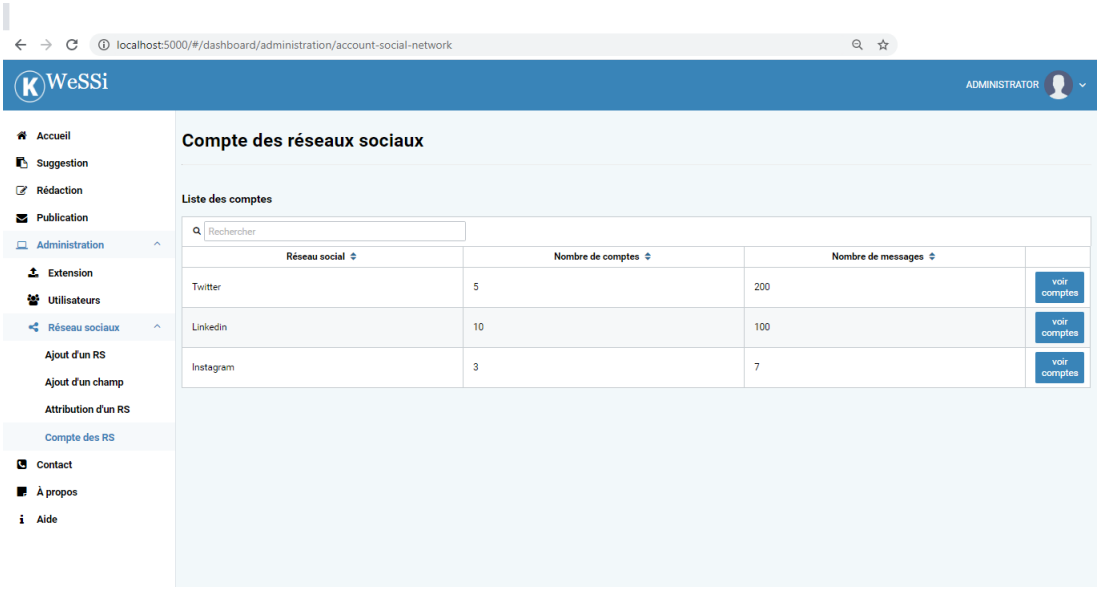



FIGURE 14.15 – écran RS et comptes.

Si on clique sur le bouton «Voir comptes» cela donne l'occasion de voir tous les comptes du réseau social sur lequel on a cliqué. Présentement nous avons cliqué sur le réseau social «LinkedIn» et l'écran suivant s'affiche.

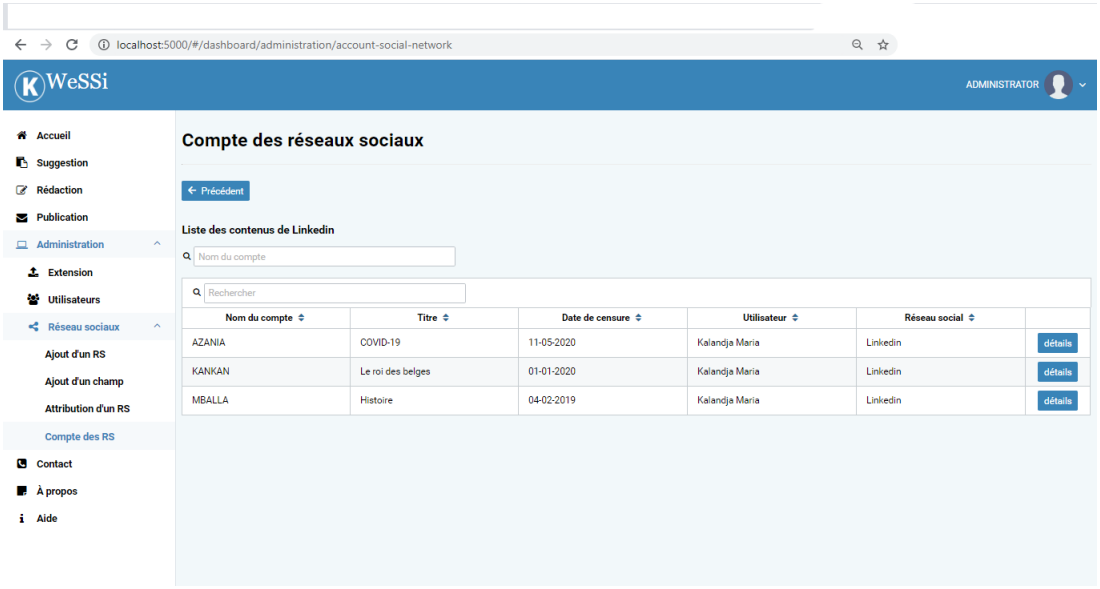



FIGURE 14.16 – écran des comptes des RS et leurs contenus.

L'écran ci-dessus montre quelques contenus qui ont été publiés sur certains comptes du réseau social «LinkedIn». Si on clique sur le bouton «détails» de la deuxième ligne de cet écran on va voir les détails du contenu tel que cela se trouve sur l'écran ci-dessous.

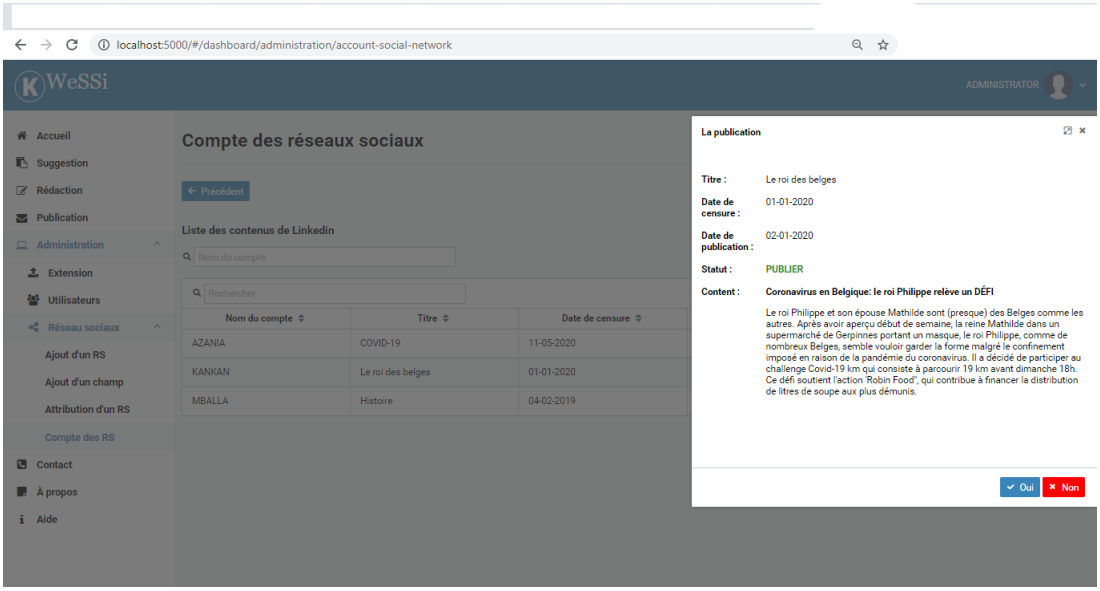



FIGURE 14.17 – écran des détails d'un compte.

## 14.6 Écran d'ajout d'un champ et ses caractéristiques

Pour ajouter un champ à un RS on crée un fichier OWL dans lequel on précise le nom du RS, les champs à ajouter et les caractéristiques qu'il y a sur chaque champ. Puis avec l'écran qui suit, on va importer le fichier créé et faire le «Merge».

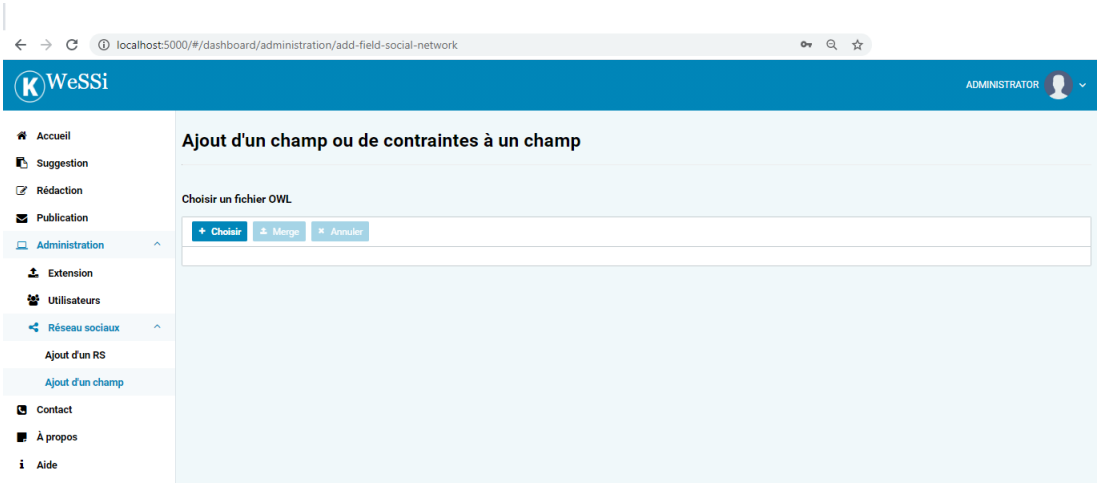



FIGURE 14.18 – écran d'ajout d'un champ à un RS.

# Chapitre 15

# Orientations techniques

## 15.1 Connexion par email plus mot de passe ou via les tokens d'accès

Nous constatons après étude des trois solutions que nous avons présentées dans le «chapitre 4», qu'il existe deux manières de se connecter sur les réseaux sociaux pour y publier du contenu. Une avec email plus le mot de passe (utilisée par Sprout Social et Agorapulse), et l'autre via les tokens d'accès des API (API Facebook, Twitter, LinkedIn, etc.), utilisée par le CMS Joomla.

La méthode de connexion avec un email et un mot de passe est plus simple et est connue sur le nom de «Basic authentication [1]» . Nous avons juste besoin de l'URL de la page de connexion du RS, combiné avec notre email et le mot de passe d'un compte sur ce RS pour se connecter et y publier du contenu. Étant donné que les comptes appartiennent à des utilisateurs, il faudrait qu'ils aient suffisamment confiance à notre plateforme pour accepter de nous fournir leurs emails et leurs mots de passe. Pour cela, il convient d'intégrer dans la plateforme KWeSSi une politique de sécurité efficace, et bien les expliquer pour que chaque utilisateur de la plateforme se sente rassurer.

Avec les tokens d'accès, on est un peu embêté par le fait qu'ils sont mise à jour régulièrement (au minimum chaque mois). Pour résoudre le problème, on pourrait mettre en place un mécanisme pour récupérer régulièrement les nouveaux tokens que génèrent les APIs des RS. La méthode des tokens s'appuie sur OAuth [2] [24] qui est un protocole de «délégation d'autorisation». Jusqu'en 2015, OAuth n'intégrait pas encore le rafraîchissement des tokens dans OAuth1.0. Dans sa version OAuth2.0, le rafraîchissement des tokens d'accès a été intégré [3]. Cela est un grand apport en terme de sécurité. Tout de même, ce protocole présente encore certaines failles qui sont relevées sur le site internet http ://www.bubblecode.net/fr/2016/01/22/comprendre-oauth2/ (consulté le 21 mars 2020).

Comment faire pour choisir entre les deux méthodes de connexion ? La réponse n'est pas évidente dans la mesure où chaque méthode répond à notre besoin. En pratique, nous avons constaté que Sprout Social et Agorapulse utilise la méthode par émail et un mot de passe. En testant cette méthode nous l'avons trouvé plus efficace par rapport aux tokens d'accès. Chaque fois, le CMS Joomla que nous avons testé pour nous connecter aux réseaux sociaux présentait régulièrement des instabilités de connexion dues au faite que les tokens se renouvelaient. Nous avons aussi constaté ce problème dans Wordpress, Drupal que nous avons aussi testé sans toutefois les présenter dans ce mémoire vu qu'ils sont semblables à Joomla. Techniquement, la connexion via tokens est un peu difficile à mettre en place et à maintenir. Dans ce mémoire nous optons pour l'approche de connexion via les tokens d'accès car elle est plus sécurisée que celle de l'email plus mot de passe et la sécurité est importante pour nous.

1. Plus d'information sur : https ://developer.twitter.com/en/docs/basics/authentication/overview (consulté le 10/01/2020)
2. Site internet : https ://oauth.net/2/ (consulté le 10/01/2020)
3. Plus d'information : https ://oauth.net/2/grant-types/refresh-token/ (consulté le 10/01/2020)

## 15.2 Sécurité

Un plugin est une application venue de l'extérieur qui s'exécute à l'intérieure d'une autre application (la principale) qui la reçoit. Un plugin a accès au code source, aux données de l'application principale, ce qui pose un problème de sécurité parce que le plugin pourrait avoir pour intention de nuire en volant les données, en paralysant le système global, etc. Pour faire face à ce problème de sécurité plusieurs solutions sont possibles.

#### Limiter l'accès aux données

Les données disponibles pour le plugin peuvent être limitées ; par le biais du programme donnant les valeurs d'initialisation et / ou les paramètres permettant au plugin d'effectuer son travail. Cela peut être assez sécurisé en raison des informations restreintes, mais on peut facilement courir le risque de fournir trop peu d'informations (même si cela peut être étendu avec le temps) ou trop d'informations (car on peut fournir beaucoup d'informations et le plugin ne les utilise pas).

#### Bien utiliser les valeurs de retour

Il faut éviter des appels directs à partir du plugin. Cela est mieux, on augmente le risque de sécurité en fonction de ce qu'on fait avec les valeurs. Par exemple, ne jamais prendre une valeur de retour et l'ajouter à une chaîne SQL, cela favorise des attaques par injection SQL ou une inconsistance de données pour ce qui est de OWL.

#### Les backups

Le backup est une opération qui consiste à dupliquer et à mettre en sécurité les données contenues dans un système informatique dans le but de relancer ce système en cas de panne. Nous l'avons utilisé pour faire des copies de sauvegarde des fichiers OWL à chaque fois qu'un plugin ajoute un nouveau réseau social.

#### Anti-virus

Les antivirus sont des logiciels développés pour identifier, neutraliser et éliminer des logiciels malveillants. Plusieurs anti-virus sont vendus sur internet notamment AVG, McAfee, Trend, Norton, etc. Pour empêcher que les schémas OWL soient corrompus par une application externe, on peut installer sur le serveur qui héberge notre PFC nommée KWeSSi un anti-virus.

#### Chiffrer les données

Il est important de chiffrer les données que l'on estime sensibles. Par exemple les mots de passe, les emails. Le chiffrement empêche que l'on lise les données tant que l'on ne peut pas les déchiffrer. Il existe plusieurs moyens de chiffrement pour le web comme TLS (Transport Layer Security) ou HTTPS. Nous n'allons pas détailler car ce n'est pas le but de notre travail. Dans ce travail nous chiffrons tous les mots de passe en utilisant Spring Security [4]. La principale classe qu'il fournit est BCryptPasswordEncoder [5] qui utilise l'algorithme de hachage BCrypt [6].

#### Politique de gestion de mots passes

En plus du chiffrement une politique de gestion de mot de passe rigoureuse est nécessaire. Nous recommandons ceci pour les mots de passe :

- Avoir au moins huit caractères.
- Avoir au moins une majuscule.
- Avoir au moins une minuscule.
- Avoir au moins un chiffre.

4. Site internet : https ://spring.io/projects/spring-security (consulté le 10/03/2020)
5. Documentation Java de la classe : https ://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BC (consulté le 10/03/2020)
6. Plus d'information sur : https ://www.bcrypt.fr/explications (consulté le 10/03/2020)

- Avoir au moins un caractère spécial (tels que :-, ;.= etc.).
- Changer régulièrement son mot de passe (après un, deux, ou trois mois par exemple)
- Ne pas communiquer son mot de passe à une tierce personne ou l'écrire quelque part.

Nous tenons à signaler que les mesures de sécurité que nous avons listées ne sont pas exhaustives. Il peut encore avoir d'autres.

## 15.3 Langage de développement

Les solutions actuelles que nous avons étudiées sont développées soit en Java, soit en PHP. Nous n'avons pas pu trouver les raisons exactes qui ont poussé les concepteurs de ces solutions à choisir l'un ou l'autre de ces deux langages. Nous pensons que chaque solution s'est située dans son temps pour faire son choix. Joomla voir le jour en 2005, période durant laquelle le langage PHP est beaucoup utilisé pour développer les sites web. À cette période PHP serait le langage par excellence, a déjà beaucoup de spécialistes pour le développement web. Agorapulse et Sprout social viennent à partir de l'année 2010, période durant laquelle Java vient en force dans le développement des sites Web. Cette nouvelle donne a peut-être guidé les concepteurs d'Agorapulse et Sprout social. Nous sommes conscients que notre argument est insuffisant d'autant plus qu'en 2010 plusieurs d'autres langages de développement web étaient fort connus et fort utilisés, notamment ASP.net (ou C#), Python.

Pour notre plateforme collaborative il n'y a pas une contrainte particulière en ce qui concerne le choix du langage de programmation. Un classement des langages de programmation les plus utilisés pour le Web a été publié par Easy partner et se trouve sur https ://www.easypartner.fr/blog/le-classement-des-langages-de-programmation-de-liee/ (consulté le 22/08/2019). Ce classement place le langage Python en tête de liste, Java en quatrième position, C# en cinquième et PHP en sixième. Parmi ces langages celui que nous maitrisons le mieux est Java et c'est l'une des raisons pour laquelle nous l'avons utilisé dans ce travail.

## 15.4 Version de quelques outils utilisés

| Outil | Version |
|---|---|
| Protégé | 5.5.0 |
| Java | 1.8.201 |
| Jena | 3.13.1 |
| PF4J | 3.2.0 |
| SpringBoot | 2.2.6.RELEASE |
| Spring | 5.2.5.RELEASE |
| Angular | 6.1.7 |
| Node.js | 10.15.1 |
| NPM (Node Package manage) | 6.14.5 |
| Maven | 3.3.9 |
| Overleaf | - |

# Chapitre 16

# Rencontre des besoins

Rendu presqu'à la fin de ce mémoire, une question nous taraude l'esprit : est-ce que nos recherches ainsi que notre solution proposée répondent au problème de départ tout en couvrant tous les besoins qui avaient été décrits ? Pour y répondre, nous allons reprendre, un à un, les besoins décrits au début dans la première partie intitulée «Description des besoins issus de la documentation», et nous allons voir si l'ensemble de ces besoins sont entièrement couverts par notre PFC.

| Besoins | Tâches inclues | Est couvert | Commentaire |
|---|---|---|---|
| **Les besoins fonctionnels** | | | |
| Inscription d'un utilisateur | Attribuer un rôle à un utilisateur. | OUI | Voir écran d'inscription |
| Traiter un utilisateur | Activer, désactiver, bloquer, refuser, supprimer. | OUI | Voir l'écran de gestion des utilisateurs et des rôles. |
| Se connecter | Modifier le mot de passe | OUI | Voir L'écran de connexion |
| Ajouter un rôle à un utilisateur | - | OUI | Voir l'écran de gestion des rôles. |
| Rédaction d'un article | Modifier, valider, publier. | | Voir l'écran de rédaction et de publication de contenus. |
| Consultation d'un article | - | OUI | Voir l'écran de publication de contenus. |
| Traiter un article | Modifier, supprimer, attribuer, valider, publier. | OUI | Voir l'écran de rédaction et de publication de contenus. Voir le diagramme de workflow. |
| Recycler ou customiser un article | Sélectionner, modifier, valider, publier. | OUI | Voir l'écran de customisation d'un contenu. |
| Traiter un réseau social | Ajouter un RS, supprimér attribuer un censeur ou un modérateur. | OUI | Voir l'écran d'ajout d'un réseau social. Utilisation de PF4J pour Spring (plugin). |
| Ajouter un champ à un article | Ajouter, modifier, supprimer. | OUI | Voir l'écran d'ajout d'un champ et ses caractéristiques. |
| Envoyer un mail | - | OUI | Présence d'un système d'envoie automatique de mails. |
| **Les besoins non fonctionnels** | | | |
| Fournir une couche de sécurité à l'application | - | OUI | Utilisation de Spring Security. Cryptographie avec BCript. Politique de gestions des mots de passe. |
| Interactivité | - | OUI | Couche GUI qui accepte les pc, smartphones, tablettes. |
| Extensibilité et flexibilité | - | OUI | Développement de plugin avec PF4J. Schéma OWL |

En dehors des besoins ci-dessus qui étaient clairement exprimés, il y avait aussi un besoin sous-jacent ayant aussi

une grande importance. Il s'agit de la gestion des restrictions de chaque réseau social concernant la publication de contenus. Les recherches que nous avons présentées dans la « partie littérature » ont montré par exemple que les réseaux sociaux acceptent une taille de vidéo, d'image différente. Le nombre de caractère d'un message à publier est différent d'un réseau social à l'autre. Cette recherche nous a montré également que la publication sur les RS fait intervenir des relations encore plus complexes entre les données. Par exemple, les relations entre photo, photo de personne, photo de faible résolution, photo libre de droit, etc. Pour gérer cette situation, le choix de OWL/RDF a été capital. En fait, ce choix nous a permis de définir des restrictions sur chaque champ du contenu et en fonction du réseau social (voir annexe). OWL a permis aussi de faire face plus facilement au problème d'extensibilité des données.

Exemple de restriction sur Facebook : Facebook n'accepte que des photos avec pour extension jpeg et png. Pour gérer cela nous avons utilisé la restriction OWL suivante sur les extensions des images du RS Facebook.

```
<owl:DatatypeProperty rdf:about="http://www.semanticweb.org/will/ontologies/2019/7/kwessi#extensionImageFacebook">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/will/ontologies/2019/7/kwessi#FacebookImageExtension"/>
    <rdfs:range>
        <rdfs:Datatype>
            <owl:oneOf>
                <rdf:Description>
                    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
                    <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string">jpeg</rdf:first>
                    <rdf:rest>
                        <rdf:Description>
                            <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
                            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string">png</rdf:first>
                            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                        </rdf:Description>
                    </rdf:rest>
                </rdf:Description>
            </owl:oneOf>
        </rdfs:Datatype>
    </rdfs:range>
</owl:DatatypeProperty>
```

FIGURE 16.1 – restrictions sur l'extension des images de Facebook.

Sur la figure suivante on essayer d'ajouter une image sur Facebook avec pour extension mp4, qui n'ai pas une extension d'image acceptable sur ce RS. Le raisonneur OWL va immédiatement détecter une inconsistance dans le schéma OWL et va afficher le message d'erreur qui se trouve sur la figure.

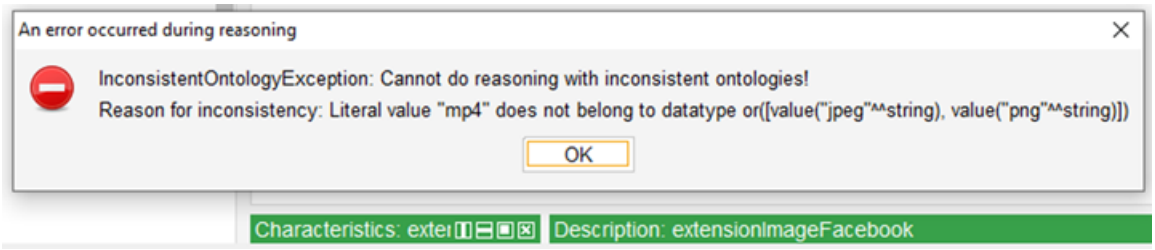



FIGURE 16.2 – message pour mauvaise extension sur l'image de Facebook.

Après avoir reparcouru tous les besoins, nous pouvons affirmer sans nous tromper que la solution que nous proposons répond à la question de recherche de départ qui est : Comment concevoir une plateforme collaborative (PFC) pour la publication de contenus sur les réseaux sociaux en couvrant certains besoins ?

# Chapitre 17

# Améliorations

L'application KWeSSi doit répondre à un certain nombre de besoins qui ont été défini dans le chapitre « Description des besoins ». L'ensemble des besoins a été couvert par le prototype que nous avons conçu. Cependant, en implémentant le prototype, nous avons eu plusieurs idées de fonctionnalités qui pouvaient améliorer l'application KWeSSi. Évidemment, ces fonctionnalités pourraient être apportées par une communauté de développeurs parce que le code source de notre l'application est ouvert.

## 17.1 Connexion aux réseaux sociaux via l'email et le mot de passe

Actuellement lorsqu'un utilisateur veut publier du contenu sur les réseaux sociaux, il utilise les access token. Cependant, les access token ne sont pas les uniques manières de se connecter aux réseaux sociaux. Nous pourrions aussi utiliser les connexions via l'email et le mot de passe.

## 17.2 Suggestion dynamique de contenus

Lorsqu'un utilisateur veut rédiger un message, il a 2 choix : soit il rédige directement le message sur n'importe quels sujets pour un ou plusieurs réseaux sociaux, soit il se dirige sur l'écran de proposition de sujet et en propose un, qu'il rédigera. Actuellement, c'est une bonne solution. Cependant, nous pouvons l'améliorer en apportant une extension qui pourrait proposer dynamiquement du contenu comme le fait l'application Sprout Social.

## 17.3 OWL/RDF

Nous avons structuré nos données avec le langage OWL/RDF parce que celui-ci présentait des avantages d'extension de données. Nous avons écrit des contraintes pour chaque réseau social afin de respecter leurs critères d'acceptation de contenu. Étant donné que c'était notre première expérience avec le langage OWL/RDF, nous n'avons pas suffisamment eu de connaissances pour mieux explorer tout le potentiel de ce langage. Nous pensons qu'avec plus de formation «https ://www.w3.org/TR/owl2-overview/» il serait possible de mieux raffiner la structure du schéma OWL du Framework de base pour ajouter des contraintes plus pertinentes ou retirer celles qui le sont moins.

## 17.4 Plugins

L'application KWeSSi se veut extensible avec l'ajout de nouveaux RS. Pour répondre à ce besoin, nous avons utilisé le Framework PF4j cité plus haut et développé en Java. Nous avons présenté des outils pour développer les plugins en Java et il y en existe sûrement d'autres. Malheureusement nous n'avons pas pu tous les analyser en profondeur afin de dénicher celui qui est le mieux adapter à un travail comme celui-ci. Cette tâche pourrait donc faire l'objet d'une autre recherche, qui complétera la nôtre.

## 17.5 Template

Actuellement, si nous rédigeons un contenu afin de le publier sur Twitter et LinkedIn par exemple, nous devons respecter les contraintes de publications de leurs champs. Par exemple, sur Twitter nous ne pouvons pas dépasser 280 caractères contrairement à LinkedIn. Pour permettre à Twitter de publier du contenu de plus de 280 caractères, nous pouvons créer des templates de rédaction de contenu dans notre plateforme. Par exemple, il pourrait avoir un template qui mets le contenu rédigé dans un fichier png, jpeg ou pdf afin de les visualiser avant et de les publier.


FIGURE 17.1 – contenu de plus 280 caractères sur Twitter.

Ci-dessus un exemple de ce type de contenus publié sur Twitter et qui a plus de 280 caractères. Le contenu a été placé dans un fichier pdf avant d'être publié.

# Conclusion

Chaque seconde une quantité indénombrable de publications sur les réseaux sociaux est effectuée. Il existe une multitude de réseaux sociaux utilisés eux-mêmes par des milliards d'utilisateurs. Les entreprises ou les particuliers possèdent souvent des comptes sur un ou plusieurs réseaux sociaux différents. Ces entreprises ou particuliers publient souvent la même information sur ces différents réseaux sociaux. Pour cela, ils doivent faire la même tâche de publication plusieurs fois. Ce qui serait parfois ennuyeux et même fastidieux s'ils veulent publier sur une dizaine de comptes à la fois. C'est dans cette problématique que ce situe notre question de recherche qui est : **Comment concevoir une plateforme collaborative pour la publication de contenus sur les réseaux sociaux en couvrant certains besoins ?** En relation avec cette question, notre promoteur nous a fourni des documents (voir annexes) dans lesquels nous avons tiré les spécifications de notre future PFC.

Ces documents que nous avons reçus nous ont permis de décrire l'ensemble des besoins de l'application. Dans cette tâche, il était important de différencier les besoins fonctionnels des besoins non fonctionnels. Dans les besoins fonctionnels, certains étaient plus prioritaires que d'autres : traiter un réseau social, rédiger un contenu et customiser un cotenu. Dans les besoins non fonctionnels, il s'agissait de l'extensibilité et la flexibilité de l'application. Ces besoins nous ont servi de fil conducteur pour la réalisation de ce mémoire.

Les différentes recherches effectuées pour la rédaction de la partie littérature nous ont menées à plusieurs pistes intéressantes : la découverte des spécificités de chaque réseau social, la découverte des solutions actuelles, le développement des compétences en OWL et sur la conception des plugins, les méthodes pour faire la synthèse de plusieurs lectures, etc. Nous avons choisi Facebook, LinkedIn, Instagram, Youtube et Twitter pour comprendre les réseaux sociaux dans leurs globalités. Ce sont les réseaux sociaux les plus utilisés dans le monde, cela était donc un choix pertinent. Tous les RS n'admettent pas la publication de contenus à partir d'une application externe. Nous avons beaux faire des recherches, mais nous sommes limités par ce que chaque RS permet de faire. Sur Facebook par exemple, l'obtention des tokens d'accès prend trop de temps pour un nouveau compte. Lorsqu'on crée un nouveau compte sur Facebook, il faut attendre environ deux jours, voire une semaine, pour que la publication de contenus à partir d'une application externe soit possible. Nous avons aussi découvert qu'il existe des PFCs comme Agorapulse ou Sprout Social qui publient du contenu sur les réseaux sociaux de manière collaborative. Malheureusement, leurs fonctionnalités ne correspondent pas à nos besoins.

En comparant les différentes solutions existantes (Agorapulse, Sprout Social, CMS) nous avons constaté que : les applications Sprout Social et Agorapulse ne sont pas extensibles étant donné qu'il n'est pas possible d'y ajouter un nouveau réseau social. Le CMS Joomla n'est pas adapté pour customiser les articles et traiter un réseau social. Cela nous a poussés à nous intéresser aux différentes architectures permettant de concevoir une PFC.

Pour étendre notre structure de données, nous avons eu 2 possibilités : les bases de données relationnelles ou OWL. Nous avons choisi OWL parce qu'il est plus extensible que les bases de données relationnelles. OWL accepte des relations plus complexes avec des couplages simples. Avec les bases de données, on est obligé de gérer le couplage fort des clés primaires, des clés secondaires ce qui peut être une limite pour l'extensibilité d'une structure de données. L'OWL n'a pas ce type de contraintes, il utilise les raisonneurs pour générer de nouvelles relations et faciliter l'ajout d'autres relations. Pour étendre notre application, nous avons choisi l'extension par plugins combinée avec la fusion de fichier OWL. Cette technique permet d'ajouter plus facilement un réseau social et/ou un champ et ses contraintes. Nous avons énormément perdu du temps dans ce chapitre en raison d'utilisation de mauvaises références nous envoyant vers des pistes non concluantes. Nous avons fini par nous réorganiser en précisant aux mieux nos mots-clés pour faire les recherches des articles de référence.

Durant la conception du prototype de l'application KWeSSi, nous avons conçu la base de connaissances de chaque réseau social sélectionné puis nous les avons intégrées entre elles pour obtenir la base de connaissances de notre application. Cela fut très difficile de recueillir les données sur les RS et qui sont nécessaires à la mise en place des différents schémas OWL. Par exemple, dans le cas de Facebook, il était primordial de différencier les informations faisant partie des champs de publications de celles qui étaient des fonctions annexes comme la géolocalisation. Ces

fonctions annexes étaient retirées des données collectées parce que nous estimions qu'elles n'entraient pas en compte dans la publication de contenus. La gestion des contraintes sur les champs n'était pas facile pour plusieurs raisons : pas évident de trouver la bonne syntaxe OWL pour chaque contrainte, difficultés pour répertorier toutes les contraintes, etc. En ce qui concerne l'architecture de l'application, nous avons utilisé Spring pour concevoir le cœur de la PFC et le Framework PF4J pour étendre notre application à l'aide de plugins. Spring tout comme PF4J était très simple à utiliser, d'autant plus que nous avons eu suffisamment de bonnes documentations sur ces outils et elles étaient bien rédigées.

La recherche et la rédaction du mémoire fut longue et difficile. Nous avons rencontré beaucoup d'obstacles que l'on a finis par franchir les uns après les autres. L'une des difficultés principale fut la compréhension du langage de structure de données OWL qui nous était totalement inconnu. La gestion de la pression professionnelle ainsi que la pression scolaire est une autre difficulté que nous avons dû surmonter. Nous avons fait régulièrement de mauvais choix non seulement sur les articles de référence mais aussi dans les choix des outils pouvant aider dans la réalisation de ce travail. Par exemple, l'outil de visualisation de graphe WebVOWL est adapté pour la visualisation des ontologies de petites tailles. Cependant, plus notre ontologie grandissait, plus cet outil s'avérait moins efficace dans la visualisation de toute la base de connaissances. Au début, nous avons eu des divergences dans la compréhension des différents sujets que nous avons abordés dans ce mémoire. En effet, chacun de nous avait un avis en ce qui concerne l'architecture de l'application ainsi qu'au niveau de la compréhension d'OWL. Cela a mis en évidence les difficultés qu'un travail d'équipe peut engendrer. Cependant, ce même travail d'équipe nous a permis d'avoir des échanges de compétences très enrichissantes.

Enfin, ce mémoire nous a appris à traiter un problème de façon scientifique. Nous avons pour la première fois fait des recherches dans des revues scientifiques que nous avons dû analyser et trier. Nous espérons que notre travail pourra être réutilisé dans le cadre de la centralisation des publications de contenus sur les réseaux sociaux. L'une des pistes pouvant améliorer notre travail concerne la mise en place des templates pour la rédaction des contenus adaptables à plusieurs RS.

# Bibliographie


[1] Thomas Coëffé. Chiffres réseaux sociaux – 2018. *Rapport Technique, Laboratoire I3S (Universite de Nicehttps ://fr.overleaf.com/project/5d72cd3cbf42370001ce21cb Sophia Antipolis/CNRS), numero I3S/RR-2008-20-FR, Sophia Antipolis, France*, 7 2018.

[2] Danah M Boyd and Nicole B Ellison. Social network sites : Definition, history, and scholarship. *Journal of computer-mediated Communication*, 13(1) :210–230, 2007.

[3] Bruno Chaudet. *Plateformes collaboratives et logiques processuelles dans l'évolution des formes organisationnelles : pour une conception étendue de l'information organisationnelle*. PhD thesis, Université Rennes 2, 2011.

[4] Michel Arnaud. Les limites actuelles de l'apprentissage collaboratif en ligne. *Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation*, 10 :7–pages, 2003.

[5] Kristin Johnson. Introducing bambu by sprout social, a new social advocacy platform for employees. 8 2015.

[6] Alban Dumouilla. We made all the mistakes during the 2000's internet bubble. 5 2017.

[7] Julien Dubois, Jean-Philippe Retaillé, and Thierry Templier. *Spring par la pratique : Mieux développer ses applications Java/J2EE avec Spring, Hibernate, Struts, Ajax...-Spring 1.2 et 2.0*. Editions Eyrolles, 2011.

[8] Rod Johnson. *Expert one-on-one J2EE design and development*. John Wiley & Sons, 2004.

[9] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5) :28–37, 2001.

[10] Jean Charlet, Philippe Laublet, and Chantal Reynaud. *Le web sémantique*. Cépaduès-Ed., 2003.

[11] Robert Hoehndorf, Joshua Bacher, Michael Backhaus, Sergio E Gregorio, Frank Loebe, Kay Prüfer, Alexandr Uciteli, Johann Visagie, Heinrich Herre, and Janet Kelso. Bowiki : an ontology-based wiki for annotation of data and integration of knowledge in biology. In *BMC bioinformatics*, volume 10, page S5. BioMed Central, 2009.

[12] Holger Knublauch, Ray W Fergerson, Natalya F Noy, and Mark A Musen. The protégé owl plugin : An open development environment for semantic web applications. In *International Semantic Web Conference*, pages 229–243. Springer, 2004.

[13] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6) :377–387, 1970.

[14] Jean-Luc Hainaut. *Bases de données-4e éd. : Concepts, utilisation et développement*. Dunod, 2018.

[15] Sean Bechhofer, Frank Van Harmelen, Jim Hendler, Ian Horrocks, Deborah L McGuinness, Peter F Patel-Schneider, Lynn Andrea Stein, et al. Owl web ontology language reference. *W3C recommendation*, 10(02), 2004.

[16] Mike Dean, Guus Schreiber, F van Harmelen, J Hendler, I Horrocks, DL McGuinness, PF Patel-Schneider, and LA Stein. Owl web ontology language reference. w3c working draft, 2003.

[17] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, and Ulrike Sattler. Representing ontologies using description logics, description graphs, and rules. *Artificial Intelligence*, 173(14) :1275–1309, 2009.

[18] Michel Gagnon. Logique descriptive et owl. *Cours dispensé à l'école polytechnique de Montréal, Canada*, 2007.

[19] Michael Breu and Ying Ding. Modelling the world : databases and ontologies. *Whitepaper by IFI, Institute of Computer Science. University of Innsbruck*, 2004.

[20] KAIS SALHI. La fusion des ontologies. 2014.

[21] Marek Dudáš, Steffen Lohmann, Vojtěch Svátek, and Dmitry Pavlov. Ontology visualization methods and tools : a survey of the state of the art. *The Knowledge Engineering Review*, 33, 2018.

[22] Fatma Ghorbel, Nebrasse Ellouze, Fay Métais, Faiez Gargouri, Noura Herradi, et al. Memo graph : an ontology visualization tool for everyone. *Procedia Computer Science*, 96 :265–274, 2016.

[23] Véronique Crémer. Une ontologie pour le profilage des sites de réseaux sociaux par rétro ingénierie.

[24] Prabath Siriwardena. Oauth 2.0 security. In *Advanced API Security*, pages 287–304. Springer, 2020.

# Annexes

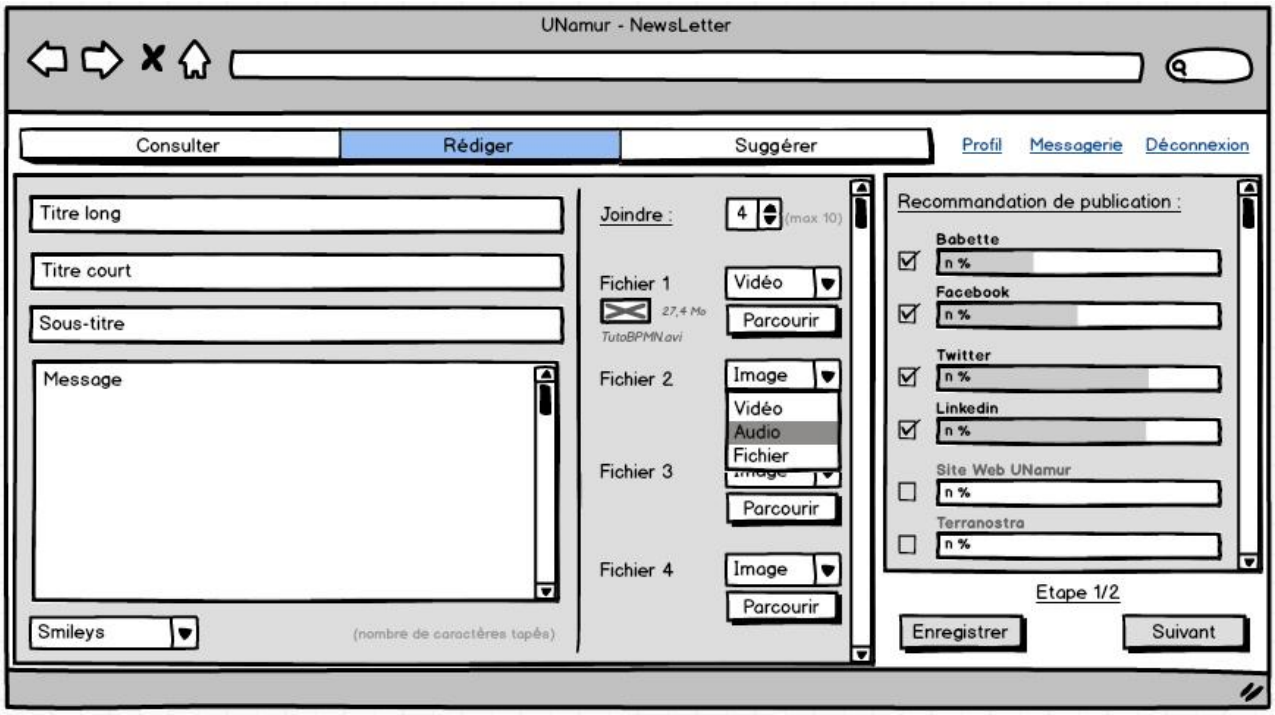
# Annexe A

# Documentation reçue

## Les écrans

### Rédiger une news

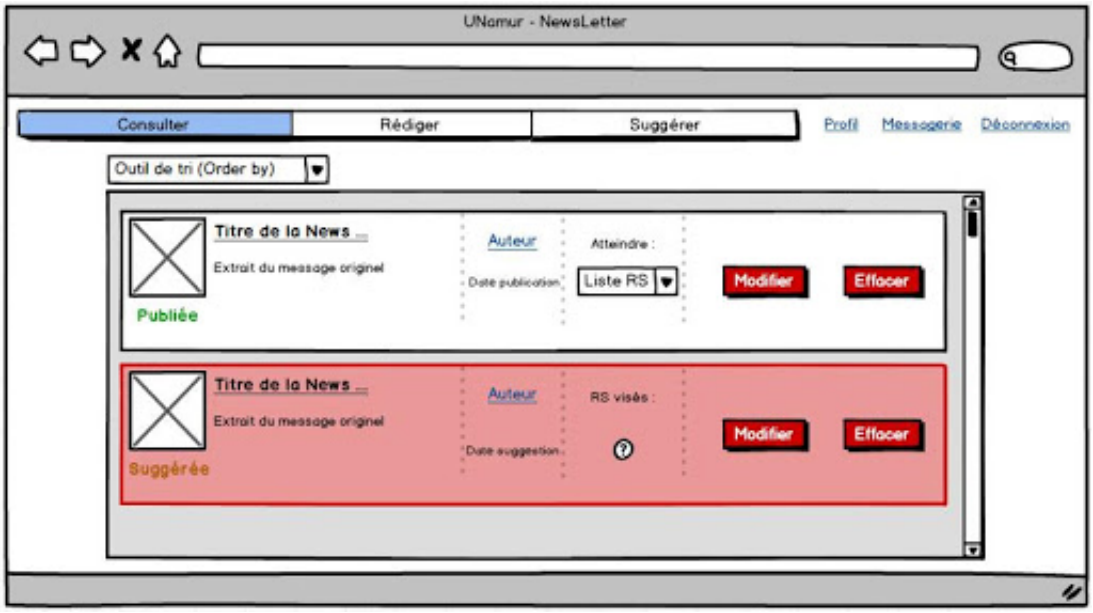
« Cette page s'atteint par le bouton de menu "Rédiger", en haut de la fenêtre. Création de la News originelle : remplissage des champs et ajout d'un ou plusieurs fichiers, le tout amplifiant ou atténuant le taux de recommandation des différents Réseaux Sociaux. »




Sur l'écran ci-dessus on peut voir plusieurs menus tels que : « Consulter », « Rédiger » ou « Suggérer » une NewLetter. Lors de la rédaction d'une NewLetter il est possible de recommander les médias sur lesquels la publié. Dans la partie droite de l'écran on voit des média comme Babette, Facebook, Twitter, Linkedin, Site Web UNamur et Terranostra.

### Consulter une news

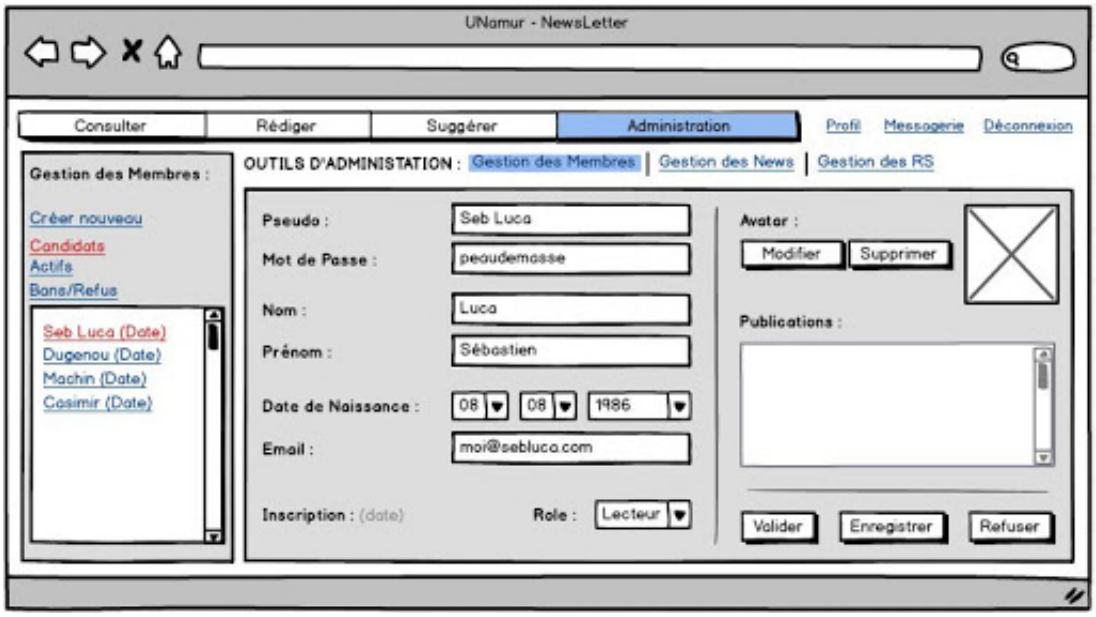
Après avoir rédigé une new, il peut être consulté via l'écran ci-dessous.




« Liste des News postées via l'application, chacune identifiable par certaines de ses informations originelles. On pourrait ici autant les consulter (tous rôles) que les modifier ou les supprimer (censeurs, modos, admins). Les News seront présentées sous forme de vignettes disposées en liste verticale et dont on pourra personnaliser l'ordre d'affichage grâce à un outil standard de tri (en haut à gauche). Pour consulter une News, on choisira via la combobox la version que l'on souhaitera lire (par rapport aux RS sur lesquels elle aura été publiée).-> Atteindre = Consulter. Les éléments colorés en rouge seront visibles des censeurs, modérateurs et admins, mais pas des simples rédacteurs. Cela comprendra les News suggérées ou récemment rédigées, et donc en attente de validation. Une fois acceptées, elles passeront à l'état de "publiée" et seront visibles de tous, dans la liste. Les boutons, quant à eux, appelleront à des rôles qui ne concernent pas non plus les simples rédacteurs : "Modifier" permettra de retoucher le contenu d'une News (globalement ou partiellement, . . . , le bouton "Enregistrer" en moins ?) ainsi que d'ajouter l'article dans un nouveau réseau social né après sa publication. ».

## Administration

« Cette page s'atteint par le bouton de menu "Administration", en haut de la fenêtre. Dans le cas de cet écran, on aura choisi l'onglet "Gestion des Membres" afin de valider la candidature d'un futur utilisateur. ».

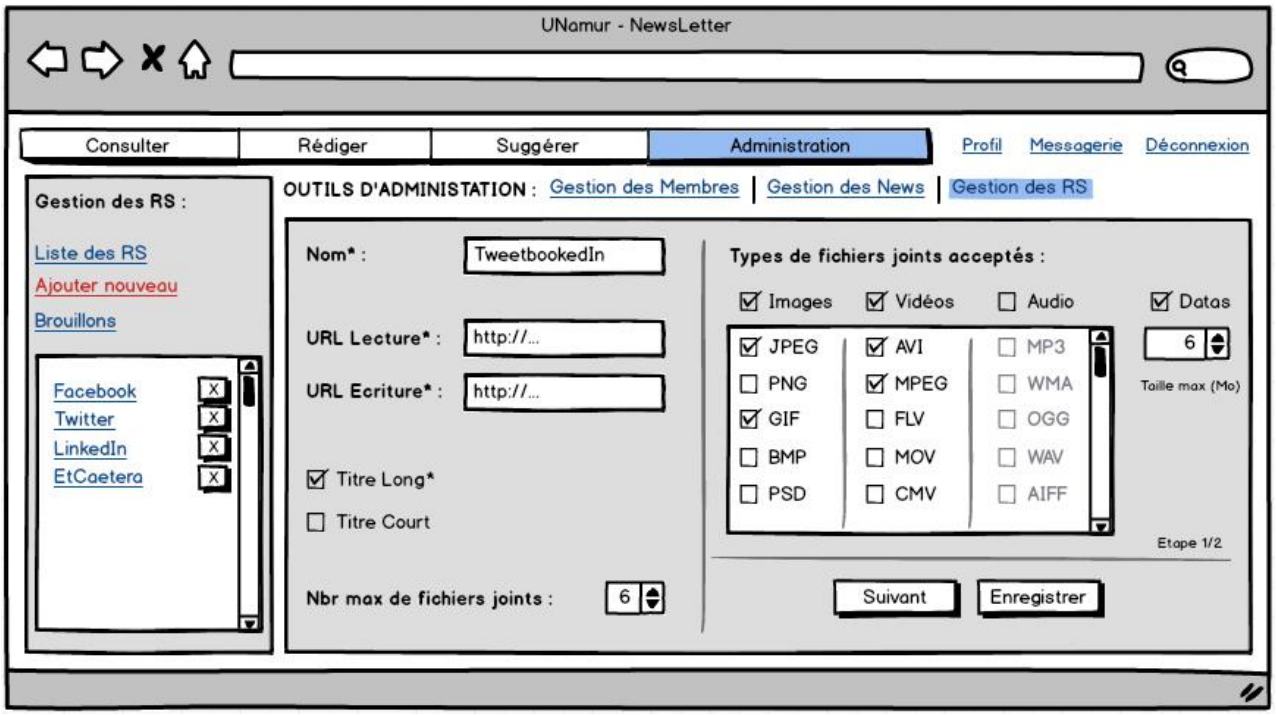



A gauche, « Les outils de sélection des membres : L'on peut parcourir la liste des utilisateurs ("Actifs") qui s'affichera dans le petit cadre en dessus. Cliquer sur un nom éditera les informations de l'utilisateur sélectionné dans le reste de la page et elles seront modifiables par l'admin». Au centre, le « Formulaire standard de création/édition de profil.

Seul un admin pourra voir et utiliser le combobox "Rôle", et donc promouvoir ou dégrader un membre ». A droite, « Suite et fin du formulaire dont nous parlions avec le choix de l'avatar. Vient ensuite un cadre listant les publications rédigées par le membre (ici, dans le cadre d'une validation de candidature, ce cadre est disabled ou inexistant».

### Ajout de réseaux sociaux

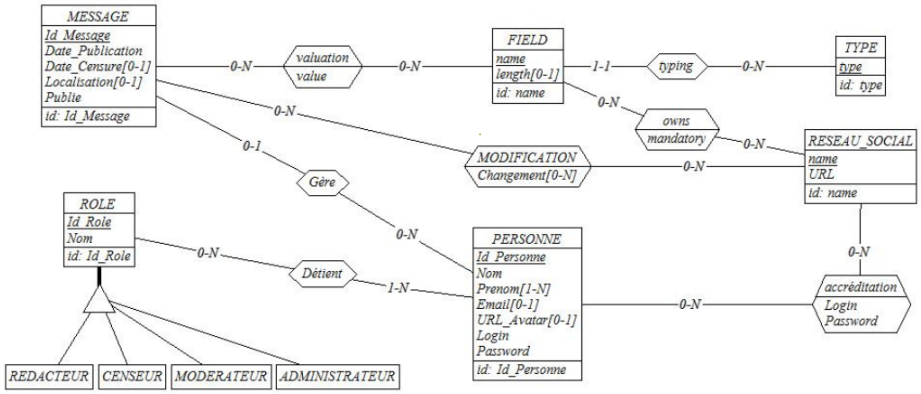
On arrive à cet écran en cliquant sur « Gestion des RS », le menu à droite de l'écran.




A gauche, «LISTE DES RS : fera s'afficher la liste des réseaux sociaux utilisés par la plateforme dans le cadre juste en dessous. Ils pourront être supprimés si l'on clique sur le bouton "X" ou, si l'on sélectionne le nom d'un des RS, ses caractéristiques pourront être modifiées. AJOUTER NOUVEAU : c'est le cas qui nous occupe, dans le contexte de cet écran. Cela affichera un formulaire vierge à compléter pour spécifier les caractéristiques d'un réseau social voué à l'ajout. BROUILLON : Contiendra les formulaires enregistrés (clic sur "Enregistrer" en cours d'écriture/édition) mais pas encore validés. Lesquels seront consultables et éditables de la même façon qu'un réseau social de la "Liste des RS"». Au centre, on a l'ensemble des champs qu'on veut donner au nouveau réseau social. Ces champs peuvent être : Nom, URL Lecture, URL Ecriture, Titre Long, Titre Court, etc. A droite, «Sélection du type de fichiers joints autorisés sur le nouveau réseau social. On en choisit le type (Image, Vidéo, Audio ou Datas) ainsi que les extensions permises pour ces fichiers. Concernant les "Datas", il s'agit de fichiers quelconques, qu'importe leur extension (cela peut même être des images, vidéos ou sons mais sous forme de données, pas en tant que médias affichés), il faudra spécifier la limite de taille (en Mo) imposée par le RS ».

## Le diagramme entité - relation

Le diagramme met en exergue les différentes relations qui existent entre les données. Ces relations sont tirées de la description qui a été faite plus haut sur les écrans. Ce diagramme d'entité relation permet de modéliser et de structurer les informations qui seront visibles sur les écrans. Le diagramme d'entité relation permet de schématiser la structure d'une base de données relationnelle (BDR).


MESSAGE
Id_Message
Date_Publication
Date_Censure[0-1]
Localisation[0-1]
Publie
id: Id_Message
valuation
value
FIELD
name
length[0-1]
id: name
typing
TYPE
type
id: type
owns
mandatory
RESEAU_SOCIAL
name
URL
id: name
MODIFICATION
Changement[0-N]
Gère
ROLE
Id_Role
Nom
id: Id_Role
Détient
PERSONNE
Id_Personne
Nom
Prenom[1-N]
Email[0-1]
URL_Avatar[0-1]
Login
Password
id: Id_Personne
accréditation
Login
Password
REDACTEUR
CENSEUR
MODERATEUR
ADMINISTRATEUR
0-N
1-1
0-1
1-N

# Annexe B

# Règles de validation du schéma à merger

| Règle 1: l'URI | |
|---|---|
| URI du schéma à importer | même que le schéma de base |

| Règle 2: les classes génériques | | | |
|---|---|---|---|
| **Schéma de base** | **Schéma à importer** | **Obligatoire** | **Signification** |
| SocialNetwork | SocialNetwork | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| Account | Account | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| Publication | Publication | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| Content | Content | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| Video | Video | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Resolution | Resolution | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| OtherFile | OtherFile | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Message | Message | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Image | Image | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Gif | Gif | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Extension | Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| Emoji | Emoji | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |

| Règle 3: les propriétés d'objet générique | | | | | |
|---|---|---|---|---|---|
| **Schéma de base** | **Schéma à importer** | **Domaine** | **Range** | **Obligatoire** | **Signification** |
| hasAccount | hasAccount | SocialNetwork | Account | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| hasPublication | hasPublication | Account | Publication | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| hasContent | hasContent | Publication | Content | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| hasImage | hasImage | Content | Image | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasVideo | hasVideo | Content | Video | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasMessage | hasMessage | Content | Message | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasOtherFile | hasOtherFile | Content | OtherFile | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasGif | hasGif | Content | Gif | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasEmoji | hasEmoji | Content | Emoji | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasImageResolution | hasImageResolution | Image | Resolution | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasVideoResolution | hasVideoResolution | Video | Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasImageExtension | hasImageExtension | Image | Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasVideoExtension | hasVideoExtension | Video | Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasOtherFileExtension | hasOtherFileExtension | OtherFile | Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |

| Règle 3: les propriétés d'objet générique | | | | | |
|---|---|---|---|---|---|
| **Schéma de base** | **Schéma à importer** | **Domaine** | **Range** | **Obligatoire** | **Signification** |
| hasAccount | hasAccount | SocialNetwork | Account | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| hasPublication | hasPublication | Account | Publication | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| hasContent | hasContent | Publication | Content | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| hasImage | hasImage | Content | Image | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasVideo | hasVideo | Content | Video | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasMessage | hasMessage | Content | Message | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasOtherFile | hasOtherFile | Content | OtherFile | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasGif | hasGif | Content | Gif | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasEmoji | hasEmoji | Content | Emoji | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasImageResolution | hasImageResolution | Image | Resolution | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasVideoResolution | hasVideoResolution | Video | Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasImageExtension | hasImageExtension | Image | Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasVideoExtension | hasVideoExtension | Video | Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasOtherFileExtension | hasOtherFileExtension | OtherFile | Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |

| Règle 4: les classes spécifiques au réseau social "Twitter" (par exemple) qu'on est entrain d'ajouter | | | |
|---|---|---|---|
| **Nom de la classe** | **Contrainte 1** | **Obligatoire** | **Signification** |
| Twitter | SubClassOf SocialNetwork | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| TwitterAccount | SubClassOf Account | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| TwitterPublication | SubClassOf Publication | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| TwitterContent | SubClassOf Content | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| TwitterVideo | SubClassOf Video | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| TwitterImage | SubClassOf Image | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| TwitterOtherFile | SubClassOf OtherFile | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| TwitterVideoResolution | SubClassOf Resolution | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| TwitterImageResolution | SubClassOf Resolution | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| TwitterMessage | SubClassOf Message | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| TwitterGif | SubClassOf Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| TwitterImageExtension | SubClassOf Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| TwitterOtherFileExtension | SubClassOf Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| TwitterVideoExtension | SubClassOf Extension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |

| Règle 5: les propriétés d'objet spécifiques au réseau social "Twitter" (par exemple) qu'on est entrain d'ajouter | | | | | |
|---|---|---|---|---|---|
| **Nom de la propriété** | **Contrainte 1** | **Domain** | **Range** | **Obligatoire** | **Signification** |
| hasTwitterAccount | SubPropertyOf hasAccount | Twitter | TwitterAccount | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| hasTwitterPublication | SubPropertyOf hasPublication | TwitterAccount | TwitterPublication | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| hasTwitterContent | SubPropertyOf hasContent | TwitterPublication | TwitterContent | oui et oui | doit être présent dans schéma à importer et avoir cette orthographe |
| hasTwitterVideo | SubPropertyOf hasVideo | TwitterContent | TwitterVideo | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasTwitterImage | SubPropertyOf hasImage | TwitterContent | TwitterImage | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasTwitterOtherFile | SubPropertyOf hasOtherFile | TwitterContent | TwitterOtherFile | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasTwitterVideoResolution | SubPropertyOf hasVideoResolution | TwitterVideo | TwitterVideoResolution | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasTwitterImageResolution | SubPropertyOf hasImageResolution | TwitterImage | TwitterImageResolution | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasTwitterMessage | SubPropertyOf hasMessage | TwitterContent | TwitterMessage | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasTwitterGif | SubPropertyOf hasGif | TwitterContent | TwitterGif | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasTwitterImageExtension | SubPropertyOf hasImageExtension | TwitterImage | TwitterImageExtension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasTwitterOtherFileExtension | SubPropertyOf hasOtherFileExtension | TwitterOtherFile | TwitterOtherFileExtension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |
| hasTwitterVideoExtension | SubPropertyOf hasVideoExtension | TwitterVideo | TwitterVideoExtension | non et oui | n'est pas obliger d'être dans le schéma à importer et s'il est présent doit avoir cet orthographe |