# OPTIMISATION DE LA SYNCHRONISATION DE DONNÉES ENTRE LES SYSTEMES INFORMATIQUES AVEC L'UTILISATION DES TECHNOLOGIES NUAGEUX

***Oleksii TUMANOV***

*Université nationale d''economie de Kharkiv Simon Kuznets, Ukraine, Université Lumière Lyon 2, France, e-mail: Oleksii.Tumanov@gmail.com*

L'article présente les résultats de la recherche et du développement des systèmes d'intégration de données. Il permet aux entreprises de déverrouiller toutes leurs données, qu'elles soient historiques, de temps réel ou émergentes.

***Mots clés**: synchronisation, gestion de données, ~~solution cloud~~*

## 1. Introduction

Le service de synchronisation de données consiste à « permettre à un utilisateur d'avoir ses données à jour, dans toutes ses applications, tout le temps » est le résultat d'une série de défis remettant en cause les pratiques actuelles en ingénierie des applications de gestion de données. Offrant à tous la possibilité de disposer d'un espace de stockage en ligne pour ses documents et l'ensemble de ses données, le Cloud est partout. La solution permet de synchroniser les données entre différentes applications fonctionnant dans le cloud. Il peut s'agir de bases de données, de solution de gestion de la relation client ou e-commerce, d'outils marketing cloud ou sur sites Internet. Pour cela, une mise à jour est effectuée régulièrement.

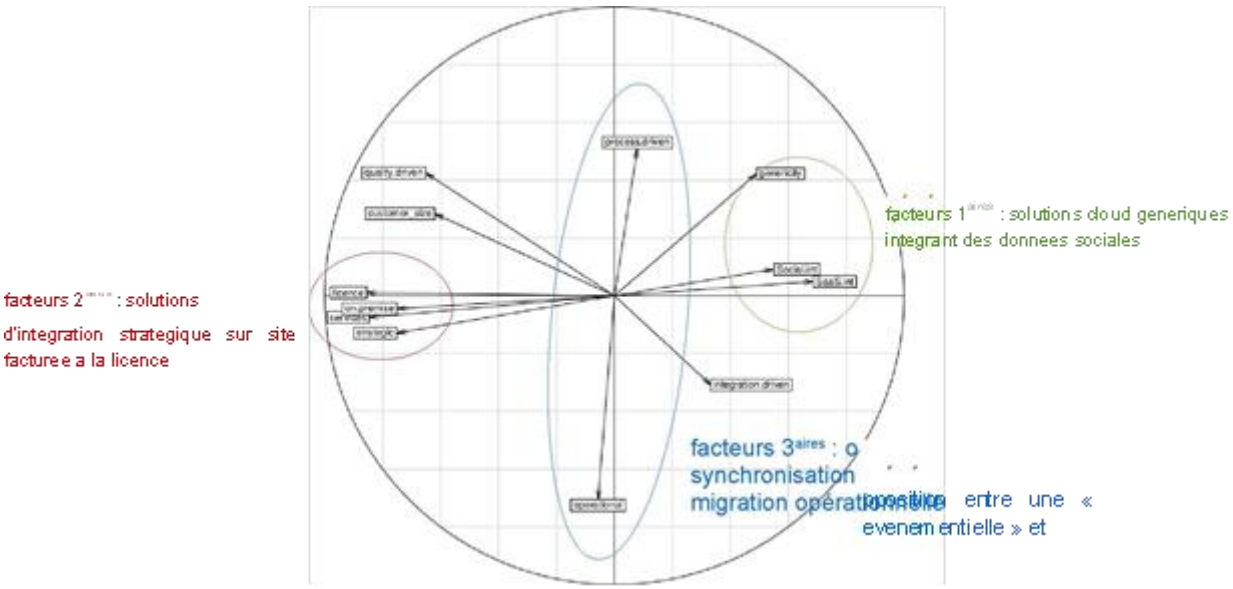
## 2. Etude de marché et positionnement de Podbox

On a réalisé une étude du marché de l'intégration de données en comparant plus d'une 40aine de compétiteurs sur des critères tels que (entre autres) :

- la réalisation d'une intégration sur poste (« on premise » en anglais : application ou prestation réalisée au sein de l'entreprise client) ou via un outil générique mis a la disposition de l'entreprise cliente lui permettant de réaliser son intégration en autonomie ;
- une prestation qui relève plus de la fusion de systèmes d'informations originellement disjoints (en cas de rachat d'une entreprise par une autre, ou les deux systèmes d'informations doivent être fusionnes) ou plutôt de l'ordre de la synchronisation (les deux SI coexistent et doivent être maintenus a jour) ;
- une intégration d'ordre stratégique (l'intégralité des données des deux SI doivent être synchronises) ou tactique (seuls quelques types de données comme les carnets de client doivent être synchronises par exemple) ;
- la migration opérationnelle de données d'un logiciel existant vers un autre (prestation plutôt ponctuelle) ou la synchronisation événementielle (propagation de modifications, création ou suppressions au fil de leur occurrence) ;
- un service facture comme une prestation ponctuelle ou comme un abonnement récurrent.

Une analyse en composantes principales des compétiteurs a permis de déterminer qu'ils se différenciaient essentiellement par (figure 1) :

- une approche « outil cloud plutôt generique » (facteurs $1^{aires}$) opposée à une approche « prestation de service spécifique sur site facturée à la licence » (facteurs 2aires) ;
- une synchronisation plutôt événementielle opposée à une intégration opérationnelle (facteurs 3aires)




~~*Figure 1. Facteurs de differentiation des competiteurs dans l'integration de donnees*~~

Si on projette ces compétiteurs en fonction de certains de ces critères (figure 2), on s'aperçoit que :

- le marché est saturé de solutions sur-site, que ce soit pour des intégrations opérationnelles ou événementielles ;
- le marche offre plutôt des solutions d'intégrations stratégique que tactique ;
- les solutions cloud se sont plus focalisées sur une approche événementielle.

.

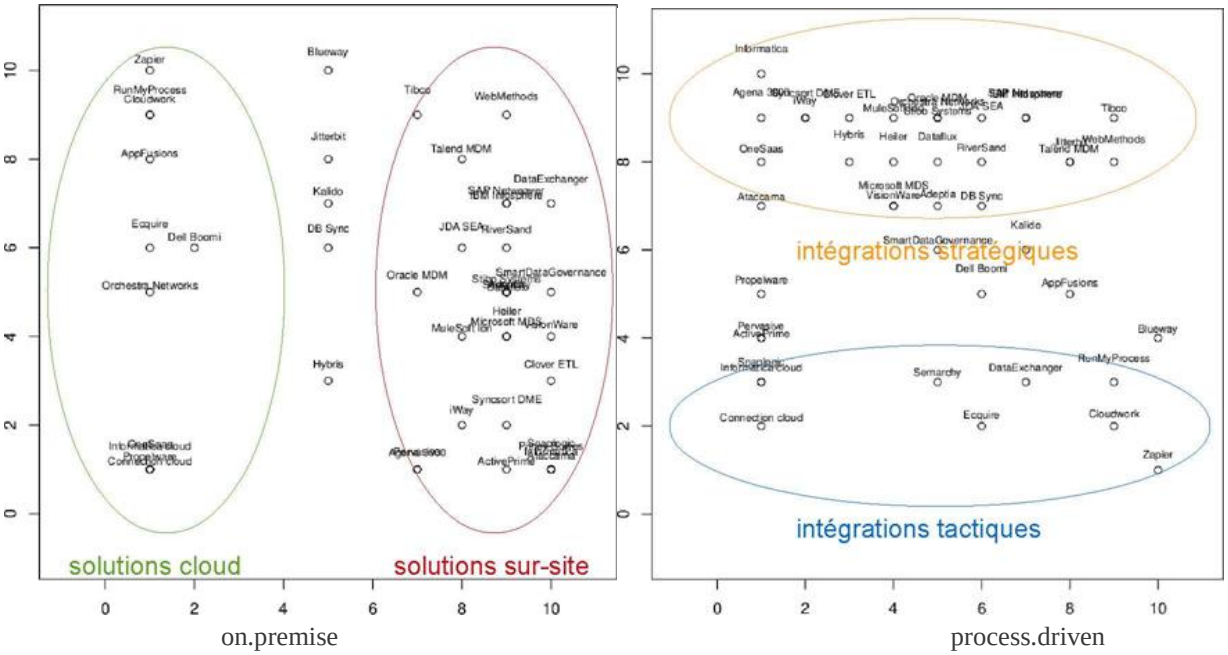



*Figure 2. Répartitions des compétiteurs selon leurs approches (sur-site / cloud, évenementielle / opéérationnelle, stratégique / tactique)*

Le développement récent d'applications professionnelles cloud vendues plutôt sur un mode d'abonnement mensuel fait que les solutions d'intégrations adaptées sont :

- également plutôt récentes ;
- focalisées sur une mécanique événementielle induite par les APIs de ces applications professionnelles, adaptées a la signalisation de données nouvellement crées ou mises à jour.

Le positionnement de Podbox consistant à proposer une solution cloud d'intégration tactique est donc particulièrement d'autant plus pertinent au vu de l'essor actuel des applications professionnelles cloud qui se

spécialisent dans une fonctionnalité donnée (CRM, emailer marketing, helpdesk de hotline, boutique en ligne, etc.) et qui n'ont besoin que d'une synchronisation partielle de données pour collaborer efficacement (synchronisation des données de contact notamment).

La place comparativement libre des solutions cloud d'intégration tactique s'explique par révolution des applications professionnelles elles-mêmes. En effet, pendant longtemps, ces applications étaient installées sur-site et vendues par un mode de licences annuelles, impliquant que les intégrateurs réalisent également des prestations sur-site, ponctuelles et complètes (stratégiques) du fait de leur coût élevé.

## 3. « Pod » : un formalisme de rupture pour la modelisation et la manipulation de données

### 3.1. L'ingénierie dirigée par les modèles et le dilemme classe / attribut

L'ingénierie dirigée par les modèles est le niveau d'abstraction le plus haut utilisé actuellement par les développeurs d'applications pour gagner en productivité. Les étapes précédentes, chacune amenant un saut de productivité conséquent, ont été :

- utilisation directe du langage machine et de son jeu d'instructions qui est un code directement exécutable par un processeur ;
- l'utilisation de l'assembleur, représentation exacte du langage machine sous une forme lisible par un humain, mais spécifique à l'architecture de chaque processeur ;
- l'utilisation de langages de bas ou haut niveau (permettant de programmer sans tenir compte des détails inhérents au fonctionnement de l'ordinateur) et de compilateurs ;
- l'utilisation de machines virtuelles pour s'affranchir de la configuration matérielle lors de la compilation.

L'IDM a débuté avec l'utilisation d'outils comme Merise ou UML pour modéliser des structures de stockage de données, des classes, des comportements de l'application ; modèles à partir desquels tout ou partie du code source est générée.

Cependant l'état de l'art actuel concernant le modèle de données manipule par l'application repose toujours sur les notions :

- de classe encapsulant des attributs ;
- d'attributs qui sont des valeurs terminales ;
- de relations entre les classes.

Ces notions de classe encapsulante et d'attribut final sont la cause de coûts (humain et financier) élevés lors de la maintenance applicative faisant intervenir des évolutions du modèle de données. Prenons l'exemple d'une application de gestion de contacts qui évolue :

- le cahier des charges de la première version de l'application spécifiait que le nom de la société de chaque contact pouvait être renseigne ;
- la deuxième version propose de renseigner le site internet de la société.

Il est donc logique, afin d'éviter une redondance de données, de créer une classe Société qui portera en attribut son nom et son site internet :

Cette opération implique cependant de profonds bouleversement dans l'application.

Concernant le stockage de données, il faut :

créer une nouvelle structure de stockage pour stocker indépendamment les sociétés des contacts ;

y migrer les données de sociétés ;

les dédoublonner ;

modifier la structure de stockage des contacts pour y inscrire les relations vers les sociétés.

Concernant le code source, il faut que : les appels à Contact.getSociete() qui renvoyaient une chaîne de caractères renvoient désormais une instance de Société.

Tous ces appels doivent être corrigés ; les interfaces graphiques permettant d'afficher ou de saisir un Contact doivent être refondues pour gérer en plus l'affichage ou l'édition de la Société reliée.

Le problème s'empire si on décide d'externaliser les e-mails pour pouvoir en associer plusieurs par contacts.

### 3.2. La solution Pod

Le paradigme de Pod repose sur :

une atomisation des données en éléments simples, un Pod, qui ne stocke qu'une seule valeur ;

un pod peut être mis en relation avec d'autres via des Relations (Rel) ;

ce pod est type par une Definition de donnée (PodDef) qui décrit le type de valeur porte (chaîne de caractères, nombre, booléen, etc.) et qui donne un sens à cette valeur via un nom de code (NOM, EMAIL, SOCIETE, etc.), un libelle destine a l'affichage, une éventuelle unité (année pour un age, km pour une longueur, etc.), un commentaire, etc. Chaque pod connaît sa définition et peut s'y référer a tout moment pour vérifier l'intégrité de sa valeur ;

les relations entre pods sont décrites par des Définitions de relation (RelDef) qui spécifient la nature de la relation (héritage, composition, agrégation, association) ainsi que les cardinalités entre les Définitions de données concernées. Chaque relation connaît aussi sa définition et peut s'y référer pour contrôler l'intégrité de la structure de données entre les pods relies (notes extrémités A et B). Les définitions de données et de relations constituent donc le modèle de données de l'application ; les pods et les relations constituent les données manipulées par l'application.

Le fait que le modèle soit embarque dans l'application : cela permet à l'application de s'y référer et de mutualiser les contrôles de validité de valeurs sans avoir a les développer dans chaque formulaire de saisie.

Ce concept permet donc de répondre au problème d'évolution du modèle de données expose précédemment sans rien casser (figure 3) :

*Figure 3. Evolution du modele de données Contact - Societe selon une approche UML classique*

Il a juste fallu :

rajouter la définition de donnée Site web ;

rajouter la définition de relation entre Société et Site web.

La couche de stockage de Pod gérant déjà les valeurs de façon atomique, il n'y a rien à modifier au niveau du stockage. Par ailleurs, la notion de Site web étant créée, elle est réutilisable lorsqu'on voudra ajouter la possibilité de renseigner un ou plusieurs sites internet pour un contact.

## ~~4. Mécanique générale de synchronisation~~

### ~~4.1. Traçabilité et versionnement des valeurs~~

Grâce à son modèle pivot évolutif, une intégration réalisée dans Podbox comprend tous les champs de données existants dans les applications qui sont synchronisées. Les correspondances de champs définies par l'utilisateur permet de spécifier si un podPath est implique lors de l'échange de données avec chaque application connectée ou non. Ainsi la figure 4 illustre l'exemple d'une intégration Podbox synchronisant trois applications (un CRM, un emailer marketing et un logiciel de support client) en montrant les champs de données qui sont impliques et le sens des flux de données :

- les champs prénom, nom et émail sont en lecture-écriture avec chaque application. Ce qui veut dire que toute modification faite dans une des applications est répercutée dans les autres ;
- les champs de données postaux sont en lecture-écriture dans le CRM et l'application de support mais ignores par l'application d'emailing ;
- l'application d'emailing fournit (en lecture seule) les informations de dernière campagne envoyée (nom de campagne, nombre d'ouvertures de l'émail pour chaque contact, la validité de l'émail). Le nom de campagne et le nombre d'ouvertures sont écrits dans la CRM. La validité de l'émail est transmis a l'application de support.

Ainsi, un champ de donnée n'a pas à être utilisé dans chaque application impliquée dans une intégration et une modification d'adresse postale ne doit donc pas impliquer la mise a jour d'un contact dans l'emailer car ces informations n'y sont pas gérées.

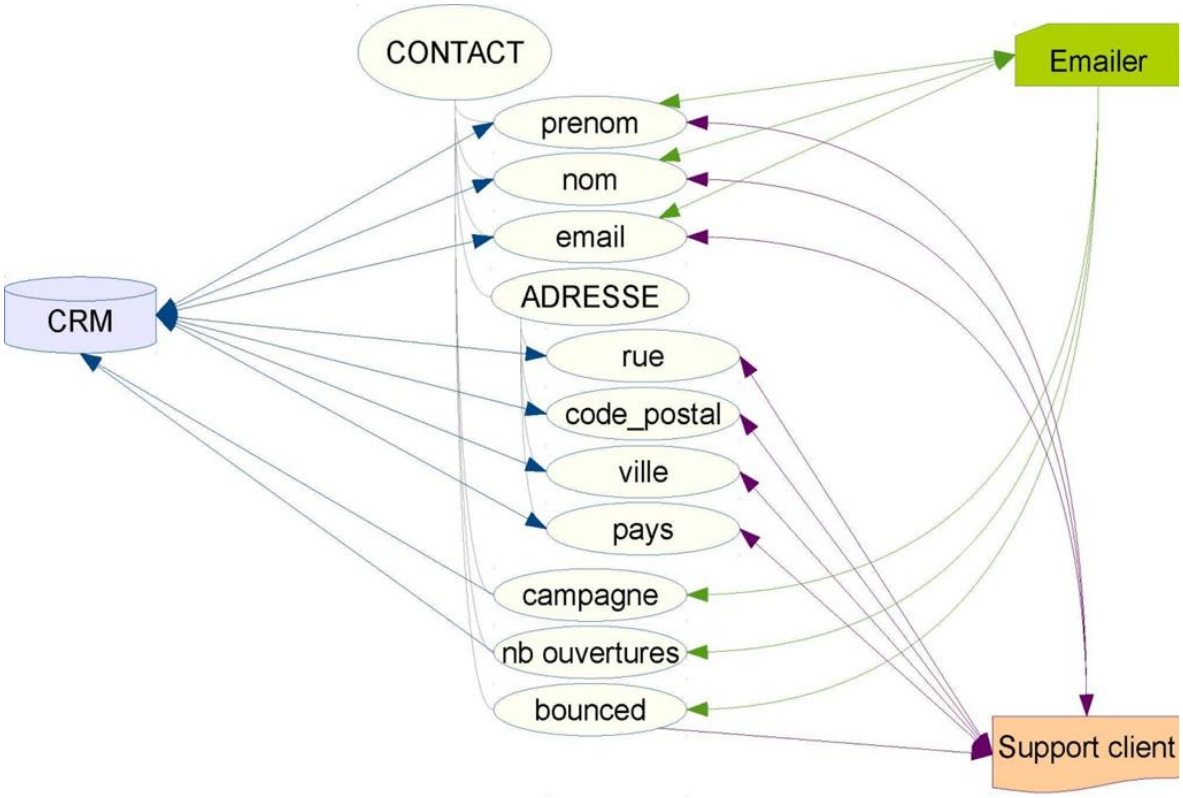



*Figure 3. Interactions entre les champs de données d'une intégration Podbox et les applications qu'elle synchronise.*

Dans une intégration, chaque pod contient donc la dernière version d'une valeur qui a été récupérée dans l'application qui en a donne la version la plus récente. Chaque pod est également associe a une autre table qui les valeurs qu'il porte (et a porte) dans chacune des autres applications et la date a laquelle cette valeur « distante » a été lue ou écrite. Cette table permet donc :

- d'avoir un historique des valeurs dans chaque application et de permettre de restaurer d'anciennes valeurs ;
- de s'affranchir des differences de format qu'une valeur peut avoir dans chaque application en stockant la valeur dans un format pivot ainsi que sa représentation dans chacune des applications. Lorsque la valeur est lue plus tard d'une de ces applications, cette valeur est comparée a la valeur « distante » au format correspondant pour détecter si elle a été modifiée. les comparaisons de valeurs se font donc dans le format natif de chaque application connectée.

### ~~4.2. Organisation des processus de synchronisation~~

Une synchronisation se déroule ainsi :

- une synchronisation comment par une phase de lecture des données pendant laquelle, type par type, les données sont récupérées a partir des applications connectées ;
- l'ordre de lecture des types de données permet de récupérer les enregistrements dans un ordre logique facilitant la création des associations entre ces types.

En effet, il est important de connaître d'abord les comptes, puis les prospects et les contacts pour savoir comment les rattacher à leur compte ; le traitement des produits (d'un catalogue d'une boutique en ligne) puis des lignes de commande et les commandes ; puis les opportunités, activités ou autres types de données indéfinis. Pour un type de données, les enregistrements sont lus et comparés application par application, sans ordre particulier prédéfini ; une phase d'écriture pendant laquelle les types de données sont traites également un par un, application par application.

### ~~4.3 Réconcilier les données pour éviter les doublons~~

En phase de mise en place d'une intégration Podbox, il est plus judicieux de faire un import de toutes les données (une phase de lecture complète, sans l'écriture) puis de faire une réconciliation pour rapprocher les enregistrements des différentes applications qui correspondent probablement aux mêmes entrées dans la vie réelle. L'utilisateur peut choisir les critères qu'il juge pertinents pour rapprocher les enregistrements provenant d'applications différentes, comme :

les prénoms et noms de contacts, ou seulement leurs adresses email ;

la raison sociale des compagnies ;

la référence des produits.

Une fois ces critères définis, Podbox remonte les doublons potentiels et laisse la possibilité à l'utilisateur de de / sélectionner les enregistrements à réconcilier.

~~4.4 Modes de synchronisation~~

Il existe les caractéristiques des deux modes de synchronisation de données qui coexistent dans Podbox en prenant l'exemple d'une synchronisation entre un CRM (Salesforce) et un emailer marketing (MailChimp).

Podbox mémorise les identifiants que chaque application affecte automatiquement à chaque contact. Par exemple, le contact Harry Cover a l'identifiant "SF-13" dans Salesforce et "MC-42" dans MailChimp. Ainsi les valeurs (de prénom, nom, email, etc.) peuvent évoluer (correction, modification) mais comme ces identifiants n'évoluent pas, Podbox conserve l'association entre le contact dans Salesforce et dans MailChimp.

Lors d'une synchronisation full, Podbox va demander à Salesforce de lui retourner TOUS les contacts :

- pour chacun des contacts retournés, Podbox va regarder si elle connaît ce contact (en regardant si elle connaît son identifiant) :
- oui, elle le connaît : Podbox va donc vérifier si des valeurs ont été mises a jour (email corrige par exemple) avec les dernières valeurs renvoyées. Si des valeurs ont été modifiées, il faudra mettre à jour ce contact dans MailChimp plus tard ;
- non, elle ne le connaît : c'est un nouveau contact lu de Salesforce, il faudra créer ce contact dans MailChimp plus tard.
- s'il y a des contacts qui avaient été lus auparavant et qui ne sont pas retournés dans ce dernier appel, c'est qu'ils ont été supprimes entre temps dans Salesforce. Il faudra supprimer ces contacts dans MailChimp plus tard.

Ensuite, Podbox demande tous les contacts de Salesforce et fait de meme (detection des valeurs modifiées, des nouveaux contacts et de ceux à supprimer).

Enfin, Podbox va propager les modifications dans une application, puis dans l'autre :

- mise à jour des valeurs modifiées ;
- création des nouveaux contacts. Lorsque Podbox demande à créer un nouveau contact dans une application, celle-ci lui retourne son identifiant, permettant a Podbox de mémoriser le lien ;
- suppression des contacts.

Avantage de la synchro full c'est ce qu'on détecte tous les changements sans incertitude (ajout, mise- à-jour, suppression), on peut se contenter de ce mode.

Inconvénients : cela prend beaucoup de temps et de ressources informatiques s'il y a 15 valeurs à vérifier sur 30 000 contacts dans chaque application alors qu'il n'y a peut-être que deux contacts modifiés et un à supprimer. Pendant ce temps de comparaison de valeurs, les applications ne sont pas à jour.

Podbox mémorise à quel moment chaque contact a été mis à jour, c'est la date de mise à jour (date de calendrier complétée de l'heure / minute / seconde du moment de la mise à jour).

### *4.4. Mode de synchronisation différentielle*

Lors d'une synchronisation différentielle, Podbox rajoute un critère de filtre a sa demande de récupération des contacts : Podbox demande alors à Salesforce et MailChimp de « lui renvoyer les contacts crées ou modifies après 12h32m15s le 13/03/2014 ». Les services web de Salesforce et de MailChimp vont faire le tri de leur coté avant de ne renvoyer que les contacts crées ou mis à jour après cette date.

Si un contact remonte précédemment n'est pas renvoyé, c'est qu'il n'a pas été modifie ou qu'il a été supprimé

Avantage : la réponse est plus légère et rapide à traiter dans Podbox qui synchronise les ajouts / modifications plus rapidement dans l'autre application.

Inconvénient : la réponse ne spécifie pas les contacts supprimés entre temps, Podbox ne peut ni les déduire, ni propager les suppressions dans l'autre application.

### *4.5. Complémentarité des modes*

Il existe deux complémentarité de modes :

- full : synchronisation autonome (tient aussi compte des suppressions) mais lente ;
- delta : synchronisation rapide des contacts nouveaux ou mis à jour mais qui a besoin d'être complétée de temps en temps par une synchro full permettant de détecter les suppressions.

### 4.6. Abstraction de la diversité des connecteurs

Le niveau d'abstraction nous permet de décrire la structure des données échangées avec chaque application indépendamment de son format d'échange.

Le choix du framework SPRING (et d'autres bibliothèques du monde Java) pour construire l'application Podbox nous permet de :

bénéficier de toutes ses bibliothèques pour convertir les données aux formats XML et JSON classiquement utilises par les API des applications professionnelles et de n'avoir à faire de développements spécifiques que pour les cas particuliers ;

de prendre en charge la diversité des protocoles :

- d'authentification : oAuth1, oAuth2, authentification HTML basique, clé d'API en paramètre de requête, etc.
- d'échange de données comme le SOAP, XML-RPC et REST.

Ainsi nous avons équipe la plate-forme Podbox d'une couche d'abstraction des connecteurs pour organiser la mécanique de synchronisation tout en gérant la diversité des technologies mises en œuvre pour chaque connecteur et bénéficier rapidement de comportements mutualises comme le traitement et le reporting des erreurs, qu'elles soient liées aux communications réseaux ou aux problématiques d'intégrité de données définies dans les API de chaque application (impossibilité d'inscrire plusieurs fois le même émail dans une mailing liste donnée, impossibilité de déclarer un contact sans nom, etc.).

## 5. Conclusion

D'après le résultat obtenu on peut faire la conclusion que le système d'information de Podbox simplifie l'intégration et fournit aux business les outils pour répondre plus rapidement aux demandes du marché, à un coût prévisible. L'utilisateur peut ajouter plus que 50 extensions pour le système populaire et régler la synchronisation seulement dans quelques minutes.

## Références

1. D. Pere, L. Sorel. Documantation et travaux de R&D Pod programming.

2. Fowler M. Patterns of Enterprise Application Architecture. Addison Wesley, 2002, 560p.

3. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования. Питер 2010, 358с.

4. Craig Walls. Spring in Action, Third edition. Manning Publications Co., 2011, 424p.

5. M. McLaughlin. MySQL Workbench: Data Modeling & Development. Mcgraw-Hill Osborne Media, 2013, 456p.

*Sous la supervision de (Під керівництвом):*

Jérôme Darmont (PhD, HdR, professeur, ERIC lab)

Дорохов О. В. (к.т.н., проф, кафедра інформаційних систем)

Керівник з іноземної мови ст. викладач кафедри іноземних мов та перекладу Безугла І.В.