ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

RAPPORT DE PROJET PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE À
L'OBTENTION DE MAÎTRISE AVEC PROJET EN GÉNIE LOGICIEL

Par
YOUSSEF RHINDI

OUTIL D'IMPORT MASSIF DOCUMENTUM

MONTRÉAL, LE 00 XXXX 2019

REMERCIEMENTS

Tout d'abord, je suis profondément reconnaissant au professeur Alain April d'avoir accepté de diriger ce projet appliqué de 15 crédits à la maîtrise, et pour le soutien, les retours rapides, et les encouragements sans fin. Son travail acharné durant mes cours de maîtrise a été une source d'inspiration et de motivation, et il restera toujours un modèle pour moi.

Je tiens à remercier le département de génie logiciel et des TI ainsi qu'à l'école de technologie supérieure (ÉTS) pour m'avoir fourni les ressources et les possibilités de développement professionnel pendant mes études.

Je tiens également à remercier Peter Vergados de m'avoir accueilli dans son équipe, m'offrir ce sujet de maîtrise, et pour ses conseils constants dans des horaires chargés. J'ai été beaucoup inspiré de mes talentueux collègues de l'équipe CCS CMS Développer de McGill durant la réalisation du projet.

Enfin, je voudrais remercier les membres de ma famille pour leur soutien dans mes études et ma carrière.

OUTIL D'IMPORT MASSIF DOCUMENTUM

Youssef RHINDI

RÉSUMÉ

Un système de gestion de contenu peut faire référence à tout processus ou système permettant à une personne ou à une organisation de gérer du contenu (dans ce cas-ci, Documentum [DCTM]). Ces systèmes peuvent être disponibles en ligne ou être installés localement, être open source ou propriétaires et ils peuvent soit gérer un type de contenu unique ou gérer une gamme de types de contenu différents.

DCTM est une plateforme de gestion de contenu qui gère les attributs de document et qui, parmi ses fonctionnalités, dispose du check-in, du check-out, du contrôle de version, de la gestion de propriété et du contrôle d'accès.

Webtop, D2 et DA sont des interfaces conviviales qui fournissent l'accès au répertoire DCTM et aux services de gestion de contenu au sein d'un navigateur Web standard sans avoir à installer une application cliente distincte.

Même lorsque l'utilisateur (dans ce cas-ci, un développeur) utilise l'utilitaire de chargement en masse de DCTM, il doit saisir manuellement de nombreuses métadonnées du document, une tâche qui peut être monotone et nécessiter beaucoup d'effort. L'automatisation des tâches manuelles répétitives est mieux gérée en utilisant un langage de programmation, tel que Java, afin de pouvoir automatiser cette tâche fastidieuse. Les avantages pour l'organisation comprennent un gain de temps considérable et une réduction des erreurs. DCTM ne permet pas l'utilisation de fichiers de contrôle comme XML, JSON pour charger en masse du contenu.

L'objectif principal de ce projet de recherche appliquée est la conception et l'implémentation d'une interface en ligne de commande (c.-à-d. CLI) nommée OIMD pour automatiser le chargement de fichiers en masse dans DCTM en utilisant un fichier de configuration XML qui va contenir les métadonnées de contenu à importer. Cette fonctionnalité sera intégrée et expérimentée dans le cadriciel McGill CMS Framework.

Mots Clés : Documentum, Webtop, D2, DA, CLI

OUTIL D'IMPORT MASSIF DOCUMENTUM

Youssef RHINDI

ABSTRACT

A content management system can refer to any process or system that allows a person or organization to manage the content (In our case Documentum [DCTM]). These systems can be online or installed locally, open source or proprietary and they can manage a single content type or manage a range of different content types.

Documentum is an enterprise content management platform, it manages document attributes and it has features such as check-in, checkout, version control, ownership and access control.

Webtop, D2, and DA are user-friendly interfaces that provide access to the Documentum directory and content management services in a standard web browser without having to install a separate client application.

Even when a developer is using the Documentum Bulk Load utility, the developer must manually enter a large number of document metadata, a task that can be monotonous and time consuming. Automation of repetitive manual tasks is best handled with programming languages, such as Java, to automate tedious tasks. The benefits to the business include considerable time savings and reduced errors. DCTM does not allow the use of control files like XML, JSON in the Bulk load documents objects process.

The main goal of this applied research project, at master level, is the design and implementation of a command line interface (CLI) named OIMD to automate the bulk loading of files into Documentum using an XML configuration file that will contain the metadata documents to import. This feature will be integrated and experimented into the McGill CMS Framework.

Keywords: Documentum, Webtop, D2, DA, CLI

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ACL	Access Control Lists
API	Application Programming Interfaces
CCM	Collaborative Content Management
CCS	Content and Collaboration Solution
CLI	Command-Line Interface
CMS	Content Management System
DA	Documentum Administrateur
DAM	Digital Asset Management
DAPI	Documentum Application Programming Interfaces
DCTM	Documentum
DFC	Documentum Foundation Classes
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
OIMD	Outil d'Import Massif Documentum
TBO	Type Based Object
UAT	User Acceptance Testing
XML	Extensible Markup Language
BNS	Business Need Statement
ITSM	IT Service Management

INTRODUCTION

1.1 Motivation

En raison de l'utilisation croissante des documents numériques, le recours aux systèmes de gestion de contenu (de l'anglais Content Management Systems) est en croissance. Pour que l'équipe des administrateurs CCS CMS de l'Université McGill augmente leur productivité, elle doit pouvoir se concentrer sur le développement de fonctionnalités de haut niveau qui sont spécifiques aux produits dont ils sont responsables.

Consacrer du temps à mettre en œuvre des solutions personnalisées pour répondre aux besoins des clients (dans ce cas-ci, les facultés) ou des utilisateurs (dans ce cas-ci, les étudiants) est souvent fastidieux et nécessite beaucoup d'effort. De plus, une grande partie de l'effort d'utilisation de cette solution personnalisée nécessite beaucoup de travail manuel supplémentaire. Les bonnes pratiques en matière d'ingénierie logicielle suggèrent que le code implémentant des solutions du même domaine d'affaires ne devrait pas être ré-implémenté pour chaque client. Pour appliquer ces meilleures pratiques, les développeurs doivent s'appuyer sur une solution générique. Un cadriciel existant, le « McGill CMS Framework » fournit du code source pouvant être réutilisé afin de résoudre un problème particulier. Bien que cette définition semble être similaire à celle des bibliothèques de logiciels, il existe plusieurs distinctions importantes entre les deux. Avant tout, une bibliothèque de logiciels orientée objet contient des classes et des méthodes que le code de développeur instancierait et appellerait afin d'implémenter une solution pour le client. Un cadriciel a généralement pour objectif de fournir certaines fonctionnalités par défaut prêtes à l'emploi, mais il requiert du code de développeur adapté et personnalisé pour réaliser sa solution (Fayad & Schmidt, 1997).

Même si les cadriciel offrent des avantages significatifs en matière de réutilisabilité du code, ils présentent un ensemble de défis différents pour le développeur. Une difficulté majeure est la courbe d'apprentissage lente. Pour utiliser efficacement un cadriciel, le développeur doit bien comprendre les abstractions que ce cadriciel offre. Un cadriciel fournit souvent diverses

fonctionnalités et chaque fonctionnalité peut nécessiter l'utilisation de plus d'une classe définie par le cadriciel. En conséquence, le cadriciel (dont la structure est complète) pourrait être très complexe. L'utilisation du cadriciel se complique à mesure qu'il devient plus mature : de nouvelles fonctionnalités deviennent disponibles et, par conséquent, le cadriciel devient encore plus complexe en raison de l'étendue des fonctionnalités. Par conséquent, sélectionner uniquement la partie du cadriciel pertinente (pour être en mesure de résoudre le problème visé par l'application cliente) n'est pas une tâche triviale (Fayad & Schmidt, 1997).

1.2 But du projet

Ce rapport de maîtrise appliquée aborde les points suivants :

- Les faiblesses des outils existants dans le but de fournir une solution logicielle pouvant être utilisée dans le contexte des opérations quotidiennes au sein du service de gestion du contenu de l'université McGill;
- Les stratégies pour surmonter les problèmes rencontrés lors de la manipulation de quantités massives de contenu à charger dans Documentum.

Ce projet de recherche appliquée de 15 crédits vise à appliquer les meilleures pratiques du génie logiciel et à remplacer l'outil développé précédemment par CCS CMS Developer, dans le but de procéder de manière plus efficace pour répondre aux besoins croissants en ce qui concerne la manipulation de grandes quantités de contenu.

1.3 Vue d'ensemble

La suite de ce rapport de projet de recherche est organisée comme suit :

- Le chapitre 2 traite des informations de base qui seront nécessaires la mise en contexte de la solution proposer plus loin dans le chapitre 3. Il commence par une brève discussion sur l'équipe CCS CMS Developer. La suite de cette section est une

description de la méthodologie Agile. La dernière partie du chapitre traite les services supportés par l'équipe CCS CMS Developer, en particulier les services Documentum;

- Le chapitre 3 traite des informations sur le serveur documentum de haut niveau dans la première section. La deuxième section présente plus d'informations sur certaines composantes de Documentum, ses composants interagissent directement avec la solution proposée plus loin dans ce rapport;

- Le chapitre 4 traite la mise en œuvre et l'expérimentation d'un modèle de conception d'architecture logicielle afin de créer une base de code extensible et facile à maintenir à l'avenir.

1.4 Conclusion

Ce chapitre, à identifier le contenu numérique non structuré, comment augmenter l'efficacité de l'équipe CCS CMS Developer, comment un cadriciel permet d'implémenter rapidement des applications et le but du projet. Finalement, une vue globale sur la structure du rapport a été listé.

Le prochain chapitre introduit l'équipe CCS CMS Developer.

CHAPITRE 2

Introduction à CCS CMS Developer

2.1 L'équipe des outils de collaboration

L'équipe CCS CMS Developer relève du directeur de plateforme des services partagés et de développement des services à l'Université McGill de Montréal. Cette équipe travaille également en étroite collaboration avec les parties prenantes, les gestionnaires de projets informatiques, les responsables informatiques et l'équipe du service à la clientèle.

Cette équipe est chargée de la mise en œuvre, de l'évolution, de la maintenance et du support des services informatiques du portefeuille CCS (voir la description aux sections 2.1.2 et 2.1.3). Elle est également responsable de la fiabilité, de la stabilité et de la disponibilité de ces services.

Peter Vergados, le gestionnaire du portefeuille CCS reçoit la mission de l'équipe au début de l'année sous forme d'objectifs et il lui revient de déterminer de quelle façon cette mission sera réalisée, d'établir les priorités, de respecter les calendriers et de déterminer les objectifs de chaque membre de l'équipe. Ce gestionnaire fournit aussi des conseils aux membres de l'équipe dans les limites des politiques de l'Université McGill, et il recommande des modifications aux stratégies ou procédures pour l'amélioration continue de l'organisation.

Son équipe compte actuellement deux séniors ECM, trois analystes ECM, un consultant et un stagiaire. Ces membres d'équipe contribuent ensemble au développement et au support des systèmes informatiques afin de s'assurer que les systèmes répondent aux besoins actuels et futurs des parties prenantes et de la communauté des utilisateurs de l'Université McGill. Les responsabilités principales de chaque membre de l'équipe sont :

- Résoudre les incidents ou problèmes du système et assurer la liaison entre les administrateurs des infrastructures et l'équipe du service à la clientèle;
- Effectuer l'analyse des besoins de l'organisation;

- Évaluer la faisabilité, fournir des estimations, concevoir des solutions et préparer des spécifications détaillées;
- Diriger et coordonner la mise en œuvre des améliorations simples des systèmes;
- Collaborer avec les autres équipes dans l'organisation ou les experts externes pour mettre en œuvre de nouveaux systèmes qui répondent aux exigences de l'organisation et seront conformes aux meilleures pratiques de l'industrie informatique;
- Élaborer des scénarios de test et effectuer des tests fonctionnels et d'intégration détaillée pour s'assurer que la programmation ou la configuration a été effectuée conformément aux spécifications fonctionnelles détaillées et aux exigences de l'organisation;
- Soutenir les autres équipes (internes au développement) et les utilisateurs lors des tests d'acceptation des utilisateurs UAT (de l'anglais User Acceptance Testing);
- Aider au développement de la documentation du manuel d'utilisateur et du matériel de formation.

2.1.1 Méthodologie de livraison : Agile

L'équipe utilise une méthodologie Agile. La durée d'une itération SCRUM est de deux semaines. Les activités principales durant une itération sont (Rauf & AlGhafees, 2015) :

1. Planification des objectifs de l'itération;
2. Réunions quotidiennes, effectuées debout pendant 15 minutes, qui permettent aux membres de l'équipe de se mettre à jour et de se préparer pour la journée;
3. Révision de l'itération, qui consiste en une réunion ou chaque membre de l'équipe fait une démonstration des réalisations effectuées durant l'itération;
4. Rétrospective de l'itération durant laquelle chaque membre de l'équipe présente ses commentaires, observations, améliorations ou recommandations.

2.1.2 Les services : Documentum

DCTM est un système de gestion de contenu qui aide les organisations à intégrer leur contenu non structuré sur une plateforme unique (c.-à-d. plus de détails seront présentés dans le chapitre 3). Le tableau 1 liste tous les services que l'équipe CCS CMS Developer est responsable de maintenir.

Tableau 1 — Les services Documentum gérés par l'équipe CCS CMS Developer

Services	Descriptions
DA	Documentum Administrateur est l'application d'administration principale du serveur de contenu. Elle permet de surveiller, d'administrer, de configurer et de gérer les serveurs et les répertoires, le tout à partir d'un navigateur Web. L'équipe CCS CMS Developer est la seule à utiliser cette application en tant qu'administrateur système.
D2	OpenText Documentum D2 est le client de Documentum orienté contenu. Ce logiciel intuitif et configurable accélère l'adoption des applications OpenText. Il permet une configuration détaillée qui élimine le besoin de code personnalisé. Il permet aussi un déploiement rapide et réduit considérablement les coûts de maintenance du système (OpenText, 2019b). Cette application comprend actuellement environ 13 000 utilisateurs actifs et environ 580 espaces collaboratifs.
Webtop	Documentum Webtop est une application Web (similaire à Documentum Desktop) qui permet aux utilisateurs d'accéder à un ou plusieurs répertoires Documentum. Cette application comprend actuellement environ 1000 utilisateurs actifs.
eCalendar	eCalendar est un logiciel développé par McGill qui permet de gérer les programmes, les cours et d'autres informations importantes du calendrier universitaire. Cette application comprend actuellement environ 350 éditeurs actifs.

eStudent Records	eStudent Records fait référence à la numérisation, au stockage et à la récupération de documents électroniques contenant des dossiers d'étudiants. Ses fonctionalités sont disponibles via l'application Web Webtop. Cette application comprend actuellement plus que 100 000 enregistrements d'élèves.
uApply	uApply est le système en ligne d'admission aux études. Ce système comprend actuellement environ 70 000 dossiers d'étudiants, et environ 600 000 fichiers.
Application Xtender	Application Xtender est un système de gestion de documents qui réduit la circulation de documents papier et les interactions manuelles permettant ainsi de réduire les coûts de transaction, de garantir la conformité aux règlementations et aux normes et d'améliorer l'accès à l'information pour une meilleure prise de décision et efficacité opérationnelle. Ce système comprend actuellement 6 applications, et environ 15 millions de fichiers.
Legal Case Management	Le logiciel Legal Case Management a été développé par McGill pour gérer ses documents juridiques. Cette application comprend actuellement environ 600 cas.

2.1.3 Les services : Office 365

Le département de soutien informatiques, de l'Université McGill, fournissent Office 356 ProPlus a toute la communauté des utilisateurs de McGill (c.-à-d. étudiants, professeurs et membres du personnel). Il comprend la version complète :

- Office 2016: Word, PowerPoint, OneNote, Excel, Access, Publisher, Outlook, Skype et InfoPath;
- Les outils de collaboration: OneDrive, Notebook, Sway, Forms, Video, Stream, Planner, Tasks, Teams, To-Do et Yammer.

Tous les services de collaboration sont actuellement en mode pilote. Conséquemment ils ne sont pas disponibles par défaut à toute la communauté en ce moment. La mise en service de chaque service est liée à la capacité de l'équipe CCS CMS Developer de fournir un support

technique de qualité. Le tableau 2 liste tous les services que l'équipe CCS CMS Developer est responsable de maintenir actuellement:

Tableau 2 — Les services d'Office 365 gérés par l'équipe CCS CMS Developer

Services	Descriptions
OneDrive	« Microsoft OneDrive est un espace de stockage personnel, illimité et infonuagique. Il permet de synchroniser les fichiers avec tous les appareils connectés et ainsi d'accéder à distance à ses documents »[1]. Ce service comprend actuellement environ 25 000 utilisateurs actifs, et environ 20 millions de fichiers.
Vidéo	« Vidéo @ McGill fait partie de l'offre Microsoft Office 365. Ce service fournit à l'Université McGill une plateforme sécurisée pour la publication, le partage et la découverte de contenu vidéo » (McGill, 2019a). Cette plateforme comprend actuellement environ 100 canaux.
Yammer	« Yammer est un réseau social d'entreprise qui est privé et sécurisé et qui permet aux utilisateurs de se connecter avec d'autres utilisateurs appartenant au même domaine de messagerie. Le site Yammer de l'Université McGill ressemble à un site privé sur Facebook spécialement pour la communauté de McGill. Yammer est offert pour partager des informations, collaborer et prendre des décisions de groupe. » (McGill, 2019b). Ce service comprend actuellement environ 700 utilisateurs actifs, en mode pilote.
Sway	Sway est une application de Microsoft Office qui facilite la création et le partage des rapports interactifs, d'histoires personnelles, de présentations et plus encore. Ce service est actuellement en mode pilote et le nombre d'utilisateurs est d'environ 100 au moment de rédiger ce rapport.

2.2 Conclusion

Ce chapitre, a brièvement introduit l'équipe CCS CMS Developer, ses clients, son chef d'équipe et ses membres. La méthodologie Agile est utilisée par ce groupe, la durée d'un Sprint et ses activités ont aussi été présentées. Finalement, les services que l'équipe supportent ont été listés, en insistant d'avantage sur ceux de Documentum, qui sont d'intérêt pour cette recherche appliquée.

Le prochain chapitre décrit le serveur DCTM et la solution proposée par cette recherche, c'est-à-dire le OIMD.

[1] https://fr.wikipedia.org/wiki/Microsoft_Office_365

CHAPITRE 3

DOCUMENTUM

3.1 Documentum

Dans une organisation, les informations commerciales existent sous plusieurs formes : documents textuels, feuilles de calcul, images, fichiers XML, pages Web, vidéos, audios, messages électroniques, messages instantanés et documents tels que les rapports, les enregistrements et les images numérisées. La gestion des documents qui possèdent un contenu non structuré sont maintenant très présents et incontournables au fonctionnement efficace de n'importe quelle organisation moderne (Abdullah & Ahmad, 2013).

Un système ECM permet de gérer ces informations non structurées. Il gère la création, la gestion, le traitement, la livraison et l'archivage de tout contenu, et ce conformément aux règles de gestion définies par l'organisation. Il établit des relations entre les éléments de contenu, ce qui permet d'utiliser le même contenu dans différents contextes. Il ajoute une intelligence, en créant un schéma de catégorisation et en assignant des métadonnées qui rendent la recherche et la récupération du contenu plus rapides et plus efficaces. Il automatise le traitement du contenu tout au long de son cycle de vie. Il facilite aussi la publication de contenu à travers plusieurs canaux (Tyrväinen, Päivärinta, Salminen, & Iivari, 2006). Par exemple, le même contenu peut être publié sur un site Web, diffusé sous forme de télécopie, imprimé sous forme de document texte et envoyé à un périphérique sans fil. Un système ECM a la capacité d'intégrer tous les services et les systèmes qui fonctionnaient d'une manière isolée par le passé (Päivärinta & Munkvold, 2005).

La plateforme DCTM offre un ensemble de produits et de services intégrés qui forment un ensemble cohérent, permettant différentes configurations, afin de mieux répondre aux besoins de gestion de contenu d'une organisation. Cette plateforme facilite la personnalisation des applications dans un but de répondre à des besoins spécifiques et à la création d'applications de contenu personnalisées à des besoins spécifiques.

La plateforme DCTM offre aussi des services de développement et d'exécution répondant spécifiquement aux besoins des applications de contenu. L'architecture de la plateforme garantit que ces fonctionnalités fonctionnent de manière orchestrée (Documentum, 2006).

Ce chapitre présente une vue d'ensemble d'un sous-ensemble de fonctionnalités de la plateforme DCTM pertinent à ce projet de recherche appliqué. Seulement les parties et les fonctionnalités qui vont interagir avec l'OIMD sont d'intérêt pour ce rapport. Il décrit également le détail des composantes et des fonctionnalités de l'OIMD. La description exhaustive de toute la plateforme DCTM n'est pas dans la portée de ce rapport. Pour des informations plus détaillées sur DCTM, la documentation officielle devrait être consutlée.

3.2 Architecture

La Figure 1 illustre une organisation de l'architecture très détaillée du serveur DCTM pour donner une vue d'ensemble de la complexité des composants de cette plateforme. On peut y voir les quatre couches de services et les fonctionnalités associées. Cette section présente sommairement les composants qui seront directement utilisés lors de cette recherche et de l'expérimentation, par exemple : le référentiel de contenu; le serveur de contenu; le modèle d'objet DCTM; et le DCTM Foundation Classes (DFC).

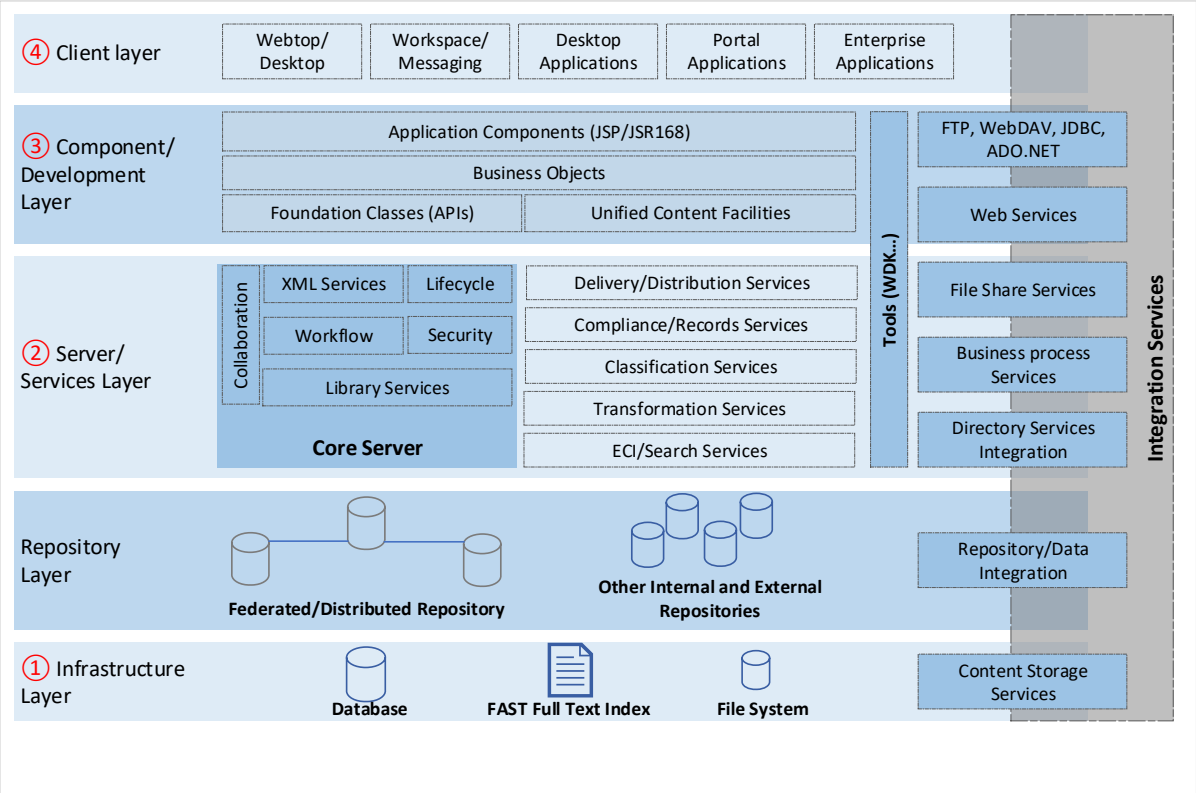


Figure 1 — L'architecture détaillée de la plateforme DCTM

Source : (réf) adapté de (Parag, 2019)

La plateforme DTCM est organisée en quatre couches (Kumar, 2010) :

1. La couche noyau (représenter par le chiffre 1) fournit la capacité principale de gestion de contenu que ce soit pour stocker, accéder ou sécuriser du contenu;
2. La couche de services d'application (représenter par le chiffre 2) fournit la capacité d'organiser, de contrôler et de distribuer le contenu depuis et vers le référentiel;
3. La couche outils (représenter par le chiffre 3) permet le développement et le déploiement d'applications personnalisées pour la gestion de contenu de l'organisation. Elle est aussi composée de DCTM Foundation Class (DFC) et d'API's qui assurent la communication entre la couche de services et les clients (c.-à-d. applications et interfaces);

4. La couche d'expériences (représenter par le chiffre 4) fournit un cadriciel et ses interfaces nécessaires pour la présentation du contenu que ce soit dans les applications de bureau ou sur le Web.

3.2.1 Le référentiel de contenu

La Figure 2 décrit le référentiel comme une unité logique comprenant des fichiers de contenu, des tables SGBDR (c.-à-d. base de données relationnelle) et un index de texte intégral.

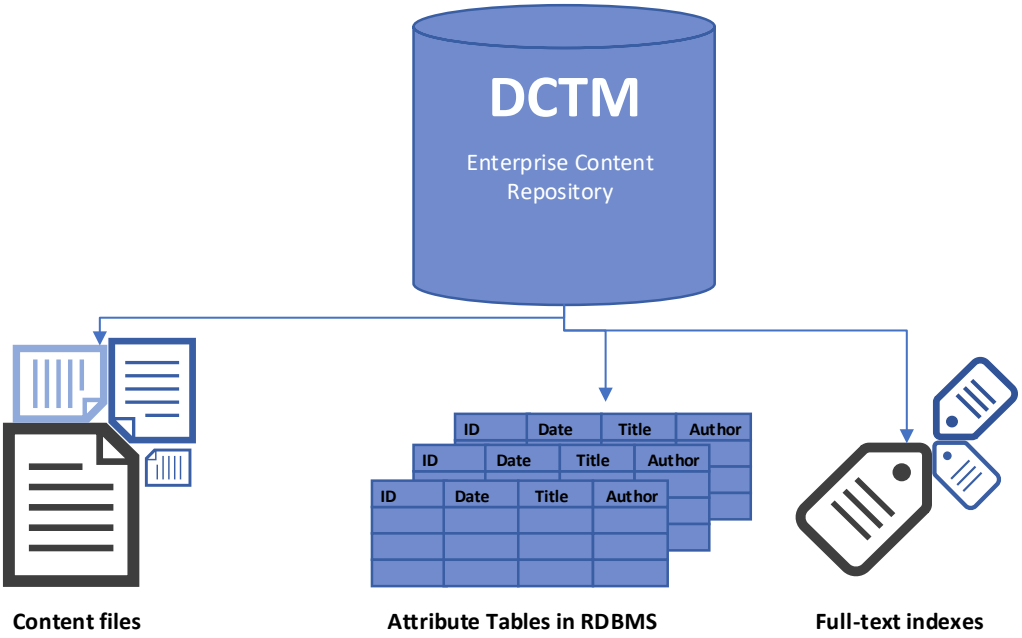


Figure 2 — Structure du référentiel de contenu
Source : adapté de (Documentum, 2006)

Le référentiel de contenu DCTM est géré par le serveur du contenu DCTM et fournit les fonctionnalités de gestion de contenu fondamentales, le référentiel utilise un modèle objet extensible pour stocker le contenu et ses métadonnées associées (Documentum, 2006).

Le référentiel de contenu est une unité logique abstraite composée de données stockées dans des sources physiques distinctes. Les objets sont composés de fichiers de contenu (c.-à-d. le contenu dans son format natif) et des attributs de contenu (c.-à-d. également appelés métadonnées ou propriétés), tels que le propriétaire, la version et la date de création du contenu. Ces attributs servent de métadonnées décrivant le contenu et les relations entre ce contenu et un autre objet du référentiel. Le référentiel utilise les métadonnées pour organiser le contenu et les utilisateurs peuvent les utiliser durant les recherches.

Les attributs de document sont stockés dans une base de données relationnelle. Elle peut inclure des attributs requis pour avoir une valeur unique, tels que l'identificateur unique du document (ID), et des attributs pouvant avoir plusieurs valeurs, tels que des mots-clés décrivant le contenu. En plus des fichiers de contenu et des attributs qui les décrivent, le référentiel inclut un ensemble d'index, en texte intégral, créé par le moteur de recherche de texte intégral qui est situé dans le serveur de contenu.

3.2.2 Le serveur de contenu

Le serveur de contenu intègre les fichiers de contenu et les métadonnées associées dans des objets de document et fournit un accès basé sur les objets aux documents résultants. Le serveur traite les fichiers de contenu et les métadonnées comme faisant partie d'une seule entité et gère les mises à jour de l'objet document comme une transaction unique : il met à jour les deux éléments simultanément. Le serveur met également à jour automatiquement les entrées d'index, garantissant que les trois types de données ne peuvent pas être désynchronisés.

Le serveur de contenu est un système basé sur les objets : tout ce que les utilisateurs manipulent (c.-à-d. documents, dossiers, profils de sécurité, processus métiers, etc.) est stocké et géré comme un objet par le serveur de contenu. Le modèle d'objet DCTM est la structure par laquelle le serveur organise le contenu et les mécanismes de contrôle des référentiels (Kumar, 2010).

3.2.3 Le modèle d'objet DCTM

La Figure 3 ci-dessous, montre un modèle orienté-objet qui permet de stocker des informations dans le référentiel. Dans un système DCTM basé sur les objets, un objet est un composant constitué à la fois de données (c.-à-d. de fichiers de contenu, d'attributs et de relations dans le cas de documents) et d'instructions relatives aux opérations disponibles sur ces données (c.-à-d. appelées méthodes). Tout comme l'ensemble des attributs, l'ensemble des opérations (c.-à-d. les méthodes) d'un objet sont configurables et extensibles à l'aide d'outils de développement DCTM. Les développeurs peuvent créer de nouveaux types d'objets qui se comportent exactement selon leurs besoins spécifiques.

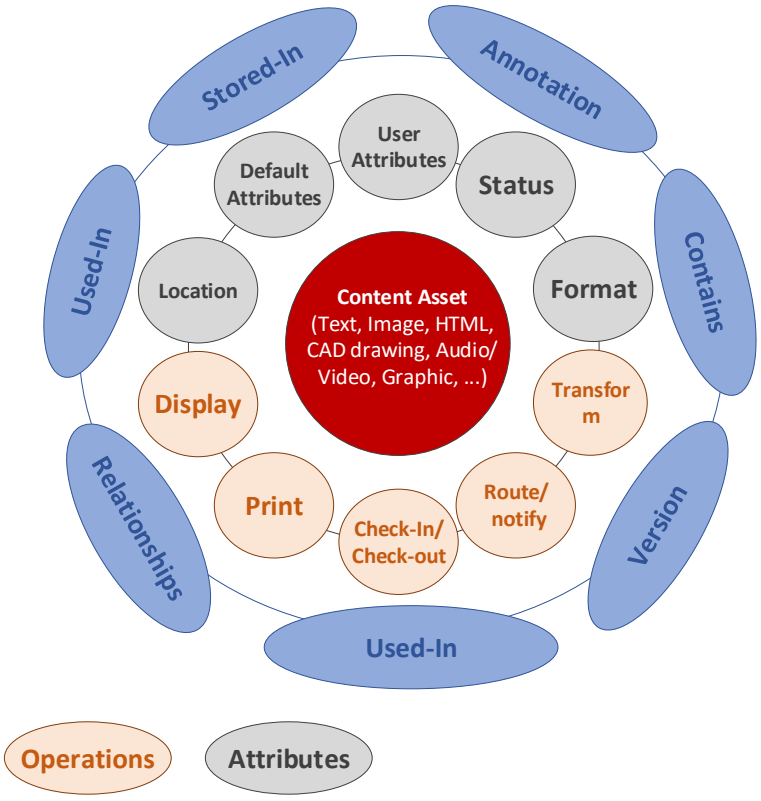


Figure 3 — Objet de document DCTM

Source : adapté de (Documentum, 2006)

3.2.4 DCTM Foundation Classes (DFC)

La Figure 4 présente un ensemble d'interfaces et de classes Java disponibles.

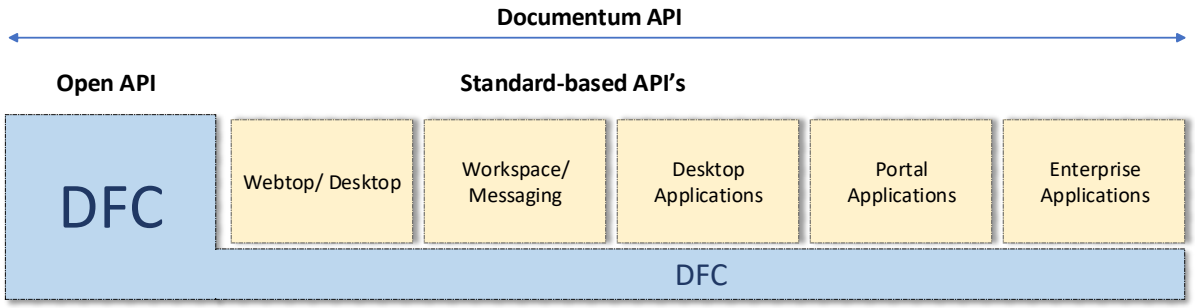


Figure 4 — Interface de programmation d'application DCTM (DAPI)
Source : adapté de (Documentum, 2006)

Les clients et les applications utilisent cette couche d'interface (c.-à-d. API) pour communiquer avec le serveur de contenu et interagir avec le référentiel de contenu. La couche d'interface est composée de DCTM Foundation Classes (DFC) et d'un certain nombre d'interfaces standards construites au-dessus de DFC. Ensemble, ces APIs forment l'interface de programmation d'application DCTM (DAPI).

DCTM Foundation Classes (DFC) représente l'API la plus riche exposant toutes les fonctionnalités de DCTM (Kumar, 2010). Les DFC fournissent une infrastructure orientée-objet pour accéder aux fonctionnalités du serveur de contenu. Les DFC exposent aussi le modèle d'objet DCTM, en tant que bibliothèque client orientée objet, que les applications de gestion de contenu peuvent utiliser. Cette API peut être utilisée à partir d'une multitude de langages, y compris Java, Visual Basic, C # et C ++.

Chaque ordinateur exécutant une application qui accède au serveur de contenu possède une copie du DFC qui s'exécute sur une machine virtuelle Java (JVM). L'application sur l'ordinateur client effectue un appel de méthode à travers la couche DFC, qui traduit l'appel

dans l'API natif du serveur de contenu. Le serveur de contenu instancie l'objet, exécute l'appel de méthode et renvoie le résultat à l'application cliente. La Figure 5 illustre le fonctionnement du paradigme requête-réponse entre les clients et le serveur de contenu à travers la couche DFC.

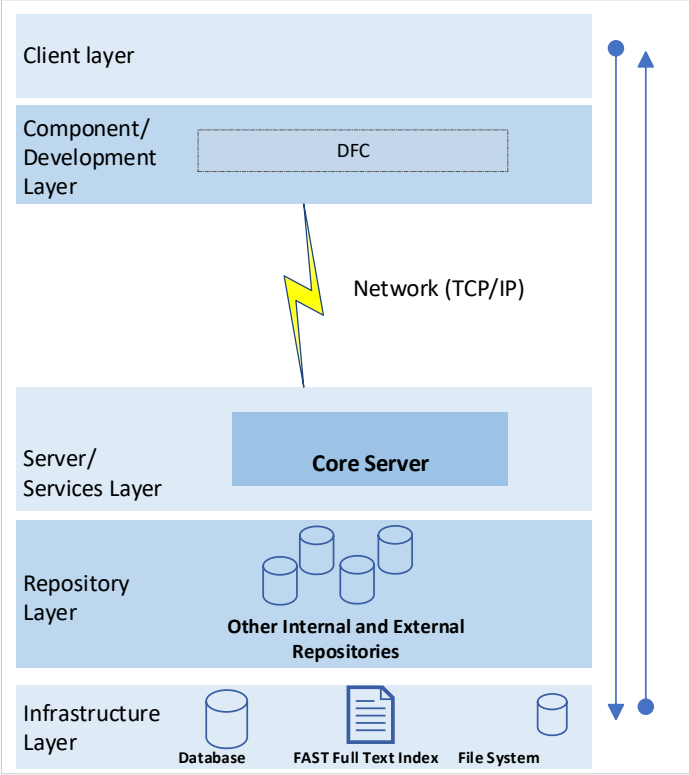


Figure 5 — L'architecture simple client et serveur

Source : adapté de (Documentum, 2006)

3.3 OIMD

Cette section présente la première version (OIMD.0.0.1) de l'interface de ligne de commande OIMD pour l'importation massive de contenu dans DCTM. Elle décrit aussi plus de détail sur les deux fonctionnalités principales de cette version : 1) la création et capture de contenus et 2) la gestion du contenu une fois importée dans le référentiel. La Figure 6 illustre ces composants et les données, types et méthodes associés.

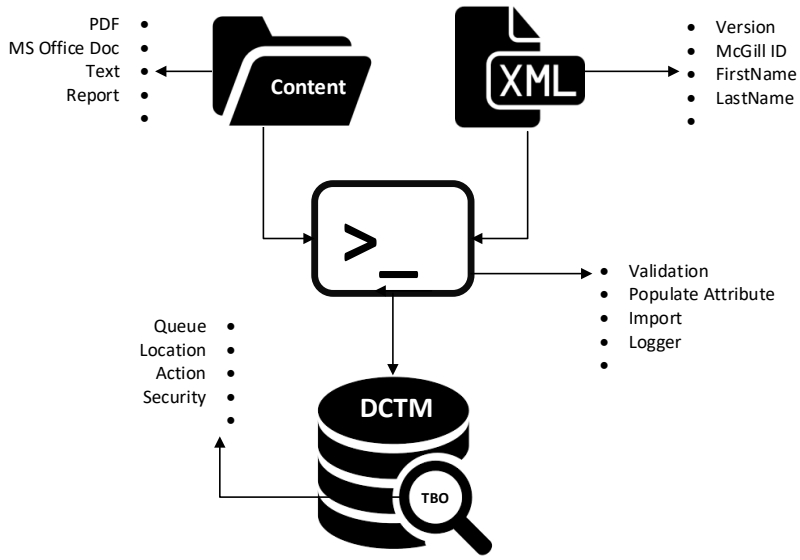


Figure 6 — Les composantes de l'OIMD

3.3.1 Création et capture de contenus

La première tâche, de n'importe quel outil de gestion de contenu, consiste à collecter le contenu pertinent et à l'ajouter au référentiel de l'organisation. Le contenu peut provenir d'une variété de sources, que ce soit interne ou externe à l'organisation ou les deux.

OIMD fournit un processus structuré pour définir, valider et importer des documents dans le référentiel DCTM. Il se compose de deux composants: 1) un CLI (voir « Content ») ; et 2) un fichier XML (voir la Figure 6). Le CLI permet de définir les données d'entrée source, la mise à jour des attributs de contenu, les requêtes de validation, l'envoi d'un courriel électronique qui contient des statistiques comme le nombre de dossiers est de fichiers importés, le temps consommé par l'opération et la taille du fichier. Le CLI génère aussi un fichier journal des résultats qui fournit plus de détails sur les étapes d'exécution de l'opération d'importation des données, ainsi que sur toutes les erreurs et les avertissements générés lors de cette étape. Le fichier XML de son côté définit le mappage des attributs DCTM sur le contenu à l'entrée. Le CLI prend deux paramètres à l'entrée : 1) le fichier XML; et 2) le lien vers le contenu à importer. Une fois l'import réussi, les fonctionnalités de l'OIMD ne s'arrêtent pas là. Il y a un

post traitement par DCTM qui va être exécuté au niveau du Type-Based Object (TBO). Plus de détails sur ce traitement sont décrits à la section suivante.

3.3.2 Gérer le contenu avec le TBO

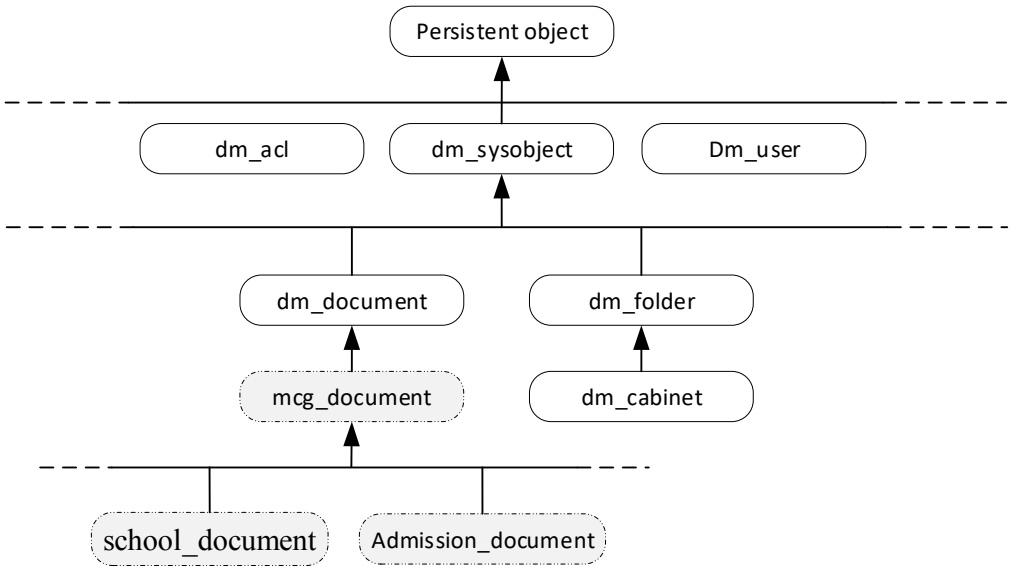


Figure 7 — hiérarchie des types DCTM avec des types personnalisés
Source : adapté de ({javaecm, 2018 #22})

Le service principal offert par un Type-Based Object (TBO) est la possibilité offerte au développeur d'ajouter, modifier et étendre le comportement d'un type personnalisé persistant à l'aide de l'ajout de code spécialisé. Par exemple, en tant que développeur, si vous souhaitez ajouter de nouveaux attributs ou modifier considérablement le comportement d'un type de document DCTM, la Figure 7 présente la création d'un type personnalisé à McGill mcg_document de type dm_document, ensuite, créer un module de type TBO pour ajouter ou modifier le comportement de ce sous-type. Après l'installation du TBO dans DCTM, le DFC crée une instance de la classe d'implémentation TBO à chaque fois que les clients demandent l'accès ou la création d'un objet de mcg_document (Corporation, 2011). Parmi les fonctionnalités de ce post traitement du TBO qui sera conçu et expérimenté dans ce projet de maîtrise appliqué, il devient possible de créer une structure de dossier, déplacer les fichiers aux bons endroits.

3.4 Conclusion

Ce chapitre a décrit comment un système ECM permet de gérer efficacement les informations non structurées d'une organisation. Il a présenté que DCTM est un système de gestion de contenu qui aide les organisations à intégrer leur contenu non structuré sur une plateforme unique. Ce chapitre a aussi abordé, dans la deuxième section, certaines caractéristiques de la plateforme DCTM en relation avec la proposition de la conception d'un OIMD, par exemple, le référentiel de contenu, le serveur de contenu, le modelé d'objet DCTM et l'interface DFC.

À la fin de ce chapitre, les fonctionnalités qui seront conçues et expérimentées dans la première version du prototype expérimental de la solution OIMD ont étés présentées à haut niveau. Dans le prochain chapitre, la conception, le développement d'un prototype de la solution OIMD est présenté.

CHAPITRE 4

CONCEPTION ET EXPÉRIMENTATION

Ce chapitre présente un aperçu de la réalisation du prototype expérimental OIMD qui a été présenté à haut niveau dans le chapitre précédent (section 3.3). Une première tentative de réalisation d'un outil semblable a été tentée quelques années auparavant, c'était une classe Java qui consiste à importer 100 documents à fois, en utilisant les mêmes métadonnées. La classe Java en question était exécutée directement dans l'IDE (c.-à-d. Éclipse). L'objectif de ce projet de recherche appliquée est de développer une solution avec beaucoup plus de fonctionnalité (section 3.3.1).

Ce chapitre présente la méthodologie de livraison, le travail réalisé lors de la conception et de l'implémentation de l'application. De plus, il présente la structure des données utilisées dans les essais et lors du processus de validation du prototype expérimental.

4.1 Méthodologies de livraison

La méthodologie de livraison agile est utilisée durant ce projet. La section 2.1.1 présente plus de détails sur les étapeseffectuées. Une étape importrante a été le « Sprint rétrospectif » qui a été effectué toutes les deux semaines. Dans cette rencontre les membres de l'équipe échangent sur les fonctionnalités ou modules implémentés, ce qui doit être amélioré où corrigé avant de commencer le cycle de l'itération suivante (c.-à-d. l'analyse, la conception, le développement et les tests de la nouvelle fonctionnalité). C'est généralement un exercice très dynamique et pratique.

4.2 Conception

Ce projet comprend trois composantes : 1) Invite de commande; c'est-à-dire l'interface graphique du système d'exploitation Linux ou Windows ; 2) McGill CMS Framework; c'est-à-dire un ensemble de modules qui visent à automatiser des interactions avec le serveur de

contenu (voir Figure-A I- 1); et 3) Documentum; la platforme de gestion de contenu. Cette section présente les décisions de conception de l'OIMD à l'aide de la structure présentée ci-dessous. Le schéma fonctionnel à implanter est illustré à la Figure 8 :

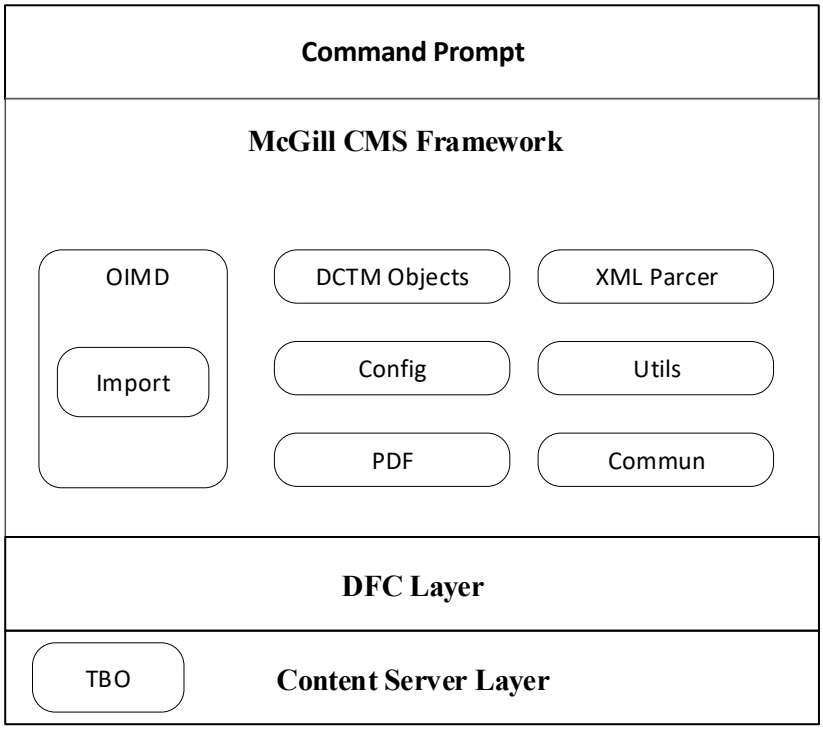


Figure 8 — Schéma fonctionnel de l'OIMD

La Figure 8 présente une vue fonctionnelle de haut niveau. Tous les cas d'utilisation planifiés pour cette première version du prototype expérimental sont représentés dans ce schéma. De plus, le schéma présente une vue synthétique du projet pour toutes les parties prenantes.

Le cadriciel « McGill CMS Framework », « désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture). »[2] l'OIMD est le premier composant implémenté dans ce cadriciel, il représente la logique métier implémentée par trois modules définissant les services,

[2] https://fr.wikipedia.org/wiki/Framework

les objets de domaines et les évènements : 1) un module d'analyse syntaxique; 2) un module de traitement de la requête; et 3) un module post-traitement (TBO).

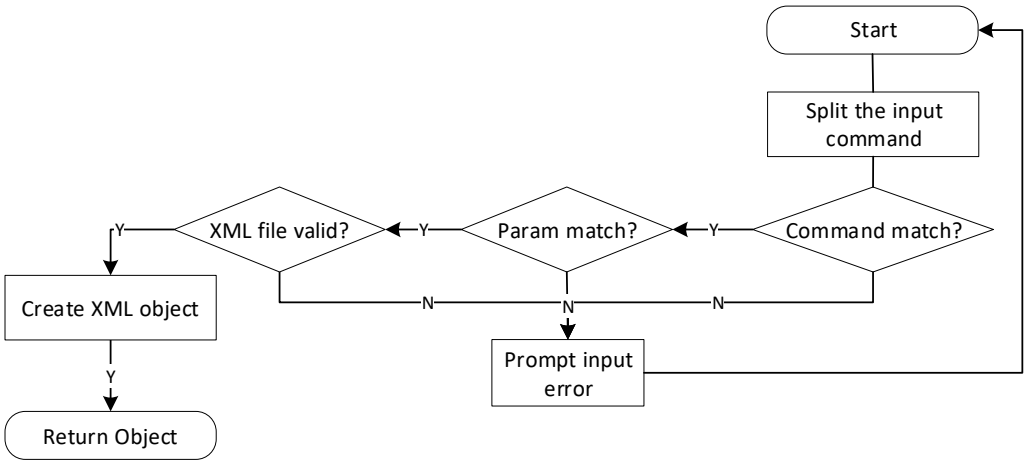


Figure 9 — Organigrammes d'analyse syntaxique de l'OIMD

La Figure 9 présente le flux d'exécution du module d'analyse syntaxique (c.-à-d. le «xmlParcer») qui analyse la commande d'entée et vérifie si le format de la commande comporte une erreur (c.-à-d. conformément à des règles de validation spécifiques). En cas d'erreur, durant cette vérification, un message d'erreur est envoyé à l'utilisateur. Dans le cas contraire, il vérifie si les paramètres de la commande existent dans le jeu de commandes. Si tel est le cas, il retourne un objet représentant le fichier XML au module d'import.

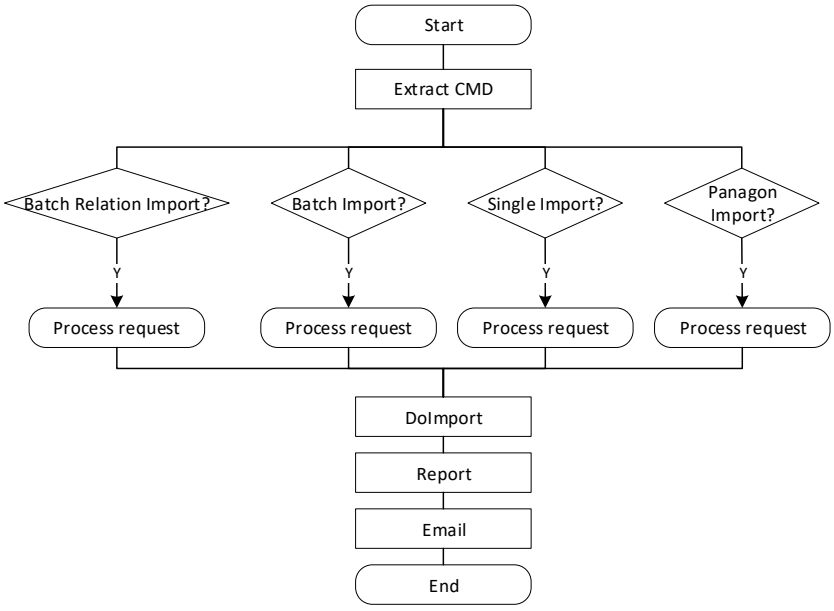


Figure 10 — Organigrammes de traitement des requêtes

La Figure 10 présente le flux d'exécution du module de traitement de la requête (c.-à-d. «Imports»). Ce dernier est responsable du traitement de la fonctionnalité visée par la requête. Dans cette première version du prototype expérimental, quatre types d'imports ont été implémentés (c.-à-d. Batch Relation Import, Batch Import, Single Import, et Panagon Import).

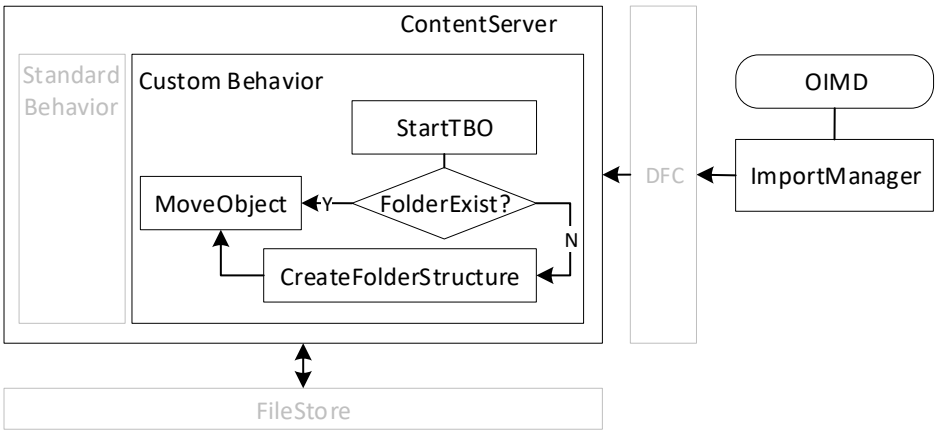


Figure 11 — Organigrammes d'exécution du TBO.

Un module pos-traitement (TBO), est installé dans le serveur DCTM. Ce module est basé sur les évènements d'un type spécifique. Chaque importation ou création dans le serveur DCTM d'un contenu possède un type spécifique. Le serveur exécute ce module. La Figure11 montre les traitements post-imports exécutés par le TBO.

La couche DFC fournit une infrastructure orientée-objet pour accéder aux fonctionnalités du serveur de contenu (section 3.2.4).

La couche serveur de contenu (c.-à-d. Content Server) intègre les fichiers de contenu et les métadonnées associées dans des objets DCTM et fournit un accès basé sur les objets aux documents résultants (section 3.2.2).

La synthèse des objectifs de cette section sont: utiliser une approche de conception orientée-objet pour diviser le logiciel en modules distincts. Cette approche de conception, provenant du génie logiciel, vise à s'assurer que l'ajout futur de nouvelles caractéristiques ou fonctionnalités n'impliquerait que peu d'effort pour son intégration dans le logiciel existant.

4.3 Implémentation

Une nouvelle interface de ligne de commande a été conçue et mise en œuvre en utilisant le langage de programmation Java afin de maintenir la cohérence avec la base de code de la platforme DCTM. Les autres technologies et outils utilisées lors de la réalisation de ce projet sont présentés avec plus de détails au Tableau-A II- 1 de l'annexe II. Cette section décrit trois parties du code source (c.-à-d. le traitement des requêtes, l'analyse syntaxique, et le TBO). La hiérarchie du code source du projet est présentée à la Figure 12 :

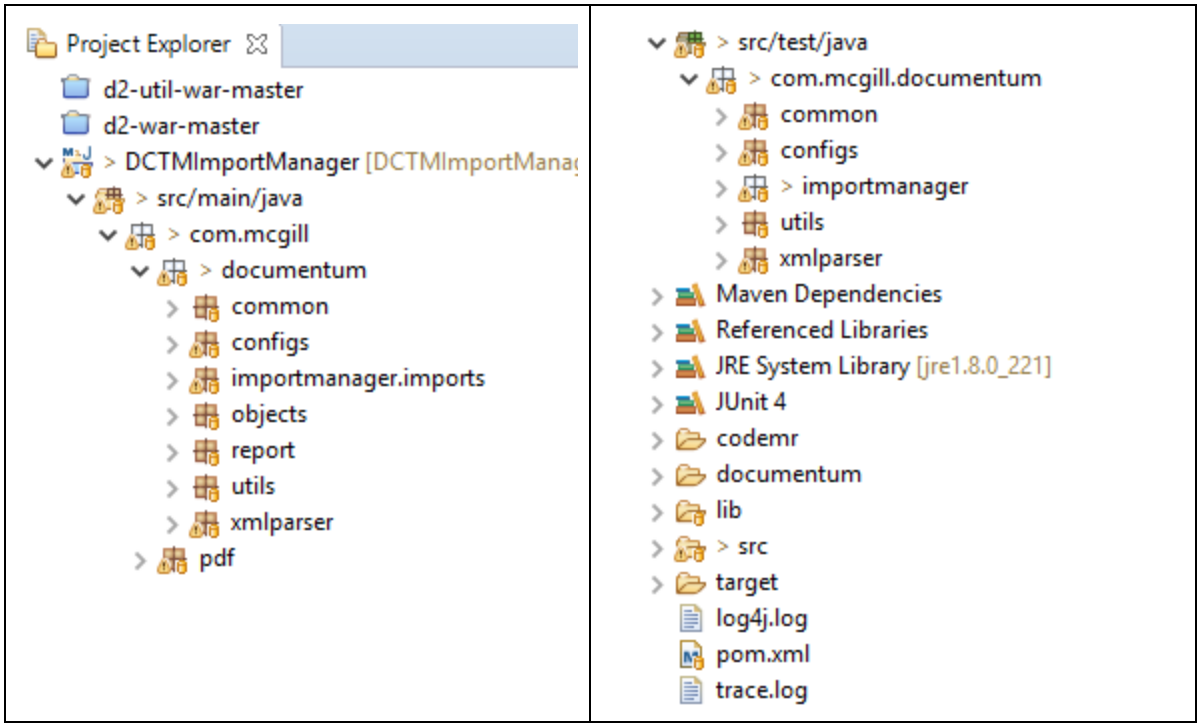


Figure 12 — Vue sur la hiérarchie du code source du projet

4.3.1 Traitement des requêtes (Import)

L'approche de conception choisie vis à exploiter certaines fonctionnalités en tant que modules, en particulier en étendant les actions de l'OIMD avec l'ensemble des opérations fournies par ces modules. En appliquant la notion d'externalisation, un module est alors conçu comme un outil que l'OIMD peut éventuellement créer et utiliser selon ses besoins. Comme le montre la Figure-A I- 2, les modules : xmlParcer, objects, common, utils, pdf et config sont des outils externes que l'OIMD peut utiliser pour effectuer son travail. Le Tableau-A II- 2, de l'annexe II, décrit en détail les fonctionnalités de chaque module. Le choix d'externaliser ces fonctionnalités rend le développement de l'application plus rapide et plus facile à gérer, nécessitant moins d'effort pour implémenter de nouvelles fonctionnalités. Ainsi les modifications peuvent être apportées, testées et déployées plus rapidement et plus facilement. Dans les deux prochaines sections, plus de détails sont présentés concernant des modules plus spécifiques au domaine d'affaire de la solution.

4.3.2 Analyse syntaxique (xmlParcer)

Les fichiers sont premièrement importés dans un répertoire du système de fichiers local et les métadonnées dans un fichier XML. Maintenant, pour lire ces données afin de créer les objets nécessaires dans DCTM, il est nécessaire d'importer les fichiers dans le serveur DCTM et les mapper à l'objet créé. En fonction du type d'objet, le serveur de contenu les stockera dans le magasin de fichiers. La Figure 9 illustre l'architecture effectuée du module d'analyse syntaxique. Tout d'abord, il est nécessaire de créer un service d'analyse qui lit les métadonnées des fichiers à importer, lit le contenu du fichier et valide le fichier à l'aide d'un schéma XML et à l'aide d'une bibliothèque supportée par Java (c.-à-d. le SAXParser). Si cette validation est réussie, un objet XML est créé. Les données de cet objet sont constituées de divers attributs (voir Figure-A II- 3). Enfin, l'objet XML est renvoyé au module «ImporManager» pour l'étape suivante.

4.3.3 TBO

Cette section, présente la structure et les décisions d'implémentation du TBO qui est développé est déployé séparément du code source de l'OIMD.

4.3.3.1 **Implémentation**

Dans cette première version du TBO, deux fonctionnalités sont implémentées. La Figure11 présente les traitements post-import exécutés par ce TBO : 1) Le TBO vérifie les métadonnées du document importé et interroge la Docbase (à l'aide de requêtes DQL) pour extraire la liste de contrôle d'accès à appliquer. S'il existe déjà une autre liste de contrôle d'accès sur le document, celle-ci est écrasée; 2) Le TBO permet également de créer la structure de dossiers si elle n'existe pas encore. Enfin, il est nécessaire de déplacer le document au dossier correspondant (c.-à-d. en fonction de l'ID).

Le code de ce module consiste à exécuter des requêtes que ce soit pour extraire ou enregistrer des données dans le serveur de contenu. La Figure 13 décrit un exemple d'une méthode

d'extraction des informations d'un utilisateur (c.-à-d. à l'aide du Nom et Prénom d'un ID de McGill valide).

```
private void getFirstLastNameAndMcgillId() throws DfException {
  IDfQuery query = new DfQuery();
  String validMcGillID = "";
  String queryString = "" +
             "SELECT first_name, last_name, mcgill_id  "+
             "FROM dm_dbo.BANNER_STUDENT_EMAILS_VIEW "  +
             "WHERE mcgill_id ='" + SD_MCGILL_ID + "'";
       query.setDQL(queryString);
       IDfCollection coll = query.execute(session,
                            DfQuery.DF_READ_QUERY);
       while (coll.next()) {
                    sdFirstName1 = coll.getString("first_name");
                    sdLastName1 = coll.getString("last_name");
                    validMcGillID = coll.getString("mcgill_id");
       }
       closeCollection(coll);
  if (validMcGillID.isEmpty()) {
     sdFirstName1= this.getString(AttributsAndLoggerMsgs.ATTR_SD_FIRST_NAME);
     sdLastName1= this.getString(AttributsAndLoggerMsgs.ATTR_SD_LAST_NAME);
             }
}
```

Figure 13 — Exemple de requête DQL en code Java

4.3.3.2 **Déploiement**

Lors de cette étape, le code TBO est déployé et configuré dans le référentiel qui l'utilise et il est chargé dynamiquement par les classes DFC selon les besoins de n'importe quelle application appelante. Les attributs TBO identifient : le type de module; sa classe d'implémentation; les interfaces qu'il implémente; et les modules dont il dépend. D'autres attributs fournissent des informations sur la version, une description des fonctionnalités du module, la version minimale de la machine virtuelle Java sous laquelle le module peut être exécuté et des informations sur le développeur.

Chaque référentiel possède un dossier System (c.-à-d. Cabinets), qui contient un dossier de niveau supérieur nommé «Modules». Par exemple, les TBOs sont installés sous « / System /

Modules / TBO ». La Figure 14 présente une capture d'écran du répertoire des modules dans l'environnement de développement (DEV_CMS_ADMIN) à McGill, tel que visualisée dans l'application Webtop.

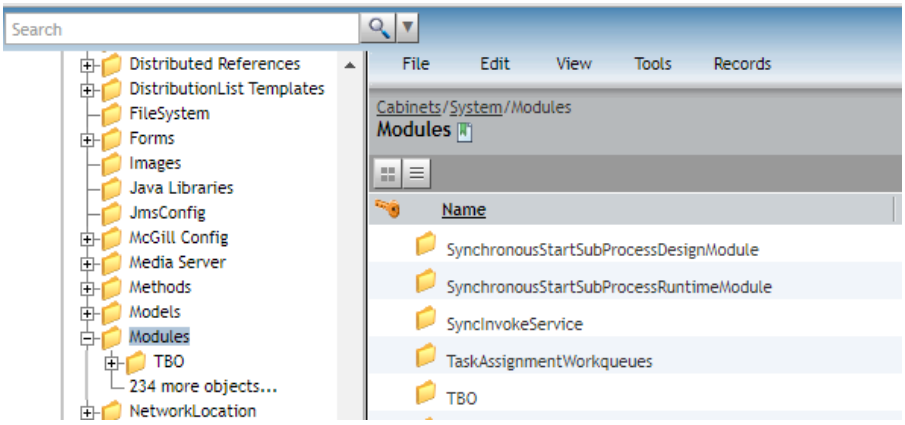


Figure 14 — Le sous-dossiers de modules représentant les types TBO.

Le module lui-même réside dans « / Système / Modules / [type de module] / [nom du module] », qui est dans ce cas « / Système / Modules / TBO / dmc_bulk_Import ». La Figure 15 montre les fichiers JAR d'interface et l'implémentation du TBO qui se trouvent dans le dossier racine du module.

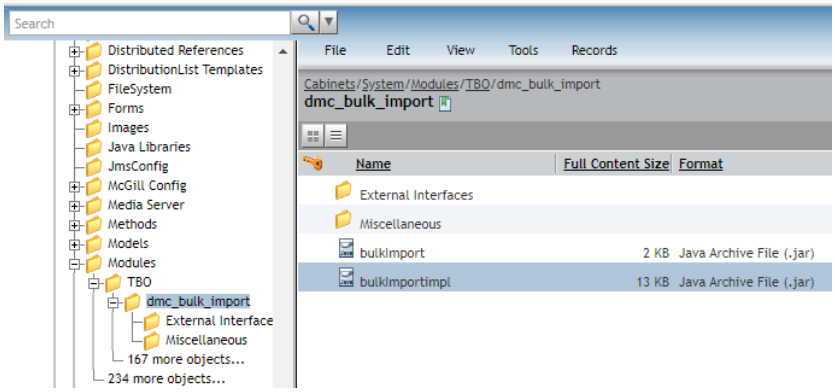


Figure 15 — Les fichiers JAR d'interface et d'implémentation TBO.

4.4 Les données d'essais

Les données utilisée afin d'évaluer la performance de l'OIMD prototypé dans cette section sont basée sur la demande de service ITSM présentée sous la forme d'un BNS (c.-à-d. un

énoncé des besoins de l'entreprise), par exemple l'école des sciences d'infirmières, tel qu'illustré à la Figure-A III- 1.

Cette demande comportait deux fichiers. Un fichier Word (Figure-A III- 2) et un fichier Excel comportant les conventions de dénomination et les acronymes de type d'élément à utiliser. Le Tableau 3 présente la convention de nommage exigée par ce client (voir le Tableau-A III- 1). L'ensemble de données qui a été importé dans le serveur de contenu pour cet essai de l'école des sciences d'infirmières est de 3190 fichiers. Tous ces fichiers étaient au format PDF et la taille des documents importés a été d'environ 1 Go au total.

Tableau 3 — La convention de nommage des documents importés

La convention de nommage	Descriptions	Exemples
Un dossier	Lastname, Firstname - ID	Smith, Jane - 260000567
Les enregistrements	ID - item type and detail - date in format yyymmdd	260000567 - STG-DEC 20170909 260000567 - STG-1APP 20170927 260899999 - LIC-CNO 20170525 260899999 - CLEV-FN 530

4.5 Vérification et validation

Dans cette section, l'accent est principalement mis sur la vérification et validation du prototype expérimental d'application OIMD. Les tests ont été exécutés dans l'environnement de développement (DEV_CMS_ADMIN) et l'environnement d'intégration (INT_CMS_ADMIN) de McGill. Deux stratégies d'évaluation, discutée ci-dessous, ont été effectuées.

4.5.1 Tests de boite blanche

Le test de boîte blanche est très efficace pour détecter des défauts de logiciel. C'est un processus d'assurance qualité logicielle qui consiste à entrer des données d'essais dans le système et à vérifier comment les énoncés du code source traitent ces données. L'idée principale est de tenter d'exécuter tous les chemins de l'algorithme. Le test de boîte blanche

est considéré comme une méthode de test pouvant être utilisée pour vérifier si l'implémentation du code source respecte la conception prévue. Dans ce projet on a utilisé le cadriciel JUnit 4, la Figure 16 offre une vue sur les classes des tests unitaires implémenté, ces tests appellent les méthodes de chaque unité et valident celle-ci lorsque les paramètres requis sont passés et dont la valeur de retour est celle attendue, la Figure-A III- 1 présente une saisie d'écran de l'exécution des tests unitaire dans l'IDE Éclipse.

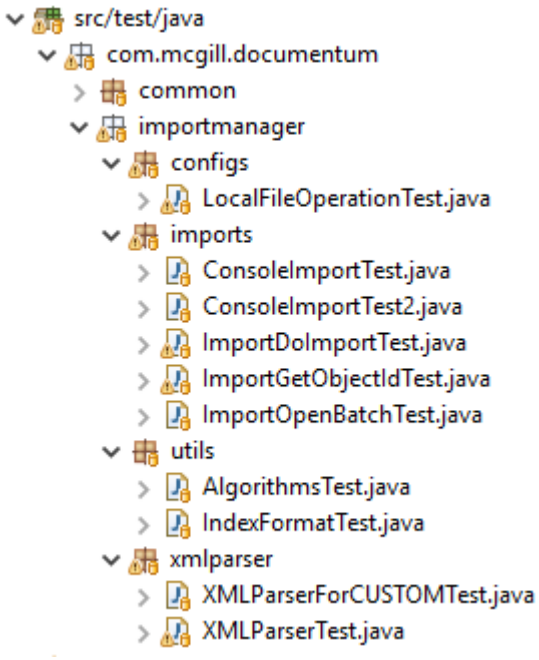


Figure 16 — Vue sur la hiérarchie du code source des tests

4.5.2 Test de boite noire

Le test de boîte noire est une technique de test basé sur les exigences de sortie et sans aucune connaissance du codage du programme. L'objectif est de vérifier dans quelle mesure le composant est conforme aux exigences. Le test de boîte noire s'assure que l'application fonctionne comme prévu une fois que tous les modules ont été intégrés. Des essais sont créés et exécutés dans le portail de test TestRail. La Figure-A III- 5 et la Figure-A III- 2 présentent une saisie d'écran de l'exécution des tests unitaire dans TestRail.

4.6 Conclusion

Ce chapitre a présenté la conception et la mise en œuvre d'une solution modulaire. La stratégie de développement a consisté à utiliser de petits modules de composants pour fournir des fonctionnalités spécifiques au prototype logiciel. Cette implémentation, tirée de l'enseignement du génie logiciel, peut améliorer : l'évolutivité, la maintenabilité, la réutilisabilité, l'extensibilité et la portabilité de l'application. Ce chapitre a aussi présenté avec plus de détails les données d'essais. La dernière section a mis l'accent sur les deux stratégies d'évaluation (c.-à-d. test boîte blanche et test boîte noire) pour vérifier et valider le prototype d'application.

CHAPITRE 5

CONCLUSION

Les principaux objectifs atteints par ce projet de recherche appliquée de 15 crédits sont :

- Mettre la base du cadriciel « McGill CMS Framework » qui supposer fournit du code source pouvant être réutilisé afin de résoudre les problèmes rencontrés lors de la manipulation de quantités massives de contenu à charger dans Documentum;
- Réalisation d'une interface de ligne de commande (OIMD) permettant de fournit un processus structuré pour définir, valider et importer des documents dans le référentiel DCTM;
- Satisfaire à une exigence de réussite au programme de maitrise en génie logiciel à l'école de technologie supérieure, dirigée par le professeur Alain April.

Dans le cadre de ce projet, nous avons tenté de résoudre ces problèmes en concevant un prototype expérimental qui utilise un cadriciel conçu spécifiquement pour la gestion de contenu de l'Université McGill. La stratégie de développement a consisté à utiliser de petits modules pour s'assurer d'une bonne modularité et maintenabilité. Cette expérimentation démontre qu'un cadriciel flexible va permettre à l'équipe CCS CMS Developer d'adapter facilement ce cadriciel en fonction de leurs besoins spécifiques et évolutifs.

L'OIMD expérimenté dans ce projet de recherche a été utilisé avec succès dans les environnements de développement (DEV) et d'intégration (INT). Il offre toutes les fonctionnalités planifiées au début du projet.

ANNEXE I

CONCEPTION DE l'OIMD

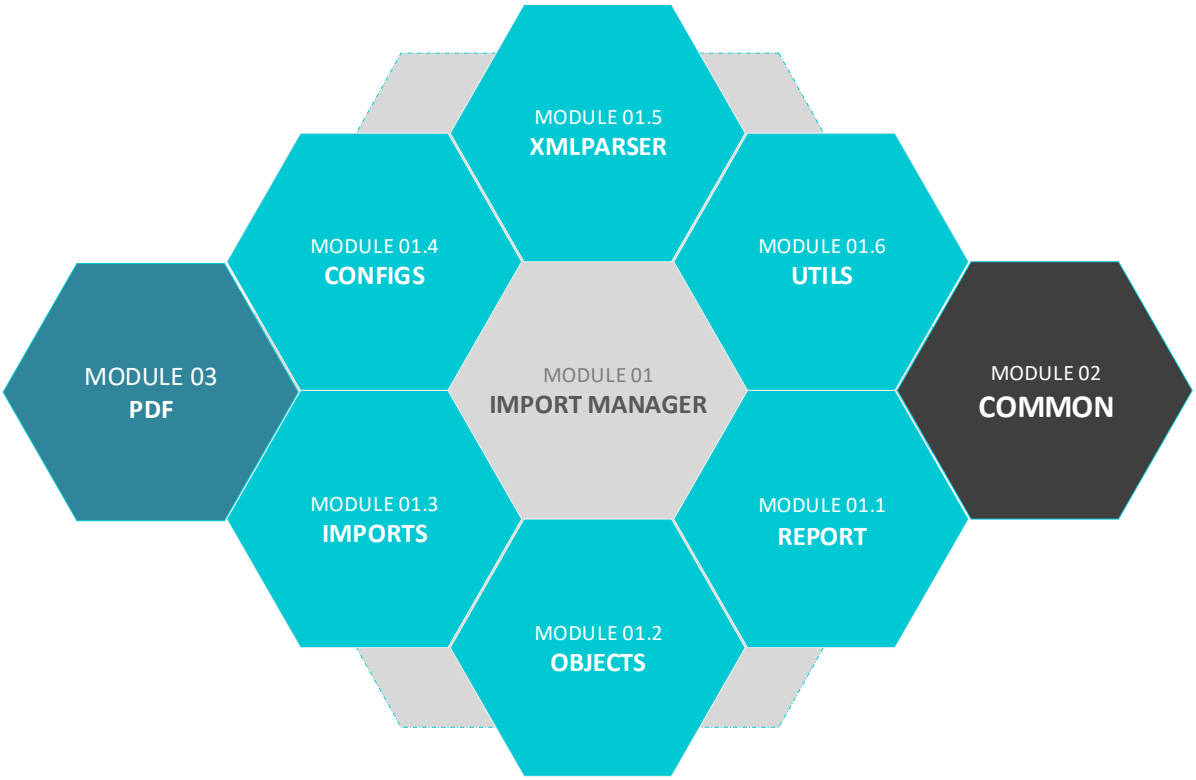


Figure-A I- 3 : Vue en modules de l'OIMD

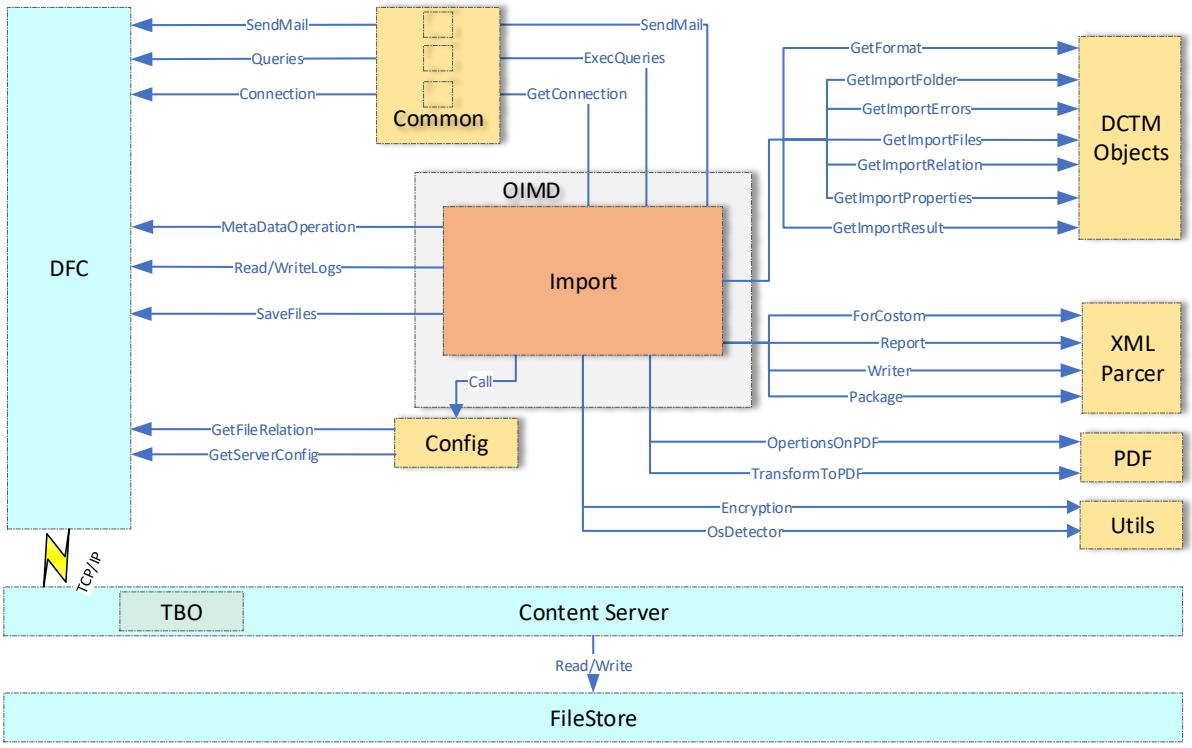


Figure-A I- 4 : Les interactions des différents composants de l'OIMD

ANNEXE II

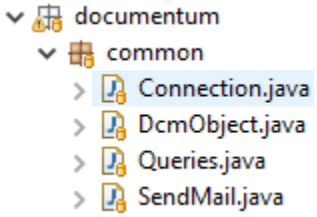
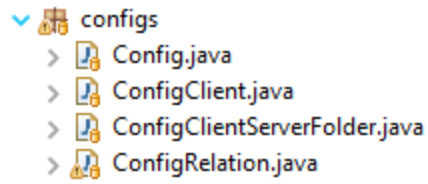
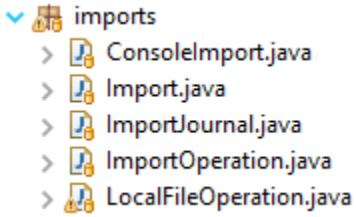
IMPLÉMENTATION

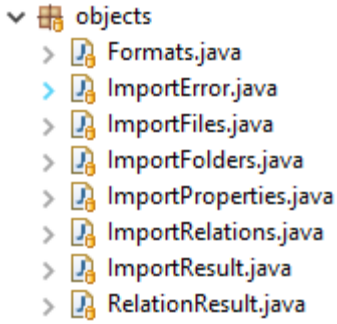
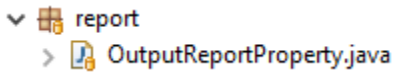
Tableau-A II- 1 : Les outils utilisés pour la réalisation de l'OIMD

Technologie	Description
Éclipse	Éclipse est un environnement de production de logiciels libres qui soit extensibles, universels et polyvalents, en s'appuyant principalement sur Java.
DFC	DFC fournit une infrastructure orientée objet pour accéder aux fonctionnalités du serveur de contenu. Aussi elle expose le modèle d'objet DCTM en tant que bibliothèque client orientée objet que les applications de gestion de contenu peuvent utiliser.
JUnit	JUnit est un Framework de test unitaire pour le langage de programmation Java.
TestRail	TestRail [3]est un outil de gestion des tests permettant aux équipes de gérer et de suivre leurs plans de tests logiciels.
Java	Java est un langage de programmation orienté objet.
XML	XML est un langage de balisage pour stocker et transporter des données.
CMD	Cmd ou l'invite de commandes est un logiciel d'interprétation des commandes DOS.
Composer	Composer est construit en utilisant la plateforme Éclipse. c'est une application pour les artéfacts Documentum. Il est conçu pour développer, construire et déployer des projets Documentum dans un référentiel DCTM.
DARDePloyer	DARDePloyer est utilisé pour installer dans le référentiel DCTM tous les artéfacts Documentum (Jobs, méthodes, modules, cycles de vie, types personnalisés, etc.) qui sont créés dans un projet Composer et intégrés dans un fichier DAR.

[3] TestRail est un logiciel Web de gestion de cas de tests pouvant être intégré à JIRA. TestRail aide les testeurs, les développeurs et les chefs d'équipe à gérer, suivre et coordonner efficacement les efforts de test des logiciels, le tout à partir d'une application Web centralisée et facile à utiliser. https://www.gurock.com/testrail/jira-test-management

Tableau-A II- 4 : Les responsabilités des modules de l'OIMD

Module	Descriptions	Responsabilités
	Ce module fournit plusieurs fonctionnalités prêtes à utiliser dans tout projet dans le même domaine d'affaires.	• Gestion de la connexion; • Exécution des requetés (DQL); • Envoi des messages électroniques; • DctmObject : ce module offre diverses fonctionnalités sur les objets DCTM.
	Ce module définit les données nécessaires pour exécuter des fonctionnalités spécifiques.	• Définissez la version du fichier importé; • Définissez la configuration locale du client; • Définissez la configuration du dossier du serveur; • Définissez la configuration de la relation.
	Ce module est responsable des actions d'importation.	• Importer en utilisant la console; • Gérer les opérations par lots; • Effectuer l'opération d'importation; • Créez un fichier journal; • Effectuer des opérations sur le système de fichiers.

	Ce module contient tous les objets Java nécessaires pour effectuer l'action requise pour l'importation d'un ou plusieurs documents.	• Il crée un objet sur le nom et l'extension d'un fichier; • Il fournit tous les messages d'erreur possibles; • Il fournit les informations nécessaires sur un fichier à importer; • Il fournit les informations nécessaires sur un dossier à importer; • Il fournit les opérateurs de comparaison des propriétés; • Il fournit les relations entre l'objet à importer et d'autres objets; • Il fournit des résultats sur les d'imports; • Il fournit les résultats d'une relation.
	Ce module est responsable de la mise en page des rapports générés.	• Extraire les valeurs des propriétés demandées dans le rapport du document injecté; • Signaler les succès de l'importation; • Signaler les erreurs d'importation; • Retourner l'entête du rapport d'importation; • Signaler les fichiers non importés; • Écrire le rapport dans un fichier; • Déplacer les fichiers de rapport.

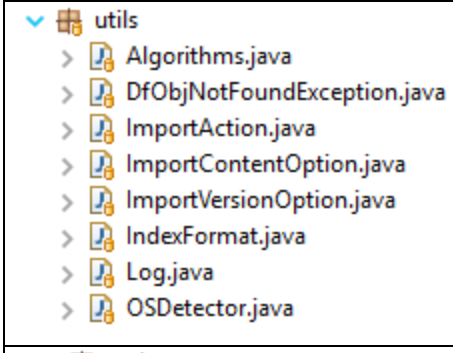
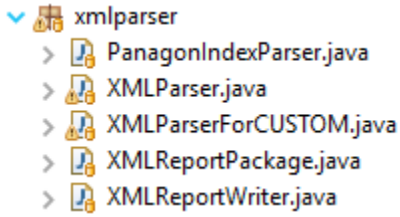
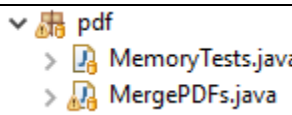
	Ce module est responsable de l'établissement de la connexion avec le Docbase.	• Crée une connexion; • Fermer une connexion; • Offrir la liste des référentiels active; • L'état de la connexion; • La journalisation.
	Ce module fournit des moyens d'analyser des documents XML.	• Transfère les informations du fichier XML à des objets Java durant l'exécution du programme; • Conserve les informations pour chaque paquet zip spécifique importé (total, succès, échecs); • Génère des rapports de journal XML contenant des informations pour certaines la variable activée (ex. no_lot_num).
	Ce module génère un fichier de rapport à partir de plusieurs fichiers PDF, un fichier sans gestion de la mémoire et un autre fichier avec l'utilisation de la mémoire.	Fusionner plusieurs fichiers PDF;

Figure-A II- 5 : Fichier de configuration d'import

```
<?xml version="1.0" encoding="UTF-8" ?>
<request        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.mcgill.ca/documentum/importmanager generic.xsd"
                xmlns="http://www.mcgill.ca/documentum/importmanager">
  <source>
        <content_folder>/opt/documentum/ContentShare/ AdmissionDocument /01/content</content_folder>
        <xml_folder>/opt/documentum/ContentShare/AdmissionDocument/01/xml</xml_folder>
         <log_folder>/opt/documentum/ContentShare/ AdmissionDocument /01</log_folder>
         <log_file_name>ControleAD_1</log_file_name>
        <listxmlerr_files>ListADXmlErrFiles_1</listxmlerr_files>
        <instance>1</instance>
  </source>
  <docbase>
        <docbase_name>DEV_CMS_ADMIN</docbase_name>
        <user_name>user_name</user_name>
        <password>password</password>
        <serverside_date_format>MM/dd/yyyy HH:mm:ss</serverside_date_format>
        <request_rendition>true</request_rendition>
        <format><name>pdf</name>  <extension>.pdf</extension>   </format>
  </docbase>
  <import_config>
        <server_start_folder>/Dossier membre</server_start_folder>
        <create_folder>true</create_folder>
         <import_option>minor_version</import_option>
        <version_option>minor_version</version_option>
        <import_action>create</import_action>
        <delete_allversions>false</delete_allversions>
        <version_qualification>%object_type% where object_name='%object_name%'</version_qualification>
        <content_option>import_content</content_option>
        <object_type>admission_document</object_type>
        <file_format>pdf</file_format>
  </import_config>
  <after_import_config>
         <source>
                <success>
                        <new_content_folder>
                                /opt/documentum/ContentShare/AdmissionDocument/01/content_done
                        </new_content_folder>
                       <new_xml_folder>
                                /opt/documentum/ContentShare/ AdmissionDocument /01/xml_done
                        </new_xml_folder>
                </success>
                <failure>
                        <new_content_folder>
                                /opt/documentum/ContentShare/ AdmissionDocument /01/content_reject
                        </new_content_folder>
                        <new_xml_folder>
                                /opt/documentum/ContentShare/ AdmissionDocument /01/xml_reject
                        </new_xml_folder>
                </failure>
                <new_log_folder>
                        /opt/documentum/ContentShare/ AdmissionDocument /Rapport
                </new_log_folder>
        </source>
        <report>
                <mail_from>ImportAdmissionDocument@mcgill.ca</mail_from>
                <mailto> ccscmsdevelopers@campus.mcgill.ca</mailto>
                <mail_subject>Import status</mail_subject>
                <enable_xml_log>true</enable_xml_log>
        </report>
  </after_import_config>
</request>
```

ANNEXE III

LES ESSAIS DE TESTS

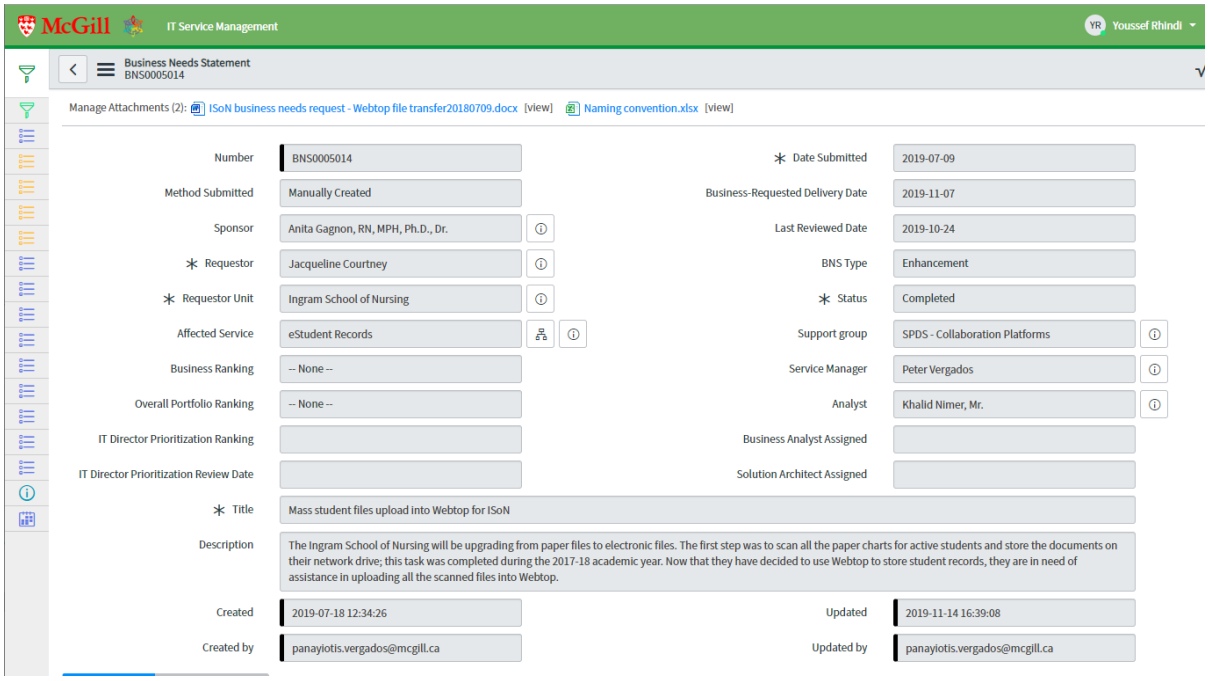


Figure-A III- 3 Business Needs Statement (demande de service)

P100 Business Need			
Business Need Title:	**Mass file upload unto Webtop**		
Business Requestor:	**Jacqueline Courtney**		
Requestor Unit:	Ingram School of Nursing		
Sponsor Name:	Dr. Anita Gagnon		
Date submitted (originally):	<2019 JUL 09>	Ticket #:	
IT Service Manager		PPM #:	

A. Business Need Description [Business]

1.1 Business Need Statement

The Ingram School of Nursing will be upgrading from paper files to electronic files. Our first step was to scan all the paper charts for our active students and store the documents on our network drive; this task was completed during the 2017-18 academic year. Now that we have decided to use Webtop to store our files, **we are in need of assistance in uploading all the scanned files into Webtop.**

1.1.1 Current Situation

Currently, are staff are using both paper and electronic files. Once we complete our Webtop training (July July 10th and July 26th), we will only use Webtop (paper files will no longer be created).

The administrative staff in the Nursing Student Affairs Office (NSAO) as well as the Program Directors and Assistant program directors have access to each file (12 people and professors who made ad hoc requests). Each person makes the request to a staff member within NSAO to obtain the file. Once they are finished consulting with the file, they hand it back to the NSAO staff member for filling.

By virtue of having each file scanned into Webtop, we will have a secure method to house all our student files. We will be able to do away with our current, inefficient process for requesting files (because all those who have been granted access will be able to access them directly**). Furthermore, having the scanned files uploaded onto Webtop will save us from having to have our staff upload each document individually.**

1.1.2 Business Needs

The ISoN needs a fast and efficient way to upload the scanned records of all our current students (currently housed on our network drive) into Webtop so that we can become paperless.

1.1.3 Desired Key Functionality (Optional)

- **Reads the folder/file names (which each include the McGill ID number) and assigns the to the appropriate student record on Webtop.**

Additional information:

- **929 student folders (741 undergrad; 188 grad)**
- **3125 files (approx.)**
- **All files are in PDF format**

File sizes (total approx. 1 GB)

- **BNI folders: 92 MB**
- **BScN folders: 514 MB**
- **Gr Certificate NP: 93.7 KB**
- **Gr Diploma NP: 5.77 MB**
- **Masters: 272 MB**
- **PhD: 96.2 MB**
- **Post Doc & Research Trainees: 27.2 MB**
- **Special NP: 2.36 MB**

1.1.4 Business Need Complexity Evaluation

Section filled out by the Portfolio Manager.

Complexity Criteria:	**True/False**
Is Development required on more than 1 system, OR	
Is this a new technology or Service for McGill, OR	
Does it require more than 20 person days of IT effort, OR	
Does the need require new integration work, OR	
Does the need require non-existing data, OR	
Does the need impact major security concerns?	

If all are “NO” then Technical Complexity is FALSE (F)	

< Portfolio Manager can add further comments or explanation about how the complexity was determined.>.

Figure-A III- 4 Business Needs Statement (Word file)

Tableau-A III- 2 Type d'objet acronymes utilisés

Acronymes	Descriptions
IFT	Inter or Intra Faculty Transfer
IUT	Inter University Transfer
T/C	Transfer credits
LOA	Leave of absence requests and supporting documents
CLEV-MT	Clinical Evaluations –midterm
CLEV-FN	Clinical Evaluations – final
Web	Web forms
FAA	Financial Aid, Awards, Prizes, Scholarships
PLAN	Course of study, modified course of study, study plan
STG	Standings
READ	Readmission
ADM	Admission
LIC	Licensure documents
PR REP	Progress Reports
STUDY	Study Away
DEC	Decision
CORR	Correspondence
EMAIL	Email
###	Three digit course number for each course evaluation
1-APP	First appeal
2-APP	Second appeal
3-APP	Third appeal
WD	Withdrawal
CNO	Canadian Nurses of Ontario
OIIQ	Ordre des infirmières et infirmer du Québec
CRNBC	Canadian Registered Nurses of BC
CFGNS	Visa screen

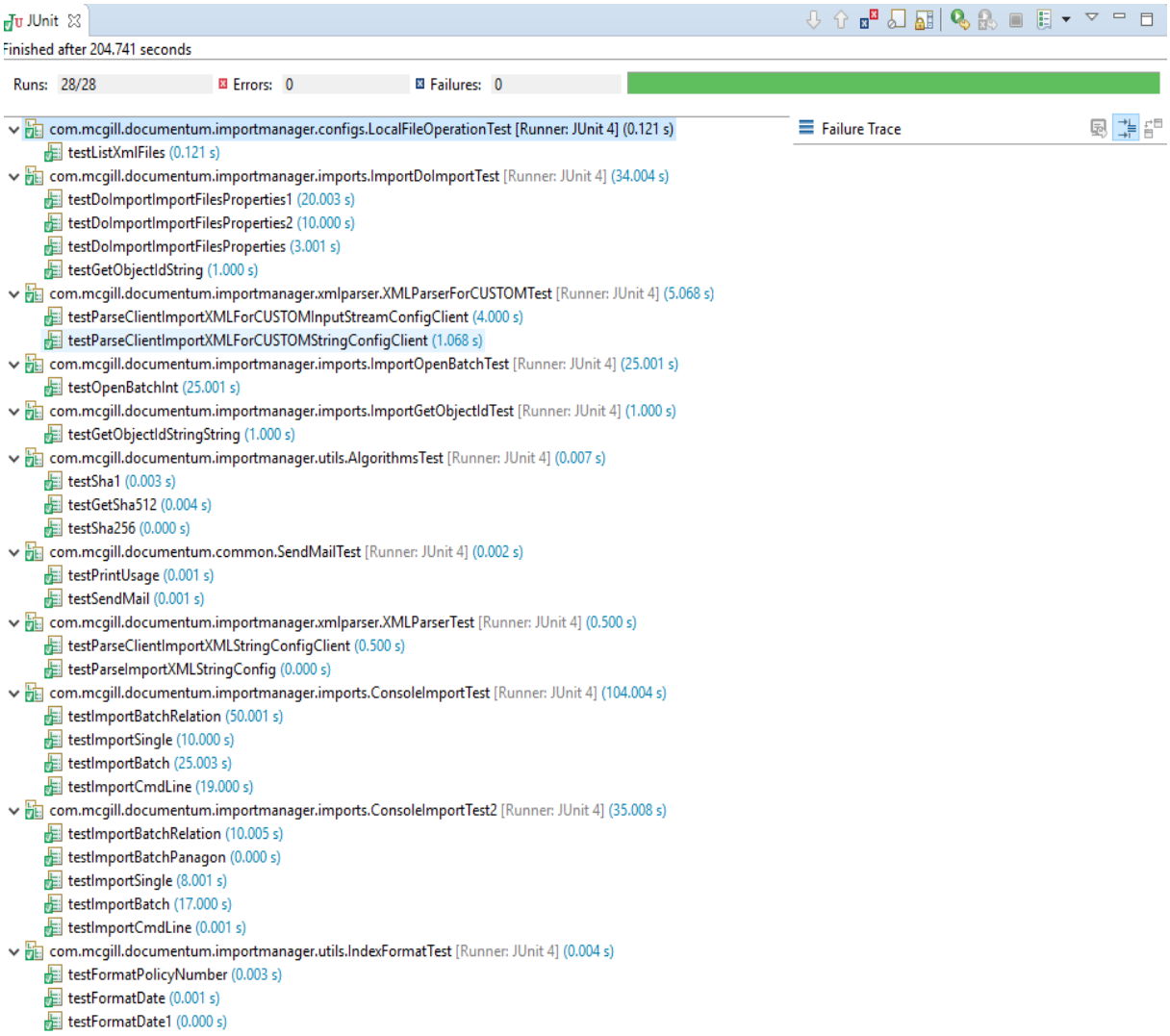


Figure-A III- 5 : Exécution des tests unitaires dans Éclipse

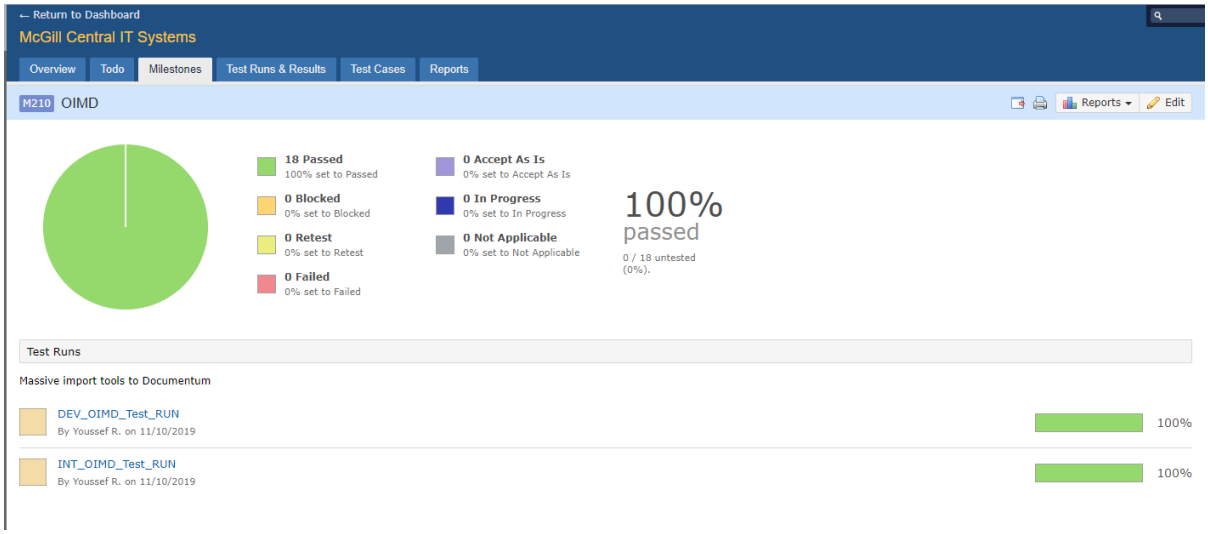


Figure-A III- 6 : Les essais de tests pour les environnements DEV et INT

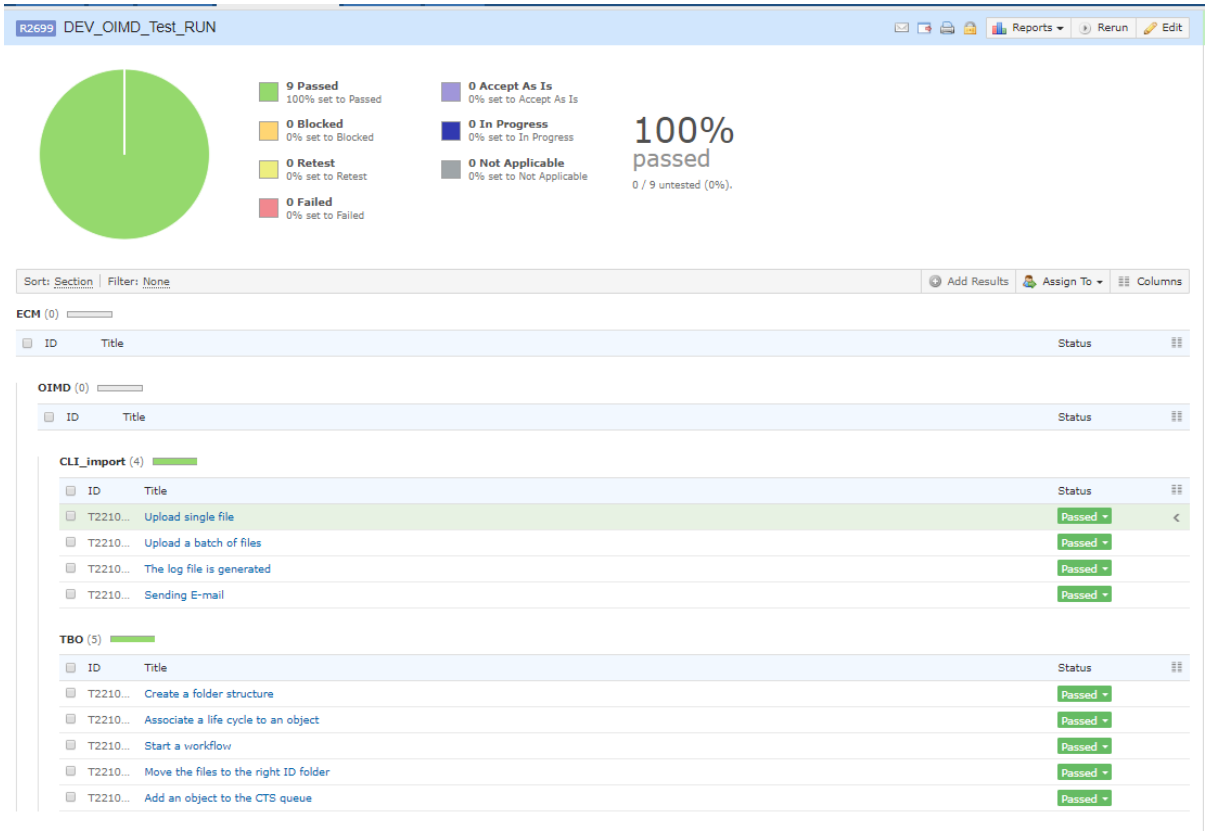


Figure-A III- 7 : Les tests fonctionnels dans TestRail pour les environnements DEV

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Abdullah, M. F., & Ahmad, K. (2013). The mapping process of unstructured data to structured data. Dans *2013 International Conference on Research and Innovation in Information Systems (ICRIIS)* (pp. 151-155). doi : 10.1109/ICRIIS.2013.6716700

Corporation, E. (2011). EMC Documentum Business Object Framework (pp. 20).

Documentum, I. (2006). Documentum 5 Architecture: A Technical Overview (pp. 89) : https://developer-content.emc.com.

Fayad, M., & Schmidt, D. C. (1997). Object-oriented application frameworks. *Communications of the ACM, 40*(10), 32-38.

Kumar, P. (2010). *Documentum 6.5 Content Management Foundations*. Packt Publishing.

McGill. (2019a). Video @ McGill McGill IT Knowledge Base : McGill.

McGill. (2019 b). Yammer - McGill's social network. McGill IT Knowledge Base : McGill.

Päivärinta, T., & Munkvold, B. (2005). *Enterprise Content Management : An Integrated Perspective on Information Management*. doi : 10.1109/HICSS.2005.244

Parag. (2019). Documentum ArchitectureParag Doshi's ECMNotes | Parag Doshi's ECMNotes.

Tyrväinen, P., Päivärinta, T., Salminen, A., & Iivari, J. (2006). *Characterizing the evolving research on enterprise content management* (Vol. 15). doi : 10.1057/palgrave.ejis.3000648