E-Vision : élaboration d'une solution de reporting pour un système intégré de gestion de bibliothèque

Samuel Alizier

MEMOIRE

présenté en vue d'obtenir

Le DIPLOME D'INGENIEUR CNAM

SPECIALITE : Informatique

OPTION : Systèmes d'information (ISI)

par

Samuel ALIZIER

E-Vision : Élaboration d'une solution de reporting pour un système intégré de gestion de bibliothèque

Soutenue le 16 Avril 2015

JURY

PRESIDENT :	**Monsieur Christophe PICOULEAU**	**Professeur des Universités - Cnam Paris**
MEMBRES :	**Monsieur Bertrand DAVID**	**Professeur des Universités - ECL Lyon** **Enseignant responsable - Cnam Lyon**
	Monsieur Claude GENIER	**Administrateur du CERN E.R.** **Enseignant-Cnam Lyon**
	Monsieur Laurent DUFOUR	**Directeur Recherche et Innovations Decalog**
	Monsieur Guillaume MESSONNIER	**Responsable développement et exploitation Decalog**

Abréviations

ABF Association des Bibliothécaires Français.

BDD Behavior Driven Development.
BDP Bibliothèque Départementale de Prêts.

CAS Le Central Authentication Service.
CMS content managment system.

ETL Extract Transform Load.

FRBR Functional Requirements for Bibliographic Records.

GPEC gestion prévisionnelle des emplois et des compétences.

HTTP Hypertext Transfer Protocol.

JSON JavaScript Object Notation first.

MDX MultiDimensional eXpressions.
MVC Modèle-Vue-Contrôleur.

ODS Operational Data Store.
OLAP On-line Analytical Processing.
OPAC Online Public Access Catalog.

PHP PHP : Hypertext Preprocessor.
POM Modèle Objet de Projet.

R&D Recherche et Développement.
REST REpresentational State Transfer.
RIA Rich Internet Application.

SaaS Software as a Service.
SIGB Système Intégré de Gestion de Bibliothèques.
SQL Structured Query Language.
SVN Subversion.

URL Uniform Resource Locator.
uuid Universal Unique Identifier.

Glossaire

Base de données relationnelle En informatique, une base de données relationnelle est un stock d'informations décomposées et organisées dans des matrices appelées relations ou tables conformément au modèle de données relationnel. Le contenu de la base de données peut ainsi être synthétisé par des opérations d'algèbre relationnelle telles que l'intersection, la jointure et le produit cartésien.

Datamart (ou magasin de données) Le DataMart est un ensemble de données ciblées, organisées, regroupées et agrégées pour répondre à un besoin spécifique à un métier ou un domaine donné. Il est donc destiné à être interrogé sur un panel de données restreint à son domaine fonctionnel, selon des paramètres qui auront été définis à l'avance lors de sa conception.

Datawarehouse (ou entrepôt de données) Un entrepôt de données (data warehouse) est une collection de données thématiques, intégrées, non volatiles et historisées pour la prise de décisions.

Dimension Un ensemble de données du même type, permettant de structurer la base multidimensionnelle. Une dimension est parfois appelée un axe. Chaque cellule d'une mesure est associée à une seule position de chaque dimension. Temps, pays, produit sont des dimensions classiques.

Fond documentaire Également appelé collection, un fonds documentaire est un ensemble construit de ressources. Les ressources qui constituent le fonds peuvent être de toutes natures : livres, archives, revues, enregistrements sonores ou audiovisuels, etc..

Hiérarchie Les positions d'une dimension organisées selon une série de relations 1-n en cascade. Cette organisation de données est comparable à un arbre logique, ou chaque membre n'a pas plus d'un père mais un nombre quelconque d'enfants.

HyperCube (ou cube) Une construction multidimensionnelle formée de la conjonction de plusieurs dimensions. Chaque cellule est définie par un seul membre de chaque dimension.

SIGB Un système intégré de gestion de bibliothèque (SIGB) est un logiciel destiné à la gestion informatique des différentes activités nécessaire au fonctionnement d'une bibliothèque (gestion des collections et des usagers, de la circulation des documents, des acquisitions, édition de rapports statistiques).

Remerciements

Je tiens à remercier toutes les personnes qui m'ont apporté leur soutien pour la réalisation de ce mémoire.

Je remercie dans un premier temps Monsieur Bertrand David, mon professeur responsable, pour sa disponibilité et ses précieux conseils pour la rédaction de ce mémoire.

Mes remerciements vont aussi à Monsieur Laurent Dufour pour m'avoir permis de choisir e-Vision comme sujet de mémoire et pour m'avoir toujours soutenu tout au long de ma formation et ce dès mes premiers jours d'embauche.

De même, j'exprime ma vive reconnaissance à toutes les personnes de l'équipe e-Paprika. Chacune de ces personnes a en effet apporté à un moment ou un autre de l'aide, des idées ou des conseils pertinents et utiles. Parmi ces personnes, je tiens particulièrement à remercier Cyril Beaumes pour sa patience et la justesse de ses analyses et Guillaume Messionnier pour sa disponibilité et sa pertinence.

Enfin, je tiens à remercier toute ma famille pour son soutien et son aide avec une attention toute particulière pour ma femme Véronique qui m'a supporté et soutenu depuis le début de mon cursus.

Introduction

Les bibliothèques sont de plus en plus informatisées pour leurs tâches du quotidien telles que la gestion du fond, des abonnés ou encore des prêts. Cette informatisation permet de faciliter les recherches, de pouvoir ouvrir le catalogue au public ainsi que d'avoir accès catalogue de documents des Bibliothèques Départementales de Prêts (BDP),. L'offre e-Paprika / e-Carthame est développée par Decalog et offre une solution complète pour répondre à ces différents besoins. Un des autres avantages de l'informatisation est de pouvoir mesurer l'activité d'une bibliothèque via les statistiques.

La solution e-Vision a été développée dans cette optique, pouvoir donner des indicateurs pertinents pour le suivi de l'activité des bibliothèques. Les bibliothécaires doivent en effet pouvoir connaitre l'état de leur fond, les exemplaires qu'il faut remplacer ou encore les abonnés inactifs. Ce mémoire traite de la solution statistique e-Vision. Ce mémoire se décompose en quatre chapitres.

Le premier chapitre présente le contexte du projet. On y trouve une description de l'entreprise Décalog et des différentes solutions proposées par celle-ci. Ce chapitre explique les objectifs et les contraintes du projet e-Vision et donne une vue d'ensemble des méthodes et outils utilisés.

Le second chapitre est un zoom sur le SIGB (Système Intégré de Gestion de Bibliothèques) e-Paprika. Il présente les différents aspects fonctionnels couverts par le logiciel. De plus, afin de comprendre les contraintes d'intégration d'e-Vision, ce chapitre décrit l'architecture du SIGB.

Le troisième chapitre est consacré à la construction de l'entrepôt de données. C'est le cœur du projet, il explique comment sont extraites et transformées les données d'e-Paprika pour pouvoir être intégrées dans e-Vision. Les solutions choisies et les principales méthodes de chargements y sont décrites.

Le quatrième chapitre décrit la solution de reporting, c'est à dire le logiciel que l'utilisateur final va utiliser pour faire ses rapports. On y décrit les principales fonctionnalités, l'architecture utilisée ainsi que les fonctionnalités qui ont été rajoutées.

Enfin, dans la conclusion, nous dressons un bilan du projet et donnons quelques perspectives d'améliorations. De plus nous y faisons un lien avec la formation du CNAM et donnons les principaux enseignements que ce projet a apportés.

Table des matières

Chapitre 1

Contexte du projet

1.1 Présentation de l'entreprise

1.1.1 Présentation générale

Créée en 1987 par Michel Denis, Decalog est une société française éditrice de logiciels de gestion de bibliothèques et de médiathèques au capital de 359 100 €. Ce fut l'une des sociétés pionnières dans le domaine de l'informatisation des bibliothèques.

Suite au rachat de l'entreprise par Jean-Louis Bezin en 2005, la stratégie d'entreprise a évolué. Decalog voulait attaquer le marché des clients de grande et de très grande taille à l'échelle nationale et internationale. C'est d'ailleurs dans cette optique que le logiciel e-Paprika a été lancé.

Le siège social se situe 1244, rue Henri Dunant à Guilherand-Granges (07 500). Décalog possède également plusieurs agences sur Besançon, Nantes, Nice, Paris et Toulouse, lui permettant d'être plus proche de ses clients. La R&D, le S.A.V. et les services généraux sont basés à Guilherand-Granges.

1.1.2 Les différents services

Le service « R&D » Ce service est le plus imposant avec une vingtaine de permanents épaulés par un roulement quasiment permanent d'alternants et de stagiaires. Il est dirigé depuis 2010 par Laurent Dufour, docteur en informatique. Il est essentiellement composé d'ingénieurs diplômés. Les axes de travail sont au nombre de 3 :

1. Les SIGB cœur de métier de Décalog
2. Les portails documentaires qui ont pris un véritable essor ces dernières années avec le Web 2.0.
3. Les systèmes de gestion des espaces publics

Le service « Commercial » a pour mission la réponse aux appels d'offres, la prospection commerciale et les recommandations de développement pour les évolutions des produits de Décalog. Le service commercial est force de proposition et de conseil auprès des prospects et clients. Le service commercial est réparti géographiquement pour être plus proche des clients avec des zones précises. L'entreprise est actuellement présente sur Valence, Paris, Toulouse, Nice, Besançon et Nantes.

Le service « Opération » prend en charge les commandes, conduit les opérations et accompagne le client jusqu'à la mise en service. Ce service comprend le déploiement, la configuration des solutions en cas de clients lourds et d'installation locale. Il prend aussi en charge la formation des bibliothécaires aux logiciels et la migration des données depuis un ancien logiciel. Les opérations de support par maintenance téléphonique et télémaintenance pour les nouveaux clients sont aussi affectés à ce service.

Le service « Support et Qualification» répond aux questions et problèmes logiciels des clients par téléphone ou télémaintenance. Il est également en charge de la validation des nouvelles versions des produits. Il peut ponctuellement réaliser des actions sur site pour régler des problèmes précis ou des formations de courte durée. Un portail de maintenance, Cerheme, a été développé pour améliorer la réponse aux clients, mieux gérer les priorités, inclure une foire aux questions et la documentation.

Le service « Administratif et ressources humaines » L'équipe administrative est chargée de la comptabilité fournisseur, de la facturation et comptabilité client, de la comptabilité analytique et du contrôle de gestion. Elle assiste la direction dans les recrutements, la formation, la gestion prévisionnelle des emplois et des compétences (GPEC) et la gestion sociale de l'entreprise.

1.2 Présentation de la solution logicielle

1.2.1 Les portails documentaires

En quelques années, le portail documentaire est devenu un élément majeur dans la stratégie des bibliothèques pour la reconquête de lecteurs. Le portail public permet aux lecteurs un accès aux informations des bibliothèques en ligne. Un portail documentaire, ou public, se décompose en deux parties :

1. L'OPAC (Online Public Access Catalog) see qui est un module de recherche documentaire et d'accès au compte lecteur.
2. Le CMS pour la création et la publication d'articles dans le but d'informer, mais aussi dans la gestion d'agendas, d'actualités, de foire aux questions, de blogs, etc.

1.2.2 Les gestionnaires d'espaces publics numériques

Dès les années 80, les bibliothèques universitaires ont mis à disposition des ordinateurs pour que les étudiants puissent effectuer par eux-mêmes des recherches documentaires. Ce service a ensuite été étendu aux bibliothèques municipales. Depuis les années 2000, ces mêmes bibliothèques ont souhaité proposer des services complémentaires : accès au réseau Internet, logiciels de bureautique, stockage de documents, impression, consultation de ressources multimédias (musique, films, jeux). Les gestionnaires d'espaces publics numériques répondent à ces besoins. Dans cette catégorie, Decalog propose les logiciels EPM en client lourd et e-EPM en client semi-léger.

1.2.3 Les portails professionnels

Lorsque l'on parle de portail professionnel, on désigne les SIGB. Les SIGB proposent une gestion intégrée de l'ensemble des fonctionnalités utiles aux opérations courantes des bibliothèques organisées en réseaux ou non.

Chaque fonction correspond à un module, on retrouve ainsi les acquisitions, le catalogue et la recherche documentaire, le prêt, le bulletinage ou encore le dépouillement des périodiques. Décalog commercialise deux types d'offres pour ses SIGB : des applications clients lourds (Paprika et Carthame) et des applications en ligne (e-paprika et e-carthame). La tendance commerciale est de remplacer les clients lourds par l'offre full web.

1.3 Expression du besoin et limites du projet e-Vision

1.3.1 e-Paprika / e-Carthame

Parmi les portails professionnels proposés, l'offre e-Paprika et e-Carthame offrent toutes les fonctionnalités pour la gestion quotidienne des bibliothèques et médiathèques. On y trouve ainsi la gestion des prêts, des réservations, du fond et des abonnés. Le public ciblé par ces logiciels sont les bibliothèques et médiathèques de toutes tailles. On trouve aussi bien des bibliothèques municipales de 3000 documents que la bibliothèque des Arts Décoratifs qui compte environ 250 000 documents.

1.3.2 Volumétrie

Les volumétries de e-Paprika sont résumées dans le tableaux ci-dessous.

TABLE 1 – Résumé des volumétries de e-Paprika au 1er Mars 2015

Nombre total de réseaux (mono-site et multi-sites)	242
Nombre de réseaux multi-sites	45
Nombre de bibliothèques	416
Nombre d'utilisateurs	750
Nombre d'abonnés	539 711
Nombre de notices bibliographiques	5 718 725
Nombre d'exemplaires	6 008 737
Nombre de prêts effectués dans le mois	683 504
Nombre de retours effectués dans le mois	610 959
Nombre de prêts en cours à ce jour	812 741

1.3.3 Besoin

e-Paprika facilite la gestion des bibliothèques, cependant, il ne permet pas de donner un état des lieux du fond ou une vision de l'activité des bibliothèques. Le but du projet e-Vision est de proposer un système de statistiques sur l'ensemble des données de e-Paprika.

Le premier objectif est de pouvoir répondre à la collecte des données statistiques des bibliothèques territoriales du Service du Livre et de la Lecture (S.L.L) du ministère de la Culture et de la Communication. C'est un rapport annuel demandé par le ministère pour connaître les comportements de lecture en France et attribuer les budgets aux bibliothèques et médiathèques. Un extrait de ce rapport est présent en annexe III. On retrouve dans ce rapport :

1. Un état des lieux du fond et des acquisitions. On peut prendre pour exemple le nombre d'exemplaires par public visé, par type de documents (sonore, vidéos, textes ...) ou encore le nombre d'abonnements de périodiques.
2. Une vision générale des abonnés. On y trouve par exemple la répartition par âge et par genre des abonnés ou encore le nombre d'abonnés actifs (ayant emprunté au moins un livre)
3. Un aperçu de la répartition des prêts par type d'abonnés ainsi que le nombre de prêts par type d'abonnés (adulte, enfant) et par type de documents.

Le second objectif est de pouvoir offrir aux bibliothécaires un outil suffisamment puissant pour analyser certains comportements tout en restant simple d'utilisation. On peut ainsi imaginer des requêtes qui permettent à un niveau macro de voir quelles tranches d'abonnés empruntent le moins pour trouver un moyen de les attirer. En effet, il peut exister certains manques dans le fond documentaire pour cette tranche d'age. Il faut donc aussi des requêtes qui permettent de connaître le comportement des prêts de ces mêmes abonnés. On peut aussi imaginer une requête qui donnerait les 10 ou 100 exemplaires les plus empruntés pour savoir s'il faut en commander de nouveaux ou pas. L'outil doit aussi permettre de faire des requêtes beaucoup plus précises. On peut prendre l'exemple du détail des prêts d'un exemplaire pour comprendre pourquoi il est en mauvaise état. Ce ne sont que des exemples isolés de ce que les bibliothécaires ont besoin pour prendre des décisions.

Le troisième objectif d'e-Vision est de proposer des listes et de pouvoir les exporter en fichiers Excel. En effet, les données de e-Vision ne sont pas anonymes, on peut donc extraire des liste utiles aux fonctionnement quotidien des bibliothèques. On peut par exemple imaginer vouloir la liste de tous les abonnés qui ont un prêt actuellement en retard, ou la liste de tous les exemplaires qui sont en mauvaise état.

Le quatrième but d'e-Vision est de proposer un système de vérification de données. Le logiciel doit pouvoir donner la liste des abonnés qui sont inscrits à une newsletter mais qui n'ont pas d'adresse mail renseignée. On peut aussi y trouver les incohérences telles que les exemplaires physiques sans cote ou encore une incohérence entre un prix, une remise et un prix remisé.

1.3.4 Contraintes

En grande majorité, les bibliothécaires n'ont pas de formation informatique ni de connaissance en statistiques, c'est pourquoi e-vision doit offrir une souplesse et une grande facilité d'utilisation.

e-Vision doit parfaitement s'intégrer à e-Paprika. L'utilisateur ne doit pas se rendre compte qu'il quitte le système de gestion de bibliothèque. Il ne doit pas non plus s'authentifier spécifiquement, et doit pouvoir naviguer de l'un à l'autre. E-Vision doit donc être vu comme une brique de l'application.

Les requêtes statistiques sont souvent gourmandes en ressources machine. e-Vision ne doit donc pas perturber le fonctionnement quotidien d'e-Paprika lorsque les utilisateurs génèrent des tableaux statistiques. De ce fait, la solution doit avoir sa propre base de données et se synchroniser en dehors des horaires d'exploitation.

Les performances d'e-Vision doivent aussi être au cœur des préoccupations des développements. En effet, sans atteindre des volumétries réellement problématiques, les données d'e-Paprika sont suffisamment conséquentes pour que la question des performances se pose. Certaines tables atteignent quelques millions de lignes et les rapports statistiques doivent pouvoir sortir en quelques secondes (voir minutes pour les plus complexes). Cette contrainte, liée au fait que les clients doivent pouvoir faire les requêtes qu'ils souhaitent, pose de réelles difficultés.

La solution doit aussi s'appuyer sur l'existant. L'équipe en place sur e-Paprika a déployé des outils pertinents pour leur solution logicielle, le projet de statistiques doit donc s'intégrer au mieux dans cet environnement. En effet, le déploiement continu, les différentes plates-formes, les rythmes de développements sont autant d'éléments qu'il faut prendre en compte.

1.3.5 Limites du projet

Ce projet n'inclut pas à proprement parlé d'études de données. En effet, les données seront manipulées pour être intégrées dans un entrepôt de données et les utilisateurs ont la possibilité de coiser les données existentes. Cependant, il n'y a pas de phase d'exploration de données (data-mining). De plus, dans un premier temps, seul e-Paprika, rentre dans le cadre du projet e-Vision.

1.3.6 Articulation du projet

Comme on peut le voir sur la figure 1, le projet se sépare en deux grands axes. Le premier est basé sur l'entrepôt de données, le second sur l'intégration du logiciel qui présente les rapports à l'utilisateur.

Le projet a été initialisé en mi-2012. J'ai repris la responsabilité de la partie technique début 2013. Le choix des solutions avait été fait avant mon arrivée sur le projet.

Pour la partie entrepôt de données, mon rôle a été réparti sur plusieurs axes.

1. Il fallait construire les nouveaux faits comme par exemple le fait des prêts fonctionnels, des visites, ou encore les exemplaires et les prêts numériques.
2. La construction des magasins de données devaient être refaite complètement car l'existante présentait beaucoup de risques de régressions.
3. Le développement sur l'entrepôt de données est aussi passé par une phase de stabilisation et d'optimisation. C'est le cas par exemple de la construction du fait des exemplaires qui demandait trop de ressources mémoires.

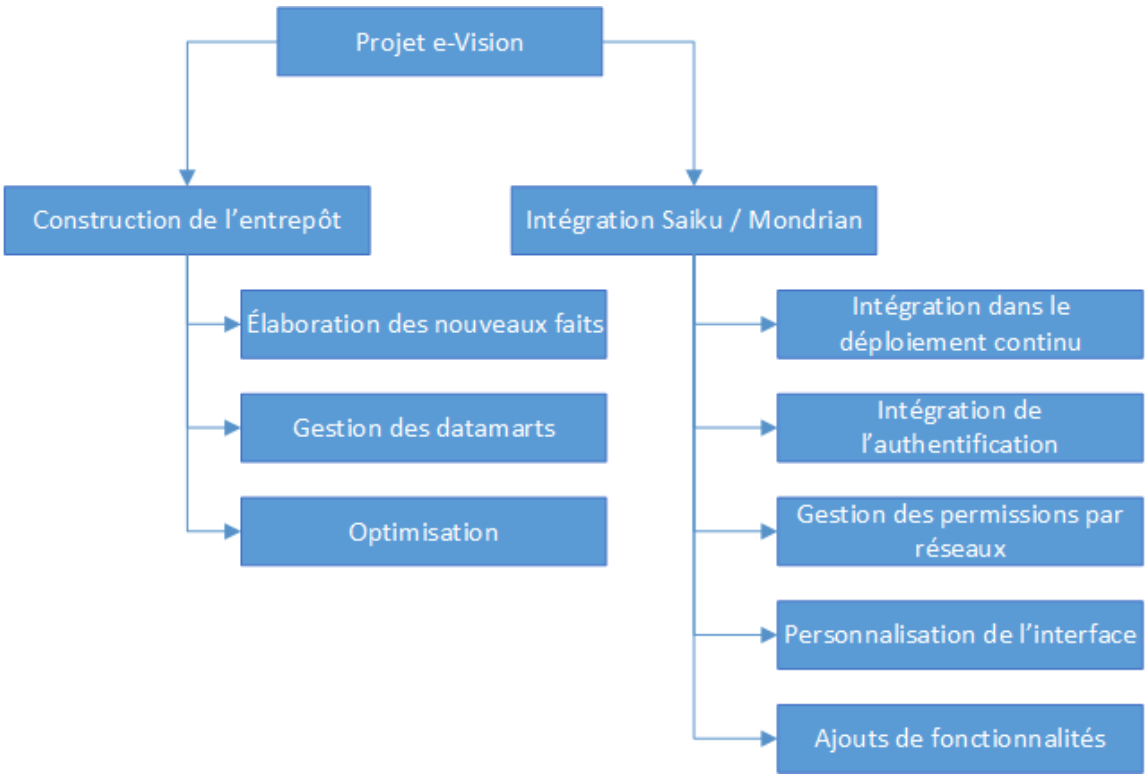


FIGURE 1 – Organigramme des tâches pour le projet e-Vision

De plus la solution de reporting a été complètement refaite suite à une montée de version de l'outil utilisé. Certaines fonctionnalités ont aussi été rajoutées. Il a fallu intégrer la solution statistique dans les processus de e-Paprika. Les principaux axes de développements sont décrits dans la partie droite de l'organigramme de la figure 1.

1.3.7 Mon Rôle

Bien que j'ai participé à l'analyse sur les différentes améliorations à apporter au produit, mon rôle principale était d'être le référent technique de l'application. Je prenais en charge les développements, le chiffrage des histoires, ainsi que les opérations d'optimisations et de maintenance.

1.3.8 Montée de version

La principale tâche que j'ai eu à accomplir a été la montée de version de Saiku. Nous somme passé de la version 2.4 à la 2.5. Il a donc fallu reporter toutes les fonctionnalités existantes au moment de la montée de version.

Faire un différentiel massif des modifications n'était pas pertinent. En effet, le code de l'application avait changé et certaines améliorations n'étaient pas compatibles avec le nouveau code. La stratégie a été de prendre toutes les fonctionnalités développées et de les reporter une par une dans la nouvelle version. Pour ce faire, je me suis appuyé sur les fiches Youtrack relatives à e-Vision. Chaque fiche correspondait à une fonctionnalité.

1.4 Outils et méthodes du projet

1.4.1 Description d'un environnement

Un environnement e-Paprika est constitué de plusieurs serveurs. On peut observer sur la figure 2 l'organisation de ces différents serveurs.

On note sur cette figure que l'utilisateur a un accès direct à deux serveurs, E-paprika et CAS. Pour rappel, CAS (Central Authentication Service) sert pour l'authentification et la sécurité du logiciel. Les communications entre les serveurs se font via le protocole HTTP (Hypertext Transfer Protocol) avec du JSON, du XML ou HTML selon les besoins. Les bases de données sont installées sur des serveurs différents. De cette manière la charge est répartie, et les ressources sont partagées au mieux. Lorsque l'on parle de serveurs dans le cas présent, on parle de machines virtuelles Toutes les machines sont dans une solution « cloud » d'OVH.

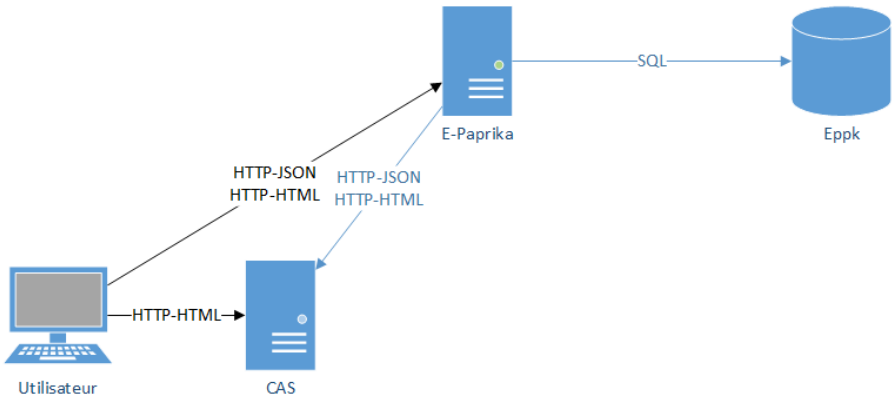


FIGURE 2 – Architecture des serveurs e-Paprika

1.4.2 Les différents environnements

Dans le projet e-Paprika, il existe plusieurs plateformes qui servent aux différentes étapes de validation et à l'exploitation du logiciel. Les postes des développeurs ne sont pas abordés dans cette section mais dans celle des méthodes et outils.

Développement

L'environnement de développement est utilisé pour tester la version en cours de développement, que l'on nomme le "trunk". C'est la version qui n'est pas encore mise en ligne pour les clients, elle sert donc pour les nouvelles fonctionnalités. Elle est déployée deux fois par jour, une fois le matin et une fois le midi. De cette manière, les tests peuvent être réalisés au plus proche du développement. Il n'existe qu'un seul serveur pour cette plateforme. Toutes les briques de e-Paprika sont sur le même serveur physique hébergé dans les locaux de Decalog.

Qualification

La plateforme de qualification sert pour la version "stable" du logiciel. Elle permet essentiellement de corriger les bugs qui sont reportés par les utilisateurs. Son architecture est sensiblement la même que celle des plateformes de production. Cependant, la plateforme de qualification n'a pas la même volumétrie que les productions et donc des ressources moindres (processeur et mémoire). Le serveur de qualification est mis à jour tous les matins.

Les productions

Il existe actuellement quatre plateformes de production. Les productions 1 et 2 présentent à elles deux 95% des clients e-Paprika, ces clients ne sont pas sur le même environnement pour des questions de répartition de charge. L'environnement de production numéro 4 sert pour les clients qui ne sont pas sur le fuseau horaire de la France. De cette manière, il est plus facile de décaler les traitements par lot hors des horaires de travail.

La pré-production

La pré-production sert essentiellement pour des tests de performances. Cet environnement dispose des mêmes ressources qu'une production. Cependant, les données présentes dans la base sont constituées du cumul des bases de production 1 et 2, on compte donc environ 5 millions d'exemplaires pour 4.5 millions de notices. Cette double volumétrie permet des tests de performances efficaces et pertinents puisque potentiellement on pourrait accueillir le double de la volumétrie actuelle sur chacun des serveurs de production.

1.4.3 Équipe en place

L'équipe fonctionnelle

L'équipe fonctionnelle est composée de quatre personnes dont un chef de produit. Elle est en charge de la rédaction des fonctionnalités et de la priorisation de celles-ci, en accord avec le comité de pilotage de l'entreprise. Au sein de cette équipe, deux personnes sont en charge des tests des fonctionnalités et du support.

Développement

L'équipe de développement est composée de cinq à six personnes. Elle est en charge des développements, des corrections de bugs. Elle est responsable des chiffrages des histoires et apporte les connaissances pour faire le lien entre le métier et l'aspect technique.

Exploitation

L'équipe d'exploitation est responsable de tout ce qui touche aux serveurs de qualification, de pré-production et de production. Elle fait les mises en production, optimise les processus et les configurations. Elle aussi assure une veille régulière. En effet, le mode SaaS (Software as a Service) ne donne pas le droit à l'erreur sur ce point.

1.4.4 Méthode SCRUM

Le projet dans son ensemble utilise la méthodologie SCRUM.

Résumé de la méthode SCRUM

Pour résumer, SCRUM est une méthodologie de développement agile. Le projet s'organise autour d'itérations. On définit une période, nommée "sprint", à la fin de laquelle, on doit être capable de présenter un livrable testé. Le périmètre du sprint est défini de manière à avoir la plus grande valeur métier possible. Dans le cas d'e-Paprika, les "sprints" durent deux semaines.

Les développements s'articulent autour d'histoires. Une histoire correspond au développement d'une fonctionnalité. Lorsqu'il y a plusieurs histoires sur un même thème, on parle alors d'"épic". L'histoire est l'élément de base des développements.

Le chef de produit, dans le cas d'e-Paprika, l'équipe fonctionnelle, doit écrire toutes les histoires. Une fois écrites, ces histoires sont placées dans ce que l'on appelle l'"Icebox". Cet élément contient toutes les histoires qui ne sont pas priorisées. La seconde étape consiste à **prioriser** les histoires pour les développeurs. La priorisation se fait dans un autre ensemble appelé le "BackLog".

L'estimation du temps de développement des histoires se fait par les développeurs selon le principe des « histoires utilisateur ». Plutôt que de chiffrer de manière classique les durées de réalisation en jours de développement, on attribue des points aux histoires. Il est pratiqué des **estimations relatives**. Par exemple, une histoire que l'on estime simple se verrait attribuer un point. Celle qui pourrait prendre deux fois plus de temps se verrait donc attribuer 2 points. Ces estimations peuvent être faites très tôt (car il est facile de les comparer à des histoires existantes) sachant que plus le nombre de points est grand, moins l'estimation est précise.

Le nombre de points qui peuvent être réalisés dans un sprint est appelé la **vélocité**. C'est la vélocité qui permet de faire des plannings à long terme. Les estimations des histoires sont définitives à de rares exceptions près, comme un changement de périmètre important et inévitable. Si des histoires présentent des temps de développements différents de l'estimation, c'est la vélocité qui évoluera et donc le planning long terme. On dit que la vélocité adapte automatiquement le planning.

Les avantages de la méthode SCRUM

Du fait du développement par itérations courtes, cette méthode permet d'identifier rapidement les problèmes. On évite donc l'effet tunnel qui est le principal inconvénient d'une méthode en cycle en "V". Un des autres aspects de la méthode est l'amélioration continue.

Une réunion très courte est faite tous les matins. Lors de cette réunion appelée "SCRUM", chacun donne son avancement et éventuellement les difficultés rencontrées. Cette réunion ne doit pas dépasser dix minutes, mais elle permet de connaître l'avancement de chacun et surtout de savoir ce que chacun fait.

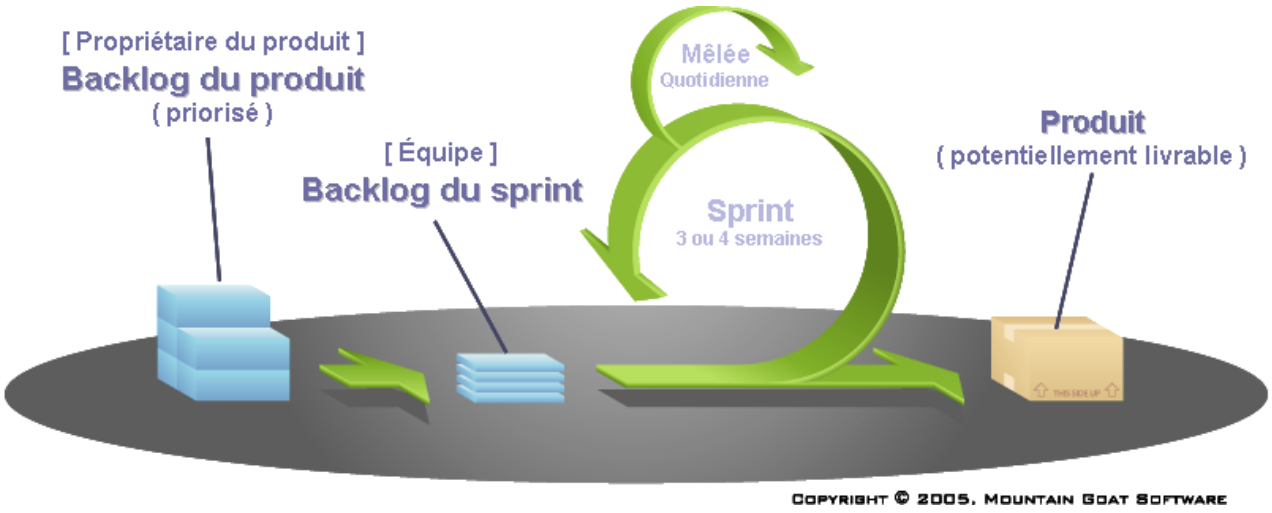


FIGURE 3 – Illustration de la méthodologie SCRUM (Wikipedia)

Dans le principe de l'amélioration continue, une réunion est faite à chaque sortie de version. Une version dans e-Paprika comprend des «Epic» complètes. Une version est installée pour les clients environ tous les deux mois. Lors de cette **rétrospective**, on aborde les différentes actions à mener pour améliorer le produit. Tous les points sont discutés, de l'ambiance de l'équipe à la stabilité de l'application en passant par la qualité de rédaction des histoires.

1.4.5 Outils de développement

Eclipse

Initié par IBM puis ouvert à la communauté Open Source, l'environnement de développement intégré (IDE) Eclipse pour développeurs Java présente l'énorme avantage de disposer d'un nombre impressionnant de plug-in l'enrichissant dans tous les domaines de la programmation : liens subversion, liens Maven, etc. Fonctionnant en Java, l'IDE Eclipse est indépendant de la plate-forme du développeur (Windows, GNU/Linux, etc) Il est actuellement utilisé pour les développements Java à Decalog.

NetBeans

C'est un autre IDE qui fonctionne sous Java. Il est donc aussi indépendant du système d'exploitation de l'utilisateur. Il est utilisé pour le développement du frontend. Il est en effet plus intuitif et plus ergonomique pour le développement en JavaScript.

Gestionnaire de sources : Subversion

Subversion (en abrégé SVN) est le système de gestion de versions actuellement utilisé à Decalog. Il permet d'avoir un historique de tous les développements, de pouvoir faire des retours arrières sans difficultés. Il sert de base au déploiement continu effectué par Jenkins.

Gestion de construction (compilation, etc.) : Maven

Apache Maven est un outil libre pour la gestion et l'automatisation de production des projets logiciels Java. L'objectif recherché est de produire un logiciel à partir de ses sources, en optimisant les tâches réalisées et en garantissant le bon ordre de fabrication. Maven utilise un paradigme connu sous le nom de Project Object Model (POM). Ce paradigme permet de décrire un projet logiciel, ses dépendances avec des modules externes et l'ordre à suivre pour sa production. Il est livré avec un grand nombre de tâches prédéfinies, comme la compilation de code Java. Maven est actuellement utilisé à Decalog pour la gestion de production des projets Java.

Intégration continue : Jenkins

L'intégration continue est un ensemble de pratiques utilisées en génie logiciel. Elles consistent à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression de l'application en cours de développement.

Jenkins est un outil open-source d'intégration continue écrit en Java, fonctionnant dans un conteneur de servlets tel qu'Apache Tomcat, ou en mode autonome avec son propre serveur Web embarqué.

Gestion du planning : Pivotal Tracker

Comme nous l'avons vu ci-dessus, e-Paprika / e-Carthame est développé avec la méthodologie SCRUM enrichie par la technique des user stories. La technique traditionnelle est d'avoir un mur avec des Post-it (un Post-it par user story). Le problème est que le projet d'e-Paprika est trop conséquent pour pouvoir prétendre reposer sur ce moyen. Dans ce contexte, la technique traditionnelle montre rapidement des limites. En effet, il n'y a aucune recherche textuelle rapide ni traçabilité ou de projection du planning.

C'est pourquoi Pivotal Tracker est utilisé, c'est un outil SaaS fourni par Pivotal Labs. Il est composé d'un tableau de bord où chaque histoire apparaît. Les histoires sont classées selon leur état dans le flux de travail. On a donc une colonne qui référence la "icebox", une pour le backlog et les histoires en cours. L'interface est cependant paramétrable et permet à chacun de personnaliser son espace de travail. Il permet aussi le calcul de la vélocité sur les trois derniers sprints, la projection du planning ou encore la simulation d'un changement de vélocité.

Gestion des spécifications et des bugs : YouTrack

Comme énoncé dans la section relative à pivotal, e-Paprika est un logiciel complexe, il faut donc un outil pour pouvoir saisir les demandes de fonctionnalités et les bugs remontés. Pivotal Tracker permet de gérer du contenu mais cette gestion est beaucoup trop superficielle. Il n'y a, par exemple, pas de priorité, ni de recherche sur les attributs, on ne peut pas joindre des fichiers facilement. C'est pourquoi on utilise YouTrack édité par la société JetBrains. YouTrack est un système de suivi des fonctionnalités, aussi appelé "bug tracker". Il présente les fonctionnalités suivantes :

- éditeur complet
- recherche plein texte et filtres sur des attributs
- workflow simple et personnalisable
- fichiers faciles à joindre, notamment un outil de capture d'écran très intuitif.

Lien entre planning et spécifications : ALMIT

Il existe donc deux principaux outils pour gérer le cycle de vie de l'application. Cependant, il est nécessaire d'avoir une homogénéité et une synchronisation entre ces outils.

Par exemple, une fonctionnalité saisie dans YouTrack doit se retrouver dans la icebox de Pivotal. Ensuite, lorsque l'histoire est prise par une personne, il faut que son état dans le workflow change dans les deux logiciels.

Il est donc hautement nécessaire d'avoir une automatisation pour maintenir cette homogénéité. C'est pour cela que Decalog a créé l'outil ALMIT (pour Application Lifecycle Managment Integration Tools) : Cet outil créé automatiquement une histoire dans Pivotal quand un ticket a été créé dans YouTrack. Il maintient ensuite automatiquement la cohérence entre les états YouTrack et Pivotal. Enfin, il ajoute un commentaire dans le ticket YouTrack à chaque commit sur le SVN.

Machine virtuel - VirtualBox

Pour l'environnement de développement, il est fait appel à des machines virtuelles. Une machine virtuelle est une émulation d'une machine complète sur un poste hôte. Par exemple, on peut avoir un environnement linux complet dans une machine virtuelle alors que le poste de travail est sous windows. Cette solution permet d'avoir des environnements de développement qui ont des configurations au plus proche de celle de production. De plus, elle permet de pouvoir allumer et éteindre les machines selon les environnements utilisés. Par exemple, lors de la correction d'un bug, on lance la machine sur laquelle est installé l'environnement stable. A contrario, lorsque l'on développe une fonctionnalité, c'est la version « trunk » qui est lancée.

Vagrant

On utilise Vagrant pour harmoniser et automatiser la configuration des machines virtuelles de chaque poste. Vagrant propose de configurer des environnements complets via un simple fichier de configuration. De cette manière, chaque machine aura les mêmes caractéristiques sur chaque poste.

Puppet

Le couple Vagrant / VirtualBox garantit que chaque machine aura les caractéristiques voulues. Cependant, l'installation des différentes couches logicielles telles que Java, MySql ou encore l'application e-Paprika doit aussi être standardisées. C'est là que rentre en jeu Puppet. C'est un gestionnaire de configurations de machines, qui permet via des fichiers de configuration d'installer tous les composants nécessaires.

L'ensemble des fichiers de configurations Puppet et Vagrant sont stockés dans l'outil de gestion de version. De cette manière, chaque développeur a juste à récupérer et à installer vagrant et VirtualBox, récupérer les sources d'e-Paprika et lancer une commande Vagrant. Ces machines virtuelles sont alors les mêmes que celles de toute l'équipe. De plus s'il y a besoin de faire évoluer une configuration, il suffit de modifier les sources sur SVN, et chaque développeur peut répercuter les changements sur son poste.

Il est à noter que Puppet est aussi utilisé pour les déploiements des environnements finaux. A l'heure où est écrit ce rapport, le serveur de développement et les postes des développeurs n'utilisent pas exactement les mêmes configurations Puppet que les productions, la qualification ou la préproduction. Cependant, un développement est en cours pour harmoniser encore plus le processus.

1.5 Synthèse

Ce premier chapitre permet présenter l'entreprise Decalog et son offre logicielle. Il décrit le besoin, les contraintes et les limites d'e-Vision ainsi que les outils et méthodes déjà utilisés.

e-Vision étant le module de statistique d'e-Paprika, le prochain chapitre va présenter un peu plus en détail ce dernier. Cette présentation s'articule autour de deux axes. Le premier axe aborde l'aspect fonctionnel afin de comprendre le métier et le but d'e-Paprika. Le second axe permet d'appréhender l'environnement technique sur lequel on a greffé e-Vision.

Chapitre 2

Description du SIGB e-Paprika

2.1 Présentation du SIGB e-Paprika et e-Carthame

2.1.1 Généralités

En 2010, Décalog a lancé un vaste projet de développement d'une nouvelle offre « dans les nuages ». Un prototype d'e-Paprika a été présenté en juin 2011 au congrès de l'ABF (Association des Bibliothécaires Français). La première mise en production a été réalisée en décembre 2011 pour un premier client, l'Institut Français de Londres. Au 1er juillet 2013, 98 bibliothèques avaient accès à cette solution novatrice par les points suivants :

1. Une approche ontologique de l'information bibliographique via le FRBR (Functional Requirements for Bibliographic Records).
2. Une approche agrégative de cette même information bibliographique.
3. Une recherche documentaire universelle et performante.
4. Une réelle interopérabilité avec le système d'information du client et ses fournisseurs.
5. Un fonctionnement « dans le nuage ».

C'est une stratégie de livraison mensuelle ou bimestrielle qui est appliquée. En effet, les processus de développement agile et d'intégration continue offrent la possibilité de générer régulièrement des versions et de les déployer rapidement sur les serveurs. Les clients peuvent ainsi bénéficier des nouvelles fonctionnalités rapidement.

e-Carthame a été présenté en juin 2013 au congrès annuel de l'ABF. Cette solution est à destination des bibliothèques exigeantes ayant besoin d'une large couverture fonctionnelle et d'un hébergement performant et hautement sécurisé. Elle repose sur le même socle que e-Paprika, mais fait partie d'une offre de services beaucoup plus complète.

Dans ce mémoire, il n'est pas fait de différence entre les deux offres. Dans la suite du rapport, on utilise donc le terme e-Paprika pour désigner le SIGB, cependant, tout ce qui est décrit s'applique aussi pour e-Carthame. Ces logiciels sont commercialisés en mode SaaS, ce qui signifie que les clients utilisent le logiciel comme un service, via internet grâce à un navigateur web dans le cas d'e-paprika. La section 1.6 revient plus en détail sur le mode SaaS.

2.1.2 Quelques aspects fonctionnels

Le monde de la « bibliothéconomie » est beaucoup plus compliquée que ce que l'on peut s'attendre de prime abord. Ce document n'a cependant pas vocation à décrire les aspects fonctionnels poussés de ce métier. Les paragraphes qui suivent présentent les principales fonctionnalités couvertes par e-Paprika sans rentrer dans les détails.

La notion d'Abonnés

Les abonnés constituent la "clientèle" de la bibliothèque. L'abonné est l'acteur principal dans la bibliothèque, il est au centre de beaucoup d'attention dans les outils de statistiques.

Ce module concentre

- La gestion des informations personnelles de l'abonné comme l'adresse, le genre ou encore la date de naissance. Il faut cependant faire attention, un abonné n'est pas forcément une personne physique mais peut aussi être un groupe comme une classe d'école par exemple.
- La gestion des modes de communications pour les relances ou les courriers d'information.
- La gestion des inscriptions, cotisations et éventuellement des dettes.
- La gestion des regroupements, en effet, les abonnés peuvent faire partie de regroupements tels que des familles.

La notion de gestion du fond documentaire

Le fond d'une bibliothèque est constitué de l'ensemble des exemplaires et des périodiques qu'elle détient. Le fond est décrit à partir de deux notions, les notices bibliographiques et les exemplaires. La notice bibliographique contient tous les éléments descriptifs des ouvrages et des documents tels que l'auteur, le titre, l'éditeur ou la date et le lieu de publication. Toutes ces informations permettent d'identifier de manière sûre et unique un ouvrage.

Une autre notion complète la notice bibliographique, c'est le document physique, dans e-Paprika, on le nomme exemplaire. On y retrouve par exemple, le code barre, l'état général de l'ouvrage ou encore la cote. Ces informations sont rattachées au document physique, donc, en cas de suppression de l'ouvrage ces données sont "effacées" puisque non pertinentes, à contrario des informations de la notice bibliographique qui elles restent pertinentes.

La notion de circulation

La circulation documentaire est l'ensemble des mouvements appliqués aux documents d'une bibliothèque. On y retrouve donc, les prêts et retours évidemment mais aussi les réservations. Ce module prend aussi en charge d'éventuelles dettes des abonnés (comme les dettes dues aux retards par exemple).

La circulation des documents fait le lien entre les abonnés et le fond d'une bibliothèque. Elle a aussi une influence sur le fond puisqu'elle introduit des notions de disponibilité et de localisation pour les exemplaires.

Module de gestion du budget

La gestion des budgets dans une bibliothèque est aussi une responsabilité du SIGB et il convient de le mentionner dans ce rapport. Cependant, le développement de ce module est prévu dans le courant de l'année 2015. Ce module sera donc intégré plus tard dans les statistiques.

Organisation en réseau de bibliothèques

Un client dans e-Paprika peut être soit une bibliothèque soit un regroupement de bibliothèques. On nomme ces regroupements des « réseaux ». On préfère le terme « site » à Bibliothèque car ce dernier est trop restrictif. En effet, les sites peuvent être des bibliothèques, mais aussi des médiathèques, des ludothèques ou encore des centres de documentation. On parle donc de sites organisés en réseau.

2.2 L'architecture technique d'e-Paprika

e-Paprika repose sur une architecture multi-couches. On observe sur la figure 4 l'architecture technique d'e-Paprika.

2.2.1 La partie frontend

e-Paprika est ce que l'on appelle un logiciel full web, c'est-à-dire que c'est via un navigateur internet que l'utilisateur a accès aux différents services fournis. L'interface utilisateur est développée sous la forme d'une application web ou RIA (Rich Internet Application). La partie fronted est présente dans le cadre vert de la figure 4.

RIA signifie qu'il n'y a pas d'actualisation complète de la page lors de la navigation : toute ergonomie repose sur des rechargements partiels du contenu de la page en Javascript. Dans e-Paprika c'est le framework ExtJs qui a été retenu pour exécuter cette tâche. Il permet d'avoir les outils nécessaires au dialogue avec le serveur. Le rôle de cette couche est de proposer une interface à l'utilisateur, elle n'est pas responsable de la logique métier.

La couche PHP est responsable des tâches suivantes

- interagir avec le backend
- conserver la session de l'utilisateur (et également l'authentification CAS, cette notion est détaillée au paragraphe 2.2.4)

2.2.2 Le backend

Le backend est codé en Java et se retrouve dans le cadre bleu de la figure 4. C'est lui qui est responsable de la logique métier. Les actions exécutées en base de données sont aussi exécutées par le backend. On distingue aussi trois couches dans cette partie du programme, les web-services, la logique métier et l'accès aux données.

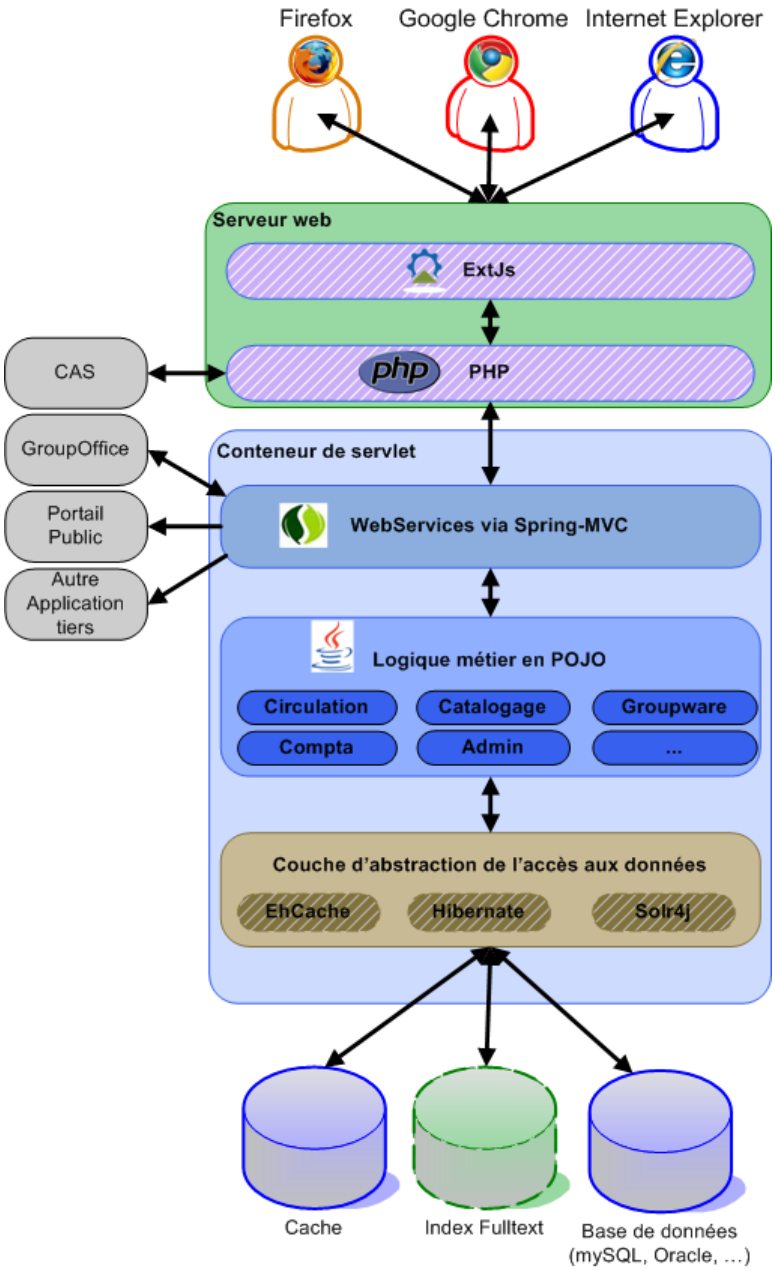


FIGURE 4 – Architecture logicielle d'e-Paprika

La couche de services web

Cette couche permet de dialoguer avec le frontend. C'est la couche la plus proche de l'utilisateur, elle propose un certain nombre de services basés sur l'architecture REST (REpresentational State Transfer).

Cette architecture est construite autour du protocole HTTP pour dialoguer entre le client et le serveur. On utilise la requête HTTP et ses composantes pour interroger le web-service. Une réponse basée sur le protocole HTTP est utilisée pour récupérer les informations. Lors de la requête, le verbe du protocole HTTP est utilisé pour décrire le type d'action qui est associé à un web-service.

On dénombre 4 actions (ou verbes) différentes dans l'application dans ce fonctionnement. Les quatre sont utilisées dans e-Paprika. Cependant, ce n'est pas directement la couche de web services qui s'occupe de ces actions. Cette couche a juste pour vocation d'interpréter la requête et de fournir la réponse renvoyée par le modèle.

- L'action GET implique une opération sans effet sur l'application, elle est dite neutre. On retrouve sous ce verbe essentiellement des accès aux données en lecture.
- L'action POST implique une création de ressource dans l'application. Ces actions initialise un identifiant pour la ressource créée de manière à pouvoir interagir avec.
- L'action PUT est le premier verbe qui permet d'interagir avec une ressource existante, il implique une mise à jour de la ressource. On retrouve donc un identifiant de ressources.
- L'action DELETE implique une suppression de ressource. On retrouve aussi un identifiant dans les données de la requête.

Les notions de création, de mises à jour ou de suppression peuvent se traduire par des actions en bases de données ou sur un système de fichiers.

En plus de l'action, il y a deux autres composantes principales dans la requête HTTP. L'URL identifie une ressource unique dans le logiciel, elle doit être le plus explicite possible quant à son utilité. Pour être plus précis, c'est le couple identifiant/verbe qui doit être unique.

Enfin, la ressource permet de véhiculer l'information. Dans e-Paprika, le format choisi est le JSON, qui permet un dialogue très facile entre les différents langages utilisés avec une structure qui est suffisamment verbeuse pour être comprise, mais qui offre de la souplesse et de la légèreté. Enfin, la couche Web-services gère la "sécurité", c'est-à-dire l'authentification et les autorisations d'un utilisateur. Chaque service vérifie les autorisations de l'utilisateur courant pour son action.

La réponse du web services s'articule autour de deux composantes essentielles. Le code réponse HTTP et le corps de la réponse. Le corps de la réponse est au format JSON et contient les informations utiles que l'utilisateur a demandées. Le code HTTP donne le statut de la réponse. On retrouve les codes les plus connus tels que

200 : Requête traitée avec succès

401 : Ressource non autorisée pour l'utilisateur actuel

403 : ressource interdite.

404 : URL non trouvée

500 : erreur du serveur

503 : service indisponible

La logique métier

La couche de logique métier implémente les fonctionnalités du logiciel. C'est un ensemble de classes Java qui est responsable de toute la logique du logiciel, les différents choix fonctionnels sont toujours pris à ce niveau.

C'est aussi la couche qui est la plus testée. Les tests dans e-Paprika se font essentiellement autour de la bibliothèque jBehave qui permet de scénariser les tests. De cette manière, le logiciel peut être développé selon le principe du développement dirigé par le comportement aussi appelé BDD (Behavior Driven Development).

L'accès aux données

e-Paprika se base sur deux sources de données. Une base MySql est utilisée pour les données métier. La solution Lucene/Solr, un moteur d'indexation de texte, est utilisé pour pouvoir faire des recherches plus rapidement. Les objets utilisés dans cette couche n'ont pas de comportement.

Dialogue entre les 3 couches

Le lien entre les trois couches se fait grâce à l'injection de dépendances. L'injection de dépendances est un patron de conception logiciel qui permet à un objet d'en utiliser un autre dynamiquement plutôt que statiquement. En effet, l'injection de dépendances se fait au moment de l'exécution et non directement dans le code. Une classe peut en utiliser d'autres via une interface.

Par rapport à l'architecture décrite dans les paragraphes précédents, on retrouve donc principalement les injections suivantes.

- Les DAOs (i.e DAO = objet pour accéder aux données) dans les Services (i.e objet pour gérer la logique métier)
- des Services dans les Controllers (i.e objet pour exposer une fonctionnalité via HTTP-JSON)

Cependant, il faut bien prendre en compte que le dialogue est unidirectionnel. En effet, la couche d'accès aux données ne peut pas utiliser les Services ni les Controllers tandis que la couche de logique métier ne peut pas utiliser les Controllers. Cet aspect unidirectionnel est le principe fondamental du backend. Ceci permet en effet de garder une cohérence dans les développements et une robustesse de l'application.

Pour résumer, l'injection de dépendances apporte trois aspects. En premier lieu, cela permet de réduire le couplage entre les différentes classes et d'augmenter la cohésion. En effet, chaque classe à une et une seule responsabilité, ce qui apporte aussi une meilleure réutilisation du code. En second lieu, elle améliore la testabilité des classes. Il est en effet facile de fournir des collaborateurs fictifs à une classe afin de prédire son comportement dans un test unitaire. Enfin, le troisième point apporté est l'augmentation de la maintenabilité. Il n'y a pas d'effet de bord lors de la modification de l'implémentation puisque ce sont des interfaces qui sont injectées.

2.2.3 Stockage des données

Types de sources de données

Il existe deux sources de données principales dans e-Paprika. La première, celle qui nous intéresse le plus, est la base de données MySql. C'est une structure relationnelle classique de troisième forme normale à quelques exceptions près. Cette base de données tourne sous MySql avec le moteur innoDb.

Il existe deux principaux moteurs pour mySql, MyIsam et innoDb. MyIsam est plus rapide en lecture et en écriture qu'innoDb. Cependant, innoDb garantit l'intégrité des données via les clés étrangères et présente un système de transactions. Lorsque l'on veut faire une modification, le moteur ouvre une transaction, fait ses mises à jour et ne les valide qu'au moment du « commit ». Si une erreur survient avant le commit, il est possible de faire un retour arrière sur les données. MyIsam ne fournit pas ces deux fonctionnalités. C'est pour ces raisons qu'innoDb a été choisi comme base de données d'e-Paprika.

La seconde source est un index Solr qui sert essentiellement à accélérer et améliorer les requêtes de recherches. Toutes les données présentes dans Solr proviennent de la base de données relationnelle.

Organisation des bases de données

Au début d'e-Paprika, il a été fait le choix de mettre tous les réseaux de bibliothèque dans la même base de données. Cependant, il existe maintenant plusieurs environnements qui tournent en parallèle. On a donc plusieurs bases de données qui contiennent chacune un ensemble de clients.

Cette décision a été motivée principalement par des contraintes d'exploitation. Ainsi, il a été mis en place un environnement qui permettait de faire des mises en production sur un serveur contenant moins de clients. Un autre environnement a été créé pour les clients qui n'étaient pas sur le même fuseau horaire que nos client Français. Ceci permet de gérer ces quelques clients au cas par cas afin de ne pas appliquer de traitements par lots en plein milieu de leur journée de travail.

2.2.4 La sécurité : CAS

e-Paprika fonctionne avec une architecture de type SSO (Single Sign On). Le backend n'a pas de notion d'authentification : le backend fait confiance à celui qui l'appelle. A l'inverse, le frontend doit s'assurer que l'utilisateur est authentifié via CAS avant de faire des requêtes au backend : le frontend est la porte d'entrée sécurisée du backend.

Cependant, c'est le backend qui fournit la logique d'authentification. C'est-à-dire que le backend ne demande aucune authentification à part le login, il est cependant capable de dire si un login correspond à un mot de passe.

Des plugins CAS ont été développés. Ils appellent cette logique pour authentifier un bibliothécaire ou un abonné. La raison principale de ce choix est la simplicité : la logique métier est accessible facilement depuis n'importe quelle application de confiance (même si cette application ne supporte pas CAS). En effet, le backend fournit des méthodes HTTP-JSON pour l'authentification, donc n'importe quelle application qui supporte HTTP-JSON peut interagir avec le backend. Cependant, CAS requiert HTTPS et donc partage la gestion de certificats SSL ainsi que la gestion des cookies.

Cela implique que le backend doit être dans une DMZ et que les accès au backend sont réservés aux adresses IP de confiance à savoir les adresses du serveur qui héberge le frontend et de celui qui héberge le portail public.

2.3 Synthèse

Ce second chapitre décrit le logiciel e-Paprika d'un point de vue fonctionnel et technique. Il permet de comprendre la finalité de e-Vision et les contraintes techniques qui guident le logiciel de statistiques.

Le prochain chapitre décrit la construction de l'entrepôt de données utilisé par e-Vision. Nous y abordons, dans un premier temps, les grands principes des systèmes décisionnels. Nous présentons ensuite les outils choisis et les différentes stratégies de constructions utilisées. Enfin nous présentons les améliorations possibles du produits qui pourraient être faites dans les prochains mois.

Chapitre 3

Construction de l'entrepôt de données

3.1 L'entrepôt de données

Ce chapitre décrit la solution adoptée pour extraire les données d'e-Paprika et les intégrer dans e-Vision. La solution retenue est une solution sous la forme d'un entrepôt de données (ou Datawarehouse). La définition la plus reconnue d'un entrepôt de données est celle de Bill Inmon (1990) : « Un entrepôt de données (data warehouse) est une collection de données thématiques, intégrées, non volatiles et historisées pour la prise de décisions. »

3.1.1 « Thématique »

Le terme thématique veut dire que la base de données est orientée vers les sujets et non plus vers les fonctions. Les thèmes abordés dans e-Vision sont donc la circulation des documents, les abonnés et le fond documentaire.

3.1.2 « Intégré »

La notion d'« intégré » implique que des informations qui proviennent de différentes sources peuvent être recoupées dans l'entrepôt. Il n'y a pas de problème majeur pour cela, car l'entrepôt de données d'e-Vision n'a pour le moment qu'une seule source. Il est cependant prévu d'y intégrer d'autres produits de Décalog dans un futur proche.

3.1.3 « Non volatiles et historisées »

« Non volatiles et historisées » signifie que les données présentes dans le système décisionnel doivent pouvoir rendre compte d'une évolution dans le temps. Il n'y a jamais de suppression de données et les mises à jour sont rares. Dans e-Vision, cette notion est respectée partiellement. En effet, prenons l'exemple de l'abonné.

Un abonné est un élément qui change régulièrement, son adresse change s'il déménage, un nom peut changer suite à un mariage. L'idéal dans ce cas de figure serait de rajouter une ligne pour chaque changement dans l'entrepôt. On présente la même problématique pour les exemplaires, leur état peut

changer (s'il est abîmé par exemple), où des informations complémentaires peuvent être saisies. Il a été fait le choix de mettre à jour ces informations sur les lignes des dimensions exemplaires. Si une information doit être gardée telle que l'état d'un exemplaire lors d'un prêt, alors cette donnée sera répercutée sur la ligne du prêt. Cela rajoute des colonnes dans une table, mais évite la gestion multiligne pour une seule et même donnée en fonction du temps. En effet, la solution serait de mettre une date de fin de validité pour chaque donnée. Cependant, cette technique demande beaucoup plus de ressources mémoire, processeur et disque.

Pour palier à ce manque d'historisation dans e-Vision, certaines archives sont générées tous les ans au 31 Décembre. De cette façon, on garde un état des lieux du fond par an. Cette méthode n'est pas parfaite puisqu'elle est moins précise qu'un archivage complet et certains états de l'exemplaire sont perdus. Cependant, elle permet néanmoins de garder les informations à date et de suivre l'évolution du fond.

3.1.4 « Pour la prise de décision »

Les systèmes « décisionnels » offrent aux utilisateurs une vision transversale de toute leurs activités. Ils facilitent de fait la prise de décision et sont orientés vers le pilotages de l'activité.

3.1.5 Choix de cette solution

La définition de Bill Inmon représente bien les raisons pour lesquelles un entrepôt de données est pertinent pour un système statistique. Il ne perturbe pas le modèle source et répond parfaitement au besoin. À la vue des volumétries importantes d'e-paprika, cette solution de stockage s'est imposée d'elle-même, il n'y avait pas réellement d'alternative à l'entrepôt de données.

3.2 Architecture générale

3.2.1 Description générale

L'architecture générale d'e-Vision est présentée en figure 5. On observe, en vert sur ce schéma, les différents composants et leurs liens avec e-Paprika.

La partie qui nous intéresse ici est la partie ETL (Extract Transform Load) en marron sur le schéma. La solution ETL détient son propre serveur de traitement et son propre serveur de base de données (l'entrepôt). Pour le moment, les magasins de données et l'entrepôt sont stockés dans la même machine physique. C'est une solution logique dans la mesure où les magasins de données sont utilisés en journée tandis que l'entrepôt n'utilise des ressources que lors de sa construction. Les ressources sont donc correctement partagées.

Les transactions entre les machines se font via des requêtes SQL. Cette méthode permet un accès direct et rapide aux données. Il implique aussi que le serveur d'ETL doit avoir un accès en lecture extérieure à la base de données d'e-Paprika. Cette condition n'est pas handicapante d'un point de vue sécurité, car toutes les machines sont dans un même réseau local sur un cloud sécurisé.

La construction de l'entrepôt débute à 5 heures du matin de manière à laisser e-Paprika finir ses traitements de nuit. De cette manière, la solution décisionnelle a accès aux données de la veille. Une des contraintes du projet vient de cet horaire. En effet, les bibliothèques commencent à se connecter à e-Paprika vers 8 heures. Ce qui laisse une marge de 3 heures pour construire l'entrepôt. La construction des magasins de données est moins dépendante de cette contrainte étant donnée qu'il ne perturbe pas le fonctionnement de la base source. Ils sont construits à partir de l'entrepôt et uniquement à partir de celui-ci.

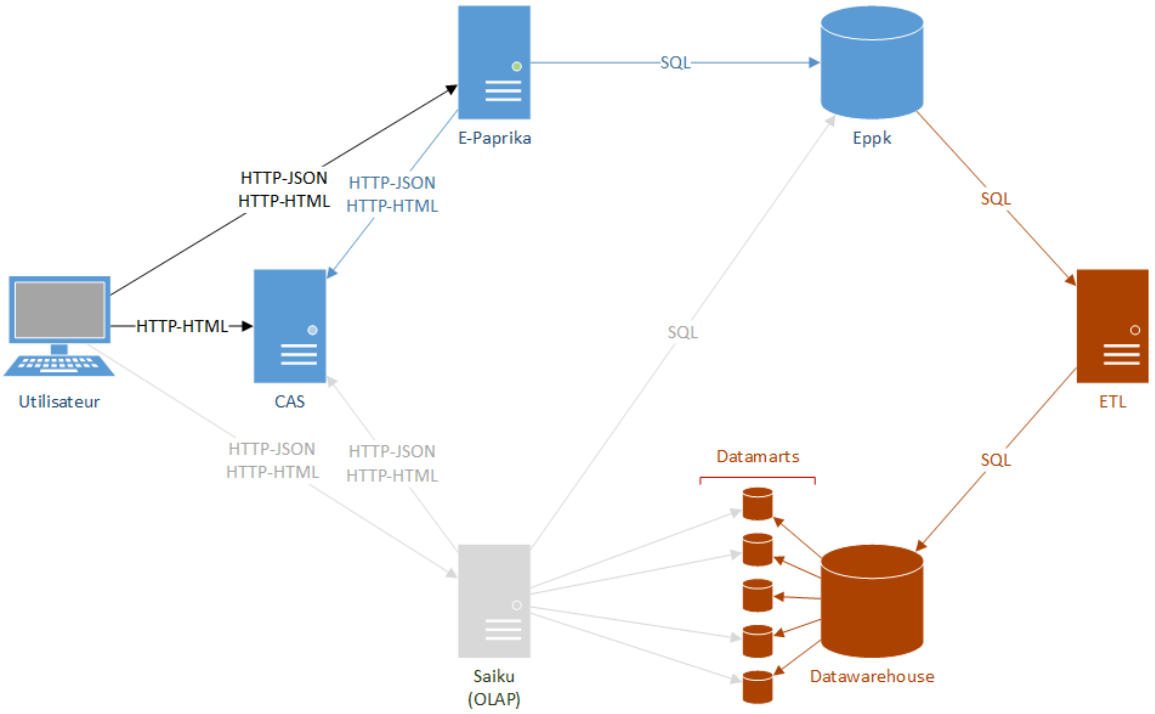


FIGURE 5 – Architecture générale d'e-Vision

3.2.2 Les magasins de données

Les magasins de données, ou datamarts, sont à opposer à l'entrepôt de données. L'entrepôt de données concentre toutes les informations pour tous les thèmes de l'entrepôt. Dans une structure décisionnelle classique, les magasins sont thématiques. Dans notre projet, on pourrait imaginer un magasin de données pour la circulation, un autre pour l'étude du fond ou encore un autre pour les abonnés.

Cependant, dans e-Vision, le choix de faire un datamart par thème est contre-productif. En effet, il semble plus logique de stocker toutes les données d'un même client sur une seule base. De cette manière, chaque réseau de bibliothèques détient son propre magasin de données et les volumétries restent raisonnables. Les plus gros réseaux contiennent au maximum 300 000 documents. Le nom de la base de données associée a un datamart sera toujours de la forme « dm_<identifiant unique du réseau> ». On prend pour l'identifiant unique du réseau, l'identifiant de la table *network* dans la base de données d'e-Paprika.

3.2.3 Les bases de données

Les bases de données utilisées pour l'entrepôt et les magasins de données sont pour le moment du MySql avec le moteur MyIsam. Etant une base opérationnelle e-Paprika nécessite une grande intégrité dans ses données d'où le choix de innoDb. La partie décisionnelle a quant à elle besoin d'avoir de très bonnes performances en lecture, surtout pour les magasin de données. Le moteur MyIsam a donc été choisi, les insertions et surtout les sélections étant plus rapides.

Ce choix de moteur s'accompagne naturellement des désavantages inhérents à celui-ci. Avec MyIsam, les clés étrangères n'étant pas vérifiées, il appartient à la solution d'ETL d'être particulièrement fiable et d'offrir des contrôles sur l'intégrité des données. Ces mécanismes de contrôle sont détaillés dans la description de construction des faits.

3.2.4 Différents environnements

Les environnements d'e-Vision se calquent sur ceux d'e-Paprika. On retrouve donc :

Un environnement de développement Cet environnement est particulier dans le sens où toutes les briques du logiciel sont sur le même serveur. De la même manière qu'e-Paprika, il est mis à jour deux fois par jour, une fois le matin et une fois à midi. Vu la faible volumétrie et le nombre d'utilisateurs, l'entrepôt peut facilement être construit à midi sans que cela dérange.

Un environnement de qualification Cet environnement reproduit les conditions des productions. On y retrouve donc un serveur propre à la solution ETL.

Un environnement de préproduction La préproduction e-paprika regroupe deux fois plus de données qu'une production moyenne. C'est donc une excellente plate-forme pour tester les performances de la construction de l'entrepôt. Il est à noter cependant que seule la partie ETL est présente sur cet environnement. La partie interface n'est pas installée.

Les environnements de productions Il a été fait le choix de ne pas créer un seul entrepôt de données qui regrouperait toutes les productions, mais d'avoir un serveur par production. Ce choix a été motivé par plusieurs critères. Le premier est un souci de performance, il est plus difficile de faire rentrer la construction d'un entrepôt en trois heures s'il doit regrouper toutes les productions. La seconde est de garder une cohérence en ce qui concerne l'exploitation. En effet, les mises en production des versions ne se font pas sur toutes les plates-formes au même moment. On peut donc avoir jusqu'à deux versions différentes simultanément, le fait d'avoir un entrepôt par production permet de garder un lien entre les versions du datawarehouse et des productions sans générer de développements supplémentaires.

3.3 Base de données multidimensionnelles

Le rôle de l'ETL est donc de transformer le modèle relationnel d'e-Paprika en modèle décisionnel. Les données sont stockées dans l'entrepôt de données dans un modèle multidimensionnel. Pour

comprendre ce que fait réellement Talend, il est utile d'avoir quelques notions théoriques sur le modèle en étoile (ou plus précisément en constellation d'étoiles dans le cas d'e-Vision) Dans le modèle multidimensionnel, on parle de « cube » et de « dimension ».

3.3.1 Les cubes

Les faits

Un cube relate un fait. Un fait est une activité propre au domaine étudié. Dans e-Vision, les faits d'activités recensés sont les suivants :

Inscription Ce fait permet de faire des statistiques sur les différentes cotisations et inscriptions d'un réseau.

Prêt retour Ce cube donne les principales informations sur la circulation. Il permet de savoir comment le fond est utilisé, quels abonnés empruntent quoi, quels sont les livres les plus empruntés, etc.

Prêt et retour fonctionnel Ce cube a un aspect plus informatif. Les abonnés fonctionnels sont une notion d'e-Paprika. Ils permettent en effet de faire des manipulations dans le logiciel, ce sont des abonnés fictifs. On peut prendre pour exemple le prêt inter-bibliothèques. Une première bibliothèque veut prêter un exemplaire à une seconde au sein du même réseau. Dans la pratique, la première bibliothèque va prêter à un abonné fonctionnel et la seconde fera le retour au moment de la réception de l'exemplaire.

Réservation Comme son nom l'indique, ce cube permet de connaître l'évolution des réservations d'une bibliothèque. Il permet entre autres de voir la disponibilité des différents exemplaires via les durées de réservations par exemple.

Visite librairie - Visite réseau Ces faits référencent les dates et heures auxquelles les utilisateurs font des actions en interaction avec le logiciel. Ces compteurs d'actions sont agrégés soit par réseau, soit par bibliothèque.

Pour autant, ce ne sont pas les seuls cubes qui sont proposés dans e-Vision. On y trouve aussi les cubes suivants dans le modèle de données :

Abonnés : Ce cube présente un état des lieux à un instant "t" des informations des abonnés des bibliothèques. Ce cube à la particularité de ne pas s'appuyer sur une date de dernière mise à jour lors de sa construction. On passe toutes les lignes relatives aux abonnés tous les jours.

Exemplaire Le cube des exemplaires est le plus volumineux, c'est un état du fond à un instant « t ». Il est cependant construit sur un différentiel quotidien.

On ne peut pas exactement dire que ces cubes sont des faits au sens activité du terme. En effet, ni l'exemplaire, ni l'abonné ne représente une action datée. Se sont des états à date, il n'y a pas de notion d'historisation. Ce sont d'ailleurs les deux faits qui sont archivés en fin d'année. Ils sont là pour pouvoir fournir des statistiques sur l'état du fond ou proposer des listes.

Le taux de rotation et le taux d'activité sont deux cubes qui donnent un ratio. Le taux de rotation donne le ratio entre le nombre de prêts sur une période et le nombre d'exemplaires existant sur cette période donnée. De la même manière, le taux d'activité donne le ratio sur une période entre le nombre de prêts et le nombre d'abonnés. Ces deux cubes ont la particularité de ne pas nécessiter de tables spécifiques. Il n'existe en effet pas de table de fait pour ces cubes, ils sont l'association de deux autres cubes. Nous revenons dessus dans la section Mondrian à la section 4.4.4.

3.3.2 Les mesures

Un fait est quantifié par des mesures. Les mesures peuvent être de simples compteurs, des sommes ou des formules. Les mesures sont donc des fonctions d'agrégation sur des champs de la base. Le tableau donne un aperçu des différentes mesures du cube de réservations. On peut voir que chaque champ donne lieu à plusieurs mesures selon la fonction qui lui est appliquée.

TABLE 2 – Liste des mesures du cube de réservation avec les fonctions associées

Mesure	Colonne	Fonction
Nombre de réservations	nbBooking	sum
Nombre de réservations (distinctes)	_idBookingSIGB	distinct-count
Nombre de réservataires (distincts)	idPublicuser_FK	distinct-count
Nombre de documents réservés (distincts)	idBibRecord_FK	distinct-count
Nombre de prolongations	nbExtension	sum
Maximum de prolongations	nbExtension	max
Minimum de prolongations	nbExtension	min
Moyenne de prolongations	nbExtension	avg
Délai moyen de satisfaction	realDuration	avg
Délai moyen d'attente	waitingDuration	avg
Délai moyen de retrait	attributionDuration	avg
Délai moyen de retrait prévu	expectedDuration	avg
Délai moyen d'envoi	sendingDuration	avg

On a une ligne par « unité » d'activité. Chaque ligne pouvant se comporter différemment selon la mesure demandée. Cependant, avec uniquement des mesures, il n'est pas possible de faire des croisements. Pour cela, nous avons besoin des dimensions.

3.3.3 Les dimensions

Les dimensions sont les axes d'analyse dans un cube. Ce sont les informations que l'on veut recouper. Chaque dimension est organisée sous forme de hiérarchie. Toujours dans le cube des réservations, voici un extrait des différentes dimensions utilisées. Nous avons décidé de ne pas fiare de liste exhaustive dans ce mémoire, car elle ne serait pas pertinente étant donnée que pour le seul cube des réservations, il y en a 130.

La première dimension que l'on présente est de la forme la plus simple. Dans le type de l'abonné (groupe ou personne), on a deux niveaux de hiérarchie.

1. Type de l'abonné
   (a) Tout
   (b) Type de l'abonné

Le type de l'abonné contient deux "membres" dans e-vision. Un abonné est en effet soit un groupe, soit une personne. Le « Tout » est une notion du système décisionnel qu'il faut interpréter par « pour tout type d'abonné ». Dans le cas des abonnées, le tout peut être assimilé à un total. En effet, un abonné est soit un groupe, soit une personne. Il ne peut pas y avoir un réservataire qui soit à la fois un groupe et un abonné.

Si on prend la dimension « Nouveauté ? ». Cette dimension contient trois membres, « oui », « non » et « non précisé ». Elle permet de savoir si une réservation porte sur une nouveauté du fond.

TABLE 3 – Illustration de la notion « Tout »

-Tout-	Nouveauté ?	Nombre de réservataires (distinct)
Tout		187
	-Non précisé-	56
	Non	162
	Oui	3

Dans ce tableau, on observe le nombre de réservataires distincts pour la dimension « Nouveauté ? ». Il y a donc au total 187 réservataires distincts qui ont réservé. Cependant, le détail donne 56 + 162 + 3 = 221. L'explication est que l'on cherche les réservataires distincts. Cependant, certains abonnés ont réservé des livres qui en nouveauté ainsi que d'autres livres qui ne le sont pas. Ils apparaissent donc dans chacune des lignes, mais il n'apparaissent qu'une seule fois dans le « Tout ». Cette notion est très importante dans le sens où le total n'a pas vraiment de sens dans ce contexte. C'est un point que beaucoup d'utilisateurs du logiciel ont du mal à appréhender.

Les deux dimensions précédentes étaient très simples, la dimension « Temps » est beaucoup plus intéressante. Il est en effet possible de la découper en de multiples hiérarchies. Les listes suivantes donnent deux des hiérarchies possibles pour la dimension temps. Elles permettent des tableaux radicalement différents.

- Date d'attribution-Outils
  - Tout
  - Année d'attribution
  - Semestre d'attribution
  - Trimestre d'attribution
  - Mois d'attribution
  - Jour du mois d'attribution
- Date d'attribution-Outils2
  - Tout
  - Année d'attribution (2)
  - Semaine d'attribution (2)
  - Jour de la semaine d'attribution (2)

Ainsi, si l'on se positionne sur le niveau du jour pour chacune de ces hiérarchies, on obtient dans le premier cas la répartition des réservations sur un mois complet. Tandis que dans le second cas, c'est la répartition des réservations par jour de la semaine quel que soit le mois qui ressort.

3.3.4 Implémentations

L'implémentation d'un modèle multidimensionnel (ou OLAP) peut se faire sous trois formats.

MOLAP est une structure en matrice. Toutes les agrégations possibles sont stockées et les différentes mesures sur ces agrégations pré-calculées. C'est un modèle qui permet une très bonne performance en sélection, cependant il est très difficile à maintenir.

ROLAP est basé sur un stockage relationnel tel que MySQL ou Oracle. Il présente un modèle en étoile et c'est la solution que nous avons adoptée.

HOLAP est basé sur le meilleur des deux précédentes solutions. Cette solution pourrait être envisagée pour des problèmes de performances à l'avenir, mais elle n'a pour le moment pas été testée.

On peut observer sur la figure 6 les tables du cube des prêts. Pour des raisons évidentes de lisibilité, l'ensemble des colonnes n'apparaît pas sur cette figure.

La table de fait

On observe au centre du modèle la table de fait des prêts. Elle dispose d'un identifiant propre à l'entrepôt. Elle contient toutes les références vers les dimensions (en bleu sur le schéma). C'est aussi elle qui contient les champs utiles pour les mesures. On peut prendre par exemple *realDuration* et *expectedDuration* qui représentent la durée réelle et la durée prévue du prêt. Ce sont des nombres sur lesquels on pourra facilement faire des fonctions d'agrégations de type somme ou moyenne.

Les dimensions en étoile

Les tables de dimensions sont les critères sur lesquels on va baser les tableaux. Leur accès en lecture est simplifié du fait qu'elles ont toutes au moins une référence directe dans la table de fait. On n'y accède donc avec une seule jointure dans 100% des requêtes. On peut cependant trouver plusieurs jointures directes vers une même table de dimensions. Par exemple, la table de dimensions temps (*dim_temps*) est la table qui est la plus utilisée en multi jointure. On retrouve une jointure vers cette table avec la date de début de prêt, mais aussi la date de fin prévue ou encore la date de fin réelle.

Les dimensions en flocon

Les dimensions en flocons sont un contournement du modèle en étoile. Dans le cube des prêts de la figure 6, on observe ces tables en orange. Se sont des tables qui dérogent à la règle de n'avoir qu'une jointure pour faire le lien entre la table de fait et les tables de dimensions. On passe en effet par deux jointures pour y avoir accès, la première pour aller sur une dimension mère et une seconde pour la dimension finale.

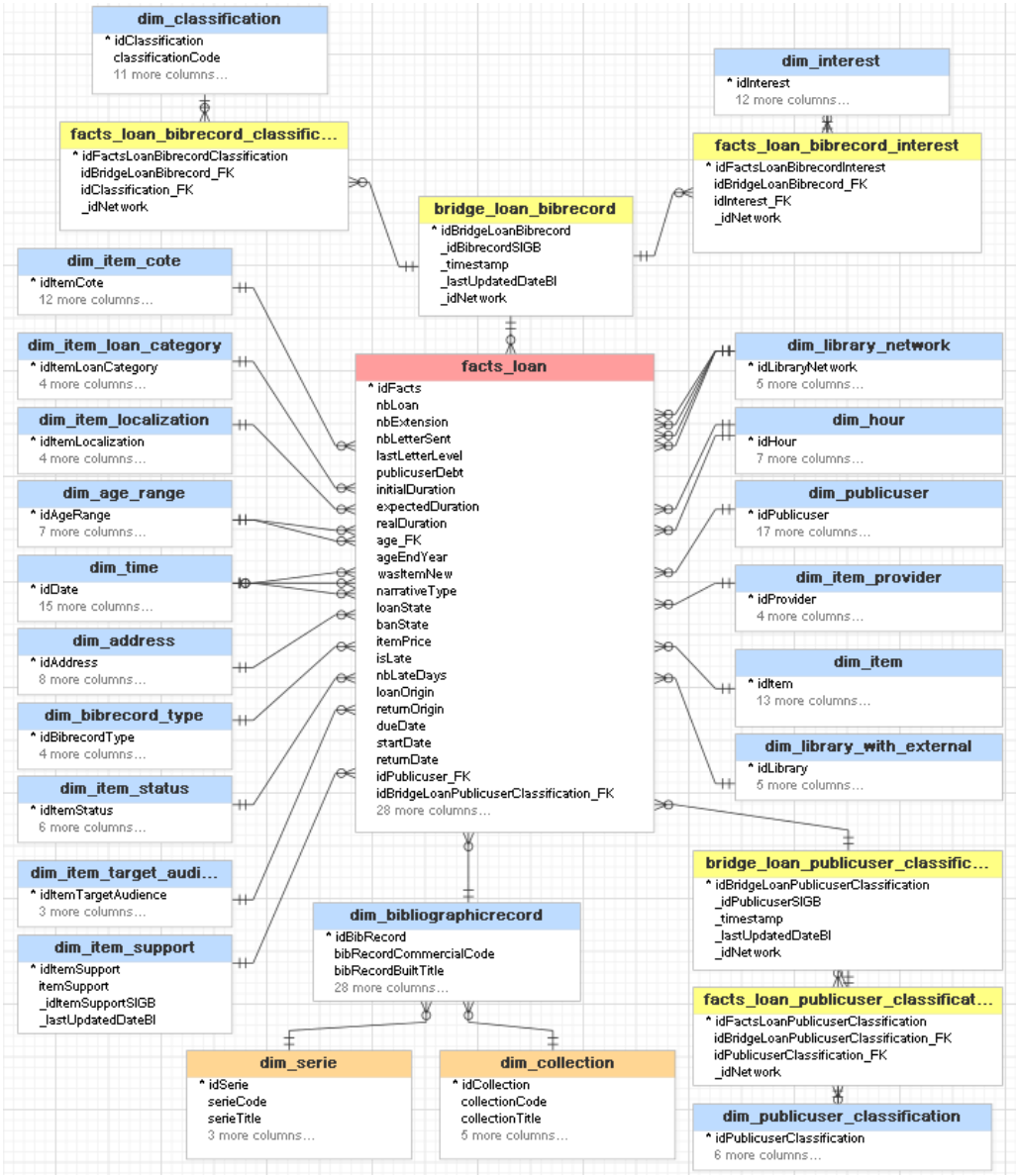


FIGURE 6 – Organisation en étoile des tables du fait des prêts

Ces tables posent le problème de ne pas être optimisées pour une sélection rapide et efficace. Leur existence se justifie cependant pour des raisons de maintenance. En effet, lorsque l'on rajoute une série sur une notice bibliographique, on veut que tous les prêts qui font référence à cette notice soient impactés. C'est une notion rétroactive or il peut être très coûteux de reprendre les prêts d'une série si celle-ci a été très prêtée. Cette justification est historique, et une étude pour faire revenir ces tables dans le modèle en étoile est en cours.

Les relations n-n

Dans le cas ou une ligne de fait peut avoir plusieurs associées dans une table de dimension, le modèle en étoile classique atteint sa limite. Il faut en effet créer deux tables intermédiaires pour gérer ce cas. On retrouve ces tables en jaune sur le schéma. Nous explicitons cette notion avec l'exemple de la dimension *dim_publicuser_classification*

La table de jointure sur le schéma 6, elle se nomme *bridge_loan_public_user_classification*. Cette table de jointure permet de donner un identifiant qui va pointer vers un ensemble de valeur. Il a été choisi dans e-Paprika de donner un identifiant de liaison (« bridge ») pour chaque utilisateur (*publicUser* dans la base de données). On a donc dans cette table la liaison entre un identifiant de liaison (*idBridgeLoanPublicUserClassification*) et un identifiant d'utilisateur (*_idPublicUserSIGB*). Les autres champs n'ont pas d'intérêt dans le mécanisme de jointure de la relation en elle-même.

La table de fait sans fait sur le schéma 6, elle se nomme *facts_loan_public_user_classification*. Cette table fait la relation entre l'identifiant de la table de dimension (*idPublicClassification_FK*) et l'identifiant de liaison (bridge) (*idBridgeLoanPublicUserClassification_FK*). On trouvera en plus un identifiant unique de relation. C'est véritablement dans cette table que ce fait la relation, on aura ainsi potentiellement plusieurs valeurs de *public_user_classification* pour un seul bridge, c'est la traduction de la relation *many-to-many* dans un format décisionnel.

Lors de la sélection, dans l'outil de rendu des tableaux par exemple, on se sert pas de la table de liaison. On fait directement le lien entre la table de fait et la table de fait-sans-fait grâce à l'identifiant de liaison. De cette manière, nous gagnons une jointure et la requête est plus rapide. Une fois cette relation faite, on se retrouve avec une liste d'identifiants qui permet d'avoir l'ensemble des valeurs de la dimension finale. Lors de cette explication, on fait le raisonnement pas à pas, il est évident que l'on fait cette opération en une seule requête.

Points communs dans la construction des tables

Il y a quatre constantes dans toutes les tables de dimensions ou de fait.

1. Elles ont une clé primaire propre à l'entrepôt et c'est un entier autoincrémenté. Bien que les clé primaire d'e-Paprika soient des UUID (Universal Unique Identifier), on ne les récupère pas dans e-Vision. Cette règle a pour objectif de garder à l'esprit que l'entrepôt a son propre référencement et que l'on pourrait avoir d'autres sources qu'e-Paprika à l'avenir.
2. La clé primaire du modèle source est stockée directement dans la table de dimension. Cette colonne va permettre de faire le lien entre l'entrepôt et les données du modèle sources lors de la construction des dimensions. Ceci pourrait éventuellement être complété par un id source si les sources venaient à se diversifier.
3. il existe un champ *_lastUpdatedDateBi* qui permet de garder la date exacte de dernière modification. Pour les dimensions, cette date n'est pas utilisée au quotidien mais facilite les opérations de maintenances
4. Si la donnée relatée dans la table dépend du réseau, alors il y a une colonne *_idNetwork*. Ceci permet de construire les magasins de données et de procéder à la copie de réseaux.

Schéma en constellation

Tous les cubes de e-Vision sont organisés de cette manière. Cependant, certaines dimensions comme la dimension temps, la dimension exemplaire ou encore la dimension abonnés sont utilisées dans plusieurs cubes. Le fait qu'il y ait des dimensions partagées par plusieurs cubes implique que l'on appelle le modèle d'e-Vision, un modèle en constellation.

3.4 Présentation de l'ETL

3.4.1 Talend

Une solution ETL (Extract Transform Load) permet d'extraire les données d'une source de données et de les adapter pour les charger dans une autre. Ici, l'ETL permet de récupérer les données d'e-Paprika, de les adapter et de les charger dans le modèle en étoile. La solution retenue pour l'ETL est Talend dans sa version TOS (Talend Open Studio) qui est une la suite logicielle open source qui permet l'intégration de données.

3.4.2 Choix du produit

Nous avons choisi Talend car c'est la solution open-source (et donc gratuite) la plus évoluée, la plus conviviale, la plus simple, la plus rapide et surtout la plus indépendante. Les alternatives étaient :

1. Pentaho Data Integration (anciennement Kettle) qui était une solution trop couplée avec pentaho
2. Jaspersoft Data Integration (repackaging Talend) qui était une solution trop couplée avec japser

3.4.3 Interfaces et principes

Talend a été développé sur la base d'éclipse. On retrouve donc beaucoup de similitudes avec cet IDE.

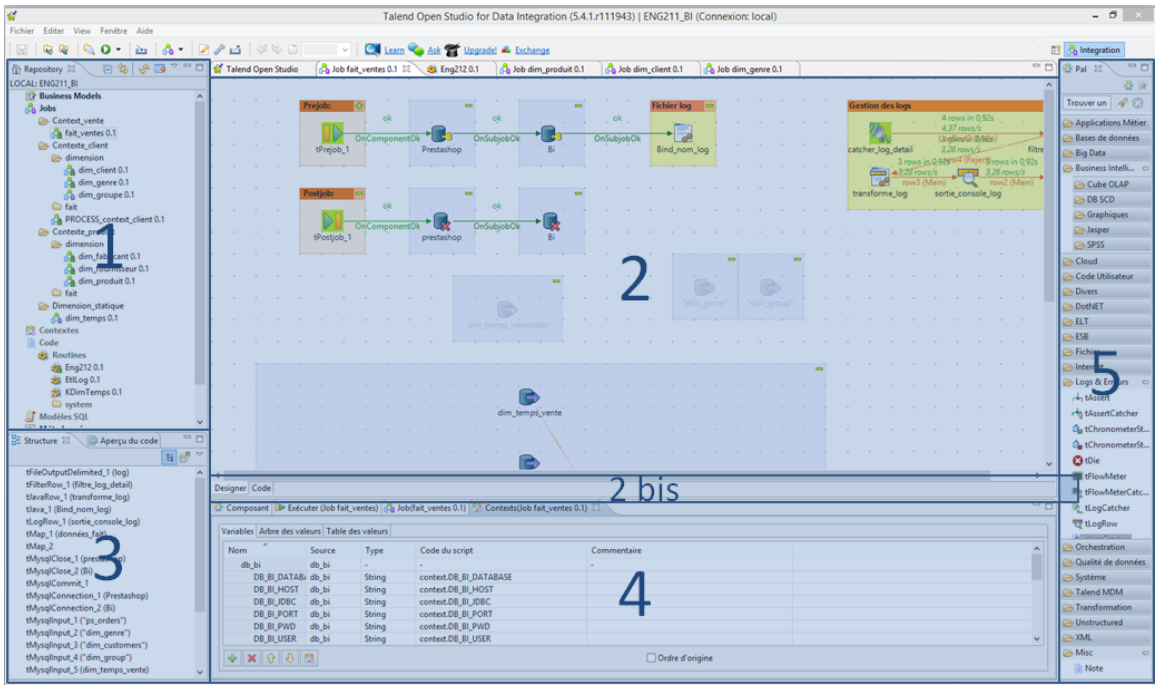


FIGURE 7 – Découpage de l'interface par défaut de Talend

On voit ici dessinées 5 sous-zones.

La zone 1 est la zone qui définit l'arborescence des différents éléments principaux. C'est la zone de « repository », nous y retrouvons les **Jobs**, les **Contextes**, les **Routines** ainsi que les **Métadonnées**. Nous revenons sur chacune de ses notions dans la section suivante.

La zone 2 est la zone de travail principale. Elle contient tous les composants d'un Job et les relations qui permettent de construire les flux de données.

La zone 2 bis permet de basculer la zone de travail. En effet, Talend se fonde sur un outil graphique pour organiser ses flux. Cependant, en cas d'erreur ou si l'on veut comprendre le fonctionnement en interne, on peut avoir accès à la vue "code". Cette vue propose le code java généré. Les erreurs de compilation, comme une égalité entre deux variables de différents types (erreur la plus courante) est mise en évidence par une puce rouge dans la barre de défilement. Cependant, le code généré n'est pas facile à lire et un job est écrit sur une seule page. Cela donne un code pouvant facilement atteindre des dizaines de milliers de lignes.

La zone 3 fait la relation entre les noms de la vue et le nom des composants. Cette vue trouve tout son intérêt lorsque l'on doit faire des investigations sur les fichiers de log.

La zone 4 concentre tous les onglets que l'on utilise lors du développement.

- La vue composant qui permet de gérer les composants individuellement.
- La vue exécuter permet de lancer le job en fonction des variables de contextes que l'on a prédéfini. Il existe aussi un mode de débuggage pas à pas qui lit le flux principal ligne par ligne et affiche le contenu au fur et à mesure.
- La vue contextes permet de rajouter des variables de contexte et de voir leur valeur. La section 3.4.4 est dédiée au contexte dans Talend.

La zone 5 est la palette, c'est dans cette zone que l'on peut sélectionner les différents composants que l'on veut utiliser. Les composants sont organisés en catégories, application métier, base de données, big data, fichier etc...

3.4.4 Les contextes

Description

Les contextes font partie des éléments que l'on trouve dans la liste des éléments principaux de la zone 1 dans le paragraphe précédent. Les contextes servent à définir des variables qui peuvent potentiellement être utilisées dans tous les Jobs. On peut voir cela comme des variables globales pour Talend.

La figure 9 donne la vue qui permet de définir le nom et le type des variables. Les noms et les types de variables sont les mêmes que ceux existants en Java. Dans l'exemple ci-dessous, on a un contexte qui permet de définir les variables de connexion aux bases de donnnées.

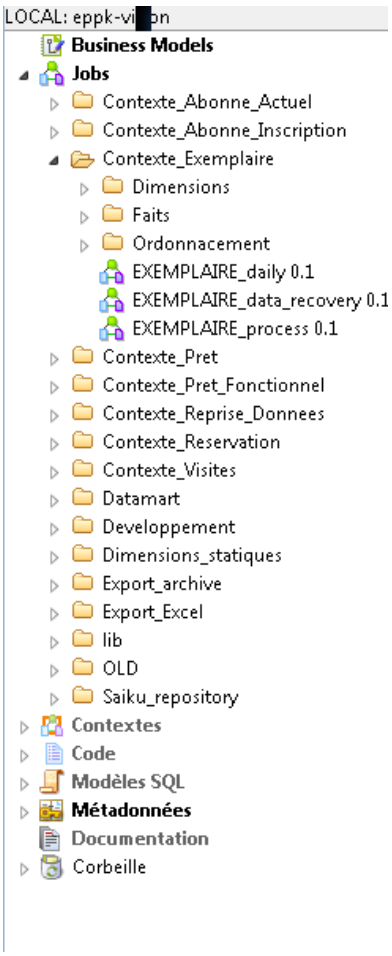


FIGURE 8 – Zoom sur l'arborescence des principaux éléments

On peut faire autant de contextes que l'on souhaite. Dans un premier temps, chaque environnement avait son propre contexte. Nous sommes revenus sur ce choix car l'exploitation était beaucoup trop liée au développement. En effet, si l'on voulait monter un nouvel environnement, il fallait modifier les sources dans Talend.

Sur la capture 9 on peut cependant voir qu'il y a deux contextes. Même si le contexte Beta n'est plus utilisé actuellement, on peut voir l'utilité d'un tel mécanisme. La gestion des contextes est précisée dans la section 3.5.3.

3.4.5 Les composants

Généralité

Les composants sont les briques de Talend. On retrouve beaucoup de composants d'entrée, de sortie ou de transformation dans Talend. Par exemple, en entrée, certains composants permettent de lire des fichiers ou de récupérer le résultat de requêtes SQL en base de données. De la même manière, on dispose de composants de sorties, des outils pour les écritures sur fichiers ou pour les insertions dans des tables d'une base de données. Nous ne faisons pas dans ce rapport de présentations de tous

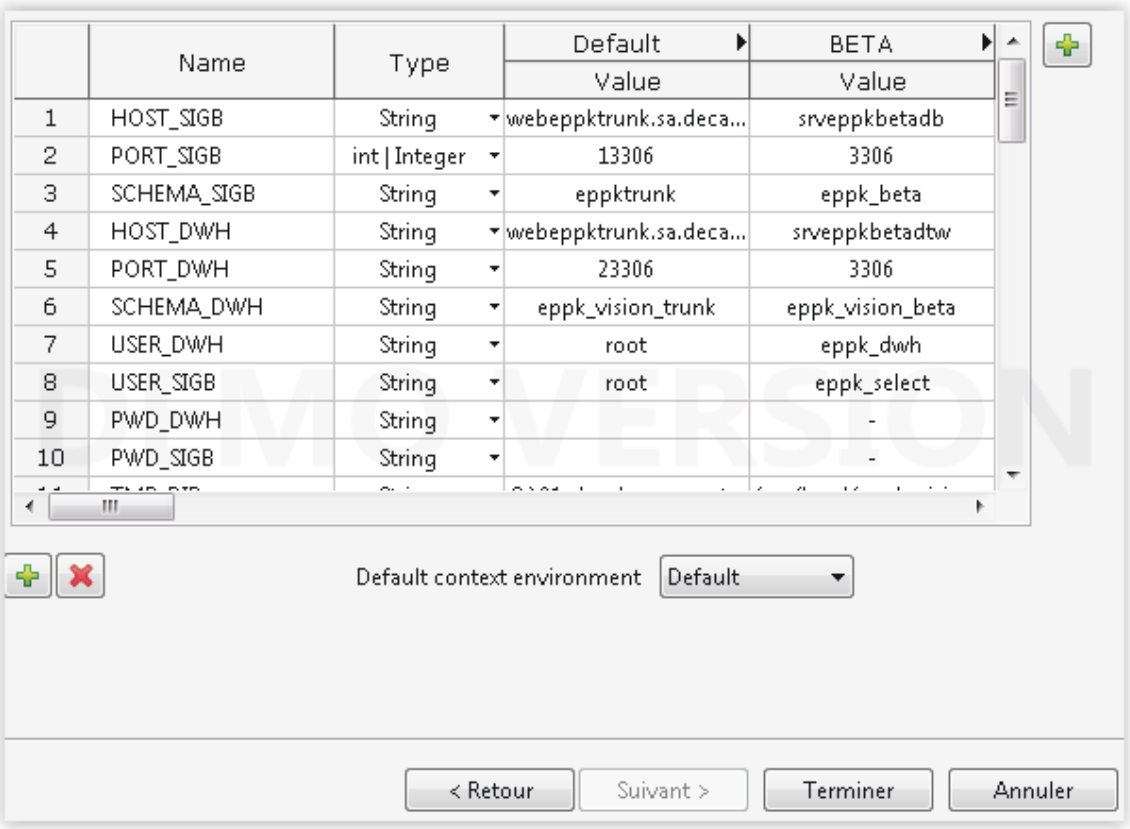


FIGURE 9 – Écran de définition des variables de contextes

les composants utilisés. Il faut cependant connaître quelques notions à propos des composants pour comprendre le fonctionnement de Talend et les différentes stratégies de constructions abordées.

Les noms

Un composant a deux noms, un pour l'interface et un autre technique. Cela peut sembler un détail, mais c'est très déconcertant au début. En effet, lorsque l'on regarde la sortie console ou les logs seuls les noms techniques tels que *tJava_1* ou *tMysql_1* ressortent. Alors que sur la vue, nous avons mis un *dim_temps* ou un *bind_nom_log* pour le nom du composant. Il faut savoir que la vue n'est qu'un fichier xml qui permet d'avoir une interface compréhensible et manipulable par le développeur. Le code généré ne reprend pas toutes les nominations présentes dans la vue.

Gestion des erreurs

Il est intéressant de noter que les composants qui produisent une erreur n'arrêtent pas nécessairement le flux (sauf erreur fatale Java de type Null Pointer Exception). Il faut pour cela cocher une case qui spécifie l'arrêt du job. Ainsi, si un fichier n'est pas trouvé, cela peut ne pas influencer la continuité du Job.

Les connecteurs

Les connecteurs sont les composants d'entrées et de sorties dans Talend. On utilise principalement deux sortes de connecteurs dans e-Vision. Les connecteurs aux bases de données qui permettent d'extraire les données des tables et les connecteurs de fichier plat.

Exemple de composant représentatif : le *tMap*

Le composant tMap est une des composantes essentielles de Talend. En effet, il permet de croiser différentes sources, de transformer certaines données. C'est un composant très souple mais très complexe et très représentatif des autres composants. Il existe des composants pour supprimer des colonnes d'un schéma, pour faire une jointure entre deux tables. Ces composants sont utiles parce que facile à maintenir et ils annoncent directement leur affectation, il est alors facile de lire la carte des composants. Cependant, toutes ces tâches peuvent aussi être exécutées avec un tmap. Nous allons détailler ce composant dans la mesure où il concentre beaucoup de notions utiles. En effet, du fait qu'il peut remplir un nombre important de tâches, il donne une très bonne idée de la stratégie de Talend.

Ce composant accepte plusieurs composants en entrée et renvoie potentiellement plusieurs sorties. Il faut au moins un composant en entrée. Le premier composant en entrée est appelé le composant principal ou *main*. C'est sur cette entrée que Talend se base pour calculer les différents autres champs et faire ses jointures. Le *tMap* est configuré via un écran spécifique. La figure 10 représente cette vue.

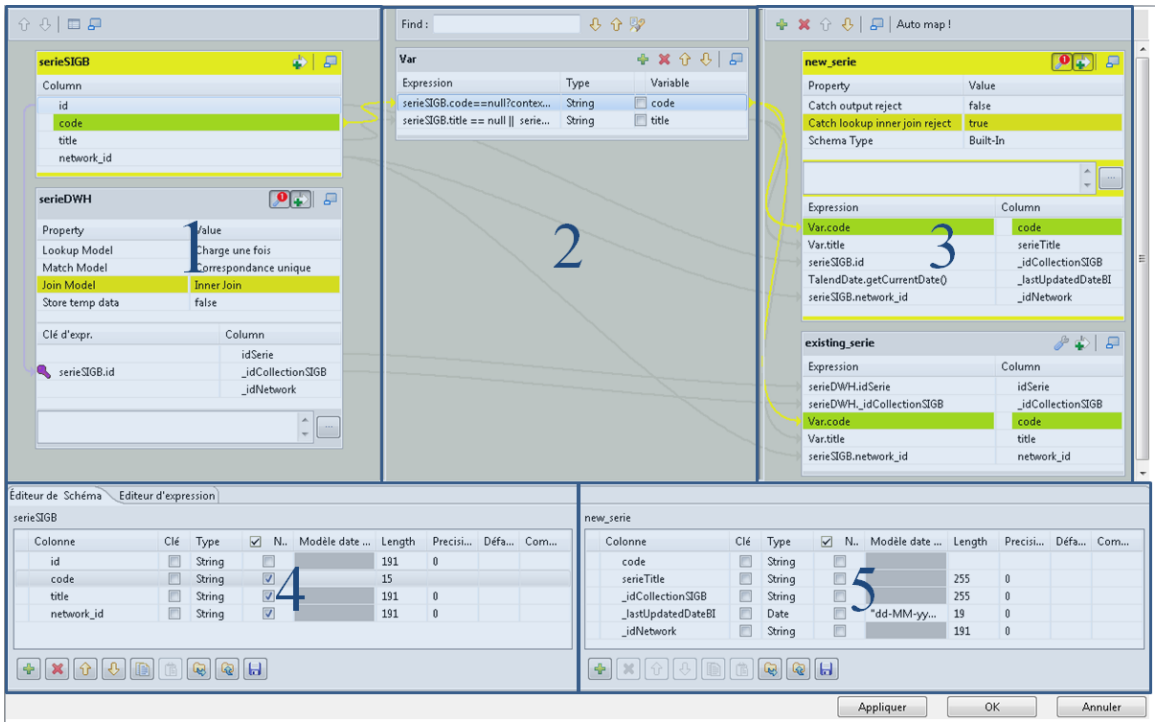


FIGURE 10 – Ecran de configuration globale du tMap

Nous présentons ici les différentes zones et leurs rôles.

- **La zone 1** contient toutes les entrées du composant, on peut y voir les schémas de données internes à Talend. On a une entrée principale en haut de la zone. C'est dans cette zone que l'on peut faire des jointures. L'exemple utilisé ici présente une jointure interne.
- **La zone 2** contient une table de transition. Elle permet de transformer certaines données avant de les envoyer vers la sortie. On applique des fonctions Java à la valeur des champs, ceci permet notamment grâce à la forme ternaire de java d'appliquer des conditions aux champs.
- **La zone 3** contient toutes les sorties. A ce stade, ce sont juste des sorties Talend, on peut très bien les injecter par la suite dans des tables mysql, des fichiers plats ou d'autres composants tels que les *tMap*. Ici, on exploite la jointure interne de deux manières, la première sortie prend en compte toutes les lignes qui satisfont les conditions de jointure. La seconde prend exactement l'inverse, c'est-à-dire tous les rejets de la jointure. Les jointures sont faites en mémoire via des hashMap Java, ce n'est pas un fonctionnement de type MySQL, on a donc la possibilité de reprendre les rejets.
- **La zone 4** permet de configurer les schémas d'entrée.
- **La zone 5** permet de configurer les schémas de sortie.

Ces différentes zones représentent parfaitement les composants Talend. Des zones d'entrée, des zones de sortie, une transformation possible et des interactions entre les divers éléments. C'est une configuration qui est très représentative. De plus, ce composant est très utilisé dans e-Vision, il convenait donc de le décrire pour pouvoir comprendre les différents processus.

3.4.6 Les liens entre les composants

Il existe plusieurs types de liens entre les composants. La figure 11 montre l'organisation générale des différents composants de la construction de la *dimension client*.

Nous nous intéressons ici aux flèches qui relient les différents ensembles. On observe ainsi trois type de flux, *onComponentOK*, *OnSubjobOk* et les flèches orange qui sont des flux. Il existe cependant une autre liaison qui existe mais que nous n'utilisons pas ici, l'*iterate*.

OnComponant

Cette liaison est une liaison que l'on peut qualifier d'ordonnancement. Il existe deux variantes, *OnComponantOK* et *OnComponantError*. Cette liaison attend que le composant source se termine et s'exécute immédiatement après le composant de destination. La différence se fait sur le retour du composant ; s'il est en succès, c'est le flux « OK » qui est exécuté sinon, on ressort sur le flux d'erreur. On peut comme dans le cas qui nous concerne n'avoir que le flux en succès et ne pas connecter le flux d'erreurs.

OnSubjob

Cette liaison est aussi un lien d'ordonnancement. Pour la comprendre, il faut comprendre la notion de sous job. Les zones bleues claires sont en effet des sous-parties du job principal. La flèche du *OnSubjobOK* ne se contente pas d'attendre la fin du composant *SIGB_Support*, il attend que tous les composants qui sont encadrés dans le carré bleu soient terminés en succès. Il existe aussi deux

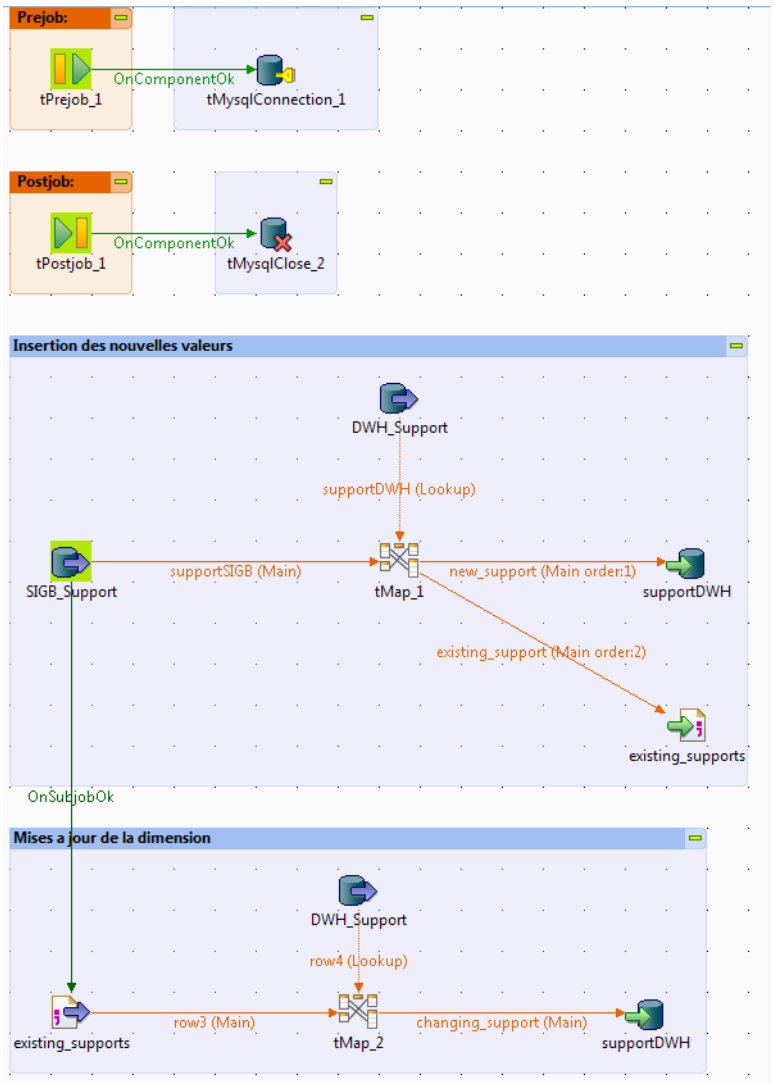


FIGURE 11 – Les principaux types de flux

variantes, le *OK* et le *Error*. Le premier est appelé lors du déroulement normal du job, le second en cas d'erreur de celui-ci. Cela permet de tracer les erreurs spécifiques. Dans e-Vision, le déclenchement d'une procédure sur une erreur est anecdotique. En effet, les composants en erreur arrêtent l'exécution du flux. On retrouve dans les fichiers de logs l'emplacement des erreurs.

Flux et schéma

Les flux ne servent pas d'ordonnancement, ils véhiculent les données. De ce fait, ils comportent un schéma. Le schéma est l'une des composantes les plus importantes de Talend.

Les schémas définissent le format des données qui circulent. Chaque ligne présente dans un schéma donné est associée à une variables Java dans le code généré par Talend. On y retrouve

- Le nom du champ qui doit être unique au sein d'un même schéma.
- Le type, c'est un type Java qui est fourni. On peut ainsi définir des entiers (*Integer*), des chaînes de caractères (*String*).

- Des informations complémentaires selon le type. On aura ainsi des formats de date, des tailles de chaînes de caractères ou des précisions pour des nombres.
- Dans certains composants de type base de données le lien entre la variable de Talend et la base de donnée se fait aussi dans le schéma.

Certains composants ont un schéma d'entrée et un schéma de sortie. D'autres peuvent avoir soit l'un soit l'autre. Mais dans tous les cas si deux composants communiquent entre eux ils doivent avoir impérativement le même schéma. C'est l'une des sources principales de problèmes lors du développement.

On peut observer sur la figure 12 le schéma de la dimension des statuts d'exemplaires.

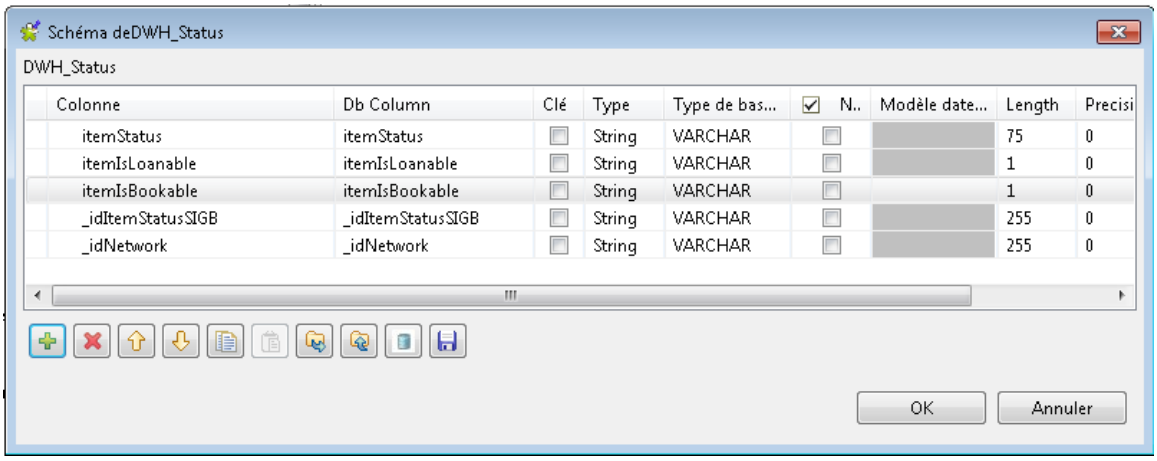


FIGURE 12 – Exemple de schéma d'un composant tMysqlOutput

Iterate

Le flux Iterate est un intermédiaire entre le flux de données et le flux d'ordonnancement. Il lit le flux avec des entrées et fait une itération dessus. Par exemple, on peut parcourir la liste des tables d'une base MySQL (grâce au composant *tMysqlTableList*) et pour chacune des lignes retournées exécuter un composant (un *tJavaRow* par exemple), ou un job.

3.4.7 Les jobs

Les jobs sont les composants d'entrées dans Talend. C'est dans les Jobs que l'on organise les flux et placer les composants.

3.4.8 Job en tant que composants

Dans cet exemple, les Jobs sont des composants comme les autres. Dans Talend, les composants d'entrées ont toujours un fond vert. Ce sont les Jobs qui n'ont pas de dépendance.

On observe des flèches entre les jobs. Ces flèches servent à ordonnancer les différents Jobs ainsi, les Jobs s'exécutent les uns après les autres séquentiellement. D'ordinaire, les flèches d'ordonnancement comme celles-ci sont plutôt orientées verticalement mais pour des questions de présentation dans le rapport, la capture a été prise avec les éléments disposés horizontalement.

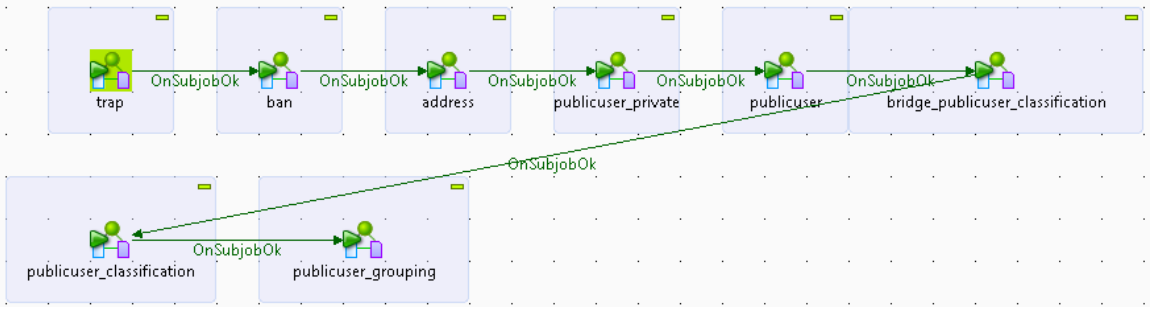


FIGURE 13 – Ordonnancement des dimensions du flux des abonnés

3.4.9 Exports des Jobs

Étant donné que ce sont les Jobs qui sont les points d'entrée de Talend, ce sont aussi eux qui sont exportés pour être exécutés sur un environnement autre que celui du développement. Il existe plusieurs types d'exportations selon les environnements de destinations. On a ainsi des exportations Java sous forme de web service ou des exportations autonomes.

L'"Autonomous Job" est une exportation qui ne nécessite pas d'application tierce telle qu'un serveur tomcat pour fonctionner. Cela nécessite uniquement une jvm sur la machine. C'est une archive zip qui contient tous les jobs et un script *sh* (pour linux) ou *bat* (pour windows) lancera les archives *jar* de manière autonome. C'est cette solution qui a été adoptée pour le projet e-Vision. Elle a l'avantage de nécessiter peu de configuration sur les serveurs de destination. Cependant, il faut avoir une méthode pour adapter les scripts de lancement et pour donner les paramètres d'entrée spécifique à chaque environnement.

3.4.10 Routines

Les routines permettent de créer des méthodes utilisables dans la plupart des composants. Une routine est une classe Java qui ne contient que des méthodes statiques. Pour appeler la fonction, il suffit de l'appeler dans n'importe quel champ de composant qui accepte du code java <NomDeLaClass>.<Méthode>. Les routines sont reconnues dans tous les jobs et sont exportées automatiquement si elles sont utilisées.

Par exemple, vérifier qu'une chaîne de caractères est soit *null* soit vide peut générer beaucoup de code redondant dans Talend, c'est une méthode qui est beaucoup utilisé dans les flux. Il paraît donc pertinent de mutualiser le code dans une fonction et de pouvoir l'utiliser simplement dans tous les flux.

3.5 Stratégie de construction de l'entrepôt de données

3.5.1 Ordonnancement générale

La construction de l'entrepôt de données se fait sur un différentiel chaque matin. Chaque processus se fonde sur une table de suivi nommée *etl_execution*. Ce n'est pas une table qui appartient au

modèle en étoile, elle n'a aucune valeur métier, il s'agit d'une table « utilitaire ».

ID	DATAWAREHOUSE	START_DATE	END_DATE	DURATION	STATUS	NB_ROWS_INITIAL	NB_ROWS_FINAL
2012	VISITS_NETWORK	2015-01-31 05:10:47	2015-01-31 05:11:27	40474	success	6577	6577
2011	VISITS_LIBRARY	2015-01-31 05:10:47	2015-01-31 05:11:27	40474	success	6685	6685
2010	RESERVATION	2015-01-31 05:10:05	2015-01-31 05:10:46	42796	success	0	0
2009	PRET_FONCTIONNEL	2015-01-31 05:09:40	2015-01-31 05:10:03	24179	success	0	0
2008	PRET	2015-01-31 05:08:51	2015-01-31 05:09:39	48359	success	0	0
2007	INSCRIPTION	2015-01-31 05:04:05	2015-01-31 05:08:49	284301	success	0	0
2006	ABONNES	2015-01-31 05:02:59	2015-01-31 05:04:04	66382	success	54424	54424
2005	EXEMPLAIRES	2015-01-31 05:00:08	2015-01-31 05:02:57	168796	success	0	0

FIGURE 14 – Ordonnancement des différents Jobs

Cette table recense pour chaque exécution d'un flux, la date de début, la date de fin le status et le nombre de lignes en entrée et en sortie. Cependant, bien que très utile pour le suivi au quotidien de l'ETL, l'utilité première de cette table est de retrouver la date de dernière construction pour chaque type de flux.

En effet, afin de tenir le délai des 3 heures pour construire l'entrepôt de données, il est important de ne chercher que les données modifiées depuis la dernière construction. C'est donc sur cette table que les processus vont chercher cette date. Dans l'exemple de la figure 14, le prochain flux exemplaire va rechercher tous les exemplaires qui ont été modifiés après le 31 janvier 2015 à 5 heure 2 et 57 secondes.

C'est l'un des développements spécifiques qui a été fait pour e-Vision dans la base d'e-Paprika. En effet, chaque table qui alimente une table de fait contient une colonne « last_updated_date ». Cette valeur est mise à jour par un « trigger » à chaque insert et delete dans les tables. Cette méthode permet d'avoir le nombre de lignes dans la table source e-Paprika (*NB_ROWS_INITIAL*) et les lignes mises à jour dans la table de fait de l'entrepôt (NB_ROWS_FINAL).

3.5.2 Ordonnancement interne des processus

Lors de la construction d'un fait, il est nécessaire à celui-ci d'avoir toutes ses dimensions à jour. On doit donc mettre à jour les dimensions rattachées à un fait avant de le construire. Cependant, le schéma d'e-Vision est ce que l'on appelle une constellation plus qu'un schéma en étoile. En effet, une dimension peut être commune à plusieurs faits, ce qui implique que l'ordre d'exécution des processus est important.

Plus précisément, on ne peut pas lancer le contexte des prêts avant celui des exemplaires. En effet, le processus des exemplaires va mettre à jour toutes les dimensions nécessaires à la construction du fait puis va construire ce dernier. Le fait des prêts, quant à lui, nécessite certaines dimensions qui sont sous la responsabilité du contexte exemplaire. Typiquement, la dimension *dim_bibliographicRecord* qui correspond aux notices bibliographiques. On ne remet donc pas ces dimensions à jour lors du processus des prêts. Celui-ci actualise uniquement les dimensions qui lui sont propres.

3.5.3 Gestion des contextes

Afin de séparer le développement de l'exploitation, il a été décider de sortir les contextes de Talend. La stratégie consiste à ne plus avoir les mots de passe directement dans les archives. Dans Talend, il n'existe plus qu'un seul contexte utile, « Default ». Ce contexte est celui qui est utilisé lors du développement, il contient toutes les variables qui permettent de faire tourner les processus. Les variables présentes dans le contexte sont de deux types.

Contenu des contextes

La première catégorie contient les variables communes à tous les environnements. Ce sont des variables qui reviennent à de multiples endroits du code et qui ont été mutualisées. On peut les faire passer d'un job parent vers tous ses jobs fils. On retrouve parmi celles-ci le nom des tables des deux bases de données qui sont fixes et identiques pour tous les environnements.

Parmi les variables que l'on souhaite faire passer dans tous les jobs d'un processus, la plus significative est sans doute la date de dernière mise à jour. En effet, le job principal de chaque processus qui génère une table de fait va aller chercher la date de dernière mise jour dans la table *etl_execution*. Cette date va donc être valorisée dans une variable de contexte qui pourra être propagée à l'ensemble des jobs et sous jobs.

La seconde catégorie de variables regroupe celles qui sont spécifiques à la machine sur laquelle est exécuté le processus. On retrouve dans cette catégorie les adresses et informations de connexion aux différentes bases de données, le chemin vers les répertoires de travail dans l'arborescence du fichier etc... C'est cette seconde catégorie de variables qui reste problématique.

Chargement dynamiques

Le chargement dynamique des contextes permet de déporter cette problématique de spécificité de plate-forme à l'exploitation. Sur la figure 15, on observe les sous-jobs en rouge les composants nécessaires au chargement des contextes. Ces composants sont rajoutés dans tous les jobs principaux des différents processus.

Le composant « context_in » est un job qui vérifie l'existence d'un fichier de configuration au format xml et les charger dans un flux de données Talend (donc dans un schéma). Ce fichier contient des noms de variables et leurs valeurs pour l'environnement. Ce flux est transmis à un composant natif de Talend le « tContextLoad_1 ». Celui-ci charge et remplace les variables du fichier dans le contexte par défaut. De cette manière, toutes les variables spécifiques à l'environnement sont écrasées. Les variables communes sont quant à elles conservées à partir du contexte « Default ». Cela a donc l'avantage de laisser au développement la liberté de créer des variables de contextes et à l'exploitation de se détacher du développement pour les mises en production ou la maintenance des plates-formes.

Ensuite, on vérifiera en début de job si les contextes sont chargés. Si l'on ne détecte pas de chargement (fichier inexistant par exemple), on interrompt le job. En effet, le risque d'exécuter le processus avec des variables de connexions erronées pourrait avoir de graves conséquences.

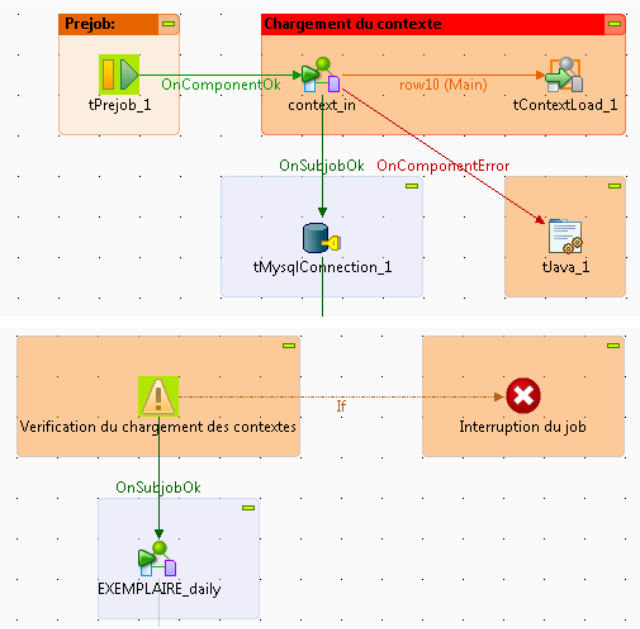


FIGURE 15 – Chargement des contextes dynamiques

Cependant, le chemin vers le fichier de configuration des contextes reste spécifique à chaque environnement. Et il est évident qu'avec cette méthode ne peut pas le charger dynamiquement. Pour cette valeur et uniquement celle-ci, les scripts de déploiement Puppet vont modifier la ligne de commande qui lance les processus.

```
java -Xms1024M -Xmx3328M -cp
$ROOT_PATH/../lib/advancedPersistentLookupLib-1.0.jar :
$ROOT_PATH/reprise_extraire_id_0_1.jar :
$ROOT_PATH/reprise_job_0_1.jar :
$ROOT_PATH/bridge_booking_publicuser_classification_0_1.jar :
eppk_vision.reservation_process_0_1.RESERVATION_process
–context=Default
–context_param CONTEXT_FILE=/var/local/talend/sources/context_params.xml
"$@"
```

Cette ligne de commande qui lance le processus de construction du fait de réservation montre bien la configuration demandée. On garde le contexte « Default » quel que soit l'environnement. On ajoute le chemin vers le fichier xml de configuration. Une autre solution aurait été de placer directement toutes les variables spécifiques dans ce script, mais dans ce cas Talend n'aurait pas la possibilité de savoir si le chargement a correctement été fait.

3.5.4 Construction d'une dimension

La stratégie de construction d'une dimension est sensiblement toujours la même dans le projet e-Paprika. On peut voir sur la figure 16 cet enchaînement de composants.

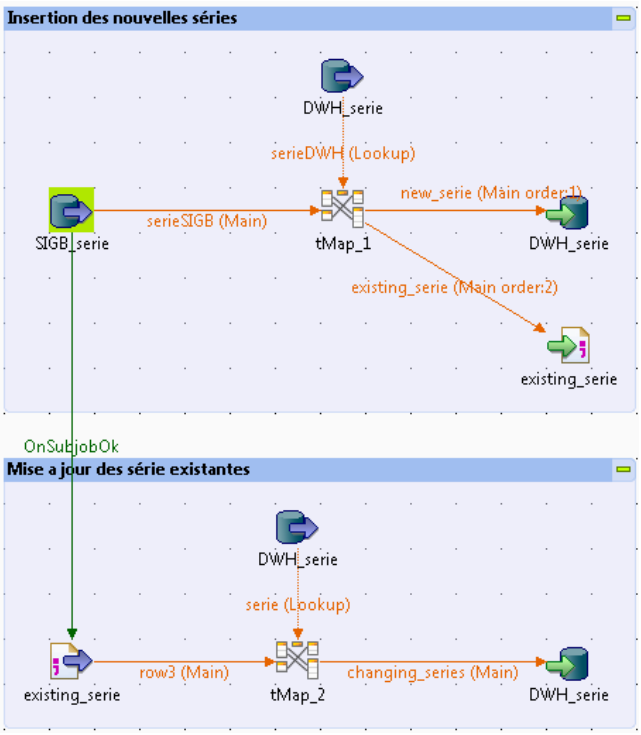


FIGURE 16 – Construction de la dimension serie

Recherche des lignes du modèle source

Le composant d'entrée en vert sur la figure 16(SIBG_serie) permet d'extraire les données du modèle e-Paprika. Ce composant cherche les informations de connections dans les variables de contexte ainsi que le nom de la table. Il se base sur une requête MySQL qui permet une projection et une sélection des champs. Dans l'exemple la volumétrie des séries est très faible, environ 30 000 lignes pour l'ensemble des réseaux, c'est pourquoi on ne filtre pas sur une date de dernière mise à jour. Sur les très grosses dimensions comme celle des exemplaires, plusieurs millions de lignes, il faut filtrer sur la date de dernière mise à jour de l'entrepôt. Cependant, le filtre sur une date dans les dimensions reste une exception. En effet, le gain ne serait pas significatif, la table source n'a donc pas de champ contenant la date de dernière mise à jour. C'est à Talend de faire la différence entre les données qui ont changées et celles qui sont restées les mêmes durant le processus.

Le second composant d'entrée est le composant « DWH_serie ». Ce composant cherche uniquement dans la table de dimension de l'entrepôt la correspondance entre l'identifiant e-Paprika (en uuid) et celui d'e-Vision (en auto-incrément).

Tris des lignes : insertion et mise à jour

Avec cette correspondance, le *tMap* peut faire une jointure interne avec l'identifiant de e-Paprika de la table série. Les lignes qui satisfont cette jointure sont les lignes dont l'id existe déjà dans l'entrepôt. Ces lignes sont donc candidates pour une mise à jour et sont exportées dans un fichier texte. Les lignes qui ne satisfont pas la jointure n'existent pas dans l'entrepôt. On insère donc ces lignes via le composant de sortie de type « mysql_output ».

A ce stade, les nouvelles données sont insérées dans la table de dimension. Cette étape dans la construction peut varier sur d'autre dimensions plus complexes. On peut en effet faire rentrer plusieurs tables dans le composant *tMap*. En effet, si on prend la dimension des notices bibliographiques, on retrouve en entrée de ce premier tMap les tables du SIGB des éditeurs, des titres ou encore des responsables.

Séparation des tâches

On aurait pu se passer ici d'un fichier texte et mettre directement le second *tMap* à la sortie du premier. Cependant, cette méthode est plus pertinente vis-à-vis des ressources de la machine. En effet, Talend libère (en partie) la mémoire utilisée par les composants entre deux sous-jobs. Les composants de type *tmap* sont très gourmands en ressource mémoire. Ainsi, même si une table de 30 000 lignes ne justifie pas une séparation entre deux sous-jobs telle que celle-ci, il a été décidé de le faire pour toutes les dimensions. On isole tous les *tmap* au maximum de manière à limiter les problèmes de mémoire, on a donc une tâche pour un sous-job.

Mise à jour

On se place ici dans le second sous-job avec en entrée le fichier plat généré lors du tri des lignes. La seconde entrée, est à nouveau la table dimension de l'entrepôt. Seulement, cette fois, ce n'est plus la correspondance des identifiants qui est remontée, mais toutes les informations qui peuvent potentiellement demander une mise à jour. Le composant de jointure essaye de faire une correspondance de type « Inner Join » entre les champs du fichier texte et ceux de la base de données. On obtient à la sortie du *tMap* ceux qui ont une correspondance exacte et les autres qui ne satisfont pas la jointure. Ces derniers sont les enregistrements à mettre à jour, on redirige donc ces éléments vers un composant de sortie MySql configuré en mode mise à jour.

3.5.5 Construction d'un fait simple

Il existe deux types de construction pour les faits dans e-vision. La première est la plus simple, elle est faite pour les faits présentant peu de volumétrie. Les faits demandant des traitements spéciaux sont abordés dans la section 3.5.6. La figure 17 nous présente la construction du fait des inscriptions.

Sélection des données

Ici, la sélection des données se fait directement dans le composant d'entrée qui recherche les données dans la table de référence du SIGB Dans l'exemple de la figure 17 c'est le composant « publicUserSubscriptionSIGB » qui remplit ce rôle. On observe sur la requête 3.1 que la sélection des données modifiées se fait sur les deux tables *publicUser* et *publicUserSubscription*. Les champs sélectionnés par cette requête constituent donc le schéma d'entrée du fait. Il est converti dans un schéma Talend qui est transmis aux composants *tmap* en tant que flux principal.

Listing 3.1 – Requête de sélection des inscriptions

```
1 SELECT
2     s . ‘ id ‘ ,
```

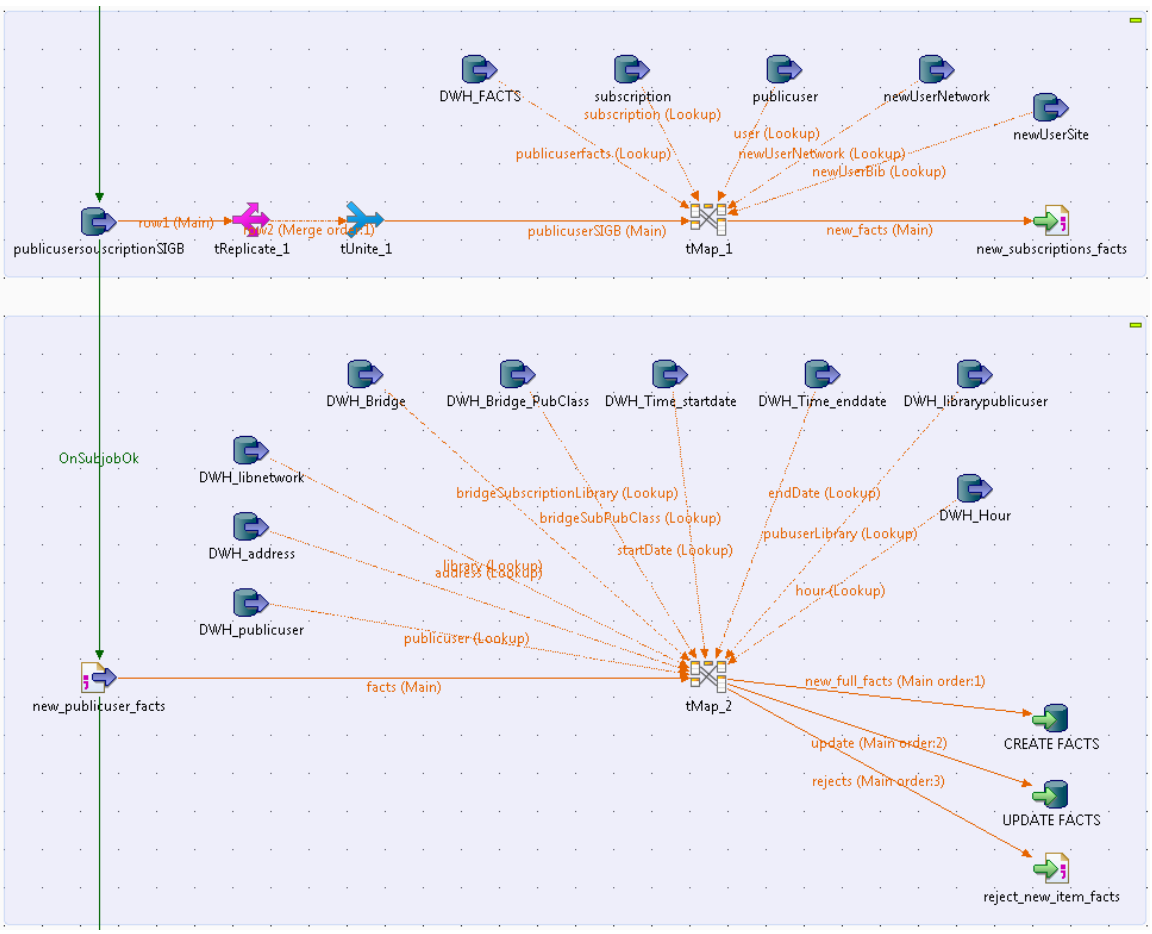


FIGURE 17 – Construction d'un fait simple (inscription)

```
    s.‘publicuser_id‘,
    s.‘subscription_id‘,
    s.‘startDate‘,
    s.‘endDate‘,
    s.‘deleted‘,
    s.‘library_id‘
FROM publicUserSubscription s,
     publicUser p
WHERE 0= 0
  AND p.id = s.publicuser_id
  AND p.func = 'N'
  AND (
    s.‘lastUpdatedDate‘ >= '" + context.LAST_DATE_SUBSCRIPTION + "'
   OR
    p.‘lastUpdatedDate‘ >= '" + context.LAST_DATE_SUBSCRIPTION + "'
  )
]
```

Enrichissement des données

La seconde étape est l'enrichissement des données, on recherche toujours dans le SIGB les données que l'on veut présenter dans le fait. C'est à dire les données du fait en lui-même plus les données des dimensions. Le tableau 4 recense les champs et leur provenance

TABLE 4 – Champs du flux de construction du fait de inscription : première étape

Champs du schéma	Table de provenance	Base de données
idFacts	identifiant du fait *publicUserFacts*	e-Vision
subscriptionName	subscription	e-Paprika
startDate	publicUserSubscription	e-Paprika
startHour	publicUserSubscription	e-Paprika
endDate	publicUserSubscription	e-Paprika
ageStartDate	publicUserSubscription & publicUser	e-Paprika
ageEndDate	publicUserSubscription & publicUser	e-Paprika
ageEndYear	publicUserSubscription & publicUser	e-Paprika
idAdressSIGB	publicUser	e-Paprika
deleted	publicUser	e-Paprika
price	publicUserSubscription	e-Paprika
newUserNetwork	publicUser	e-Paprika
newUserLibrary	publicUserSubscription (2)	e-Paprika
publicUserIdSIGB	publicUserSubscription	e-Paprika
idLibraryNetworkSIGB	publicUser	e-Paprika
idPublicUserLibrarySIGB	publicUser	e-Paprika
idSubscriptionSIGB	publicUserSubscription	e-Paprika
_idPublicUserSubscriptionSIGB	publicUserSubscription	e-Paprika
_idNetwork	publicUserSubscription	e-Paprika

Ce tableau nous montre que les données sont complétées avec la bases d'e-Paprika. Certains champs sont issus de fonctions qui dépendent de deux tables. De plus on peut voir que le champ *newUserLibrary* a été repris de la même table que la table source mais à partir d'un autre composant d'entrée. Ce champ nous dit si l'utilisateur est nouveau sur le réseau, or avec uniquement les données modifiées depuis la dernière construction de l'entrepôt, il n'est pas possible de le savoir.

Le composant « newUserSite » recherche l'inscription la plus ancienne de l'utilisateur pour la comparer à celle qui doit être modifiée en base. Ces exemples nous montrent que la première partie de la création du fait est une manipulation de données dans le but de mettre en forme les enregistrements. Ainsi, il est plus facile de les intégrer dans la table de fait. Cette partie aurait pu être déportée dans un ODS (Operational Data Store ou magasin de données opérationnelles), mais il a été jugé inutile de créer un ODS pour e-Vision étant donné qu'il n'y avait qu'une seule source. La manipulation des données est de ce fait assez légère. C'est un état de fait qui pourrait être appelé à changer dans les prochains mois, si d'autre sources de données viennent à alimenter e-Vision.

Tri des données

Il n'y a pas à proprement parler de tri des données dans cette phase. Cependant, on peut voir dans le tableau 4 l'identifiant de la table de fait. Cet identifiant est valorisé de la même manière que les lignes sont triées dans les dimensions. Nous avons un composant d'entrée « DWH_FACTS » qui donne la correspondance entre l'id de e-Paprika et les id de l'entrepôt. Le composant *tMap* va essayer de faire la jointure pour associer un id de fait à une ligne e-Paprika. La différence est que c'est une jointure externe qui est opérée. On aura donc toutes les lignes en sortie, et cet identifiant est valorisé pour les lignes qui satisfont la requête, sinon il est égal à 0.

Insertion dans le fait

Arrivé à ce stade de la construction du fait, le plus gros du travail de transformation est fait. Il faut alors récupérer les identifiants des tables de dimensions pour les associés aux lignes de la table de fait. Toutes les tables qui rentrent dans le *tMap* sont soit des tables de correspondances entre les id d'e-Paprika et ceux de l'entrepôt, soit la dimension temps. Ces dimensions permettront de faire le lien entre les informations extraites de e-Paprika et les dimensions. En effet, toutes les informations incluses à ce stade dans le flux de données ont normalement une correspondance dans une table de dimension. Avant de construire un fait, toutes les dimensions associées ont été valorisées.

On peut observer sur la figure 17 qu'il existe trois sorties sur le tMap.

- La première est le flux d'insertion des données. Elle correspond aux lignes qui ont un idFacts égal à 0 (voir section 3.5.5).
- La seconde est le flux de mise à jour pour les lignes qui ont déjà une correspondance dans la table de fait (et donc un idFatcs différent de 0).
- La troisième sortie est une vérification des erreurs. En effet, normalement, chaque information du flux doit avoir une correspondance dans une table de dimension. Donc le *tMap* opére uniquement des jointures internes entre les champs du flux principal et les tables de dimensions. Si une ligne ne satisfait pas à une jointure, elle est insérée dans le fichier de rejet et fait l'objet d'une investigation.

Contrôle d'erreur

Outre le fait d'avoir le fichier de rejet, chaque ligne qui est insérée ou mise à jour contient une date de dernière mise à jour. Cette date est utilisée pour compter le nombre de lignes impactées par le flux. A la fin du processus, c'est le nombre de lignes avec une date de mise à jour supérieure à la date du début du flux qui est répercuté dans la table *etl_execution* (voir section 3.5.1). On a donc une vision rapide des erreurs survenues sur les flux via cette table.

Limite de la solution

Une des limitations de cette solution est que potentiellement, certains enregistrements passent dans le processus jusqu'au moment de l'update sans qu'il n'y ait aucun changement sur la base (en dehors de la date de dernière mise à jour de l'entrepôt). En effet, la date de dernière mise à jour d'e-Paprika garantit que la ligne a été mise à jour dans la base source, mais pas que la donnée mise à jour est répertoriée dans e-Vision. Ceci génère une volumétrie qui peut ne pas être négligeable dans le cas de certains traitements par lot. Cependant, vérifier que les mises à jour sont pertinentes demanderait

TABLE 5 – Champs du processus de construction du fait de inscription : état final

Champs d'entrée	Champs de sortie	Dimension associée
idFacts	identifiant du fait *publicUserFacts*	e-Vision
subscriptionName	subscription	e-Paprika
startDate	publicUserSubscription	e-Paprika
startHour	publicUserSubscription	e-Paprika
endDate	publicUserSubscription	e-Paprika
ageStartDate	publicUserSubscription & publicUser	e-Paprika
ageEndDate	publicUserSubscription & publicUser	e-Paprika
ageEndYear	publicUserSubscription & publicUser	e-Paprika
idAdressSIGB	publicUser	e-Paprika
deleted	publicUser	e-Paprika
price	publicUserSubscription	e-Paprika
newUserNetwork	publicUser	e-Paprika
newUserLibrary	publicUserSubscription (2)	e-Paprika
publicUserIdSIGB	publicUserSubscription	e-Paprika
idLibraryNetworkSIGB	publicUser	e-Paprika
idPublicUserLibrarySIGB	publicUser	e-Paprika
idSubscriptionSIGB	publicUserSubscription	e-Paprika
_idPublicUserSubscriptionSIGB	publicUserSubscription	e-Paprika
_idNetwork	publicUserSubscription	e-Paprika

un temps plus conséquent encore. Il faudrait importer toutes les données des faits dans le second sous-Job pour les comparer avec ceux de l'entrepôt existant comme on le fait pour les dimensions, ce qui imposerait de rajouter un troisième sous-job.

Cette solution est adaptée, comme il est précisé en début de chapitre, pour les faits qui ne présentent pas une trop grosse volumétrie. En effet, les deux composants de jointure, les *tMap* sont très chargés, il n'y a pas de filtre sur les données qui y entrent. Cette solution a commencé à poser des problématiques dans deux cas. La première est la reprise de données, en effet, lorsque l'on fait des évolutions, il arrive que l'on doivent rajouter plusieurs colonnes dans un fait. Il faut alors repasser le flux sur l'ensemble de la base e-Paprika et les données en entrée saturent les composants qui sortent en erreur mémoire.

Une solution aurait été de forcer le *tmap* à faire ses opérations sur le disque plutôt qu'en mémoire. Cette solution consiste en une simple case à cocher sur les composants « tMap ». Cependant, lorsque l'on a fait les tests de performance de cette solution, les processus duraient entre 4 et 5 fois plus longtemps. Les flux quotidiens n'auraient plus tenu en trois heures. Il a donc été préféré de remanier la stratégie de construction des faits.

3.5.6 Construction d'un fait avec optimisation pour la mémoire

Lors de la construction d'un fait, il y a deux composants problématiques, les deux composants de transformation *tMap*. En effet, ces composants sont très gourmands en ressources mémoires et

plus les volumétries augmentent, plus il faut rajouter de la mémoire. La solution d'augmenter les ressources machines n'est évidement pas viable sur le long terme. Il y a deux axes d'amélioration pour réduire la consommation de RAM.

Filtrer les entrées secondaires

La première optimisation consiste à réduire les volumes des entrées secondaires des composants de transformation. En effet, lorsqu'on importe les entrées secondaires (surtout sur le second *tMap*) on ramène toutes les données de la base pour une dimension. Si, pour le moment, cette solution est acceptable pour des faits comme les inscriptions, elle n'est pas viable pour un flux comme celui des exemplaires.

En effet, le fait des exemplaires contient deux tables problématiques en entrée du composant de transformation. La dimension des exemplaires et celle des notices bibliographiques contiennent chacune un peu plus de deux millions de lignes. Cela a pour conséquence que la mémoire utilisée par ces deux tables dans le composant est constante quelle que soit la volumétrie du flux principal. Avec une faible volumétrie, les flux se déroulaient correctement, mais avec l'arrivée de nouveaux clients, le flux quotidien ne passait plus. Le but de cette amélioration est donc de faire un filtre avec uniquement les données qui ont une correspondance dans le flux principal. Cette solution présente cependant la difficulté de faire une jointure entre une table d'e-Paprika et une autre d'e-Vision. On ne peut donc pas le faire directement dans le composant d'entrée.

Réduire le flux principal

La problématique précédente est très utile sur le flux quotidien. Cependant, il peut arriver que dans certains cas, la volumétrie du flux principal atteigne le volume critique. C'est le cas si un très gros réseau est importé, ou en cas de gros traitements par lots (suite à l'installation d'une nouvelle version par exemple). Le but de cette optimisation est donc de limiter le nombre de lignes en entrée du flux. Pour ce faire on découpe le flux principal pour qu'il ne dépasse pas un certain nombre de lignes. On traite par lot de 500 000 lignes pour les productions. Le nombre de lignes varie selon les plateformes et sera paramétré dans les fichiers externes de contexte (voir section 3.5.2)

Implémentation des solutions

La première étape consiste à valoriser dès le début du fait deux tables de références. Cette étape est faite avant la construction des dimensions, c'est un job indépendant nommé « fill_reference_table ». La première est une table qui contient tous les identifiants des exemplaires et des notices bibliographiques impactés par le flux quotidien. Le nom de cette table est *BUFFER_ITEM_FULL_ID*.

La seconde est la table qui contient toutes les données à extraire pour le flux quotidien. C'est la table *TMP_ITEM_SIGB* qui remplira ce rôle. Potentiellement, cette table peut être très volumineuse, mais une fois les données extraites, on dépendra beaucoup moins du système source. Les deux tables dépendent donc de la date de dernière mise à jour du processus

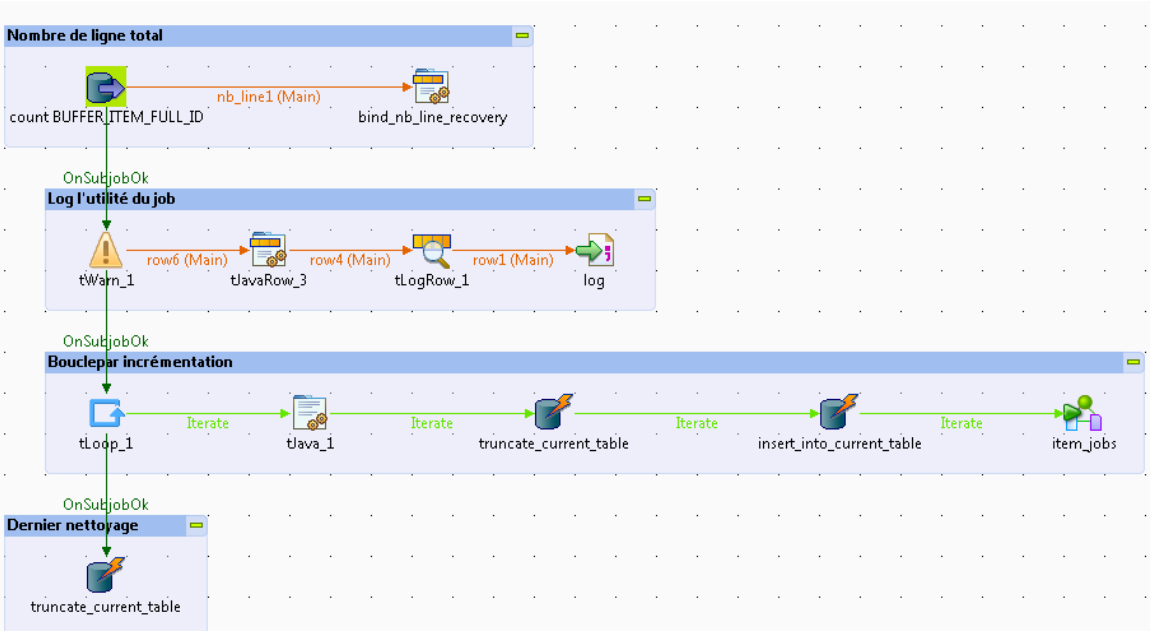


FIGURE 18 – Boucle pour la construction du fait exemplaire

La seconde étape reste classique, on construit les dimensions qui n'ont pas besoin des données chargées dans les tables temporaires. La troisième étape est plus intéressante. On va ordonnancer les différents jobs qui vont utiliser les processus. Pour cela, on va se servir du composant *tLoop*, illustré sur la figure 18.

Ce composant se comporte comme une boucle *tant que*. L'initialisation se fait en Java avec la formule, on valorise la variable de boucle avec le nombre de lignes contenues dans la table *BUFFER_ITEM_FULL_ID*. La boucle continue tant qu'il reste des lignes à traiter (la variable de boucle est supérieure à 0). On décrémente la boucle de la valeur du pas que l'on veut lui donner, par exemple 500 000 en production. C'est la variable de contexte *NB_MAX_LINE_ITEM* qui donne cette valeur.

A chaque itération de cette boucle, on valorise une table temporaire *BUFFER_ITEM_ID* avec le nombre de lignes souhaitées. Cette table sert ensuite à filtrer les dimensions entrantes dans les *tMap*. De plus, les données du flux principal sont extraites à partir de la table *TMP_ITEM_SIGB* et restreintes sur les id de la table filtre *BUFFER_ITEM_ID*. Le filtre se fait grâce à un composant *tJoin* de talend. De cette manière, le flux principal et les grandes dimensions sont proportionnels aux données que l'on veut traiter et non plus au volume total de la base.

3.5.7 Reprise de données

Utilité de la reprise de données

La reprise de données est un mécanisme qui est utile dans deux cas. Lors d'une montée de version, on peut vouloir rajouter des colonnes dans la base de données. Pour valoriser ces colonnes, il faut refaire passer la totalité des enregistrements dans le processus de mises à jour.

La seconde éventualité est arrivée lors d'un très gros traitement par lot qui avait mis l'ensemble des lignes de la table exemplaire à jour. Toutes les lignes de la table des exemplaires avaient donc une date de mise à jour supérieure à la date de l'entrepôt. Le flux quotidien n'a pas pu recalculer la totalité des enregistrements pendant les 3 heures de disponibilité. On a donc annuler la construction de l'entrepôt. Pour se faire, nous avons décalé la date de mise à jour de l'entrepôt d'une journée. La reprise a été nécessaire pour récupérer la journée perdue.

Implémentation de la solution

Le besoin de développer la reprise de données s'est produit au même moment que le besoin d'optimiser la gestion de la mémoire pour le flux des exemplaires. C'est pourquoi certains mécanismes ont été mutualisés. Après l'optimisation de la mémoire, la construction du fait s'appuie sur des tables temporaires pour se construire. Il y a trois tables importantes :

1. La table *BUFFER_ITEM_ID* qui regroupe ces mêmes identifiants mais uniquement ceux qui seront traités lors d'une occurrence. Elle sert à filtrer les données des dimensions et du fait
2. La table *TMP_ITEM_SIGB* qui contient les données complètes à traiter. Elle sert à valoriser le flux principal du fait.
3. La table *BUFFER_ITEM_FULL_ID* qui regroupe l'ensemble des identifiants exemplaires et des notices bibliographiques n'est pas utilisée directement dans le fait. Elle est cependant utilisée pour l'ordonnancement.

La stratégie pour la reprise est de remplir ces trois mêmes tables pour pouvoir réutiliser le job de construction du fait. Cependant, si on garde bien une limite arbitraire du nombre de lignes traitées, on découpe les données de manière un peu plus complexe. A la différence du flux quotidien ou dans tous les cas toutes les données doivent être traitées le même jour, la reprise de données peut se dérouler sur plusieurs jours.

On va donc avoir un découpage par réseau lors de la reprise de données. En effet, un tel découpage permet de ne pas avoir de donnée incohérente pour un réseau. On préfère avoir un réseau non traité que partiellement traité, en effet le suivi et la maintenance sont de ce fait facilités. On s'appuie pour cela sur une table de travail *etl_data_recovery_network*. Cette table est composée de deux champs utiles en plus d'un identifiant unique.

1. *id_network* qui recense tous les réseaux que l'on veut pouvoir reprendre.
2. *id_cube* qui donne l'identifiant du cube que l'on veut reprendre.

On utilise donc une boucle « tant que » avec deux conditions. La première est évidente, tant qu'il y a encore des réseaux à traiter, on continue. La seconde est un impératif d'exploitation, en effet, sur les tests de pré-production, une reprise complète peut durer jusqu'à 10 heures avec les volumétries actuelles. Ce flux est lancé le dimanche après-midi, une période ou il n'y a pas d'activité. Il faut donc avoir une sécurité pour que le flux ne déborde pas sur les traitements de nuit, on va donc rajouter une condition. Le flux ne doit pas dépasser une heure butoir. On met ainsi dans les variables de contextes externes une variable « DATA_RECOVERY_MAX_HOUR » qui permettra de stopper le flux.

A chaque itération de la boucle, on recherche le nombre de lignes associées aux réseaux dans la table de référence. Par exemple, pour le flux des exemplaires, on compte le nombre d'exemplaires pour chaque réseau. On optimise ainsi pour chaque itération les réseaux de manière à se rapprocher le plus possible du nombre de lignes maximum autorisé. On extrait ainsi une liste d'identifiants réseaux qui sert à valoriser les tables temporaires.

Ainsi les tables *TMP_ITEM_SIGB* et *BUFFER_ITEM_ID* sont valorisées avec les données qui nous intéressent. Les dimensions et le fait peuvent donc les utiliser de manière transparente. Cela évite de dupliquer les comportements métiers, de garder l'aspect optimisation et de pouvoir faire la reprise de données à moindre coût.

3.5.8 Magasin de données (Datamarts)

Généralité

Les magasins de données sont des bases de données qui sont directement attaquées par le requêteur OLAP (On-line Analytical Processing). Dans e-Paprika, il a été fait le choix de faire un magasin de données par réseau. De cette manière, chaque client a accès à ses données et tout est cloisonné.

Stratégie générale

La construction des magasins de données est basé entièrement sur le fichier de configuration de Mondrian. Ce fichier décrit le cube OLAP, les relations entre les tables utilisées par le requêteur. La section 4.4.4 décrit plus en détails la structure de ce fichier. Nous décrivons cependant les bases ici pour une meilleure compréhension et une facilité de lecture.

Mondrian est un fichier xml qui permet de décrire une structure OLAP au dessus du modèle relationnel. Il associe des faits, des dimensions, des hiérarchies aux différentes tables de la base de données. On retrouve aussi le lien entre les tables (quels champs joindre pour créer des jointures correctes). La structure de Mondrian se divise en cubes. Un cube peut avoir plusieurs dimensions propres ou utiliser les dimensions génériques. Les dimensions classiques sont associées à une table, tandis que les dimensions en flocon et les relations n-n (voir section 3.3.4 pour plus de détail) sont associées à deux tables. Une dimension générique utilise aussi une table, mais celle-ci est déclarée à l'extérieur des cubes et peut donc être utilisée dans n'importe quel fait.

Un magasin de données est copie de l'entrepôt, mais avec uniquement les valeurs qui concernent un réseau. Il a donc fallu trouver un moyen de construire ces entrepôts qui génèrent le moins possible de maintenance. La première solution a été de copier les tables une à une et d'inclure un filtre sur le réseau dans les tables qui nécessitaient un tel traitement. Cette solution s'est avérée contraignante pour les développements. En effet, à chaque fois que l'on souhaitait créer ou supprimer une table, modifier une colonne, il fallait aussi revenir sur les processus de création des magasins de données. La stratégie consiste ici à parcourir le fichier Mondrian pour extraire toutes les tables qui sont utilisées par les cubes.

Implémentation

De cette manière, il n'est plus nécessaire d'intervenir sur ce flux lorsque l'on rajoute ou que l'on modifie des tables. Une fois que l'on a la liste des tables utilisées, on créé une double boucle d'itération.

La première itère sur la liste des réseaux que l'on récupère dans la base e-Paprika. Ainsi pour chaque identifiant de réseau, on exécute les actions suivantes :

1. Supprimer la base de données correspondante au réseau. Le nom de la bases de données est de la forme dm_<identifiant du réseau>. Les magasins de données sont reconstruits entièrement chaque jour, l'étape de suppression de base de données est donc la plus rapide.
2. La seconde étape est de recréer une base de données vide sur ce même nom de réseau. La base de données est créée en MyIsam avec le jeu de caractère *UTF8* en *general_ci*
3. Une fois ces deux étapes effectuées, on créé une itération sur la liste des tables précédemment récupérées. Et ainsi pour chaque table :
   (a) On créé la structure de la table avec une requête mysql de type "like" de la forme : *CREATE TABLE '<Nom table>' LIKE '<Nom base de l'entrepôt>'.'<Nom table>'*. Cette requête permet de garder la structure de la table et les indexes de l'entrepôt.
   (b) On regarde ensuite pour la table si une colonne *_idNetwork* existe. Si celle-ci existe, on créé une clause de sélection pour l'id du réseau de l'itération en cours.
   (c) Enfin, on insère les données dans la table du magasin de données avec une requête de type INSERT INTO '<nom table>' SELECT * FROM '<Nom base de l'entrepôt>'.'<Nom table>' WHERE 1 = 1 <Clause de restriction sur l'identifiant réseau>

Contraintes de la méthode

Il existe trois contraintes principales pour cette méthode. La première porte sur le nom de la colonne donnant l'identifiant du réseau qui doit absolument être standardisé. En effet, le test se base sur le nom de la colonne dans la base de données *information_schema*. C'est donc une contrainte au niveau du développement qui doit être identifiée et sur laquelle on ne peut pas avoir facilement de contrôle.

La seconde contrainte est que la base de données doit tourner avec un moteur qui ne vérifie pas les contraintes d'intégrité tel que MyIsam. En effet, l'ordre des tables n'est pas vérifié, et on peut insérer des données dans la table de fait avant les dimensions.

La troisième contrainte est que l'entrepôt de données doit être sur le même serveur que les magasins de données . En effet, les requêtes de création et d'insertion des données que l'on utilise impliquent que les deux bases soient sur le même serveur.

3.5.9 Archives de fin d'année

Stratégie générale

Les archives de fin d'année ne sont pour le moment pas encore totalement automatisées. Elles concernent deux cubes, les *exemplaires* et les *abonnés*, ce sont les deux cubes qui ne relatent pas de fait en tant qu'activité. Le processus se déroule en deux temps. Le premier est la construction d'un entrepôt avec un périmètre réduit à ces deux cubes. Dans un second temps, on construit les magasins de données pour chaque réseau. Cette stratégie permet de garder les données de l'entrepôt en date du 31 décembre afin de pouvoir opérer des corrections de données si le besoin s'en fait sentir dans l'année. On peut réutiliser le processus de construction des entrepôts de données décrit dans la section précédente.

Pour exécuter cette stratégie, on part d'un fichier Mondrian qui ne contient que les données des deux cubes. La construction de ce fichier n'est pour le moment pas automatisée, elle se fait manuellement. En effet, si Talend peut facilement construire le fichier, il n'a pour le moment pas accès au logiciel de gestion de version (SVN (Subversion)) lorsqu'il est exécuté sur une production.

Implémentation

La première étape consiste à créer la base de données d'archive. Le nom de la base de données calculé est construit sous la forme <nom de la base de l'entrepôt>_<année d'archivage>. L'année d'archivage est configurée dans les variables de contextes qui sont externalisées à l'exploitation. Cependant, l'archivage de fin d'année est considéré comme une étape critique du module statistique. On rajoute donc une sécurité dans le cas où le nom de la base de données existerait déjà (le service d'exploitation aurait par exemple oublié de changer l'année dans sa configuration). On retrouve donc, grâce à la base système de MySql *information_schema*, le nom des différentes bases de données existantes. Si l'on retrouve le nom de la base de données dans cette liste, on modifie le nouveau nom en ajoutant un suffixe "_"<numéro de séquence>. On préfère avoir une base de données mal nommée plutôt que de ne pas avoir les informations correctes au jour J.

La valorisation de l'entrepôt se fait sensiblement selon la même méthode que la construction des magasins de données décrit dans la section 3.5.8. On parcours donc le fichier Mondrian pour en extraire les tables utiles. Puis on fait une itération pour créer les tables et les valoriser avec les données. La procédure est plus simple dans la mesure ou l'on garde toutes les données de chaque table. On ne filtre pas ici sur les réseaux.

La construction des magasins ne varie que très peu de la construction quotidienne. La méthode est la même à ceci près que l'on se fonde sur l'entrepôt d'archive et sur le fichier Mondrian réduit aux deux cubes qui nous intéressent.

3.5.10 Copie de réseaux

Le besoin

Comme e-Paprika, e-Vision est installé sur plusieurs plates-formes en production, or, on doit pouvoir faire migrer un réseau d'une plate-forme à une autre. Cette action peut être motivée par des problèmes de performance ou d'équilibrage de volumétries. Le transfert d'un réseau d'une plate-forme à une autre doit s'opérer avec une simple ligne de commande.

La problématique

La vraie problématique de cette action vient des clés primaires et étrangères d'e-Vision. Dans e-Paprika, le système des identifiants est fondé sur les uuid. Ce n'est pas le cas dans e-Vision où tous les identifiants sont des entiers autoincrémentés. La copie brute d'un entrepôt vers un autre est dès lors impossible. Il faut recalculer pour chaque clé sa nouvelle valeur dans la base de destination. Or la base de données est en MyIsam et donc ne référence pas les clés étrangères en tant que telles. Elle se contente de créer des index sur les colonnes pour optimiser les jointures, mais les relations n'existent pas.

La stratégie

La copie de réseau est développée dans un job Talend d'entrée de manière à pouvoir le lancer avec une seule ligne de commande. La stratégie de copie s'articule autour de cinq étapes principales.

1. Vérifier la version des bases de données.
2. Créer un schéma de données en innodb avec les relations réelles.
3. Valoriser ce schéma avec les données du réseau que l'on veut copier.
4. Modifier les identifiants pour qu'il soient cohérents entre eux et avec les données de la base de destination.
5. Vérifier l'intégrité des données et garder une trace des données non insérées.
6. Insérer les données dans la base de destination en forçant les clés primaires.

Prérequis

Il y a un prérequis pour la copie de réseau. Il faut en effet connaître les variables de connexions des deux bases, la source et la destination. Le script est lancé à partir de la base d'origine, donc les variables de connexions se trouve dans le fichier de contexte comme sur tous les autres Jobs. Il faut donc fournir au job le fichier de contexte de l'environnement de destination. C'est une des concessions que l'on a dû faire lorsque l'on a fait sortir les contextes de Talend pour les externaliser à l'exploitation.

Implémentation

Le Job principal récupère les deux contextes. On créé la base de données temporaire en innoDb. Il utilise ensuite le script Liquibase pour créer les tables. Liquibase contient toutes les constructions

de clés étrangères de la base de données. C'est la plus grosse contrainte de cette méthode. On doit en effet maintenir les fichiers liquibase pour que toutes les relations soient créées quand bien même elles ne seront pas utilisées au quotidien.

Listing 3.2 – Requête de sélection des inscriptions

```
<changeSet author="DEV" id="08">
    <preConditions onFail="MARK_RAN">
        <sqlCheck expectedResult="1">
            SELECT ENGINE != 'MyISAM'
              FROM information_schema.tables
             WHERE table_schema = DATABASE()
               AND TABLE_NAME = 'facts_loan_bibrecord_interest';
        </sqlCheck>
    </preConditions>
    <addForeignKeyConstraint
        constraintName="facts_loan_bibrecord_interest_idBridge_FK"
        baseTableName="facts_loan_bibrecord_interest"
        baseColumnNames="idBridgeLoanBibrecord_FK"
        referencedTableName="bridge_loan_bibrecord"
        referencedColumnNames="idBridgeLoanBibrecord"
        onUpdate="CASCADE"/>
</changeSet>
```

On peut voir sur le code 3.2 que la création dans la base d'une clé étrangère. Pour ne pas avoir a créer cette clé dans les schémas de l'entrepôt, on met une précondition sur le type de moteur utilisé. De cette manière, cette instruction n'est exécutée que pour la table temporaire de la copie de réseau. Le script liquibase peut donc créer une structure base de données complète avec les relations n-n.

Pour valoriser les tables dans la base de données temporaire, on ne peut pas exécuter la requête utilisée pour la création des tables des magasins de données. En effet, il n'est pas possible pour MySQL de faire une requête sur deux serveurs différents de manière simple et performante. Talend permet de transférer facilement les données d'une base à l'autre à condition de définir un schéma. Cette solution nous obligerait a créer un export pour chaque table (un composant d'entrée, un de sortie et le schéma spécifique à chaque table). La version entreprise de Talend permet de faire cette manipulation avec un schéma dynamique, mais nous utilisons actuellement la version TOS. C'est l'un des deux grands manques de la version gratuite (avec le manque de suivi de version).

Il a donc fallu créer une solution pour automatiser ce transfert de données. Pour ce faire, on construit pour chaque table les requêtes d'insertion via des composants Talend.

1. On récupère la variable de configuration *max_allowed_packet* qui nous donne la taille maximale de la requête d'insertion en masse.
2. On récupère le nom des colonnes dans la base *information_schema* pour construire la requête d'insertion

3. Chaque ligne de la table source est sélectionnée et transformée de manière a pouvoir être insérée dans la requête d'insertion. On échappe donc les champs, on les met entre quote, puis on les concatène avec des virgules. De cette manière, chaque ligne est présente sous forme d'une chaîne de caractères prête à être rajoutée à une requête d'insertion.
4. Une fois chaque ligne transformée, on créé des requêtes d'insertion de taille inférieure à la taille récupérée dans l'étape 1.
5. La dernière étape consiste alors à insérer les lignes dans la base de données en exécutant les instructions d'insertion. On ne gère pas l'ordre des tables lors de cette insertion, on désactive donc les clés étrangères lors de cette étape.

La base de données contient à ce stade toutes les données nécessaires à la copie de réseau. Cependant, étant donné que l'on a désactivé les clés lors de l'insertion, il faut vérifier l'intégrité de la base de données. On créé donc un Job qui parcourt toutes les clés étrangères listées dans la base *information_schema* et vérifier que chaque référence est correcte. Si certaines références ne sont pas satisfaites, on supprime les lignes du modèle et on écrit dans un fichier texte pour permettre une investigation manuelle post-traitement.

Après avoir transféré toutes les données, il faut modifier les clés primaires de manière à pouvoir insérer les données dans le schéma de destination. Comme le montre le code 3.2, on a rajouté la propriété *onUpdate = "CASCADE"* sur chacune des clés. Cela permet de laisser le moteur mysql gérer les mises à jour des colonnes. On récupère donc pour chaque table l'identifiant maximum et mettre à jour les clés du modèle temporaire à partir de celui-ci. On passe par une colonne temporaire sur la base de données pour faire la transition. La colonne temporaire est remplie avec la valeur de la clé précalculée, et on fait ensuite une mise à jour de masse. Le paramètre "CASCADE" va faire que toutes les références à cette clé sont automatiquement mises à jour et nous n'avons pas à tout recalculer.

La dernière étape consiste à intégrer les données dans le modèle de destination. Cette fois, les deux bases de données sont sur le même serveur, on peut donc utiliser une simple insertion comme lors de la construction des magasins de données.

3.6 Améliorations possibles et en cours

3.6.1 Base de données orientée colonnes

Infobright

Comme il est décrit dans le chapitre suivant sur l'exploitation de ces données, l'utilisateur doit pouvoir croiser n'importe quelle dimension d'un fait. Cette contrainte pose le problème des index dans la base de données. En effet, pour optimiser toutes les requêtes, il faudrait créer beaucoup d'index et cela risquerait d'être contre-productif. Toutes les clés étrangères sont couvertes par des index pour faciliter les jointures. Cependant, on ne peut pas optimiser toutes les requêtes de la sorte.

La solution en cours de développement est donc de faire appel à une base de données orientée colonnes pour les magasins de données. Infobright est normalement destiné à des bases de données beaucoup plus volumineuses que celle d'e-Vision. Il est cependant destiné aux applications décisionnelles et peut pallier le manque de performances de MyIsam/Mondrian sur certains point.

Il est développé sous la forme d'un moteur d'accès pour MySQL. La version retenue ici est la version communautaire qui présente quelques limitations. La première est que l'on ne peut pas utiliser les méthodes de mises à jour (*update*) et d'insertion de masse (*bulk insert*). La seconde est qu'on ne peut utiliser qu'un seul cœur des processus.

La première est un obstacle mineur, en effet, les magasins de données sont reconstruits tous les jours, nous n'avons donc pas besoin de mise à jour. Pour les insertions, il suffit de créer des fichiers d'insertion et de les charger dans les bases. Le second point peut être plus problématique. Nous avons en effet peu de retours sur l'utilisation réel du produit et les accès concurrents qu'il doit supporter.

À l'heure où est écrit ce document, la compatibilité d'Infobright avec les différents outils du système décisionnel a été testée. Cependant, les tests de performance et de montée en charge n'ont pas encore été exécutés.

MonnetDb

Cette base de données présenterait des performances meilleures que celle d'Infobright. Cependant, les test de compatibilité et de faisabilité n'ont pas encore été faits. Je dois faire une étude comparative de ces deux produits.

De prime abord, l'avantage de la solution et que ce n'est pas sur une version bridée. La solution est libre de droit et complète. L'inconvénient principal est que ce n'est pas une base MySQL, je ne peut donc pas dire sans faire de tests si Talend et Mondrian se comportent correctement avec ce produit.

3.6.2 Changement de moteur de l'entrepôt de données

À l'origine, le moteur MyIsam a été choisi pour ses meilleures performances en sélection. Cependant, si l'on bascule les magasins de données sur des bases orientées colonnes, les sélections sur l'entrepôt se résumeront à sa mise à jour quotidienne et à la construction des magasins. Or ces étapes devraient pouvoir être assumées par un moteur tel qu'innoDb. Cela permettrait d'avoir un véritable contrôle d'intégrité géré par la base de données.

Le gros travail sur ce changement consisterait à gérer les transactions dans chaque flux de Talend. De plus, il faudrait gérer les rejets unitairement, pour ne pas rejeter des blocs de lignes suite à une seule erreur.

3.6.3 Tests automatisés

C'est la plus grande faiblesse du projet actuellement. Le processus de développement inclut des tests de qualité, cependant il n'y a pas de test unitaire automatisé. L'idée serait de pouvoir prévenir des régressions en automatisant tous les cas de tests. Cette problématique n'a pas encore été étudiée dans le détail. L'un des impératif serait que les tests puissent être exploitables directement par Jenkins. De cette manière le projet s'intégrerait bien mieux dans le processus de déploiement continu et pourrait bénéficier des méthodes de développement de type TDD.

3.6.4 ODS

Comme le décrivent les sections 3.5.4 à 3.5.6, on interroge tout au long du processus la base de données source. Cette méthode présente l'inconvénient de dépendre de la base de données jusqu'à la fin de la construction. De plus, si l'on veut intégrer d'autres sources de données, il sera plus difficile de croiser les informations. Il sera sûrement nécessaire d'aller rechercher une nouvelle fois les données dans la base source.

La solution consisterait à créer un ODS (Opérationnal Data Store). C'est une base de données tampon qui permet de centraliser les données des différents systèmes. Le but est de centraliser les données et d'uniformiser les formats avant de les intégrer dans l'entrepôt de données. L'intégration d'autres logiciels dans e-Vision est à l'étude et devrait arriver au troisième trimestre 2014.

Cette opération pourrait aussi se révéler utile pour les développements à venir concernant les prêts numériques en bibliothèque. En effet, ce cube va devoir interroger des sources externes au projet et le recoupement d'informations serait plus facile avec un ODS.

3.7 Synthèse

Ce troisième chapitre décrit donc dans les grande lignes la construction de l'entrepôt de données utilisé par e-Vision. Toutes les transformations décrites forment le premier pré-requis pour pouvoir permettre à l'utilisateur une lecture transversale de ses données. Cependant, bien que l'entrepôt représente la partie la plus importante du projet, il ne représente pas la totalité de la solution statistique.

Le but du quatrième chapitre est de décrire la solution de reporting qui exploite les données de l'entrepôt. Nous présentons dans ce dernier chapitre les principes d'un moteur OLAP et décrivons l'implémentation de la solution choisie.

Chapitre 4

Outils de requêtage

4.1 Généralités

L'élaboration des tableaux de synthèse est gérée par une application OLAP. Dans le principe, un moteur OLAP est sensiblement la même chose qu'un moteur SQL (Structured Query Language) : on lui envoie des requêtes et il renvoie des résultats. Le langage utilisé par les requêtes OLAP est le MDX (MultiDimensional eXpressions). La grande difficulté consiste à fournir une interface graphique qui permet de facilement écrire ces requêtes MDX et d'afficher les résultats. Nous avons choisi Saiku, car c'est le seul outil open-source (et donc gratuit), actif, ergonomique et dont le périmètre fonctionnel correspond à nos besoins. Il n'intègre en effet pas d'autre solution telle qu'un ETL ou une solution de tableaux de bords.

Les alternatives étaient :

1. Pentaho est une solution trop large et trop complète. Elle couvre le besoin mais propose beaucoup de fonctionnalités non requises. Saiku peut d'ailleurs être intégré en tant que plugin dans la solution de Pentaho.
2. JasperSoft a été écarté pour les mêmes raisons. Cette solution proposait beaucoup plus que ce que l'on avait besoin.

L'intégration au système existant se fait parfaitement. En effet, Saiku partage la même conception qu'e-Paprika. On retrouve une interface RIA, une communication avec le serveur via l'architecture REST et une fonctionnement interne par couches.

Dans les faits, Saiku n'est pas en soi un moteur OLAP. C'est une application complète qui intègre le moteur OLAP Mondrian. En simplifiant à l'extrême, on peut dire que Saiku est « juste » une application qui permet d'interfacer Mondrian avec l'utilisateur final.

Mondrian et Saiku sont fortement couplées. En effet, il ne serait pas possible de remplacer Mondrian. Cependant, ce dernier est un projet open-source indépendant. Il est actuellement la référence des moteurs OLAP open-source. Il n'existe pour ainsi dire pas de solution alternative libre aussi performante à ce jour.

4.2 Présentation de Saiku

4.2.1 Architecture générale

La figure 19 reprend la figure 5 en y intégrant Saiku (en vert sur l'image). L'application Saiku a donc son propre serveur. De la même manière qu'e-Paprika, elle utilise le serveur CAS pour gérer l'authentification des utilisateurs. L'utilisateur se connecte directement via une interface web et communique via des services basés sur une architecture REST.

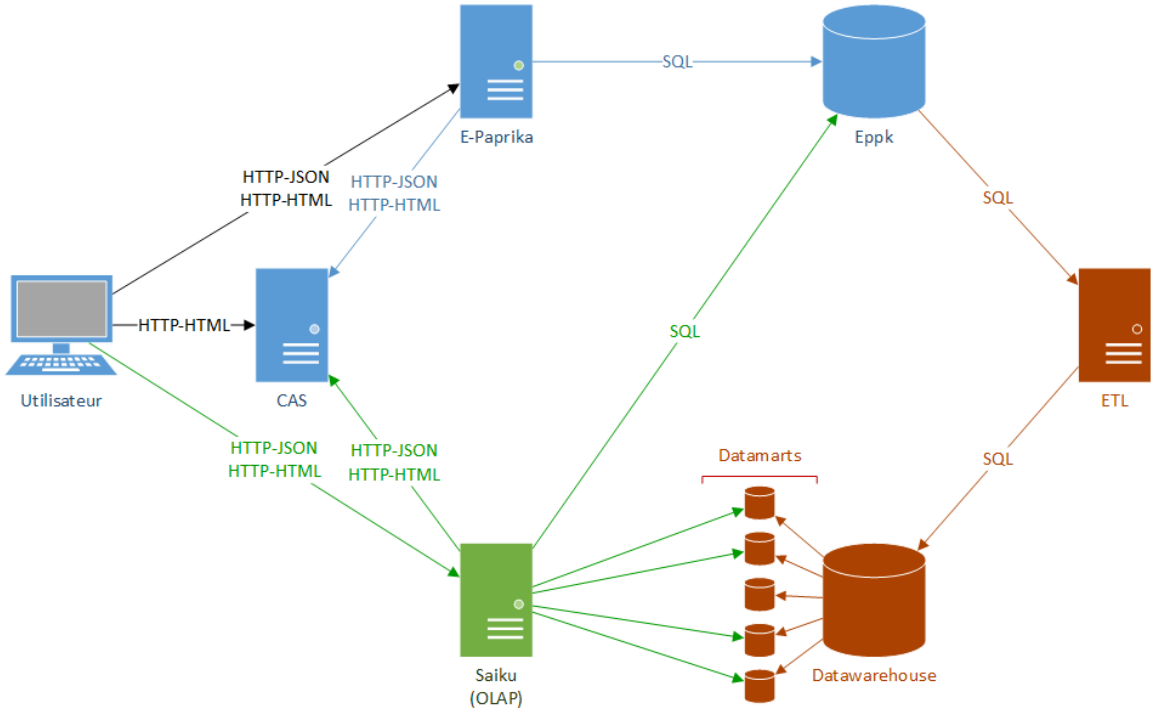


FIGURE 19 – Architecture technique du projet avec l'intégration de Saiku

L'application Saiku communique avec les magasins de données et la base du serveur e-Paprika. Elle ne communique pas avec l'entrepôt de données.

La communication avec le serveur de base de données e-Paprika est très réduite. Elle permet de lister les réseaux éligibles aux statistiques. En effet, il existe dans e-Paprika un paramètre qui indique pour chaque réseau si les statistiques sont activées. Cette liste permet de proposer la liste des réseaux sur l'écran d'accueil administrateur et de pouvoir écrire le nom du réseau sur le bandeau d'entête de la page principale.

4.2.2 L'interface

Navigation entre les pages

Tout comme e-Paprika, Saiku est une application RIA. De ce fait, il y a peu d'enchaînement entre les pages, la figure 20 nous donne une vue d'ensemble des pages.

Il existe trois points d'entrée pour e-Vision.

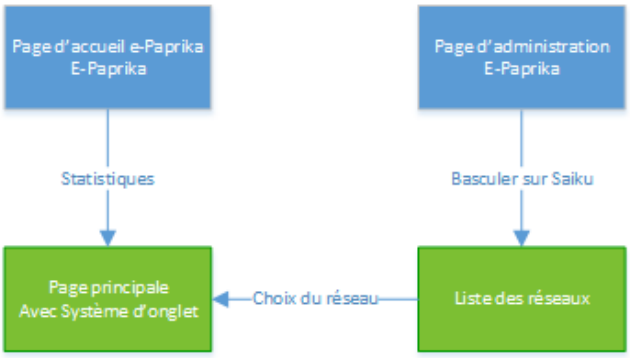


FIGURE 20 – Enchaînement des pages de l'application

1. Le premier est la page d'accueil d'e-Paprika. C'est par ce biais que tous les clients se connectent, c'est un simple lien HTML qui pointe vers l'adresse des statistiques. Ce point d'entrée nécessite d'être authentifié sur un réseau. L'utilisateur et le réseau sont donc connus et la page principale d'e-Vision est directement présentée.
2. Le second est l'interface d'administration d'e-Paprika. Dans ce cas, l'utilisateur administrateur de l'application est connecté. Le point d'entrée est donc une interface spécifique qui fournit la liste des réseaux disponibles. En effet, l'administrateur peut se connecter sur n'importe quel réseau. Une fois le réseau sélectionné, il arrive sur la même page principale que les utilisateurs.
3. Le troisième point d'entrée consiste à taper directement l'URL du site dans un navigateur. Pour ce troisième cas, une redirection vers l'écran de login cas est opérée. Suivant le profil et le réseau associé au profil, on applique les mêmes règles que précédemment. Le super-administrateur se voit proposer la liste des réseaux, tandis qu'un utilisateur classique arrive directement sur son réseau.

Bandeau d'entête

FIGURE 21 – Page des rapports

Le bandeau principale est un élément commun à toutes les pages. Il permet de connaître l'identification de l'utilisateur. Dans la figure 22, on est connecté en super-admin, on a donc la version du logiciel, le nom du réseau sélectionné et l'id du réseau. Lorsque l'on est connecté avec un utilisateur classique, la version et l'identifiant qui sont des informations techniques n'apparaissent pas.

Le menu de dernière mise à jour donne la date de dernière construction de chacun des cubes. Ces dates permettent d'expliquer certaines données incohérentes lorsque la construction des entrepôts ne s'est pas correctement terminée. Le menu *module* permet de créer un nouvel onglet et de retourner sur l'accueil de e-Paprika.

En bleu clair sur la figure 21 on trouve les onglets qui servent à la navigation. Il y a l'onglet des rapports et un onglet par requête ouverte.

Page des rapports

L'onglet des rapports est le premier onglet qui est présenté lorsque l'on accède à l'application. Il est spécifique à un réseau. Chaque réseau a un répertoire nommé *Mes Tableaux* dans lequel les utilisateurs peuvent enregistrer leur requêtes. Ils peuvent aussi créer leur propre arborescences dans ce répertoire, cet espace est le leur. Les autre dossiers contiennent des rapports qui sont faits par le service de qualification.

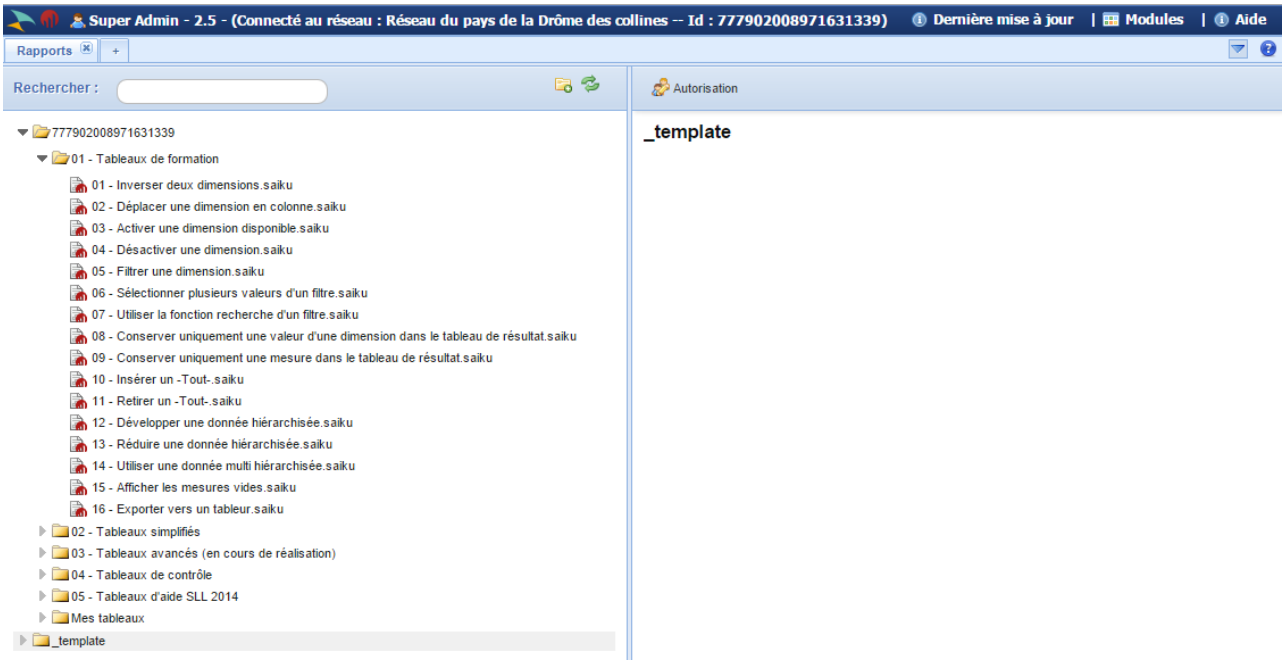


FIGURE 22 – Page des rapports

Le répertoire *_template* sert justement au service de qualification pour créer les rapports génériques. Ce répertoire n'est accessible que si l'on est connecté en super-administrateur. Tous les rapports qui y sont enregistrés sont déployés tous les matins sur les différents serveurs.

La manipulation est faite par un *job* Talend qui parcourt tous les rapports, les modifier et les copier dans des répertoires spécifiques à chaque réseau. Le dossier *_template* est en effet un répertoire qui est commun à tous les serveurs de qualification et de productions. Il existe un répertoire par version du logiciel. Le Job Talend va ainsi passer sur tous les serveurs prendre chaque rapport, adapter l'entête pour chaque réseau et le copier dans le répertoire adéquat. La structure des rapports est précisée dans la section 4.4.5.

Page de requête

La figure 23 montre la page de travail principale pour les requêtes. Cette page est constituée de trois parties principales.

- La colonne de gauche permet de choisir le contexte sur lequel on veut travailler. Il contient aussi toutes les dimensions et les mesures du cube sélectionné. On appelle cette partie le *builder*, il n'est accessible qu'avec les privilèges d'administrateur.

- La partie centrale permet de gérer la requête en cours.
- La colonne de droite permet de passer sur le mode graphique plutôt que sur le mode tableau. Elle permet aussi d'avoir un aperçu des principaux indicateurs sur la requête en cours (Somme totale, valeur médiane par mesure).

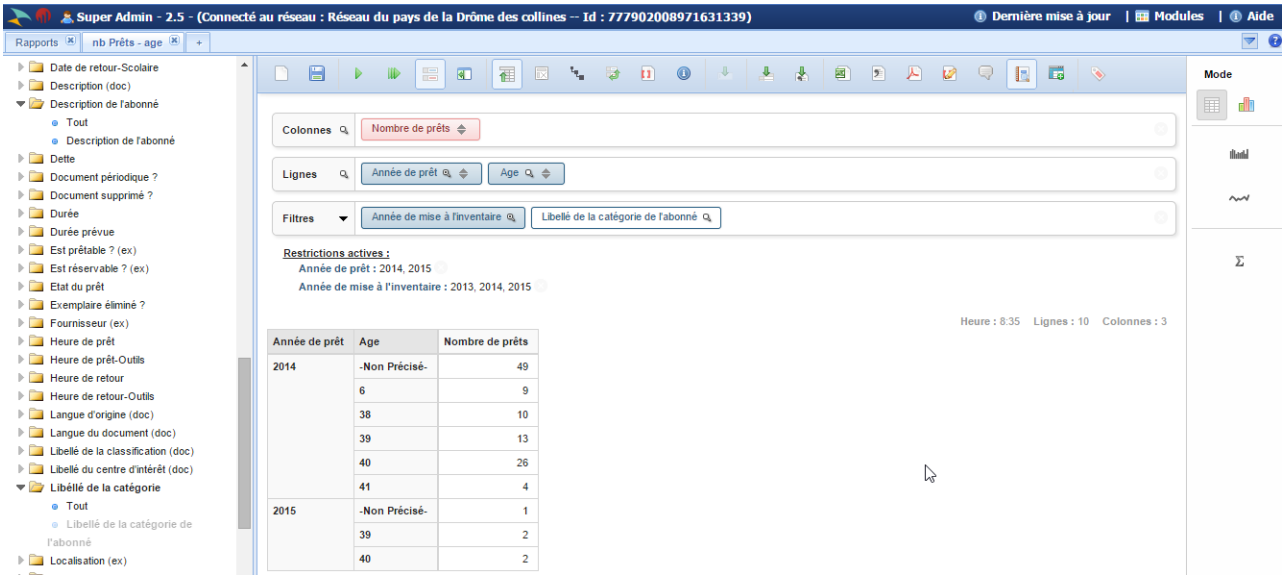


FIGURE 23 – Page des rapports

La partie principale permet de construire la requête. On peut observer la barre d'outils qui permet de sauvegarder ou d'exécuter une requête. Elle contient aussi les boutons qui permettront les différents exports. On peut ainsi exporter au format excel, CSV, et pdf l'ensemble d'un tableau.

En dessous, le bloc de trois lignes, sert de structure pour la requête. On trouve trois Colonnes, Lignes et Filtres. L'interface fonctionne beaucoup sur la base du glisser-déposer. Il suffit en effet de faire glisser un niveau de dimension à partir de la colonne de gauche et de le placer dans un des blocs de réception. Il est aussi possible de glisser-déposer des niveaux de dimensions d'un bloc à l'autre ou au sein d'un même bloc.

Les champs ligne et colonne permettent de donner les champs que l'on veut voir apparaître dans le tableau (respectivement en ligne et en colonne). Les filtres servent à réduire les enregistrements qui seront concernés par la requête sans pour autant que la dimension ne soit présente à l'affichage. On observe sur la figure 23 qu'il y a un filtre bleu et un filtre blanc. Le filtre bleu est un filtre actif tandis que le blanc n'a aucune action sur la requête, c'est un filtre neutre. Ce choix a été fait car on ne propose pas de builder pour l'utilisateur final. Cela permet de lui laisser la possibilité de modifier ces tableaux en activant des dimensions qui ne sont en filtres blancs à l'origine.

En cliquant sur un niveau de dimension, il est aussi possible d'activer l'interface qui permet de sélectionner des filtres. Il existe deux types de filtres, les filtres sur des valeurs et les filtres par intervalles. En dessous des trois blocs qui servent à la construction, on trouve les restrictions actives. Ici, on voit les filtres qui ont été faits sur l'année de prêt et sur l'année de mise à l'inventaire de l'exemplaire.

Enfin, le dernier élément principal de cette partie centrale est le tableau de résultats en lui-même. Le tableau est dynamique, on peut ainsi sur une dimension hiérarchisée comme le temps ajouter un niveau directement à partir du tableau. Il suffit pour cela de cliquer sur la valeur désirée (par exemple 2014) et de choisir *ajouter le niveau* dans le menu contextuel comme le montre la figure 24.

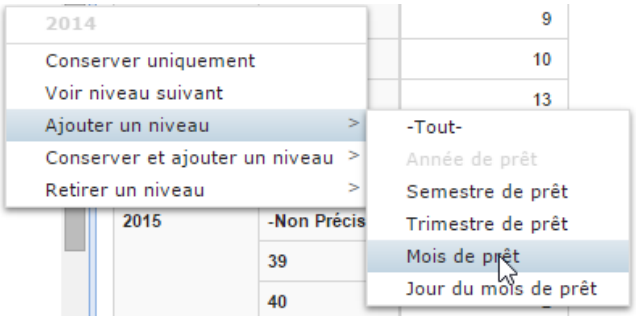


FIGURE 24 – Architecture interne Saiku

4.3 Architecture interne

Tout comme e-Paprika, Saiku est un projet en Java construit grâce à Maven et utilise le framework Spring. L'architecture de Saiku s'organise autour de plusieurs couches. L'utilisateur final a accès à l'application via le module *saiku-ui*. La figure 25 montre l'architecture interne du logiciel.

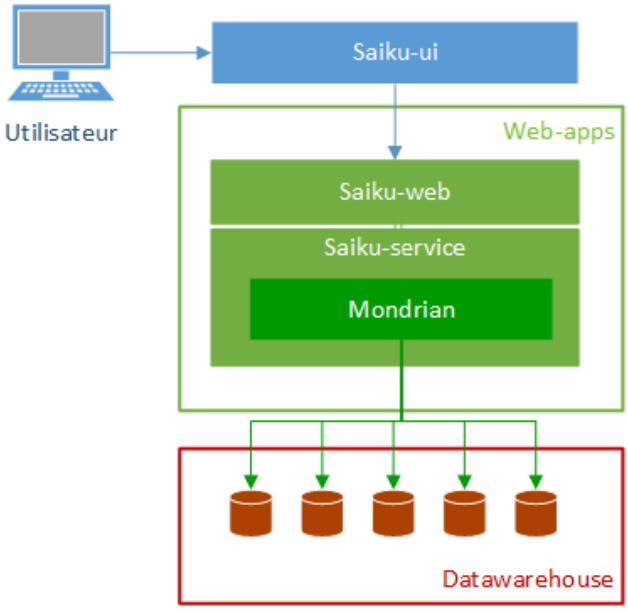


FIGURE 25 – Architecture interne Saiku

4.3.1 Organisation des classes

Le projet est articulé autour de Maven. Maven permet de construire le projet en gérant les dépendances, les artéfacts et les instructions de constructions. Les configurations des Maven sont décrites

dans un fichier POM (Modèle Objet de Projet). e-Vision est organisé en module, cette arborescence en donne l'organisation du projet :

1. **dk-Saiku** : C'est le projet principal, la racine du projet.
   (a) **saiku-core** est le module qui sert de conteneur à toute l'intelligence du logiciel
      i. **Saiku-web** est le module qui contient tous les web-service REST de l'application. C'est cette couche qui va être appelée par l'interface.
      ii. **Saiku-service** est le module contient l'intélligence du logiciel. Il permet de faire les requêtes, de les traiter et de rendre les résultats. C'est dans ce module que l'on retrouve la dépendance de Mondrian.
   (b) **saiku-webapp** est un module qui contient tous les fichiers de configuration et de connexion. C'est le module qui est déployé en tant qu'application web. Il admet dans ses dépendances saiku-web et saiku-service.
   (c) **saiku-ui** est le seul module qui ne soit pas du Java. Maven se contente ici de fournir les instructions de construction. Ainsi, le projet est archivé au format zip lors de la compilation du projet.
   (d) **saiku-server** est un module que l'on n'utilise pas dans e-Vision. Ce module embarque les modules *saiku-webapp* et *saiku-ui* ainsi qu'un serveur apache-tomcat. De cette façon, on fournit une solution tout-en-un qui est prête à être déployée sur les serveurs.

Saiku-ui

Saiku-ui est responsable de toute la couche de présentation de l'application. C'est une application qui se base sur les technologies javascript backbone.js, underscore.js et JQuery.

BackboneJs se présente comme un cadre logiciel de type Modèles, Vues, Collections. Les modèles et les collections de modèles sont synchronisés directement avec les données provenant de l'application coté serveur. La synchronisation des modèles se fait grâce à une API REST. Il suffit d'ailleurs de lier les modèles à une URL qui pointent vers les services e-Vision pour faire la synchronisation

UnderscorJs est une librairie présente dans les dépendances de BackboneJs. Elle offre la possibilité de manipuler des objets, des collections ainsi que des tableaux. Elle offre aussi des fonctions d'aides aux développements.

JQuery permet aussi de se servir de templates pour pouvoir gérer le rendu. Chaque élément présent dans le rendu de e-vision provient d'un template JQuery qui sont tous centralisés dans l'*index.html*. Il permet aussi de gérer dynamiquement le DOM de la page.

Saiku-web

Ce module est responsable de tous les services REST de l'application. Touts les services sont construits autour d'URL dans des classes Java. Il n'y a pas de difficulté particulière sur ce module. Les principaux services sont listés ci-dessous.

BasicRepositoryResource2 : Cette ressource regroupe toutes les actions qui sont utiles pour la gestion des rapports. On trouve ainsi les services pour supprimer, renommer, déplacer des rapports ou des répertoires. En ce qui concerne la création, on ne peut pas créer de rapport via *service*, juste les dossiers. La création de rapport dépend du webservice *QueryResource*.

DataSourceResource Cette ressource gère les connections aux différentes bases de données. Ce service a accès aux fichiers de configuration disponible dans le module *webApps*.

EppkSessionResource : Cette ressource sert à récupérer les informations de l'utilisateur connecté de manière à les afficher à l'écran. C'est l'un des rares services à avoir un lien direct avec e-Paprika e-Paprika.

EppkResource : C'est le second service qui a accès à e-Paprika pour récupérer la liste des réseaux. On retrouve aussi dans ce module les dates de dernières mises à jour et les numéros de version de l'application.

OlapDiscoverResource : Cette ressource permet de récupérer toutes les informations sur les cubes qui ont été configurés dans Mondrian. On peut ainsi y trouver la liste des dimensions, des hiérarchies et la description de tous les membres. Il n'y a pas de requête de mise à jour ou de création. En effet, les cubes de Mondrian sont fixés et ne sont pas modifiables par l'utilisateur.

QueryResource : Cette ressource permet toutes les actions sur les requêtes. Toutes les actions que l'on peut faire à l'IHM on une url associée dans cette classe. On pourra ainsi y trouver la création et la sauvegarde de requête, l'ajout et la suppression des dimensions, des mesures ou de filtres ou encore l'exécution en vue d'obtenir un tableau de résultats.

Saiku-service

Aucune des ressources évoquées dans la section précédente ne fait directement de traitement. Chacune d'entre elles délègue les actions à des services. Nous ne faisons pas ici la liste des services associés aux différentes ressources. Cependant, nous expliquons plus en détail le fonctionnement du service qui gère les requêtes.

En effet, ce service est le point central de l'application puisqu'il permet de générer les tableaux. Dans ce service, on utilise une interface *iQuery* qui est implémentée par les objets de requêtes. Les deux objets qui implémentent cette interface sont OlapQuery et MdxQuery. Celui qui nous intéresse ici est l'*OlapQuery* qui sert à gérer les requêtes construites par l'interface. L'objet *MdxQuery* sert pour l'éditeur MDX de saiku qui n'est pas utilisé ou uniquement à des fins de développement.

L'objet OlapQuery est donc l'objet que l'on manipule lorsque l'on veut ajouter ou modifier des comportements dans la requête. Tout comme Mondrian, cet objet utilise l'api Olap4j pour manipuler et générer le MDX. Cet objet se présente comme une surcouche de la requête Olap4J pour accepter les manipulations qui sont faites à l'écran. On trouve ainsi des méthodes qui changent les dimensions d'axes, qui ajoutent ou suppriment des filtres ou des mesures. L'exécution de la requête est quant à elle déléguée à Mondrian. C'est Mondrian qui prend la requête MDX générée dans le service pour la transformer en requêtes SQL et retourner le tableau.

Dans le développement, on essaie de ne pas modifier en profondeur les comportements du logiciel de référence. On préférera ajouter des comportements que modifier ceux existants dans la mesure du possible. Ceci permet de faciliter les montées de versions du logiciel. Un autre point d'attention concerne Mondrian. Initialement, ce projet est une simple dépendance Maven. J'ai du importer le projet pour corriger certains bugs présent dans la version que l'on utilisait. Cependant, cette application a une logique interne très complexe pour générer les requêtes SQL à partir du MDX. J'ai donc

fait le choix de ne pas intervenir sur le code sauf pour corriger des bugs, je n'apporte jamais de modification aux fonctionnalités de Mondrian. Toutes les nouvelles fonctionnalités sont donc intégrées au niveau de Saiku

4.3.2 Les fichiers de configuration

Les fichiers de configuration se trouvent dans le projet *webapps*. Les principaux fichiers de configuration sont utilisés pour les connections aux bases de données, la sécurité ainsi que pour les fichiers *Mondrian*. Ci-dessous, se trouve la liste des principaux fichiers et leur rôle.

applicationContext-spring-security.xml définit la méthode de connexion au logiciel. Dans e-Vision, c'est *Cas* qui est utilisé.

saiku-datasources/eppk définit les paramètres de connexion aux différents magasins de données. C'est un fichier générique qui est utilisé par tous les clients quel que soit le réseau de bibliothèques. La procédure pour traiter ce cas est définie à la section 4.5.2.

eppk.properties définit les paramètres de connexion à la base de données e-Paprika. Ces paramètres sont utilisés pour retrouver la liste des réseaux et les informations de connexion des utilisateurs.

saiku.properties définit des variables utilisées par le projet Saiku. On retrouve ainsi le format des nombres que l'on veut utiliser dans les tableaux et les exports excel. Les *locales* qui définissent la langue utilisée dans le projet sont aussi définies dans ce fichier. La *locale* utilisée dans le projet est fr_Fr pour le français.

mondrian.properties contient toutes les valeurs utile au fonctionnement de mondrian. Ce fichier permet de paramétrer le cache, de fixer un temps après lequel les requêtes s'arrêtent ou encore le nombre de requêtes qui peuvent être exécutées simultanément.

saiku-mondrian/eppk_mondrian.xml est la structure des différents cubes et des liens avec le modèle de données. On retrouve un description de ce fichier dans la section 4.4.4

4.4 Le fonctionnement de Mondrian

4.4.1 Les entités dans le modèle OLAP

Hypercube (cube)

Le cube représente un fait. C'est le contenant principal de toutes les dimensions et des mesures. Le cube a un nom unique dans le schéma.

Dimension

Les dimensions organisent les informations dans un cube, ce sont les axes d'analyse des indicateurs. MDX considère que les dimensions sont mutuellement indépendantes. Une dimension peut contenir plusieurs organisés en hiérarchie. Dans cette section, nous prenons l'exemple de la date de mise à l'inventaire d'un exemplaire. Cette dimension a pour identifiant unique *[dateMiseInventaireOutils1]*. On ne précise pas le nom du cube, en effet, on ne peut pas croiser deux cubes au sein d'une même requête.

Hiérarchie

Les hiérarchies sont contenues dans les dimensions. Une dimension peut contenir plusieurs hiérarchies, l'exemple le plus significatif est la dimension temps. La dimension temps peut être découpée avec une hiérarchie de type *année - mois - semestre - trimestre* et une autre de type *année - semaine - jour* de la semaine. Dans e-Vision, pour des questions de simplification, chaque dimension contient une seule hiérarchie. Dans l'exemple précédent sur les dates, il y a deux dimensions, que l'on nomme outil1 et Outil2. Dès lors, on fait l'amalgame entre les deux dans le projet.

Niveau de hiérarchie (level)

Les niveaux de hiérarchie sont les sous-ensembles des hiérarchies. Dans l'exemple de la dimension temps, on retrouve donc l'année, le semestre ou encore le trimestre dans les niveaux de hiérarchie. Un nom de *level* est unique dans une dimension. L'identifiant unique d'un niveau est donc le nom de la dimension suivi du nom du niveau tel que ceci *[dateMiseInventaireOutils1].[Annee]*.

Membre (member)

Le membre sont les unités des niveaux de dimension. Dans l'exemple précédent, on trouve les membres 2015 pour l'année, *mars* pour le mois. Ce sont les valeurs sur lesquelles on affecte des filtres. On appelle le plus souvent un ensemble de membres via le terme anglais « set ».

Dans le MDX de e-Vision, on utilise trois méthodes pour récupérer des membres dans un niveau. Si l'on reprend l'exemple de l'année de mise à l'inventaire, on peut :

- Sélectionner l'ensemble des membres du niveau avec la fonction *Members* sur le nom unique du niveau. Ainsi avec *[dateMiseInventaireOutils1].[Annee].Members*, on trouve la liste exhaustive de toutes les années présentes dans la dimension.
- Sélectionner un membre en particulier avec *[dateMiseInventaireOutils1].[2015]*
- Sélectionner un ensemble de membres : *[dateMiseInventaireOutils1].[2014] , [dateMiseInventaireOutils1].[2015]*

Dans le cas d'un membre qui appartiendrait à un niveau inférieur tel que le semestre, on applique la fonction *.children* sur un membre du niveau supérieur. Pour sélectionner le premier semestre 2014, on applique la valeur : *[dateMiseInventaireOutils1].[2014].[1er semestre 2014]*

4.4.2 Structure des requêtes MDX

Structure de requête MDX

En étudiant la structure du langage, on peut faire le parallèle avec l'interface et comprendre le fonctionnement interne du logiciel. La structure du langage MDX s'articule autour de 4 instructions clés :

1. Le **SELECT**.
2. Le **FROM**.
3. Le **WHERE**.
4. Le **WITH**.

Le MDX étant un langage de requêtes, il est en effet normal qu'il y ait des similarités avec le langage SQL dans ses mots-clés. Cependant c'est la seule ressemblance que l'on peut trouver, en effet, les deux langages sont totalement différents dans leurs stratégies d'approche et de rendu. Le langage MDX est orienté vers les requêtes sur des cubes. La clause WITH qui sert à créer des membres calculés n'est pas utilisée dans le projet.

La clause SELECT

La clause *SELECT* est sans doute la compliquée à appréhender dans ce langage. En effet, cette clause ne donne pas uniquement les colonnes et les lignes que l'on veut voir apparaître dans le rapport final comme dans le SQL. La sélection est une projection au sens OLAP du terme, c'est à dire que ce *SELECT* donne les cases du cube à rendre, il agit donc en tant que filtre au niveau des données et influence grandement les résultats rendus. Le *SELECT* se divise en deux catégories, les lignes et les colonnes. On peut inclure dans chacune de ces catégories des mesures, des niveaux ou bien des membres.

Il existe cependant certaines restrictions, chaque niveau d'une même hiérarchie doit être sur le même axe (colonne, ou ligne). De la même manière, cette restriction s'applique aussi sur les filtres. Une hiérarchie qui est représenté sur un axe ne peut donc pas figurer dans la clause *WHERE*.

Dans le cas d'insertion de membre ou d'ensemble de membres dans les lignes ou les colonnes, le moteur OLAP opère une projection. On réduit le nombre de données éligibles dans la requête. Cette opération est appelée « slice ».

Le parallèle avec l'interface de la clause « select » est évident. On retrouve bien les axes ligne et colonne. On peut y trouver des mesures, des niveaux de hiérarchie ou des dimensions filtrées.

La clause FROM

La clause from donne le cube sur lequel on exécute la requête MDX. Contrairement à SQL, il n'y a pas possibilité de croiser plusieurs Cubes dans cette clause. Cette restriction est due au modèle OLAP et au principe du décisionnel qui ne prévoient pas de jointure entre les différents cubes.

La clause WHERE

La clause *Where* sert de *Filtre*, c'est à dire qu'elle restreint la requête selon un ou plusieurs membres. On a vu que le *SELECT* sélectionne une partie du cube, le *WHERE* restreint encore plus le périmètre des données sélectionnées pour le rapport final. Cette clause sert donc à filtrer sur des données que l'on ne veut pas voir apparaitre dans le rapport final.

4.4.3 Les fonctions MDX utilisées dans e-Vision

On peut utiliser différentes fonctions dans les lignes et les colonnes des requêtes. Cette section présente les différentes fonctions utilisées dans le projet. Ce n'est pas une liste exhaustive, mais ce sont les fonctions qui ont été rencontrées lors des différents développements.

Fonction *Hierarchize* : Organiser les membres selon une hiérarchie

La fonction *hierarchize* est la fonction de présentation par défaut de Saiku. Elle permet d'organiser les données d'une hiérarchie pour avoir un résultat cohérent. Le langage MDX fait le rapprochement entre ces deux ensembles comme étant de la même dimension et il les organise pour avoir un meilleur rendu final. On observe bien ici une des particularités de MDX, c'est un langage dirigé vers la présentation des rapports.

TABLE 6 – Sélection par hiérarchie

Année de demande	Semestre de demande	Trimestre de demande	Nombre réservations
2014			50
	1er semestre		32
		1er trimestre	11
		2nd trimestre	21
	2nd semestre		18
		3ième trimestre	13
		4ième trimestre	5

Union et crossJoin

La fonction *Union* retourne l'union de deux ensembles en supprimant les doublons. La fonction *Crossjoin* retourne le produit croisé de plusieurs ensembles de données. Grâce à cette fonction, on peut croiser deux dimensions.

La figure 26 montre la requête construite à l'écran qui génère un produit croisé sur deux dimensions, l'année de début de prêts et le genre de l'abonné qui a emprunté. Le premier ensemble croisé est constitué des années de prêts filtrées sur 2014 et 2015. Le genre n'est pas filtré.

FIGURE 26 – Requête de crossJoin à l'IHM

La requête 4.1 montre la requête MDX générée. On observe ici l'utilisation des deux fonctions *Union* et *CrossJoin*. Pour chaque valeur de la première dimension, on exécute un *crossJoin* sur l'ensemble des membres de la seconde. La requête fait ensuite la jointure pour rendre l'ensemble des résultats tel que le montre le tableau 7.

Listing 4.1 – Exemple de requête CrossJoin

```
1 SELECT
2 NON EMPTY {
3     Hierarchize(
4         {[Measures].[Nb loan]}
```

```
5       )} ON COLUMNS,
6   NON EMPTY
7       Hierarchize(
8           Union(
9               CrossJoin(
10                  {[dateDebutOutils].[2015]},
11                  [genre].[sexe].Members
12              ),
13              CrossJoin(
14                  {[dateDebutOutils].[2014]},
15                  [genre].[sexe].Members
16              )
17          )
18      ) ON ROWS
19  FROM [pret]
```

TABLE 7 – Sélection par hiérarchie

Année de prêt	Sexe	Nombre de prêts
2014	femme	35
	Homme	139
2015	femme	5

Les deux fonctions *union* et *crossJoin* sont les deux plus utilisées dans le projet. Ce sont aussi celles qui ont présenté le plus de difficultés lors les différentes améliorations qui ont été apportées au projet.

4.4.4 Le fichier de description de Mondrian

Mondrian a besoin d'un modèle de données de type OLAP pour fonctionner. Ce modèle de données est décrit dans un fichier xml. Ce fichier de configuration fait le lien entre le modèle OLAP décrit dans ce fichier et le modèle relationnel présent dans l'entrepôt de données. On retrouve un extrait du cube des prêt en annexes I.

La racine du fichier est la balise <*Schema*>. Sans modification dans Saiku, cette balise a pour attribut un nom du schéma. Le fichier mondrian est un fichier générique, il doit cependant être décliné pour chaque réseau. C'est pourquoi, dans e-Vision, cette balise ne définit pas un réseau fixe mais utilise une variable. La procédure est expliquée dans la section 4.5.2.

Les cubes

Le schéma contient tous les cubes d'e-Vision. Un cube est associé à une table de fait du modèle de données. Les cubes sont indépendants les uns des autres et ne peuvent pas communiquer entre eux. Chaque cube contient les dimensions et les mesures qui lui sont associées.

Listing 4.2 – Définition d'un cube dans Mondrian

```
<Cube name="Exemplaire"
      defaultMeasure="Nb Exemplaire"
      caption="Exemplaire">
    <Table schema="dm_\%networkid" name="facts_item"/>
                ...
</Cube>
```

Le cube est donc défini par son nom unique, et un « caption ». Le caption est le libellé qui est affiché à l'écran pour l'utilisateur final. Dans e-Vision, on a fait le choix de mettre systématiquement un libellé propre sur chaque entité du schéma. De cette façon, on garde une indépendance entre le coté schéma OLAP et le coté présentation. Comme on peut le voir sur l'XML, le schéma de la table dépend aussi de la variable network (voir la section 4.5.2 pour l'utilisation de cette variable).

La mesure par défaut sert dans deux situations. La première situatio se retrouve lorsque l'on lance une requête MDX sans mesure, le résultat remonte la mesure par défaut. De plus pour faciliter les créations de requêtes pour le service de qualification, lorsque l'on créer une nouvelle requête, on ajoute à l'ouverture cette mesure à l'écran. C'est un exemple de petits développements qui ont été faits pour rendre l'interface plus ergonomique.

Les Dimensions

Les dimension font le lien entre la table de faits et les tables de dimensions. Cela permet à Mondrian de pouvoir créer des requêtes SQL à partir d'une requête MDX. L'extrait 4.3 montre la construction d'une dimension dans Mondrian.

Outre les informations propres à la dimension telles que le nom ou le libellé, on retrouve les liens vers le modèle de données. On retrouve en attribut de la balise <*dimension*> la clé étrangère qui correspond à la colonne de la table de fait qui servira pour la jointure. Comme il a été précisé dans la section 4.4.1, chaque dimension ne contient qu'une seule hiérarchie. Cette hiérarchie inclue la balise <*table*> qui donne le nom de la table ainsi que la clé primaire de celle-ci (attribut *PrimaryKey*).

Les niveaux sont inclus dans la hiérarchie. Dans l'exemple 4.3 il n'y a qu'un seul niveau, cependant on peut en imbriquer plusieurs. Les niveaux donnent la colonne de la table de fait à rendre lors de la requête. La notion de membre unique est une particularité du modèle décisionnel. Si l'on sait qu'une valeur est unique dans une dimension, on place le paramètre *uniqueMembers* à *true*. De cette manière Mondrian va optimiser les requêtes SQL.

Listing 4.3 – Dimension classique dans Mondrian

```
<Dimension name="sitePret"
           caption="Site de pret"
           foreignKey="idLoanLibrary_FK">
    <Hierarchy hasAll="true"
               allMemberName="Tout"
               primaryKey="idLibraryNetwork">
```

```
7           <Table name="dim_library_network"/>
8           <Level name="sitePret"
9                  column="library"
10                 uniqueMembers="true"
11                 caption="Site de pret"/>
12      </Hierarchy>
13  </Dimension>
```

Il existe un autre type de dimension qui ne fait pas appelle à une table spécifique. Elles sont appelées dimensions dégénérées et ne se basent que sur les données de la table de fait. C'est par exemple le cas de la *cotisation principale*, il n'existe pas de table pour cette dimension, elle est directement stockée dans la table de fait des abonnés.

Les Dimensions sur des relations m-n

Le code 4.4 montre la construction d'une dimension m-n dans le fichier de configuration. La section 3.3.4 montre les tables exploitées dans la relation. Chaque enregistrement de la table de fait correspond à plusieurs lignes de la table de dimension *dim_user_classification*. Pour faire cette jointure, on précise à mondrian quelle est la table utilisée pour faire la relation dans l'attribut *facts_loan_user_classification*. Ici, c'est la table *facts_loan_user_classification* qui fait la jointure. On l'appelle la table de fait sans fait.

Dans la balise de jointure <*join*> qui contient les deux tables, on a deux champs qui sont expliqués. La clé de gauche est le champ de la table de jointure, tandis que celle de droite est la clé primaire de la table de la dimension finale.

On remarque ici que la table de bridge qui sert aussi à la relation sur le schéma n'est pas utilisée pour la jointure. Son utilité se situe au niveaux de la construction et de l'intégration du modèle de données. En effet, la table de bridge permet d'avoir un identifiant unique qui est une clé primaire. Cela permet de ne pas avoir de clé étrangère et donc de jointure qui soit sans clé primaire associée.

Listing 4.4 – Dimension sur une relation M-n dans Mondrian

```
1  <Dimension name="abonneclassificationlibelle"
2             foreignKey="idBridgeLoanUserClassification_FK"
3             caption="Nom de la categorie">
4      <Hierarchy  hasAll="true"
5                  allMemberName="Tout"
6                  primaryKey="idBridgeLoanUserClassification_FK"
7                  primaryKeyTable="facts_loan_user_classification">
8          <Join leftKey="idUserClassification_FK"
9                rightKey="idUserClassification">
10             <Table name="facts_loan_user_classification"/>
11             <Table name="dim_user_classification"/>
12         </Join>
13         <Level name="libelleCategorie"
```

```
                table="dim_user_classification"
                column="name" uniqueMembers="false"
                caption="Nom de la categorie de l'abonne"/>
    </Hierarchy>
</Dimension>
```

Les dimensions génériques

Les dimensions génériques sont des dimensions que l'on veut utiliser dans plusieurs cubes. L'utilisation de ces dimensions se fait en deux temps. Dans un premier temps, on déclare la dimension en dehors de tous les cubes. C'est la même déclaration qu'une dimension classique à l'exception de l'attribut de clé étrangère (*foreignKey*) qui n'apparaît pas.

L'exemple 4.5 donne l'utilisation d'une telle dimension. Cette balise se trouve dans un cube. L'attribut *source* de cette balise doit être le nom de la dimension générique que l'on veut utiliser. L'attribut *name* est le nom de la dimension dans le cube. L'attribut foreignKey permet de définir le nom de la colonne qui servira pour faire la jointure avec la dimension générique.

Listing 4.5 – Utilisation d'une dimension générique

```
<DimensionUsage name="dateCreationDocOutils1"
                source="dateCreationDocOutils1"
                foreignKey="idBibRecord_FK"/>
```

Les cubes virtuels

En MDX il n'est pas possible de croiser deux cubes comme pourrait le faire une requête SQL. Cependant, il a été formulé le besoin dans e-Vision de pouvoir calculer un taux de rotation. Le taux de rotation correspond au ratio entre le nombre de prêts sur une période donnée et le nombre d'exemplaires pouvant être empruntés sur cette même période. Ce taux de rotation doit pouvoir être calculé sur toutes les informations présentes dans l'exemplaire ou le prêt. Par exemple, on trouve parmi les taux les plus consultés, le taux de rotation par classification ou par public visé.

Pour faire ce croisement, Mondrian propose des cubes virtuels qui sont l'association de deux cubes « physiques ». Pour le taux de rotation, on fusionne donc le cube des prêts avec celui des exemplaires. Le XML 4.6 montre la structure d'un cube virtuel. Cet exemple ne comprend pas l'ensemble des dimensions et des mesures, il sert juste a expliquer la structure utilisée.

Listing 4.6 – Description d'un cube virtuel

```
<VirtualCube name="Taux de rotation"
             defaultMeasure="Taux de rotation"
             caption="Taux de rotation (activite)">
    <CubeUsages>
        <CubeUsage cubeName="Exemplaire"
                   ignoreUnrelatedDimensions="true" />
        <CubeUsage cubeName="pret"
```

```
                ignoreUnrelatedDimensions="true" />
    </CubeUsages>

    <VirtualCubeDimension cubeName="pret"
                          name="dateDebutOutils"
                          caption="Pret - Date de pret-Outils"/>
    <VirtualCubeDimension cubeName="pret"
                          name="dateDebutScolaire"
                          caption="Pret - Date de pret-Scolaire"/>

    <VirtualCubeDimension cubeName="Exemplaire"
                          name="publicViseExemplaire"
                          caption="Ex - Public Vise" />
    <VirtualCubeDimension cubeName="Exemplaire"
                          name="siteacquereur"
                          caption="Ex - Acquereur" />

    <VirtualCubeMeasure cubeName="Exemplaire"
                        name="[Measures].[Nb Exemplaire]"
                        visible="false"
                        caption="Nombre d'exemplaires"/>
        <VirtualCubeMeasure cubeName="pret"
                        name="[Measures].[Nb loan]"
                        visible="false"
                        caption="Nombre de prets"/>

    <CalculatedMember name="Taux de rotation" dimension="Measures">
        <Formula>
            [Measures].[Nb loan] / [Measures].[Nb Exemplaire]
        </Formula>
    </CalculatedMember>

</VirtualCube>
```

On créé donc un cube virtuel avec la balise *virtualCube*. La balise *CubeUsages* permet d'identifier les cubes que l'on veut croiser. Ici, on ajoute le cube exemplaire et le cube des prêts.

Les balises *VirtualDimension* permettent d'importer les dimensions des cubes physiques dans le cube virtuel. De la même manière, la balise *VirtualCubeMeasure* permet d'importer les mesures. On définit le nom du cube d'origine pour chacune d'elle.

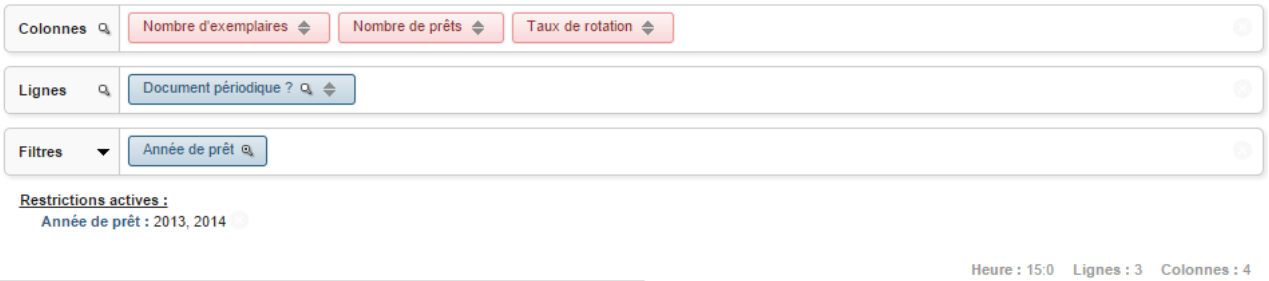
Si une dimension existe dans les deux cubes physiques (i.e. porte le même nom), les filtres appliqués à cette dimension sont effectifs pour les deux cubes. Si la dimension n'apparaît que dans un seul des deux cubes, alors seul ce cube est filtré, l'autre cube ignore le filtre. Pour illustrer ce mécanisme, la figure 27 montre le taux de rotation sur la dimension *document Périodique ?*. Cette dimension est

présente dans les deux cubes physiques *prêt* et *exemplaire*. On observe sur le tableau de la figure 27 que chaque ligne est correctement filtrée sur la dimension.

Document périodique ?	Taux de rotation	Nombre de prêts	Nombre d'exemplaires
Non	0,119	319	2 675
Oui	0,052	13	248

FIGURE 27 – Exemple de requête sur le cube virtuel du taux de rotation

Sur la requête de la figure 28, on ajoute un filtre sur l'année de prêt. Cette dimension n'apparaît que dans le cube des prêts. On observe donc que l'on garde le même total pour les exemplaires, mais que le nombre de prêts a diminué car le filtre a été appliqué.



Document périodique ?	Taux de rotation	Nombre de prêts	Nombre d'exemplaires
Non	0,097	260	2 675
Oui	0,052	13	248

FIGURE 28 – Exemple de requête sur le cube virtuel du taux de rotation

Le paramètre *ignoreUnrelatedDimensions* permet d'ignorer les dimensions qui sont dans un cube mais pas dans l'autre. Par exemple, la date de prêt n'est pas dans le cube des exemplaires. Lorsque l'on filtre dessus, il n'y a pas de correspondance dans le cube exemplaire. Cependant, Mondrian ne retourne pas 0 pour la mesure exemplaire, il remonte le nombre total d'exemplaires.

4.4.5 Les requête sauvegardées

Structure d'une requête

Après avoir élaborer une requête, Saiku donne la possibilité de la sauvegarder dans une arborescence. On appelle ces sauvegardes des « rapports ». Les rapports sont sauvegardés dans des fichiers xml qui reprennent la structure de la requête MDX et donc de l'interface.

Listing 4.7 – Structure du xml d'un rapport dans e-Vision

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```
<Query name="2666706A-4136-1155-BF31-DAE2DDAE9D99"
       type="QM"
       connection="NomResau"
       cube="[Taux de rotation]"
       catalog="9182904506546534672"
       schema="9182904506546534672">
    <QueryModel>
        <Axes>
            <Axis location="ROWS" nonEmpty="true">
                    <Dimensions> ... </Dimensions>
            </Axis>
            <Axis location="COLUMNS" nonEmpty="true">
                <Dimensions> ...  </Dimensions>
            </Axis>
            <Axis location="FILTER" nonEmpty="false">
                <Dimensions> ... </Dimensions>
            </Axis>
        </Axes>
    </QueryModel>
    <MDX> ... </MDX>
    <Properties> ... </Properties>
    <Description>
        <Title>titre </Title>
        <Comment>commentaire </Comment>
    </Description>
</Query>
```

On peut observe dans cet extrait du fichier xml que les attributs de la balise « query » donnent toutes les informations pour se connecter au datamart. On retrouve ensuite le détail de toutes les dimensions et des mesures par axes dans les balises « Axes » et « Dimension ». Dans l'extrait, le détail des dimensions a été enlevé pour des questions de présentation car la structure est très verbeuse. On retrouve un exemple plus complet en annexe II.

La balise « MDX » permet de récupérer la requête MDX associée à la structure en cours. Cette balise est présente uniquement à titre informatif et à des fins de debuggage, elle n'est pas utilisée lors de l'ouverture de rapport. La balise « Properties » regroupe les informations utiles à Saiku lors de l'ouverture de la requête. On trouve par exemple les paramètres pour savoir si :

1. la requête doit être exécutée à l'ouverture ou non
2. on doit avoir des mesures vides ou remplir artificiellement les cases avec des 0
3. on affiche ou non les filtres actifs à l'écran.

La balise description permet de sauvegarder un titre et un commentaire sur la requête. C'est un développement spécifique d'e-Vision qui permet d'écrire une aide pour les rapports

Dans e-Vision, il existe deux types de rapports, les rapports communs à tous les réseaux de bibliothèques, et ceux spécifiques à chaque client. La figure 29 donne l'arborescence des répertoires telle qu'elle apparaît pour un super-administrateur.

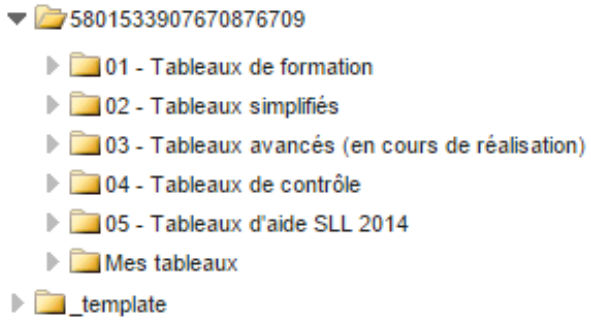


FIGURE 29 – Arborescence des rapports vue par le super-administrateur

Les fichiers spécifiques à chaque client sont stockés dans un répertoire « Mes Tableaux ». Les tableaux génériques sont élaborés sur la plateforme de qualification dans le répertoire nommé « _-template ». Chaque matin, toutes les autres plateformes synchronise leur dossier « _template » avec celui de la qualification. Ensuite, un job Talend va copier dans le répertoire de chaque client, les rapports en modifiant les attributs de la balise « Query » pour y rentrer les informations spécifiques au client.

4.5 Principaux développements

Cette section donne une liste des principaux développements qui ont été faits sur la partie reporting d'e-Vision. Cette liste n'est pas exhaustive, mais concentre les plus gros apports.

4.5.1 Montée de version

La principale tâche que j'ai eu à accomplir a été la montée de version de Saiku. Nous sommes passé de la version 2.4 à la 2.5. Les principales motivations pour la montée de versions était de garder à jour le logiciel pour garantir sa stabilité d'une part. D'autre part, les fonctionnalités ajoutées étaient très importantes pour le fonctionnement au quotidien de Saiku. Les deux plus gros apports ont été la possibilité de rechercher les 10 résultats les plus grands pour un tableau et la possibilité de faire des tris qui respectent la hiérarchie des dimensions. Cette version apportait aussi beaucoup de petites améliorations sur l'ergonomie du logiciel.

La stratégie a été de reprendre toutes les fonctionnalités développées dans *Youtrack* et de les reporter une par une dans la nouvelle version. Faire un différentiel massif des modifications n'était pas pertinant. En effet, le code de l'application avait changé et certaines améliorations n'étaient pas compatible avec le nouveau code. De plus, cette approche a permis d'optimiser certaines fonctionnalités, de les adapter au nouveau produit.

4.5.2 Organisation par réseau de bibliothèques

Créer dynamiquement des connexions

Il existe normalement dans Saiku, un fichier de connexion par magasin de données. Cependant, dans notre cas, on ne veut pas avoir à créer un fichier de connexion et un fichier de configuration Mondrian pour chaque client. Il a donc fallu trouver un mécanisme qui permette de simuler les fichiers dynamiquement. Dans le fichier de configuration de Spring qui sert à charger les dépendances bean.xml de Saiku, on retrouve la configuration 4.8. Cette configuration indique que la classe *EppkClassPathResourceDatasourceManager* est utilisée pour gérer le chargement des fichiers de connexion aux bases de données.

Listing 4.8 – Paramétrage pour le chargement des connexions aux datamarts

```
<bean id="classpathDsManager"
      class="org.saiku.service.EppkClassPathResourceDatasourceManager">
    <property name="path" value="res:saiku-datasources"/>
</bean>
```

Cette classe récupère l'ensemble des réseaux de bibliothèques éligibles aux statistiques dans e-Paprika. Pour chacun de ces réseaux, elle retrouve l'identifiant unique et remplacer la chaine de caractères « [NETWORK] » par celui-ci dans le fichier de connexion décrit dans l'extrait de code 4.9. Ainsi, dans la chaîne de connexion, *dm_[NETWORKID]* devient *dm_9182904506546534672* ce qui permet de se connecter à la base du client dont l'identifiant est 9182904506546534671.

Listing 4.9 – Connection aux bases de magasins de données

```
type=OLAP
name={[NETWORKID]}
driver=mondrian.olap4j.MondrianOlap4jDriver
location=jdbc:mondrian:Jdbc=jdbc:mysql://webeppktrunk.sa.decalog.net
/dm_[NETWORKID]?useUnicode=true&amp;
DynamicSchemaProcessor=net.decalog.eppk.EppkDynamicSchemaProcessor;
UseContentChecksum=true;
characterEncoding=UTF-8;
Catalog=res:saiku-mondrian/[NETWORKID].xml;
JdbcDrivers=com.mysql.jdbc.Driver;
username=root
password=
```

Simuler les fichiers Mondrian pour chaque réseau

Ce mécanisme permet donc de ne pas avoir un fichier de connexion pour chaque client. Cependant, il faut aussi créer dynamiquement les fichiers *Mondrian*. En effet, le fichier de connexion 4.9 indique *res :saiku-mondrian/[NETWORKID].xml* en tant que catalogue. Avec le traitement décrit dans le paragraphe précédent, on obtient par exemple *res :saiku-mondrian/9182904506546534672.xml*,or le fichier *saiku-mondrian/9182904506546534672.xml* n'existe pas physiquement.

Pour pouvoir pallier à ce problème, Saiku offre un mécanisme de chargement dynamique des schémas. Ce mécanisme est activé via le paramètre *DynamicSchemaProcessor* dans le fichier de connexion. On attribue à ce paramètre une classe Java qui va permettre ce chargement dynamique. La classe utilisée dans e-Vision se nomme *EppkDynamicSchemaProcessor*. Elle permet de prendre le fichier *Mondrian* décrit dans la section 4.4.4 et de remplacer les chaînes de caractères *%networkid* par l'identifiant du réseau. De cette manière, les schémas sont associés à la bonne base de données et un fichier <*identifiant de réseau*>*.xml* sera simulé.

4.5.3 Les filtres non actifs

Les filtres non-actifs sont les filtres blancs. On peut observer un de ces filtres sur la figure 27. L'année de prêt apparaît en blanc parce qu'il n'y a pas de filtre actif appliqué à la dimension. Ce n'est pas une fonctionnalité native de Saiku.

La stratégie utilisée pour arriver à ce résultat est de modifier la requête juste avant de la faire exécuter par Mondrian. Avant d'être traité par Mondrian, l'objet de la requête contient tous les nœuds qui forment une requête MDX. La stratégie a été de sortir ces filtres de tous les nœuds concernés avant l'exécution de la requête. De cette manière, on envoie une requête modifiée sans les filtres et on ne touche pas à Mondrian. C'est l'un des principes de développement d'e-Vision. On essaie de ne pas modifier son comportement et sa logique interne, sauf pour corriger des bugs.

4.5.4 Filtres « inférieur à » et « supérieur à »

Pouvoir faire une requête qui comprend des critères « inférieur à » et « supérieur à » semble trivial surtout lorsque l'on raisonne en SQL. Cependant, ce n'est pas une fonctionnalité de Saiku et Mondrian ne la gère pas nativement. De plus, parmi les membres d'une dimension, Mondrian gère mal la notion d'ordre. Cette tache a donc été dévolue à l'objet OlapQuery de Saiku. On recherche toutes les valeurs de la dimension qui correspondent au critère et on les ajoute en tant que filtres dans la requêtes. De cette manière, la recherche des valeurs concernées par la clause de filtres sont de la responsabilité du code spécifique à e-Vision. De plus Mondrian reste dans son domaine fonctionnel.

4.6 Synthèse

Ce dernier chapitre décrit les grands principes des moteurs OLAP et plus particulièrement de Mondrian que l'on utilise dans e-Vision. On y trouve aussi une description du langage MDX qui est le langage spécifique au solution de reporting. On peut ainsi observer que toute la structure logiciel depuis l'interface jusqu'au fichier de sauvegarde des requêtes est très proche de la structure du langage MDX. La solution permet donc de présenter une interface qui permet à l'utilisateur de manipuler des données de manière simple et intuitives.

Conclusion

e-Vision a atteint son but premier en se positionnant comme le module statistique de e-Paprika. Le logiciel permet de remplir les rapports du Service du Livre et de la Lecture (SLL). De plus, il propose une interface suffisamment riche pour que les clients puissent saisir les tableaux qui les intéressent au quotidien sans avoir de notion d'informatique ou de statistiques. Les fonctionnalités ajoutées à l'interface en font un logiciel complet. Il existe ainsi peu de demande d'amélioration en terme de fonctionnalités.

Cependant, certains choix techniques pourraient être remis en question dans un futur proche. C'est par exemple le cas du moteur MyIsam que l'on pourrait remplacer aussi bien pour l'entrepôt que pour les magasins de données. Une des autres améliorations possibles serait d'ajouter les tests unitaires pour éviter les régressions lors des phases futures de maintenance applicative. Il reste donc une phase de stabilisation à faire pour que le projet arrive réellement à maturité. Cette maturité est nécessaire si l'on veut pouvoir inclure d'autres sources de données dans e-Vision. En effet, l'architecture utilisée permettrait d'insérer les données d'autres logiciels développés par Décalog. Les statistiques pourraient ainsi rapprocher les activités du portail professionnel et celles du portail public pour proposer une vue d'ensemble aux clients.

Ce projet m'a permis d'aborder la chaîne complète d'un système décisionnel, de la construction de l'entrepôt de données jusqu'à la restitution des rapports à l'utilisateur final. Bien que je n'ai pas initialisé le projet, j'ai travaillé sur chacun des éléments. J'ai ainsi pu intervenir sur l'optimisation de la construction de l'entrepôt, des nouveaux faits et de la maintenance applicative pour la partie entrepôt de données. De plus, la montée de version de Saiku, m'a permis de monter en compétence sur le produit et de comprendre son fonctionnement interne. D'un côté, le fait d'être le seul intervenant technique sur le projet e-Vision m'a aussi donné une grande autonomie dans mon travail. D'un autre côté, le fait de travailler en étroite collaboration avec l'équipe e-Paprika m'a permis de me greffer sur un projet en développement Agile et de garder un lien avec le travail en équipe.

Ma formation au CNAM notamment les unités d'enseignements de spécialisation en système décisionnel m'ont grandement aidé à prendre de la hauteur sur le projet. J'ai ainsi pu mettre en pratique mes connaissances théoriques sur les entrepôts de données, le modèle OLAP ou encore le MDX. De plus, sur le plan professionnel, ce projet m'a permis d'avoir une réelle première expérience dans les systèmes décisionnels. Je souhaiterais à l'avenir approfondir ces connaissances, d'une part pour traiter de la partie exploitation des données suivant des modèles statistiques (data-mining) d'autre part pour continuer dans l'intégration de données en y ajoutant une plus grande dimension sur la consolidation (utilisation d'ontologie par exemple).

Bibliographie

[1] WIKIPEDIA, « Multidimensional expressions **[en ligne]** », Janvier 2014. Disponible sur <http ://fr.wikipedia.org/wiki/ >(consulté le 11/03/2015).

[2] R. KIMBALL et M. ROSS, *The Data Warehouse Toolkit : The Complete Guide to Dimensional Modeling*. New York, NY, USA : John Wiley & Sons, Inc., 2nd éd., 2002.

[3] Y. GRIM, « Olap, les fondamentaux **[en ligne]** », Janvier 2014. Disponible sur <http ://-grim.developpez.com/articles/concepts/olap/ >(consulté le 11/03/2015).

[4] E. METAIS, « Encyclopædia universalis : SystÈmes informatiques - systèmes d'aide à la décision **[en ligne]** », 2015. Disponible sur <http ://www.universalis.fr/encyclopedie/systemes-informatiques-systemes-d-aide-a-la-decision/ >(consulté le 11/03/2015).

[5] WIKIPEDIA, « Datamart **[en ligne]** », 2015. Disponible sur <http ://fr.wikipedia.org/wiki/Datamart >(consulté le 11/03/2015).

[6] QUARTETFS, « Mdx query basics and usage exampl **[en ligne]** », Janvier 2014. Disponible sur <http ://quartetfs.com/en/mdx-query-basics-and-usage-example >(consulté le 11/03/2015).

[7] PENTAHO, « Mondrian documentation **[en ligne]** », Mai 2011. Disponible sur <http ://mondrian.pentaho.com/documentation/mdx.php >(consulté le 11/03/2015).

[8] J.-P. RIEHL, « 10 requêtes mdx utiles **[en ligne]** », Janvier 2011. Disponible sur <http ://labs.bewise.fr/Article/10-requetes-MDX-utiles/ >(consulté le 11/03/2015).

[9] K. AOUICHE, « Mdx **[en ligne]** », 2013. Disponible sur <http ://eric.univ-lyon2.fr/ kaouiche/inf9002/mdx.html >(consulté le 11/03/2015).

[10] TALEND, « Documentation talend **[en ligne]** », 2014. Disponible sur <https ://-help.talend.com/display/HOME/Talend+Open+Studio+for+Data+Integration >(consulté le 11/03/2015).

[11] T. BARBER, « Documentation saiku 2.x **[en ligne]** », Aout 2014. Disponible sur <http ://-wiki.meteorite.bi/display/SAIK/Saiku+2.x+Documentation >(consulté le 11/03/2015).

[12] T. DASU et T. JOHNSON, *Exploratory Data Mining and Data Cleaning*. New York, NY, USA : John Wiley & Sons, Inc., 1 éd., 2003.

[13] WIKIPEDIA, « Système intégré de gestion de bibliothèque **[en ligne]** », 2015. Disponible sur <http ://fr.wikipedia.org/wiki/Syst>(consulté le 11/03/2015).

[14] ENSSIB, « Qu'est ce qu'un fonds documentaire ? **[en ligne]** », 2013. Disponible sur <http ://www.enssib.fr/content/quest-ce-quun-fonds-documentaire >(consulté le 11/03/2015).

[15] WIKIPEDIA, « Base de données relationnelle **[en ligne]** », 2014. Disponible sur <http ://fr.wikipedia.org/wiki/Base_de_données_relationnelle >(consulté le 11/03/2015).

Table des figures

Liste des tableaux

Annexes

I Extrait du cube des prêts dans le fichier Mondrian

```
<Cube name="pret" defaultMeasure="Nb loan"
      caption="Prêt - retour (activité)">
  <Table schema="dm_%networkid" name="facts_loan"/>
    <!--  *** Dimension Adresse *** -->
  <DimensionUsage name="rue" source="rue"
                  foreignKey="idAddress_FK"/>
  <DimensionUsage name="zoneGeographique" source="zoneGeographique"
                  foreignKey="idAddress_FK"/>
  <DimensionUsage name="ville" source="ville"
                  foreignKey="idAddress_FK"/>
  <DimensionUsage name="district" source="district"
                  foreignKey="idAddress_FK"/>
    <!--  *** Dimension Abonné *** -->
  <DimensionUsage name="codebarres"
                  source="codeAbonne"
                  foreignKey="idPublicuser_FK"/>
  <DimensionUsage name="genre" source="genre"
                  foreignKey="idPublicuser_FK"/>
  <DimensionUsage name="dateNaiss" source="dateNaiss"
                  foreignKey="idPublicuser_FK"/>
  <DimensionUsage name="typeAbonne" source="typeAbonne"
                  foreignKey="idPublicuser_FK"/>

  <Dimension name="HeureDebutPretOutils"
             foreignKey="startDateHour_FK"
             caption="Heure de prêt-Outils">
    <Hierarchy hasAll="true"
               allMemberName="Tout"
               primaryKey="idHour">
      <Table name="dim_hour" />
      <Level name="trancheHoraireJour"
             ordinalColumn="rangeHalfDayOrdinal"
             column="rangeHalfDayLabel"
             uniqueMembers="false"
             caption="Tranche horaire (1/2 journée) de prêt"/>
      <Level name="trancheHoraireHeure"
             ordinalColumn="rangeHourOrdinal"
```

```
            column="rangeHourLabel" uniqueMembers="false"
            caption="Tranche horaire (heure) de prêt"/>
    <Level name="trancheHoraireDemiheure"
            ordinalColumn="rangeHalfHourOrdinal"
            column="rangeHalfHourLabel"  uniqueMembers="false"
            caption="Tranche horaire (1/2 heure) de prêt"/>
  </Hierarchy>
</Dimension>

<Dimension name="dateRetour"
            foreignKey="returnDate" caption="Date de retour">
  <Hierarchy hasAll="true" allMemberName="Tout"
              primaryKey="idDate">
    <Table name="dim_time" />
    <Level name="retour"
            column="dateType"
            ordinalColumn="fullDate"
            uniqueMembers="false"
            caption="Date de retour"/>
  </Hierarchy>
</Dimension>

<Dimension name="dateRetourOutils" foreignKey="returnDate"
          type="TimeDimension" caption="Date de retour-Outils">
  <Hierarchy hasAll="true" allMemberName="Tout" primaryKey="idDate" >
    <Table name="dim_time"/>
    <Level name="Annee" caption ="Année de retour"
            column="year" uniqueMembers="false" levelType="TimeYears"/>
    <Level name="Semestre" caption="Semestre de retour"
            column="halfyear_year" uniqueMembers="false"
            levelType="TimeHalfYears"/>
    <Level name="Trimestre" caption="Trimestre de retour"
            column="quarter_year" uniqueMembers="false"
            levelType="TimeQuarters"/>
    <Level name="Mois" caption="Mois de retour"
            column="month_year"
            uniqueMembers="false" levelType="TimeMonths"
            ordinalColumn="month"/>
    <Level name="Jour" caption="Jour du mois de retour"
            column="dateType" uniqueMembers="false"
            levelType="TimeDays" ordinalColumn="fullDate"/>
  </Hierarchy>
</Dimension>

<Dimension name="dateRetourOutils2" foreignKey="returnDate"
            type="TimeDimension"
            caption="Date de retour-Outils2">
  <Hierarchy  hasAll="true" allMemberName="Tout"
  primaryKey="idDate" >
    <Table name="dim_time"/>
```

```
      <Level name="Annee" caption ="Année de retour (2)"
             column="year" uniqueMembers="false"
             levelType="TimeYears"/>
      <Level name="Semaine" caption="Semaine de retour (2)"
             column="weekLabel" ordinalColumn='week'
             levelType="TimeWeeks" uniqueMembers="false"/>
      <Level name="Jour" caption="Jour de la semaine de retour (2)"
             column="dayOfWeekLabel" ordinalColumn='dayOfWeek'
             levelType="TimeDays" uniqueMembers="false"/>
    </Hierarchy>
  </Dimension>


  <Measure name="Nb loan" column="nbLoan"
           aggregator="sum"
           caption="Nombre de prêts" />
  <Measure name="Nb retour"
           column="nbLoan" aggregator="sum"
           caption="Nombre de retours" />
  <Measure name="Nb pret distinct" column="_idLoanSIGB"
           aggregator="distinct-count"
           caption="Nombre de prêts (distincts)" />
  <Measure name="Nb retour distinct"
           aggregator="sum"
           caption="Nombre de retours (distincts)">
    <MeasureExpression>
      <SQL dialect="generic">
        (if ( `facts_loan`.`returnDate` != 0,  1, 0 ))
        </SQL>
    </MeasureExpression>
  </Measure>
  <Measure name="Nb abonne distinct" column="idPublicuser_FK"
           aggregator="distinct-count"
           caption="Nombre d'abonnés actifs (distincts)" />
  <Measure name="Nb doc distinct" column="idBibRecord_FK"
           aggregator="distinct-count"
           caption="Nombre de documents actifs (distincts)" />
  <Measure name="Nb ex distinct"
           column="idItem_FK" aggregator="distinct-count"
           caption="Nombre d'exemplaires actifs (distincts)" />
  <Measure name="Nb prolongation" column="nbExtension"
           aggregator="sum" caption="Nombre de prolongations" />
</Cube>
```

II Structure complète d'une requête sauvegardée

```
<?xml version="1.0" encoding="UTF-8"?>
<Query name="2666706A-4136-1155-BF31-DAE2DDAE9D99" type="QM"
       connection="Argentat" cube="[Taux de rotation]"
       catalog="9182904506546534671" schema="9182904506546534671">
 <QueryModel>
  <Axes>
   <Axis location="ROWS" nonEmpty="true">
    <Dimensions>
     <Dimension name="docPeriodical" hierarchizeMode="PRE"
                hierarchyConsistent="true">
      <Inclusions>
       <Selection dimension="docPeriodical"
             node="[docPeriodical].[estPeriodique]"
             operator="MEMBERS"
             type="level" />
      </Inclusions>
      <Exclusions />
     </Dimension>
    </Dimensions>
   </Axis>
   <Axis location="COLUMNS" nonEmpty="true">
    <Dimensions>
     <Dimension name="Measures">
      <Inclusions>
       <Selection dimension="Measures"
             node="[Measures].[Taux de rotation]"
             operator="MEMBER"/>
      </Inclusions>
      <Exclusions />
     </Dimension>
    </Dimensions>
   </Axis>
   <Axis location="FILTER" nonEmpty="false">
    <Dimensions>
     <Dimension name="dateDebutOutils">
      <Inclusions>
       <Selection dimension="dateDebutOutils"
                  node="[dateDebutOutils].[2013]"
                  operator="MEMBER" type="member"
                  level="[dateDebutOutils].[Annee]"
                  isInComposedDim="false" />
       <Selection dimension="dateDebutOutils"
                  node="[dateDebutOutils].[2014]" operator="MEMBER"
                  type="member"
                  level="[dateDebutOutils].[Annee]"
                  isInComposedDim="false" />
      </Inclusions>
      <Exclusions />
```

```
     </Dimension>
    </Dimensions>
   </Axis>
  </Axes>
 </QueryModel>
 <MDX>SELECT&#xD;
NON EMPTY {[Measures].[Taux de rotation],
           [Measures].[VirtualExemplaire],
           [Measures].[VirtualPret]} ON COLUMNS,
NON EMPTY {Hierarchize({[docPeriodical].[estPeriodique].Members})} ON ROWS
FROM [Taux de rotation]
WHERE {[dateDebutOutils].[2013], [dateDebutOutils].[2014]}</MDX>
 <Properties>
  <Property name="org.saiku.query.explain" value="true" />
  <Property name="saiku.olap.query.toggle_fields" value="true" />
  <Property name="saiku.olap.query.nonempty.columns" value="true" />
  <Property name="saiku.olap.query.nonempty.rows" value="true" />
  <Property name="saiku.olap.query.fill_table" value="false" />
  <Property name="org.saiku.connection.scenario" value="false" />
  <Property name="saiku.olap.query.automatic_execution" value="false" />
  <Property name="saiku.olap.query.toggle_filter" value="true" />
  <Property name="saiku.olap.query.drillthrough" value="true" />
  <Property name="saiku.olap.query.filter" value="true" />
  <Property name="saiku.olap.query.limit" value="true" />
  <Property name="saiku.olap.query.nonempty" value="true" />
 </Properties>
 <Description />
</Query>
```

III Extrait du rapport SLL

C - Accès et installations

C1 - Accès et installations

Nombre d'heures d'ouverture hebdomadaires tous publics		C101	Nombre de jours d'ouverture annuels		C102
Nombre de places assises disponibles		C103	Nombre d'heures d'ouverture annuelles		C113
Nombre de postes informatiques publics		C105	avec accès internet		C106
Nombre de postes informatiques professionnels		C107	avec accès internet		C108

C2 - Catalogue

Le catalogue de la bibliothèque est-il informatisé ?	Oui ○	Non ○		C201
Nom et version du système de gestion de la bibliothèque				C206

C3 - Surfaces des locaux

Surface utile nette totale en m²		C301

D - Collections

D1 - Imprimés

Livres imprimés	Fonds (hors réseau BDP)		Acquisitions		Éliminations	
Adultes		D101		D102		D103
Enfants		D116		D117		D118
Total		D128		D129		D130
dont nombre de dons (ou autres acquisitions à titre gracieux, ...)				D131		
Autres documents (partitions, cartes, ...)		D321		D322		D320

Suppr D135 à D137	Adultes		Enfants		Total	
Nombre d'abonnements en cours		D201		D203		D211

D4 - Documents audiovisuels

	Fonds		Acquisitions		Éliminations	
Documents sonores		D409		D410		D418
Documents vidéo		D411		D412		D419
Éliminations des documents sonores et vidéo						D415
Documents numériques (cdroms)		D517		D518		

E - Usages et usagers de la bibliothèque

E1 - Usagers

	Inscrits actifs		Nouveaux inscrits		Emprunteurs actifs	
Particuliers						
Enfants (de 0 à 14 ans)		E110		E111		E112
Adultes (de 15 à 64 ans)		E119		E120		E121
Adultes (de 65 ans et plus)		E128		E129		E130
Total adultes		E137		E138		E139
Total		E101		E102		E103
dont résidents dans la commune ou les communes adhérentes au réseau		E140		E141		E142
Collectivités				E143		E144
Fréquentation : entrées dans l'établissement		E147				

E2 - Prêts

	Adultes		Enfants		Total		Prêt aux collectivités	
Livres		E201		E202		E203		E204
Publications en série imprimées		E205		E206		E207		E208
Documents sonores		E233		E234		E235		E236
Documents vidéo		E217		E218		E219		E220
Autres documents		E221		E222		E223		E224
Total		E237		E238		E239		E240

E5 - Services et ressources électroniques proposés par la bibliothèque

Site internet de la bibliothèque	☐	E501	Nombre de sessions	Dans la bibliothèque		E502
				Hors de la bibliothèque		E503
Catalogue en ligne	☐	E504	Nombre de sessions	Dans la bibliothèque		E505
				Hors de la bibliothèque		E506
Ressources électroniques	☐	E515	Nombre total de contenus téléchargés			E516
Bases de données	☐	E527	Nombre de sessions			E528
Accès à internet dans la bibliothèque	☐	E529	Nombre de sessions			E530

« E-Vision : Élaboration d'une solution de reporting pour un système intégré de gestion de bibliothèque »

Mémoire présenté en vue d'obtenir
Le DIPLOME D'INGENIEUR CNAM
SPECIALITE : Informatique
OPTION : Systèmes d'information (ISI)
Lyon, 2015

RESUME

Ce rapport traite de l'élaboration de la solution logicielle e-Vision développée par Decalog pour ajouter un outil de reporting sur son SIGB e-Paprika. Le projet s'appuie sur les méthodes agiles. La solution proposée utilise les principaux outils des systèmes d'information décisionnels.

Ce mémoire aborde les principales méthodes de création des cubes OLAP et la stratégie de construction des magasins de données implémentées grâce à l'ETL Talend. L'intégration de la solution Saiku en tant qu'outil de reporting y est aussi expliquée ainsi que les points d'améliorations possibles.

Mots clés : Système d'information décisionnel,Système Intégré de Gestion de Bibliothèque, ETL, OLAP, Entrepôt de données, Magasins de Données, Saiku.

SUMMARY

This report deals with the development of e-Vision, a software solution developed by Decalog in order to add a reporting tool on its Integrated Library System e-Paprika. The project is based on agile methods. The proposed solution uses the main tools of decision-making information system.

This dissertation explains the main methods used for OLAP cube creation and for datamarts construction implemented using ETL Talend. Integrating Saiku solution as reporting tools is also explained as well as the development of possible improvements.

Key words : decision-making information system, Integrated Library System, ETL, OLAP, Datawarehouse, Datamart, Saiku.