1. **Module Social et l’Intégration SPAD RealTime**
   1. **Objectifs**

Les réseaux sociaux interviennent en tant que nouveaux canaux de communication au sein de Coheris CRM. Ce qui mène l’équipe R&D CRM à développer leur propre connecteur social qui répond aux fonctionnalités attendues par Coheris CRM.

Dans le cadre de la nouvelle offre qui propose coheris, il a été prévu une migration totale du connecteur social de coheris CRM en un module indépendant qui s’appuie sur la nouvelle architecture DDD en ajoutant de certaines évolutions. L’intégration du textmining dans le module était une des primordiales évolutions qui a été planifié sur le module social à travers l’appel aux fonctionnalités de la solution SPAD RealTime.

La part de mes travaux dans cette partie, n’était pas la migration du connecteur social en module DDD. Elle était dans une première partie l'initiation des développements sur le module par la création de la structure nécessaire basée sur l’architecture DDD. Et Dans une second partie, elle était de trouver une approche d’intégrer les model de textmining dans la solution en se basant sur le connecteur social existant.

* 1. **Etude de l’existant**

Connecteur social Coheris CRM :

La brique Social apporte la connectivité à diverse fonctionnalités dont on peut citer par exemple :

* Administration des profils Facebook et Twitter des utilisateurs CRM,
* Management des Fanpages Facebook : Consultation des Posts publiés sur les Fanpages administrées, réponse aux posts sur le mur, identification et gestion du cycle de vie des statuts des publications…
* alimentation de dossier client sur CRM : Recherche du profil Facebook d'une personne, Recherche du profil Twitter d'une personne, Récupération des indicateurs sociaux (Facebook : Likes, Friends, Post/Comment, Fanpage | Twitter : Followers, Tweets) pour la mesure d'influence de la personne…
* ciblage et Reporting : Usage des données sociales dans le Ciblage ou le reporting, usage des indicateurs sociaux dans le ciblage…

Le connecteur est une brique applicative indépendante (on-premise, saas). Il offre une API REST qui répond aux fonctionnalités décrit-y dessus.

En effet, il a sa propre base de données qui stocke les informations et les données associées aux profils Facebook et Twitter renseignés par les utilisateurs.

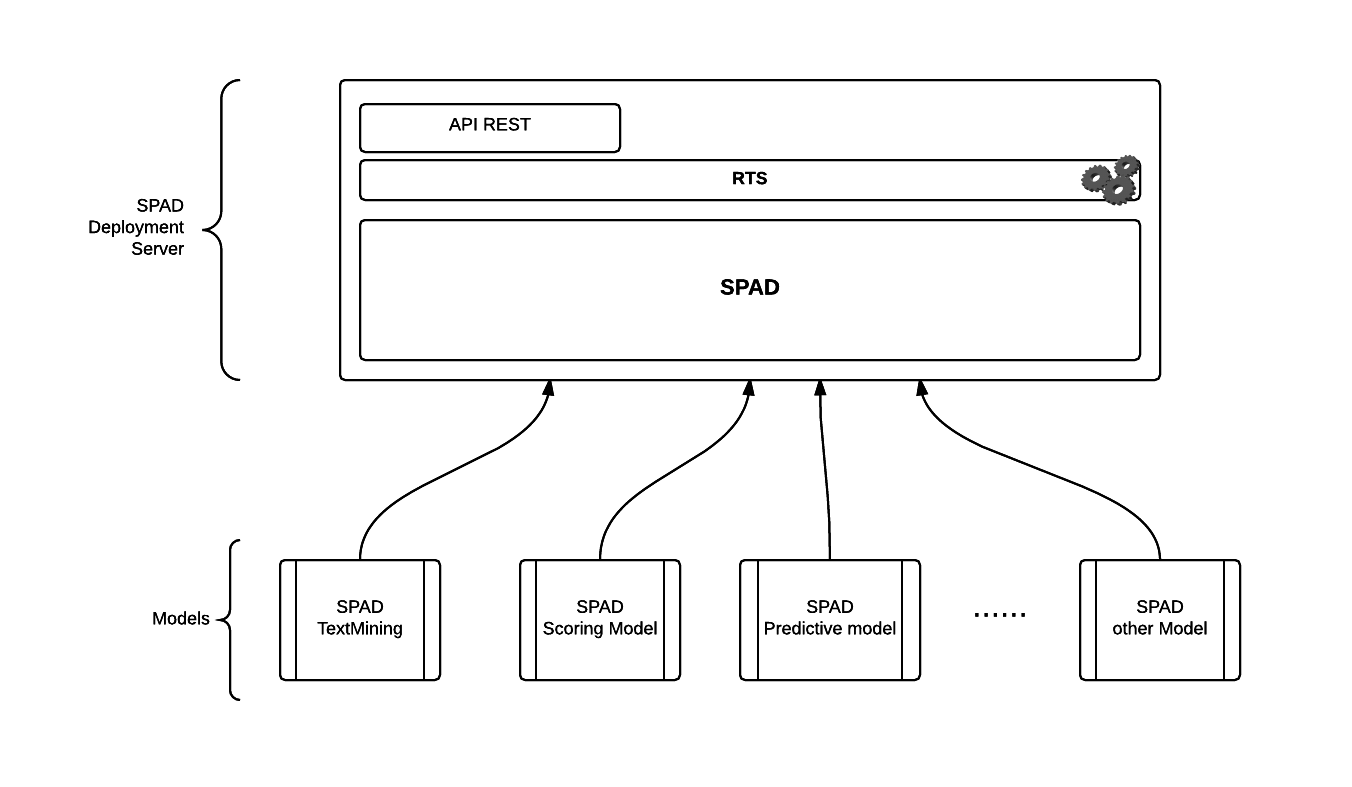
SPAD RealTime

Ce module permet d’utiliser les modèles SPAD pour répondre en temps réel aux requêtes de scoring ou de typologie. En effet, et après des travaux qui ont été réalisées en parallèle de mon stage, le serveur a été enrichi par des modèles de fouille de textes(Textmining) basées sur des algorithmes d’apprentissage bayésiennes.

*Architecture*

Le module offre une API REST qui utilise un dictionnaire JSON pour les paramètres en entrée et en sortie et qui répond aux fonctionnalités décrit-y dessus.

La mise en œuvre du modèle se fait par la création d’un fichier « .model ». Chaque type de modèle a une représentation qui différente aux autre type.



*Model Bayesien pour champs texte:*

Le modèle bayésien est utilisé pour affecter un texte libre dans une catégorie parmis N catégories prédéfinis. Dans le cas des commentaires et les Feed Facebook, on peut classifier automatiquement le message selon une catégorie de sollicitation : Demande, Opportunité, Intervention, lead. Le fichier conteneur du modèle doit être sous le format suivant :

modelClass=maestro.web.bayes.TextBayesModel

modelName=bayesien Motif

possibleValues=demande,opportunity,lead,intervention

minLength=3

lang=fr

targetName=target

keysToParse=text

learnFile=<com.spad.modelsdir>\text\posts-synopsis.txt

##Modéle de bayésien naïfs en mode random.

##pour tester l'intégration avec CRM.

##renvoie une des valeurs au hasard parmis la liste des possible

|  |  |  |  |
| --- | --- | --- | --- |
| **Propriété** | **Valeur** | **Défaut** | **commentaire** |
| modelClass | maestro.web.bayes.TextBayesModel | **requis** | obligatoire indique la classe qui implémente ce modèle |
| modelName | <libre> | **requis** | nom du modèle dans le serveur, il sert d'identifiant pour les appels REST et doit être unique |
| possibleValues | liste de valeur séparée par des "," | **requis** | Le nombre de valeur possible que prédit ce modèle.  il est aujourd'hui impossible de prendre en compte des valeurs contenant des "," |
| keysToParse | liste des clefs à parser | "text", "title" | ensemble des variables du dictionnaire passé en paramètre qui doivent être parsées pour l'analyse textuelle.  Toutes les valeurs sont considéré comme le même texte concaténé. Si il n'est pas spécifié on parsera les valeurs associées aux clef "text" et "title". |
| targetName | nom de la variable cible pour l'apprentissage | "target" | nom de la variable cible dans le dictionnaire passé à la méthode d'apprentissage. Les valeurs associées à la clef doivent être dans possibleValues.  Si il n'est pas spécifié on regarder la valeur associée à la clef "target" |
| lang | langage pour le parsing | fr | langage pour les stop word du parsing du texte. Par défaut on est en français, si le code est différent de "fr" on est en anglais. (2 langages supportés seulement pour l'instant). |
| minLength | taille minimum d'un mot | 3 | tous les mots de taille inférieure à cette valeur sont supprimés après le parsing. |

Base de données de tests

Afin de tester fonctionnellement l’efficacité de l’algorithme bayésien des model textmining, on a besoin d’une base de données de test qui contient des commentaires et des posts réels.

En effet, L’existant connecteur social a été branché sur des pages Facebook de tests. Elle ne contient pas de véritable message qui ont de sens sur la page. Alors elle nécessite de choisir une page Facebook publique pour alimenter nos bases de données. Cela nécessite une connaissance de l’API Graph du Facebook qu’elle nous permet d’extraire les données publique sur une page choisie son administrateur. De plus, on pourra même afficher le nom de l’internaute et son image publique sur Facebook.

* 1. **Réalisation**

L’essentiel du travail a consisté d’appliquer les mêmes travaux réalisés dans le module scheduler. Mais la différence était de ne pas aborder les travaux de migration du connecteur social CRM vers le nouveau module DDD. Ces derniers ont été planifiés pour des sprints qui commencent à la fin de mon stage.

Finalement, le but était de mettre en œuvre la testabilité des alogrithmes de textminig du serveur RealTime SPAD sur des sources de donnée réel.

Branchement avec l’existant connecteur social CRM

D’un point de vue structurel, et comme dans l’exemple du Framework Quartz dans le module scheduler, les interfaces responsables aux appels aux web services externes ont été définies dans la couche infrastructure comme étant une ressource externe du nouvel module social.

Afin de récupérer les posts Facebook et les commentaires d’une fanpage publique, j’avais besoin d’étendre l’api du connecteur social CRM avec des nouvelles fonctionnalités.

Image

Tous ces appels ajoutés, elle prend le nom de la fanpage comme un paramètre de recherche dans le graph API facebook. De plus, il était primordiale de migré les sources des entités métier, les porteuses des informations, du connecteur social CRM vers la couche shared-kernel afin de ne pas tomber sur des conflits entre les objets passés en JSON et les entités Domain de la nouvelle architecture.

En fin, pour alimenter la base de données du connecteur social, j’ai créé une tache scheduler qui fait la mise à jour périodiquement avec des nouvelles posts et commentaires d’une page public.

Intégration de SPAD RealTime

De la même manière, et comme pour l’existant connecteur social CRM, le branchement était cotés infrastructure. Mais avant de commencer, il était nécessaire de faire des tests sur l’api de spad RealTime pour savoir les paramètres entrées/sortie des requêtes.

Prenant l’exemple du modèle de qualification des messages selon leur motif « bayesien motif » :

Fichier bayesienMotif.model

modelClass=maestro.web.bayes.TextBayesModel

modelName=bayesien motif

possibleValues=demande,opportunity,lead,intervention

minLength=3

lang=fr

targetName=target

keysToParse=text

##Modéle de bayésien naïfs en mode random.

##pour tester l'intégration avec CRM.

##renvoie une des valeurs au hasard parmis la liste des possible

Les paramètres d’entrées sous format json

{modelName=bayesien Motif, nameToValueMap={text=Cela fait maintenant 2 mois que je me plaind de ne plus pouvoir commander sur le net}}

Retour sous format json

["intervention",32.38115350352977,3.8666739556296896,22.402458516347743, 41.349714024492805]

On s’intéresse plutôt au retour qu’il nous a fourni le serveur pour une qualification d’un message. En effet, la liste comme on le constate contient la décision du serveur à propos du contenu de message *intervention* suivi de pourcentage de chaque Target dans le même ordre qui a été écrit dans le fichier du model. Dans un premier temps, et suivant la spécification fonctionnelle du module, on a besoin juste de sauvegarder le motif décisif de la requête dans un champ *BusinessMotif* que je l’ai ajouté pour l’entité commentaire et l’entité Post.

A noter dans les développements des interfaces responsables aux appels webservice au serveur SPAD RealTime, on peut passer une liste de model à jouer sur un message.

Image classe interface wrapper liste

Prototype IHM

En fin, et afin de visualiser le resultat de mon travail du le module social, j’ai étais mené à réalise une IHM qui affiche les messages provenant du post ou du commentaire avec leur qualification selon le champ *BusinessMotif*.

De plus, il m’a ´été demandé de rendre le champ de la qualification éditable pour re-qualifer une fonction dans le champ qui affiche

* 1. **Bilan**