Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

Projet de diplôme

INFOSCIENCE

Sylvain Egger

Département	Technologie de l'information et de la communication
Filière	Informatique
Mots clés	Python, Infoscience, Curator, gestion bibliographique, MARC XML, WebService

Du 10 septembre au 14 novembre 2007

Sous la responsabilité de

Omar Abou Khaled *(EIA-FR - TIC)*
Houda Chabbi Drissi *(EIA-FR - TIC)*
Pierre Crevoisier *(EPFL)*
Gregory Favre *(EPFL)*

Expert

Jean-Yves Le Meur *(CERN)*
Christine Vanoirbeek *(EPFL)*

PRÉFACE

Introduction

Ce rapport décrit mon travail de diplôme, réalisé à l'École Polytechnique Fédérale de Lausanne (EPFL), dans le cadre de mes études au sein de l'École d'Ingénieurs et d'Architectes de Fribourg (EIA-FR).

Le but de ce travail est de développer et d'adjoindre des modules d'aide à la maintenance d'Infoscience, plateforme de gestion de bibliothèque numérique de l'EPFL, afin de faciliter essentiellement le travail des bibliothécaires.

Conventions

1. Il existe également des notions de personnes, par souci de clarté, voici ce qu'elles signifient :
    - Utilisateur : utilisateur du site du Curator, cela peut être autant un administrateur, un curateur qu'un responsable de laboratoire
2. Tous les mots suivis d'un chiffre en exposant comme ceci : « Exemple[1] » sont référencés en fin de document.
3. Les éléments encadrés, comme ci-dessous, correspondent à du code informatique

```
def hello():
    print "Hello World!"
```

Contenu

- 1 Rapport technique
- 1 Dossier d'annexes, adjoint au rapport technique
- 1 CD

Contacts

Responsables à l'EIA-FR

Omar Abou Khaled
EIA-FR / TIC
Bd de Pérolles 80 – CP 32
CH – 1705 Fribourg
Téléphone : +41 26 429 65 89
Fax : +41 26 429 66 00
Email : Omar.Aboukhaled@hefr.ch

Houda Chabbi Drissi
EIA-FR / TIC
Bd de Pérolles 80 – CP 32
CH – 1705 Fribourg
Téléphone : +41 26 429 65 60
Fax : +41 26 429 66 00
Email : Houda.Chabbi@hefr.ch

Responsables à l'EPFL

Pierre Crevoisier
EPFL / PL-DIT
MA C1 644 – CP 121
CH – 1015 Lausanne
Téléphone : +41 21 693 49 94
Email : Pierre.Crevoisier@epfl.ch

Gregory Favre
EPFL / PL-DIT
MA C1 622 – CP 121
CH – 1015 Lausanne
Téléphone : +41 21 693 22 88
Email : Gregory.Favre@epfl.ch

Experts

Jean-Yves Le Meur
CERN / IT-UDS
CH - 1211 Genève 23
Téléphone : +41 22 767 47 45
Email : Jean-Yves.Le.Meur@cern.ch

Christine Vanoirbeek
EPFL / IC CGC-GE
BC 102
CH – 1015 Lausanne
Téléphone : +41 21 693 25 75
Email : Christine.Vanoirbeek@epfl.ch

Etudiant

Sylvain Egger
Les Cornettes 10
CH – 1482 Cugy
Téléphone : +41 79 360 86 64
Email : Sylvain.Egger@gmail.com

REMERCIEMENTS

Ce travail de diplôme est le fruit de trois longues années d'études. Il n'aurait jamais pu se réaliser sans l'aide de nombreuses personnes ! En essayant d'en oublier aucune, voici une liste de personnes que je tiens à remercier.

Tout d'abord mes parents, qui m'ont soutenu tout au long de mes études. Rien n'aurait été possible sans eux !

Plus professionnellement, j'adresse un grand merci à Omar Abou Khaled qui m'a offert l'occasion d'effectuer ce travail hors EIA-FR. Et dans la continuité, je remercie Pierre Crevoisier et Gregory Favre qui m'ont très bien accueilli et encadré. Plus précisément Pierre Crevoisier pour ses conseils et son suivi du projet. Quant à Gregory Favre pour les conseils et aides techniques durant le projet. Il a également contribué à la relecture de ce document.

Je n'oublie évidemment pas mes collègues de classes qui tout au long de ces trois années m'ont permis de venir aux cours avec le sourire et la motivation ! L'ambiance et la solidarité de notre classe est un merveilleux plus lors d'études de ce niveau.

Enfin, je remercie Houda Chabbi Drissi qui a supervisé mon projet en plus d'Omar Abou Khaled. Son suivi et ces conseils sur le présent document ont été très précieux.

INDEX

1 INTRODUCTION

1.1 L'EPFL

L'EPFL est l'une des deux Écoles Polytechnique fédérales en Suisse. Comme sa sœur zurichoise, l'ETHZ, elle a trois missions: la formation, la recherche et la valorisation au plus haut niveau international. Associées à plusieurs instituts de recherche spécialisés, les deux Écoles forment le domaine des EPF, qui dépend directement du Département fédéral de l'intérieur.

L'EPFL est une université technologique qui accueille aujourd'hui près de 6.000 étudiants, dont plus de 1'000 doctorants. 3'000 chercheurs et scientifiques y développent des enseignements et des recherches en mathématiques, physique, chimie, management de la technologie, sciences de l'ingénieur, sciences de la vie, architecture.

Dans le nouvel environnement universitaire de la réforme de Bologne[1] et de la globalisation de l'enseignement supérieur, et s'appuyant sur sa bonne réputation scientifique, l'École s'est fixé l'objectif de se situer parmi les meilleures en Europe et dans le monde, pour la qualité de sa recherche comme de son enseignement.

En 2004, pour faire suite à une demande de la Direction de l'École de remplacer et renouveler le rapport scientifique produit entre 2002 et 2004, le service informatique et la bibliothèque ont reçu le mandat de mettre en place et tester une solution destinée à collecter et valoriser le patrimoine scientifique de l'EPFL.

Src : http://presentation.epfl.ch

1.2 Infoscience

1.2.1 Contexte

Après 6 mois consacrés à la comparaison de solutions existantes, de test et d'installation d'un outil logiciel adapté, Infoscience, à la fois service et serveur de l'archive institutionnelle était ouvert.

Le logiciel open source CDS Invenio (anciennement CDSware[2]), développé par le Centre Européen pour la Recherche Nucléaire (CERN), a été choisi pour gérer la plateforme technique servant de support à ce service.
À noter que tout au long du rapport, CDS Invenio sera abrégé par l'appellation simple « Invenio »

En septembre 2006, Infoscience présente le profil de plus de 1'200 chercheurs et les publications de 129 laboratoires, sur les 240 qui composent l'EPFL. 26'000 publications y sont signalées, 10'000 fulltexts y sont stockés, dont l'ensemble des thèses de l'EPFL. De plus, Infoscience est destiné à devenir l'interface d'interrogation du catalogue collectif des bibliothèques de l'EPFL.

C'est donc une plateforme de services liés au patrimoine scientifique humain (constitué de profils des personnes), produit (constitué des publications scientifiques et pédagogiques) ou acquis (constitué des documents reçus ou achetés) que vise à offrir Infoscience. L'ensemble de ces contenus, services et fonctions est décrit plus en détail dans la suite de ce document.

1.2.2 Vue globale

Infoscience est à la fois un outil et une plateforme de services

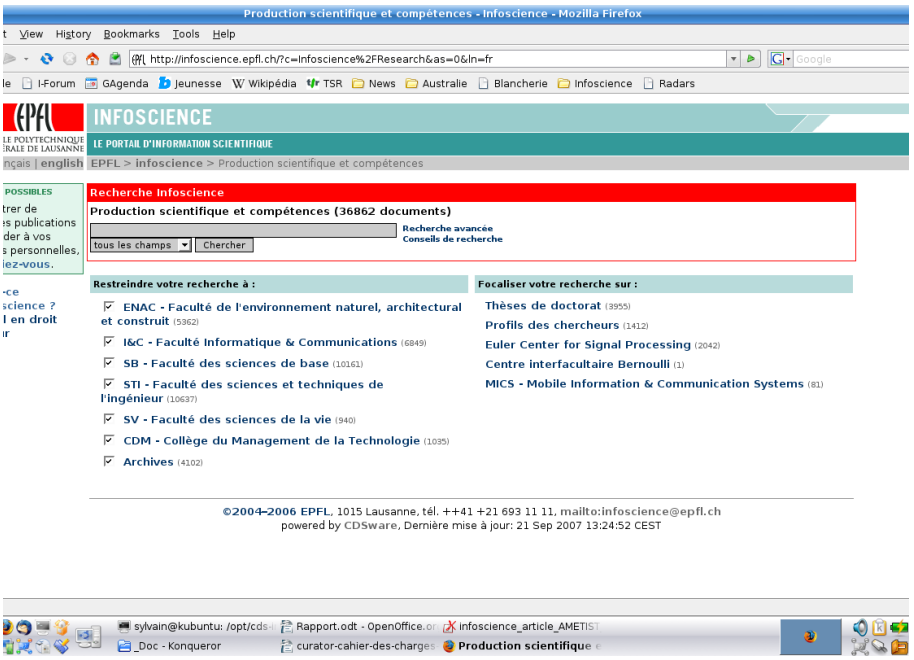


Figure 1 : Aperçu de la plateforme Infoscience

Infoscience fournit les contenus suivants :

- Références et textes intégraux des publications scientifiques
- description et identification des personnes et de leurs compétences
- collections de documents numérisés ou numériques
- le catalogue collectif des bibliothèques

Infoscience fournit les services suivants :

- Interfaces de recherche Google-like[3] et avancée
- réutilisation des données par les chercheurs eux-mêmes, selon leurs besoins
- indexation des publications et profils de personne dans Google et Google Scholar[4] pour une bonne visibilité sur le web
- interface de gestion de la qualité
- conseil en droit d'auteur
- intégration dans le moteur de recherche interne de l'EPFL

Src : http://empc51.epfl.ch/infoscience/infoscience, document AMETIST

1.3 Curator

1.3.1 Contexte

Infoscience est la plateforme de gestion bibliographique de l'EPFL. Dans le cadre de ce projet, un certain nombre de modules ont été développés afin de permettre de maintenir la qualité des données bibliographiques. Ces modules sont rassemblés sous le nom de « Curator ».

Dans le contexte actuel d'Infoscience, il n'existe que deux types de droits permettant la modification des notices. Le premier est celui d'**administrateur** qui permet de modifier n'importe quelle notice de n'importe quel laboratoire. Le second est celui de **personne accréditée au sein d'un laboratoire**. Si une personne est accréditée dans un ou plusieurs laboratoires, alors elle peut modifier les notices de ce (ou ces) laboratoire(s).

Curator propose les rôles suivants:

- administrateur général (maximum 2-3 personnes),
- curateur (un par faculté, soit actuellement six au total),
- responsable Infocience dans un laboratoire (un par labo),

Les **administrateurs** et les **curateurs** ont le droit **administrateur** pour l'ensemble des laboratoires. Les **responsables de laboratoires** n'ont que le droit **accrédité** pour les laboratoires dans lesquels ils sont actifs.

1.3.2 Vue globale

Curator est donc une suite de modules offrant différentes fonctionnalités. La liste de ces fonctionnalités selon le cahier des charges actuel du Curator est la suivante.

1 **Uploader un fichier de notices**
Mettre à jour des notices par lot en chargeant, à travers l'interface web, un fichier au format XML MARC 21[5] ou BibTeX[6].

2 **Gestion des doublons**
Permettre aux curateurs de traiter et supprimer les doublons. Un doublon est une paire de notices (parfois 3, voir 4...) se ressemblant très fortement.

3 Archivage d'un laboratoire

Après la fermeture d'un laboratoire, changer son statut d'actif vers archivé.

4 Définir le format d'autorité d'un nom d'auteur

Choisir le format de nom (généralement Nom, Prénom) qui doit être utilisé et associer les autres formes comme étant des synonymes.

5 Transférer les publications d'une personne

Transférer les notices d'une personne d'un laboratoire vers un autre.

6 Modifier le type d'une notice

Une notice a été insérée avec le mauvais type (article, papier de conférence, etc.), et celui-ci doit être changé.

7 Supprimer une notice d'un laboratoire

Une notice n'est plus valide et doit être supprimée d'un laboratoire.

8 Ajouter ou modifier un journal scientifique à la base de référence

Ajouter ou modifier une référence d'un journal scientifique (Journal.xml) connu du système Infoscience.

9 Ajouter un livre à la bibliothèque d'un laboratoire

Ajouter ou modifier les références d'un livre dans le contenu de la bibliothèque d'un laboratoire.

Src : http://empc51.epfl.ch/infoscience/Curator, Cahier des charges v1.0

1.4 Tâches

Le projet se découpe en deux phases importantes.

La première est de définir une architecture solide capable d'accueillir les différents modules du Curator.
Elle doit gérer la communication entre Infoscience et le Curator afin d'effectuer les traitements demandés par les différents modules. Il faut également que cette gestion de la communication soit générique afin que le Curator puisse s'assembler à d'autres plateformes similaires à Infoscience.

La seconde est le développement d'un module permettant de valider l'architecture. Ce module à été choisi en fonction du temps à disposition, il s'agit du module « Uploader un fichier de notices », voir point 1.3.2.

La description détaillée des tâches se trouve en annexe dans la spécification.

1.5 Structure du rapport

Ce rapport est composé de 5 parties principales. La première est l'introduction qui présente de manière générale la bibliothèque numérique de l'EPFL, Infoscience et Curator, l'outil directement impliqué dans ce projet. La seconde traite l'analyse du problème et propose une approche pour le résoudre. La troisième établit une stratégie de conception pour la réalisation du Curator. La quatrième fournit des informations sur la réalisation et les résultats obtenus et la dernière partie, la conclusion, propose des améliorations futures et donne une impression personnelle sur le sujet.

En fin de document se trouvent une webographie et un glossaire. Les annexes sont attachées à la suite de ce rapport.

2 ANALYSE

2.1 Introduction

La partie analyse de ce projet va se concentrer sur l'élaboration d'une architecture pour accueillir les modules du Curator. Cette analyse est découpée en deux parties. La première, plus restreinte, est l'analyse du Curator Beta. La seconde partie présente la nouvelle architecture qui a été mise en œuvre avec toutes les étapes pour arriver à cette structure finale.

2.2 Technologies utilisées

La question des technologies à utiliser ne s'est pas vraiment posée. Infoscience, Invenio ainsi que le Curator Beta étant implémenté avec Python[7], il va de soi que la suite va continuer dans cette optique. Cependant, il paraît judicieux de tout de même argumenter le choix de ce langage. En effet, le choix de Python avec le module Apache[8] mod_python[9] au lieu des plateformes habituelles J2EE ou .NET peut surprendre. Cependant, Python est un langage fournissant énormément de possibilités !

Quelques arguments…

Facilité d'apprentissage

Dans un environnement comme l'EPFL, les équipes de développement peuvent régulièrement changer. Les personnes qui intègrent le projet doivent être capables de développer le plus vite possible.

Langage multiparadigme

Python est aussi bien un langage orienté objet que procédural ou encore fonctionnel. Ceci rend agréable la possibilité d'utiliser l'objet seulement quand il est adapté et de coder de manière procédurale lorsque cela est plus simple.

Rapidité d'implémentation

Python est un langage concis. Aucun point-virgule, pas d'accolades pour les blocs d'instructions, pas nécessaires de définir des classes, etc. De plus, le grand nombre de fonctions déjà incluses aident beaucoup le développement rapide de fonctionnalités complexes. Le fait que Python soit un langage interprété évite le cycle de compilation.

Cependant, Python pêche sur certains points…

Lenteur

Python est donc un langage interprété, donc plus lent qu'un langage compilé.

Non standard

Il n'existe pas de standard ANSI pour Python. Cela pourrait devenir un problème avec le temps.

Python évolue sur une plateforme Apache, celle-ci devra être mise en place par la suite sur le serveur qui accueillera le Curator. Pour ce projet, Apache et Python ont été installés sur une station locale pour permettre le développement du Curator.

Du côté de la base de données, c'est MySQL[10] qui est utilisé pour Invenio, donc cette technologie va être conservé pour le Curator.

2.3 Synthèse du Curator Beta

Le Curator Beta est en fait un prototype du Curator final qui n'a jamais eu pour but de passer directement en version finale. Certes, le code qui y est présent pourra être réutilisé, mais la structure mérite d'être entièrement revue.

Voici un schéma général qui présente l'organisation des fichiers du Curator Beta :

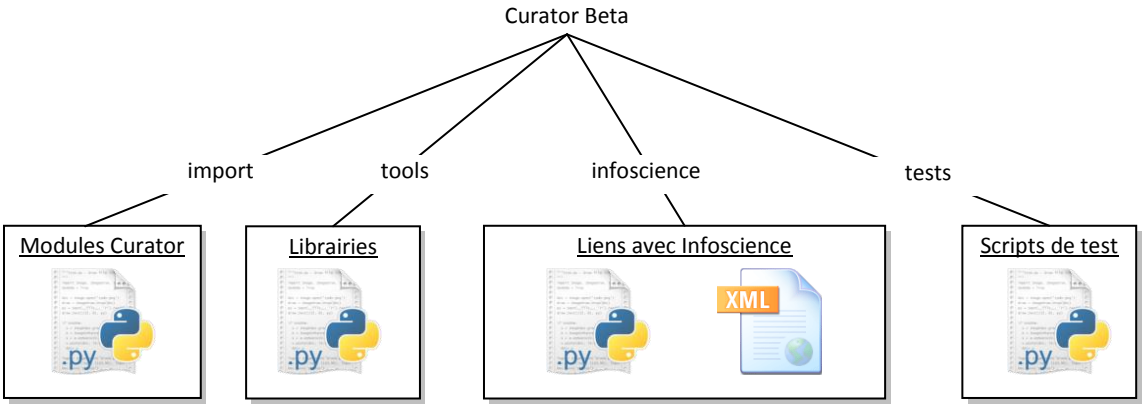


Figure 2 : Organisation des fichiers pour le Curator Beta

Cette organisation ne ressemble en rien à celle d'Invenio par exemple. L'organisation d'Invenio sera probablement celle adoptée pour le nouveau Curator et sera donc expliquée par la suite.

2.4 Nouvelle architecture

2.4.1 Introduction

La nouvelle architecture doit prendre en compte certains points importants. Ceux-ci sont :

- L'authentification du Curator sur Invenio afin de pouvoir récupérer des données et lui fournir des tâches à exécuter.
- Procurer aux modules différentes méthodes pour qu'ils puissent se greffer facilement sur le Curator.
- Définir le flux de communications entre Invenio et Curator. Le document en annexe « Synthèse sur le flux de communications entre Invenio et Curator » décrit ce point.

2.4.2 Cas d'utilisation

Selon le cahier des charges général établi par Pierre Crevoisier et Gregory Favre, disponible à l'adresse suivante → *http://empc51.epfl.ch/infoscience/Curator*, un diagramme de cas d'utilisation peut être ressorti.

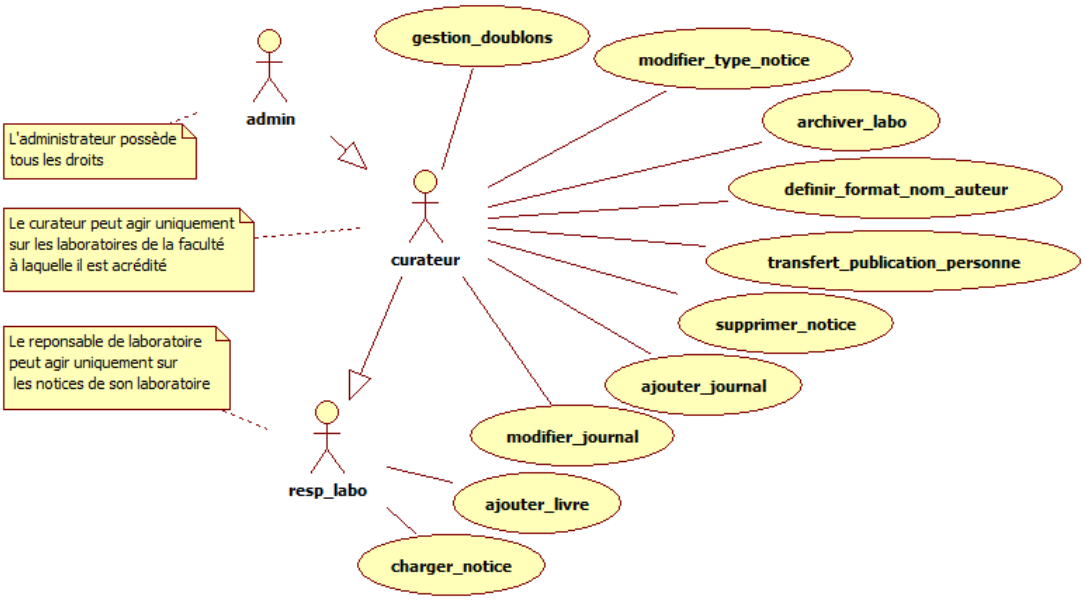


Figure 3 : Use Case du Curator

On peut y voir les trois rôles définis dans le cahier des charges. L'administrateur, le curateur et le responsable de laboratoire. Ceux-ci ont chacun différents droits décrits dans les notes ci-dessus.

Chaque cas d'utilisation décrit un des futurs modules du Curator. L'architecture doit supporter ce diagramme de cas d'utilisation.

2.4.3 Composants

Les composants du Curator sont, à quelques détails près, les mêmes que ceux d'Invenio. Nous avons besoin d'un serveur web pour faire fonctionner les scripts Python et d'une base de données pour sauvegarder certains paramètres comme le format de nom d'auteur principal avec ses synonymes ou encore les doublons faux-positifs.

Le Curator sera sur un serveur séparé d'Invenio. Principalement à cause de la charge de travail occasionnée par le module « Gestion des doublons » qui effectuera régulièrement une analyse des doublons présents dans la base Infoscience. Cette base de données étant tout de même conséquente *(plus de 60'000 publications à ce jour)* et sans cesse en expansion.
Le seul problème de déduplications nous forçant déjà à utiliser un serveur séparé pour des raisons de puissance de calcul, il n'est pas nécessaire d'argumenter plus pour justifier cette décision.

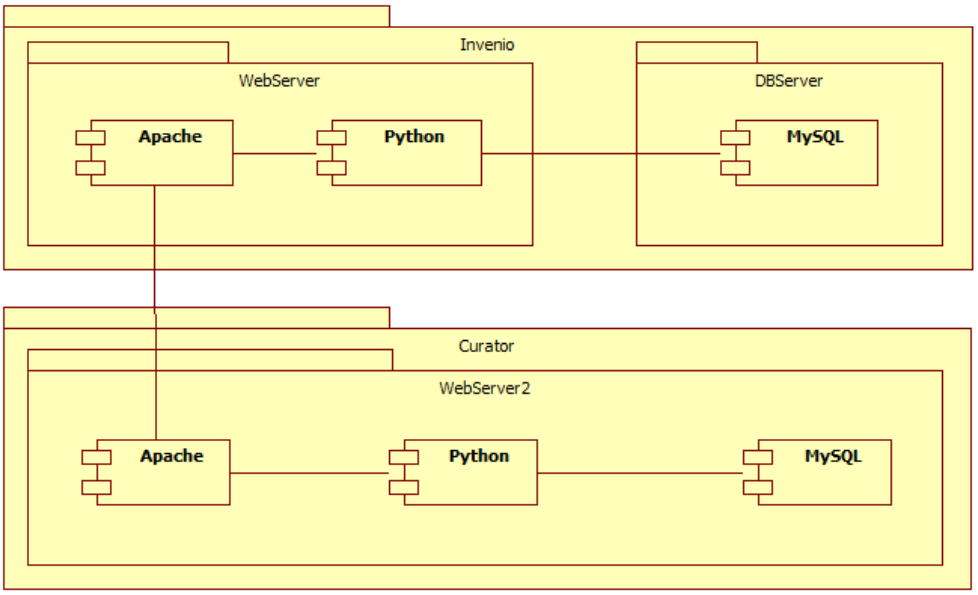


Figure 4 : Diagramme des composants

2.4.3.1 Choix de la base de données

La base de données sera utilisée par les modules du Curator et non par le noyau lui-même. C'est pourquoi une question intéressante s'est posée.

« Est-ce qu'une base de données unique devrait être utilisée ou est-ce que chaque module pourrait disposer de sa propre base. »

Ceci dans l'optique que peut-être un jour, quelqu'un souhaiterait utiliser une base de données SQL plutôt que MySQL.

Après une petite discussion avec Gregory Favre, la décision de n'utiliser qu'une seule base, de type MySQL, a été prise. Ceci simplifie les choses étant données que les modules qui vont être développé à l'EPFL par la suite utiliseront MySQL, comme sur Invenio.

2.4.4 Organisation générale

L'organisation générale des fichiers se présente de manière semblable à Invenio. Celle-ci reprend l'organisation de base du système Unix[11] avec les dossiers : bin, etc, lib, src, var, www.

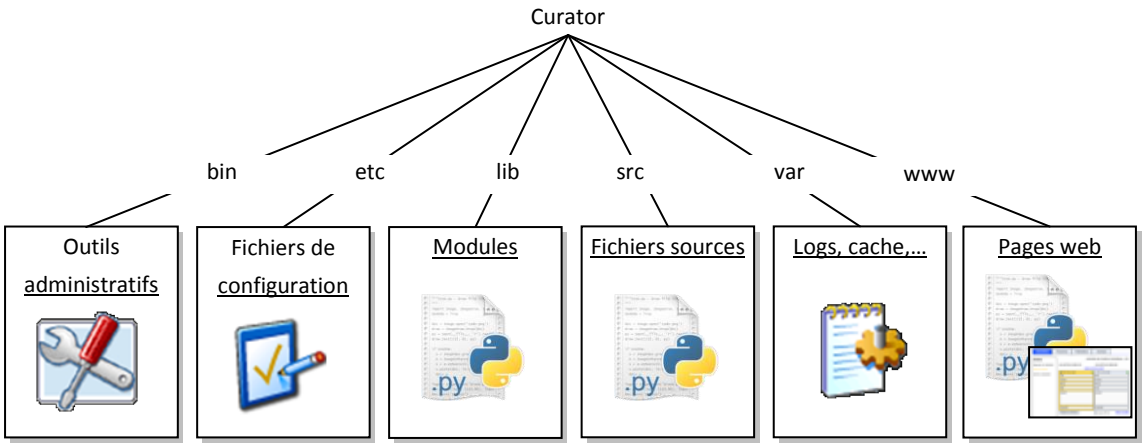


Figure 5 : Organisation des fichiers pour le Curator

- **bin**

  En règle général, le dossier « bin » contient divers outils d'administration du système ou de l'application en question.

- **etc**

  Ce dossier contient un ou plusieurs fichiers de configuration. On y trouve par exemple les paramètres spécifiques à Infoscience concernant le XML MARC.

- **lib**

  Ce dossier est le noyau des modules. C'est ici que se trouve les algorithmes de traitements.

- **src**

  Ce dossier contient les sources du Curator.

- **var**

  Ce dossier récolte les logs de tout ce qui se passe sur le Curator. Erreurs, avertissement, événements,…etc.

  Il pourrait également contenir des données mises en cache pour accélérer certains processus, ceci restant à définir spécifiquement par les modules.

- **www**
  Et enfin le dernier dossier qui accueille les pages web destinées à l'affichage pour l'utilisateur final.

Tous ces dossiers ne seront pas forcément utilisés. Cependant, le fait de respecter le standard Unix pour l'organisation des fichiers permet d'avoir une vue claire, y compris pour les personnes externes au projet, à condition de connaître le système Unix.

2.4.5 Méthodes spécifiques à fournir aux modules

L'architecture se doit de fournir certaines méthodes aux différents modules du Curator. Ces méthodes sont récurrentes à chaque module, c'est pourquoi il serait inadéquat, par un souci de redondance, de les développer indépendamment pour chaque module.

On peut citer les méthodes suivantes :

- **L'authentification**
  Fournit aux modules la possibilité de s'identifier sur Invenio afin d'effectuer des traitements *(voir méthodes suivantes)* sur les notices.

- **Le chargement de données**
  Une fois identifié, les modules souhaitent tous récupérer diverses données de la base Infoscience. Cette méthode fournit la possibilité d'effectuer ces requêtes selon différents paramètres d'entrée.

- **L'envoi de données**
  Semblable au chargement de données mais dans le sens inverse, cette méthode offre la possibilité d'envoyer à Invenio des ordres d'insertion, de mise à jour ou de suppression de notices.

- **Gestionnaire d'événement**
  Le suivi des événements n'est pas un besoin en soi, mais le fait d'avoir un fichier qui archive tous les événements spéciaux qui se passent sur le Curator est très utile en cas de problème quelconque.

Il n'est pas impossible que d'autres méthodes viennent s'ajouter à l'architecture de base selon les besoins. Cela pourrait arriver au cours du développement de futurs modules.

Cependant avec les modules prévus actuellement, les méthodes ci-dessus répondent parfaitement aux besoins.

2.4.6 Authentification

L'authentification pourrait paraître un problème anodin, mais il se trouve que c'est, au contraire, une tâche complexe de l'architecture du Curator. Le problème vient du fait que les modules du Curator devront effectuer des requêtes sur la base de données Infoscience et, du fait que le Curator sera sur un serveur séparé, les modules ne peuvent pas accéder directement aux APIs d'Invenio.
Si le Curator se trouvait sur le même serveur, Python nous permettrait d'utiliser les librairies d'Invenio, ce qui faciliterait grandement la tâche. Cependant, Curator sera sur un serveur séparé pour les raisons évoquées au point 2.4.3.
Pour ces raisons, il faut maintenant fournir une solution pour s'identifier sur Invenio et accéder à ces librairies qui nous permettent de faire des requêtes sur la base de données de notices telles que des insertions, modifications, suppressions et recherches.

2.4.6.1 Solution par intégration de Curator sur une nouvelle instance d'Invenio

Lors d'une séance au CERN *(voir PV n°6 du 17.10.07 en annexe à la fin du rapport)*, Jean-Yves Le Meur et Tibor Simko, respectivement responsable et architecte du projet Invenio, ont proposé l'idée d'implanter le Curator sur une instance d'Invenio totalement séparée de celle sur laquelle se trouve Infoscience.
Cette idée apporterait son lot de bonnes choses comme par exemple le fait que le Curator aurait directement accès aux librairies d'Invenio pour la recherche, l'authentification, l'insertion de notices,…etc. Donc le problème de l'authentification serait résolu. Cependant cette solution demanderait un temps d'analyse et de développement beaucoup plus conséquent. En effet, le Curator devrait être développé en tant que « plugin » Invenio et non comme une application stand-alone[12].
Un autre problème à cette solution est que l'un des objectifs du Curator est sa généricité par rapport envers d'autres plateformes qu'Invenio. En greffant le Curator sur une instance d'Invenio, on perd cette généricité. Pour ces différentes raisons, cette solution n'est pas envisageable dans ce projet.

2.4.6.2 Solution d'un Web service Python

Une seconde solution, qui permettrait de garder le Curator comme application stand-alone, serait d'ajouter à Invenio un service web[13] qui ferait la connexion entre le Curator et Invenio.

Beaucoup plus simple à mettre en place, ce service web, appelé depuis le Curator, prendrait en paramètres le login et le mot de passe du curateur, de l'administrateur ou du responsable de laboratoire afin de l'authentifier sur Invenio. Il permettrait également d'accéder aux libraires d'Invenio telles que la recherche sur les notices et les manipulations de notices d'Infoscience. Qui plus est, l'affichage de certaines notices peut être soumis à des questions de droits. Une telle solution garantirait de pouvoir récupérer également ces notices, chose que le curateur beta actuel ne peut faire.

Cette solution répond donc parfaitement aux besoins du Curator, c'est pourquoi elle à été choisie.

Les différents points sur la mise en place de cette solution seront exposés dans le chapitre 3, « Conception ».

2.5 Conclusion

Durant cette analyse, certains points comme les technologies à utiliser ou encore l'organisation générale des fichiers ont été rapidement définis, soit pas convention, soit par les travaux déjà effectués autour de ce projet.

Les réflexions importantes ont surtout été effectuées sur l'authentification et sur les outils que l'architecture devra fournir aux modules sous forme de méthodes.

Ces différents points étant résolus, le prochain chapitre, à savoir la conception, définit précisément la façon donc l'architecture est implémentée.

3 CONCEPTION

3.1 Introduction

Grâce à l'analyse effectuée, la conception peut définir précisément la façon dont va être implémenté le Curator. Cette partie détermine le diagramme de classe qui est utilisé pour l'implémentation du Curator ainsi que les diagrammes de séquences des actions principales de l'architecture. On y trouve également la spécification du web service pour l'authentification.

3.2 Communication entre Curator et Invenio

L'authentification et les échanges de données entre le Curator et Invenio se font donc à l'aide d'un web service. Celui-ci sera implémenté sur Invenio par Gregory Favre. Le Curator ne fait que l'appeler avec différents paramètres en fonctions de besoins et reçoit en retour les informations nécessaires.

Le schéma ci-après démontre la procédure d'authentification par le web service.

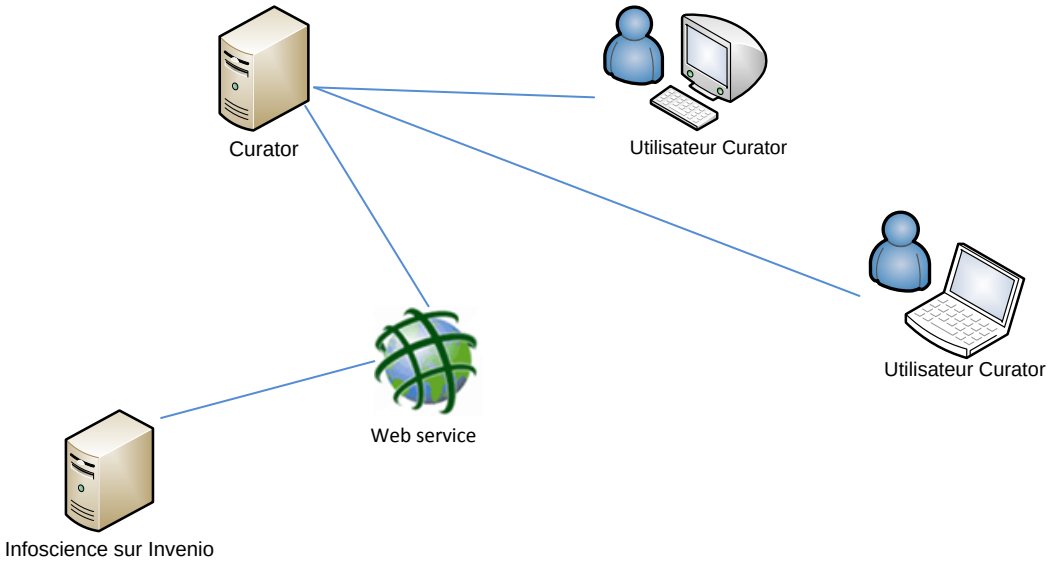


Figure 6 : Schéma du web service entre Invenio et Curator

On peut voir sur ce schéma les deux serveurs bien distincts, Curator et Invenio. Ceux-ci communiquent uniquement à l'aide du service web, tandis que les utilisateurs du Curator accèdent à celui-ci par une interface web. Lorsqu'ils souhaitent s'identifier ou insérer une notice par exemple, ils envoient l'ordre à Curator depuis l'interface web et Curator se charge d'appeler la bonne fonction sur le service web pour interagir avec Invenio. Curator reçoit de l'utilisateur des paramètres selon l'action souhaitée, par exemple son login et son mot de passe pour l'authentification ou un mot clé dans le cas d'une recherche de notice. Le détail de ces transactions est plus précisément défini dans les diagrammes de séquences.

3.2.1 Besoins du web services

Nous connaissons désormais les fonctions que doit nous fournir le web service. Celui-ci sera implémenté sur Invenio, il est nécessaire de définir la spécification de ce web service.

Liste des fonctions du web services

- ***Authentification***

  L'utilisateur souhaite s'identifier sur Invenio.

  L'API :

  - `login(user, password)`

    ```
    @user       : nom d'utilisateur de la personne
    @password   : mot de passe de la personne

    @return     : les droits de l'utilisateur
    ```

- ***Chargement de données***

  L'utilisateur souhaite rechercher des notices selon différents critères.

  L'API :

  - `search(pattern, field, user, password)`

    ```
    @pattern    : mot clé utilisé pour la recherche
    @field      : champs sur lequel le mot clé doit agir
    @user       : nom d'utilisateur de la personne
    @password   : mot de passe de la personne

    @return     : résultat sous forme de flux xml marc
    ```

- ***Manipulation de données***

  L'utilisateur souhaite manipuler des notices (insertion, modification ou suppression)

  L'API :

  - `execute(action, xml_marc, user, password)`

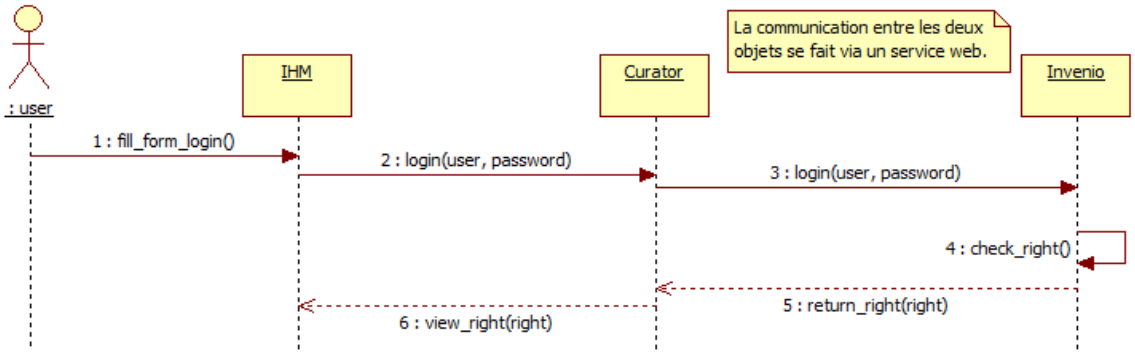
    ```
    @action     : insert, update ou delete
    @xml_marc   : flux xml_marc à traiter
    @user       : nom d'utilisateur de la personne
    @password   : mot de passe de la personne

    @return     : acquittement de l'exécution
    ```

Le fait que l'on retrouve à chaque fois le nom d'utilisateur et le mot de passe est nécessaire. En effet, à chaque appel du web service, Invenio doit à nouveau savoir qui demande une action et vérifier ses droits. Dans ce cas, la méthode « login() » du web service semble inutile. Mais au contraire, elle est utile puisque l'utilisateur Curator ne va s'identifier qu'une seule fois et ses données personnelles seront enregistrées dans un objet. C'est à partir de cet objet que le nom d'utilisateur et le mot de passe sont récupérés pour chaque appel suivant au web service.

3.3 Diagramme de séquence

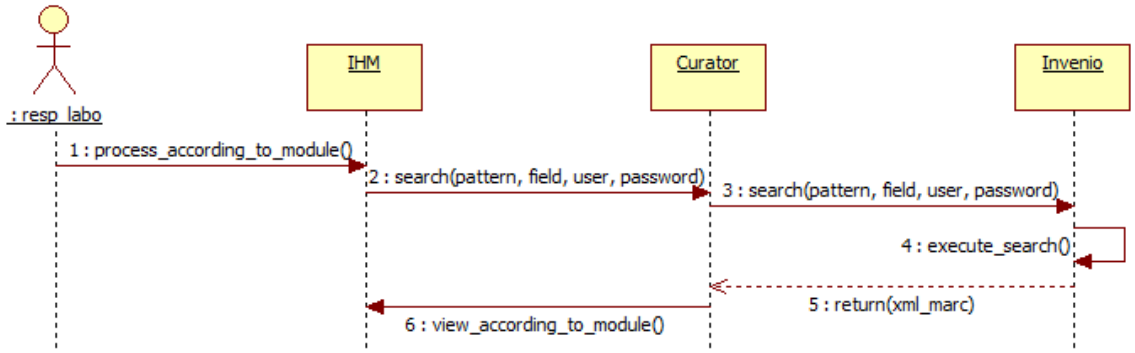
Nous retrouvons régulièrement dans le Curator les séquences correspondantes aux fonctions de base du Curator. A savoir, l'authentification, le chargement de données et l'envoi de données, qui englobe insertion, modification et suppression. Ces séquences sont expliquées ci-après. Chaque module utilisera ensuite ces fonctions et ces séquences se retrouveront intégrées au fonctionnement des modules.

3.3.1 L'authentification



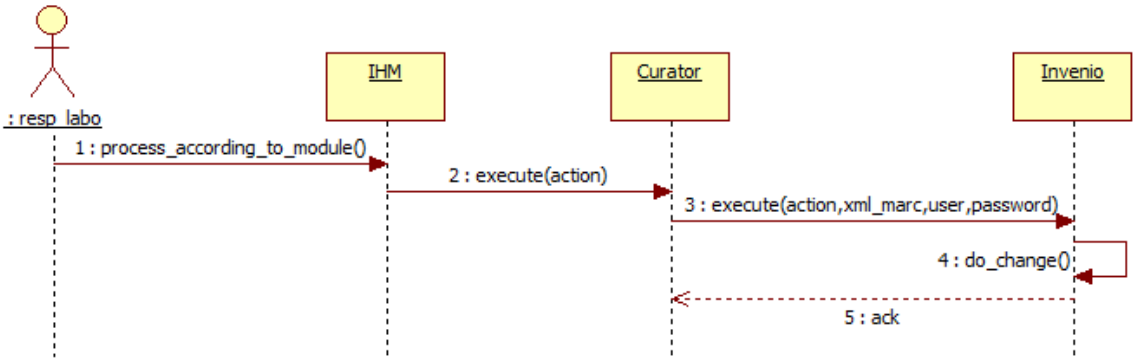
L'utilisateur saisit son nom et son mot de passe sur l'interface du Curator. Celle-ci, à travers le service web fourni par Invenio, va transmettre les données d'utilisateur à Invenio afin qu'il vérifie les droits de la personne. Si le nom d'utilisateur et le mot de passe sont corrects, Invenio retourne les droits de la personne. Dans le cas contraire, Invenio retourne une valeur de droit à « None » ce qui signifie que cet utilisateur n'est pas valide pour le Curator.

3.3.2 Le chargement de données



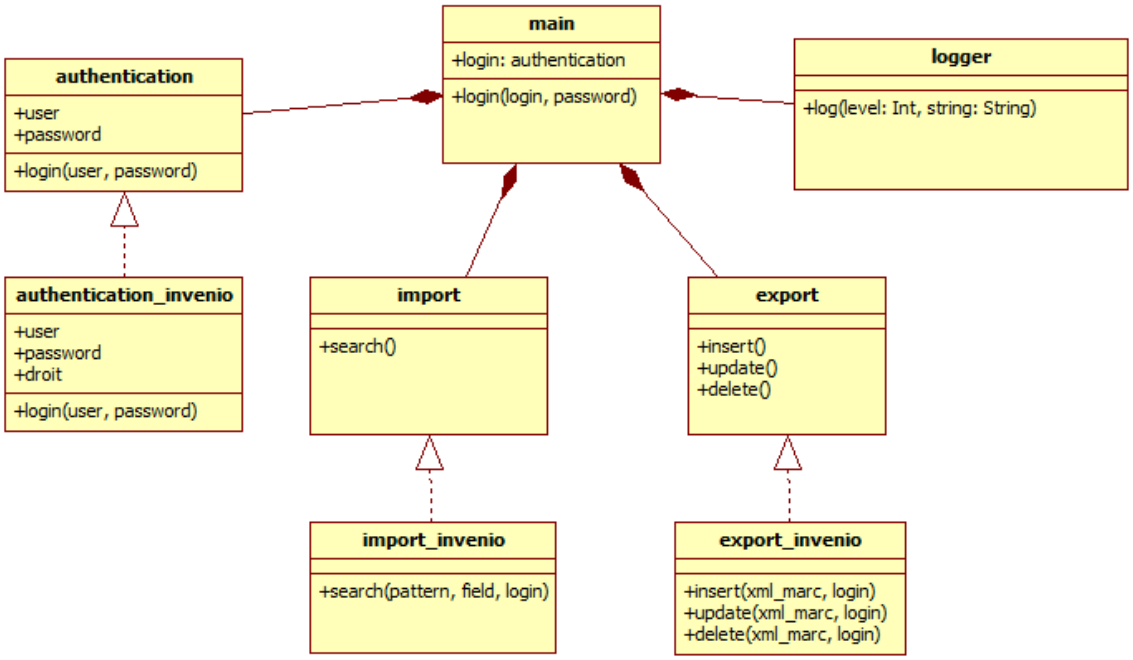
Le chargement de données s'applique lorsqu'un module souhaite charger des notices afin de les traiter (visionnage, modification, suppression). Dans un premier temps, l'utilisateur donne des critères de recherche pour les notices, ceci en fonction du module en action. Le Curator ayant reçu ces critères, les transmet à Invenio par le service web qui va rechercher les notices correspondantes pour les retourner en XML MARC au Curator.

3.3.3 L'envoi de données



Imaginons que le module « Gestion des doublons » soit en action. L'utilisateur gère un doublon, choisi la notice « Master », modifie des champs de la notice. Une fois ce traitement terminé, depuis l'interface web du Curator, il confirme la modification. L'IHM envoie au Curator les modifications sur la notice. Le Curator qui a maintenant les données MARC XML à jour les envois à Invenio à travers le service web. Invenio effectue ensuite l'action demandée pour la ou les notices et renvoie un acquittement ou une erreur, le cas échéant.

3.4 Diagramme de classes



L'architecture est basée sur une classe « main » qui sera le noyau central. A partir de ce point, les autres classes pourront être appelées.
Comme on peut le voir, les classes « authentification », « import » et « export » sont abstraites, ce qui permet d'être générique. Ensuite celles-ci sont spécialisées en fonction du moteur sur lequel elles vont interagir. Dans notre cas, Invenio.

3.4.1 Description des classes

Main

La classe « main » est le point d'entrée du Curator. Elle permet d'instancier les autres classes afin d'utiliser leurs fonctions.

Authentification

L'authentification, comme son nom l'indique, permet à l'utilisateur de s'identifier sur Invenio. Elle détermine les droits de l'utilisateur sur les différents modules qui seront implémentés pour le Curator. C'est grâce à cette authentification que le Curator peut utiliser les outils d'Invenio, ou d'un autre moteur, afin d'agir sur les notices de celui-ci.

Import

La classe « import » offre des méthodes de chargement de notices pour les futurs modules du Curator. Les données importées passent par Pybliographer[14] afin de pouvoir les manipuler dans le but d'effectuer des traitements comme l'insertion, la modification et la suppression de notice.

Export

Une fois les notices traitées, c'est la classe « export » qui permet d'envoyer les changements au moteur spécifique, Invenio dans notre cas.

Logger

La classe « logger » permet de garder une trace de tous les événements apparus durant l'exécution du Curator. Ces événements peuvent être des erreurs, de simples informations ou des « warnings ». Ces traces sont sauvegardées dans un fichier nommé « curator.log » présent dans le dossier « var » *(voir pt. 2.4.4)*.

3.5 Conclusion

La conception a permit de définir précisément la façon dont va être implémenté la communication entre Invenio et Curator, ce qui implique l'authentification, l'envoi et le chargement de données.

On peut résumer la conception du Curator en interaction avec Invenio par le schéma suivant.

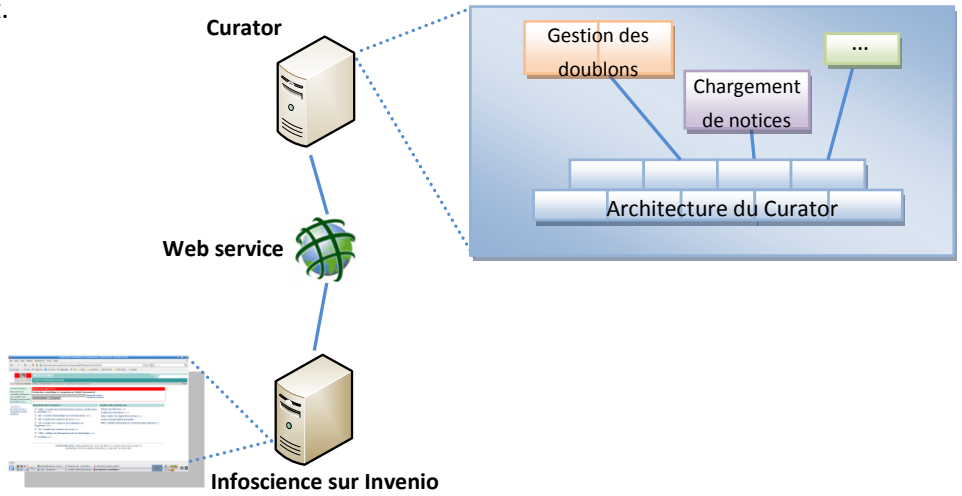


Figure 7 : Schéma global d'Invenio et Curator

On retrouve ainsi le service web qui permet la communication entre Invenio et Curator. On voit bien l'importance de l'architecture sur laquelle les modules viendront se greffer.

Le chapitre suivant qui traite de la réalisation apportera quelques précisions sur le code implémenté.

4 RÉALISATION

4.1 Introduction

Après une analyse et une conception solide, il est temps de parler de la réalisation et de l'architecture avec ce qu'elles comportent ainsi que du module « Chargement de notice ».
Ce chapitre parle notamment du service web qui permet la communication entre Invenio et Curator qui a une importance capitale dans cette architecture et également de l'utilisation de Python de manière générale. On trouve sur la fin de cette partie, une analyse des éléments réalisés.

4.2 Python

Le départ de la réalisation commence par un apprentissage du langage Python, qui au début du projet m'est encore inconnu.

Python est un langage de programmation interprété. Il autorise la programmation impérative structurée, orientée objet, et fonctionnelle. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions.

Le langage Python est placé sous une licence libre et fonctionne sur la plupart des plates-formes informatiques, de Linux à Unix en passant par Windows et MacOS, avec java ou encore .NET. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser. Il est également apprécié par les pédagogues qui y trouvent un langage où la syntaxe clairement séparée des mécanismes de bas niveau, permet une initiation plus aisée aux concepts de base de la programmation.

4.2.1 Python et le web

A la base, Python n'est pas un langage orienté web. C'est-à-dire que, par défaut, un fichier Python ne peut pas être interprété par un navigateur web tel qu'Internet Explorer ou Firefox. Pour permettre ceci, après avoir installé un serveur apache, il faut adjoindre à Python un module nommé « mod_python ». Ce module fournit des librairies qui permettront de communiquer avec Apache afin de générer des pages web à partir de nos scripts Python.

Exemple :

```
from mod_python import apache
def index(req):
    try:
        req.content_type = "text/html; charset=utf-8"
        req.write('Hello Word !!!')
    except Exception, err:
        req.write(str(err))
```

On remarque, au début de l'exemple, l'importation de la librairie « apache » du module Python. Il faut ensuite impérativement déclarer une méthode « index », qui sera le point d'entrée de la page web. La variable « req » est un objet « Request » qui permet bon nombre d'actions sur Apache.

A noter que le langage Python est basé sur l'indentation ! Il n'y a pas de « ; » pour terminer les lignes ou encore de « { } » pour les blocs d'instructions. Tout ceci est défini par l'indentation. Si celle-ci est mauvaise, une erreur est générée lors de l'exécution. Ceci donne l'avantage d'avoir un code plus propre et structuré, du moins visuellement.
On peut également signaler que les lignes commençant par un « # » sont des lignes de commentaire.

4.2.1.1 Point d'entrée du Curator

Le point d'entrée du Curator se situe au niveau de la classe « Main » *(cf. point 3.4, Diagramme de classe)* dans le fichier « main.py » du dossier « lib » *(cf. point 2.4.4, Organisation générale).* Le « Main » va se charger d'appeler le contenu HTML, qui est séparé de la logique métier, se trouvant dans le dossier « www » du Curator.

Le « Main » choisit le contenu à afficher en fonction de paramètres « _GET[15] ». Une fonction Python reprise d'Invenio permet de parser facilement tous les paramètres d'entrées et de leur attribuer un type correct. Cette fonction, `wash_urlargd(form, content)`, se trouve dans le fichier « url_handler.py » du dossier « lib ».

Cette fonction permet également de traiter des paramètres provenant de formulaires, comme pour l'identification.

Par exemple, lorsque l'on souhaite afficher la page du module « Chargement de notices », le lien est composé de la manière suivant :

- http://chemin_vers_curator/main.py?content=load_notice

Voici le code qui permet de récupérer la variable « content » et d'agir en fonction

```
#Choose the content to show according to the first GET argument
argd = wash_urlargd(args, {'content': (str, "NA")})
#choice_content(req,argd['content'],args,html)
try:
    choice_content(req,argd['content'],args,html)
except Exception, err:
    logger.log(40, 'Error in the choice of html content : ' +
     str(err))

def choice_content(req, content, args, html):
    if content=='load_notice':
        #do_work_for_loading_notice
```

4.2.2 Gestion des sessions

Le protocole HTTP est un protocole non connecté, cela signifie que chaque requête est traitée indépendamment des autres et qu'aucun historique des différentes requêtes n'est conservé. Ainsi le serveur web ne peut pas se "souvenir" de la requête précédente, ce qui est dommageable dans des utilisations telles que le Curator, pour lequel le serveur doit mémoriser le nom d'utilisateur et son mot de passe d'une page à l'autre.

Il s'agit donc de maintenir la cohésion entre l'utilisateur et la requête. Pour ceci, Apache fournit la possibilité de créer une session. Une session est en fait un objet qui peut contenir des paramètres que nous pouvons définir selon nos besoins.

Pour le Curator, nous avons besoin du nom d'utilisateur, de son mot de passe et de ses droits. Ces valeurs seront sauvegardées dans la session lors de l'authentification de l'utilisateur. Ainsi, il pourra naviguer de page en page, effectuer différentes actions dans les différents modules sans avoir besoins de saisir à nouveau ses informations personnelles.

Fonctionnement des sessions dans Curator

```
def handler(req):
    """
    This method handle the authentication session of user on
Curator
    @param req: see index() method from Main.
    @return: The authentication session to check the rights and
the state of the user
    """
    session = Session.Session(req)
    session.load()
    try:
        session['username']  == None
    except:
        session['username']  = None
    if session['username'] == None or session['username'] == '':
        try:
            session['username'] = auth.username
            session['password'] = auth.password
            session['right']    = auth.right
        except Exception, err:
            logger.log(40,  'Error  during  the  restauration  of
session : ' + str(err))
```

```
    session.save()
    return session
```

A chaque changement de page, la fonction ci-dessus est appelée. Elle se charge de créer une session et d'y charger les données de la session en cours.
Si l'utilisateur est identifié, la ligne « `session.load()` » aura chargé les paramètres dans les variables de session. Autrement dit, le nom d'utilisateur, le mot de passe et les droits seront présents dans la variable « `session` » et accessible comme ceci : `session['username']`
Si l'utilisateur n'est pas encore identifié, elle met simplement les paramètres de session à « None », ce qui signifie qu'ils ne sont pas déclarés, par l'intermédiaire de l'objet « `auth` » qui est une instance de la classe « authentification ».

Ce mécanisme permet donc à l'utilisateur de s'identifier une unique fois sur le Curator. Ses données seront sauvegardées en session et lorsqu'il fera appel au service web qui nécessite ces mêmes données, elles seront récupérées depuis la session.

4.2.3 Service Web Python

Pour créer un service web Python, c'est la librairie SOAPpy[16] qui a été choisie. Elle fournit à Python une API complète et facile d'utilisation. On aurait pu choisir XML RPC qui fournit également les fonctions nécessaires, cependant, le fait qu'XML RPC soit l'ancêtre de SOAP indique que ce dernier est donc plus évolué. De plus Python possède la librairie SOAPpy, très facile d'utilisation et stable. Elle permet également le transfert sécurisé (HTTPS), qui sera utilisé dans notre cas. SOAP est également très recommandé par la communauté Open Source.

Tout d'abord le service web a été implémenté et enregistré coté serveur, Invenio dans notre cas. Pour ce faire, SOAPpy fournit des méthodes qui se chargent de l'enregistrement et de la pérennité du service web. Le code ci-dessous présente la structure de la logique métier du service web ainsi que les appels à SOAPpy pour enregistrer le service web sur le serveur.

Squelette du code du coté serveur *(Invenio)* :

```
import sys
import SOAPpy

namespace = "http://authentification-curator"

class web_service:
    def login(user, password):
        droit = null
        return droit #null, admin, curateur ou resp_labo
    def search(pattern, field, user, password):
       return xml_marc
    def execute(action, xml_marc, user, password):
        return ack

server = SOAP.SOAPServer(("localhost", 8888))
ws = web_service ()
server.registerObject(ws, namespace)
server.serve_forever()
```

Pour accéder au service web depuis une machine distante, le Curator dans notre cas, il faut également utiliser les méthodes de SOAPpy. Celles-ci permettent de se connecter au service web distant afin d'utiliser ses fonctions comme si celles-ci se trouvait en locale. Ainsi, on peut récupérer très facilement les paramètres que nous retourne la fonction distante.
Le code ci-après démontre la connexion au service web dans le but d'authentifier l'utilisateur du Curator sur Invenio. On donne au service web le nom d'utilisateur et le mot de passe, Invenio vérifie les droits pour la personne et retourne ceux-ci au Curator.

Code de l'authentification par le service web du coté client *(Curator)* :

```
from curator.lib.authentification import *
from SOAPpy import SOAPProxy

class authentification_invenio(authentification):
    logger, username, password, droit = None, None, None, None
    def __init__(self, logger):
        authentification.__init__(self)
        # Define a logger to store all events
        self.logger   = logger

    def login(self,login, password):
        url = 'https://path_to_server:XX/path_to_webservice'
        n   = 'http://authentification-curator'
        try: #Connexion to Invenio webservice
            server = SOAPProxy(url, namespace=n)
        except Exception, err:
            self.logger.log(40,  'Error  during  the  access  to
webservices : ' + str(err))

        try: #Access to method of Invenio webservice
            droit = server.login(user, password)
            self.droit    = droit
            self.username = login
            self.password = password

        except Exception, err:
            self.logger.log(40,  'Error  during  the  access  to
webservices method : ' + str(err))
```

4.3 Module : « Chargement d'un fichier de notices»

Pour rappel, ce module permet de mettre à jour des notices par lot en chargeant, à travers l'interface web, un fichier au format XML MARC 21[17] ou BibTeX[18]. Il est utilisé lorsque par exemple un laboratoire souhaite publier des notices qu'ils possèdent au format XML MARC ou BibTeX.
Cependant, seul le chargement de fichier au format XML MARC, au dépend de BibTeX, est actuellement disponible dans le Curator, c'est donc celui-ci qui est expliqué.

L'utilisateur fournit donc, via l'interface web du Curator, un fichier XML MARC.

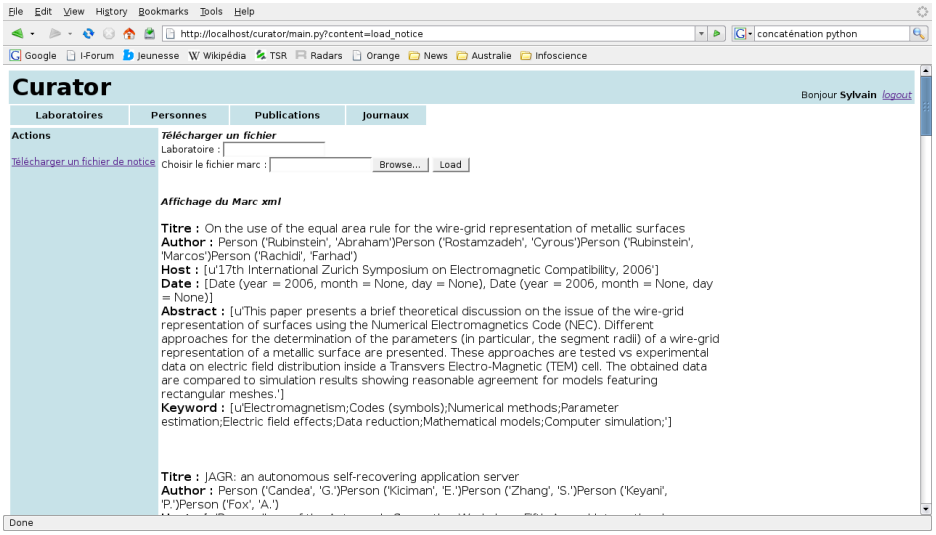


Figure 8 : Capture d'écran du module "Télécharger un fichier de notice"

Ensuite, on peut immédiatement voir tout l'utilité de l'architecture du Curator. C'est en effet une des fonctions de base de cette architecture qui va traiter ce flux XML MARC. L'architecture récupère ce flux dans un buffer temporaire afin de l'envoyer à Invenio via la méthode d'insertion de la classe « Export ».

4.4 Conclusion

Lors de la réalisation des différents points prévus à la conception, un certain nombre de problèmes sont venus entraver mon parcours.
Dans un premier temps, l'apprentissage de Python m'a quelque peu retardé. Ce n'est pas la syntaxe en elle-même mais certaines particularités, essentiellement l'intégration de la logique métier avec la sortie html ou encore la gestion des arguments, des sessions qui doivent demeurer pendant la navigation de page en page.

La mise en place du service web m'a pénalisé sur le rendu final de l'implémentation. C'est en effet à cause du retard pris pour mettre en place ce service web que l'implémentation, autant de l'architecture que du module, n'est pas terminée.
En effet, le service web devait être implémenté et mise en place avec l'aide de Gregory Favre et le fait qu'il soit surchargé en cette période, due à une migration importante d'Infoscience, a fait que nous avons dû reporter à plusieurs reprises la réalisation de ce service web. Bien évidemment, Gregory Favre n'est en rien responsable de ceci.

Même si au moment de rendre le rapport, l'implémentation n'est pas terminée, Gregory et moi allons tout mettre en œuvre pour que celle-ci soit fonctionnelle pour la fin du mois de novembre, en vue de la présentation de mon projet aux experts et responsables.

5 CONCLUSION

5.1 Résultats

Le projet Curator m'a permis de parcourir et d'avoir une bonne vue des possibilités du langage Python. Cependant, le fait de devoir se coupler avec une application tierce, Invenio, a posé quelques problèmes, notamment pour l'implémentation du service web.

Voici une vue détaillée des résultats obtenues lors de ce projet :

Objectifs	**Résultats**
Prise en main de l'environnement Invenio/Infoscience/Python	ok
Analyse du Curator Beta	100%
Analyse et conception de l'architecture du nouveau Curator	100%
Implémentation de l'architecture	90%
Implémentation du module choisi	80%

Comme on le voit, l'implémentation n'est pas complètement terminée. Le gros du travail est évidement fait. Les points manquant au moment de rendre ce document sont liés au service web, comme expliqué dans la conclusion du chapitre « Réalisation » à la page précédente.

5.2 Améliorations futures

5.2.1 Terminer l'implémentation

Sans prendre en compte les futurs modules du Curator, il faudrait dans un premier temps achever la réalisation de l'architecture et du module choisi. Ceci ne demande plus un temps conséquent étant donné l'état actuel de ces parties. Une fois le service web terminé et mis en place, il sera très rapide de terminer l'architecture. Quant au module de chargement de notice, il s'appuie sur l'architecture et ne nécessite plus que peu de temps à le terminer.

5.2.2 Optimisation du code

Du fait que je ne sois malheureusement pas encore un expert dans le langage Python, il serait judicieux d'effectuer des tests sur le code et de l'optimiser. Cependant, ceci sera plus facile à juger une fois quelques modules intégrés au Curator. Ceux-ci permettront de réaliser des tests concrets. Il serait notamment intéressant d'exploiter des outils de métrique sur le code du projet, tels que pylint (http://www.logilab.org/857)

5.3 Impressions personnelles

Le fait de pouvoir effectuer ce travail hors EIF, à l'EPFL fut très enrichissant. Cela m'a permit de découvrir un autre cadre que celui de l'EIA-FR que je connais déjà très bien. J'ai pris beaucoup de plaisir à travailler avec Gregory Favre et Pierre Crevoisier qui m'ont soutenu tout au long du projet.

Même si malheureusement je n'ai pas pu remplir à 100% toutes les tâches que je souhaitais, je suis satisfait de l'avancement effectué autant sur le plan personnel que pour la suite du projet Curator.

L'apprentissage de Python et mon aisance actuelle sur les systèmes Linux ne peuvent être que bénéfiques dans la suite de ma carrière professionnelle.

Je tiens encore à remercier mes responsables à l'EIF-FR, Omar Abou Khaled et Houda Chabbi Drissi, qui m'ont offert l'opportunité d'accomplir ce travail au sein de l'EPFL.

6 WEBOGRAPHIE

6.1 Générale

- http://wiki.epfl.ch/eggers
  Le wiki que j'ai créé et maintenu tout au long du projet. On y retrouve toute la documentation concernant le projet. Je ne peux malheureusement pas garantir la pérennité de ce site.

- http://presentation.epfl.ch
  Quelques informations sur l'EPFL utilisées pour l'introduction.

- http://empc51.epfl.ch/Infoscience/Infoscience
  A cette adresse se trouve un document qui est paru dans la revue Ametist.inist.fr. Ce document concerne le projet Infoscience.

  Infos documents
  L'archive institutionnelle de l'Ecole Polytechnique Fédérale de Lausanne : Etat actuel et perspectives.
  Publié dans la revue scientifique « Ametist.inist.fr »
  Auteurs : David Aymonin (EPFL), Pierre Crevoisier (EPFL), Frédéric Gobry (Google)

- http://cdsware.cern.ch/invenio
  Site officiel du moteur Invenio, développé et maintenu par le Cern. On y trouve les sources d'Invenio ainsi que le manuel d'installation qui m'a été grandement utile.

- https://twiki.cern.ch/twiki/bin/view/Inspire/EnrichmentScripts
  Le wiki du projet qui va dans le même sens que le Curator pour la base bibliographique du CERN. La suite du travail sur le Curator va d'ailleurs être fait avec une communication permanente avec le CERN afin de ne pas faire le travail à double.

- http://fr.wikipedia.org
  Encyclopédie gratuite qui fournit de nombreuses informations très utiles sur tout ce qui touche à l'informatique, entre autres. Utiliser pour les définitions de la plupart des termes du glossaire qui suit.

- http://packages.ubuntu.com
  Sources officielle et très complètes de paquets pour Linux Ubuntu. Très utile pour l'installation d'Invenio ou de divers logiciels.

- http://www.osalt.com
  En tant qu'utilisateur de Windows, le saut sur Linux peut-être simplifié par ce site qui fournit un comparatif des logiciels Windows habituels vers des alternatives open source pour Linux.

6.2 Technique

- http://fr.wikibooks.org/wiki/Exemples_de_scripts_Python
- http://wikipython.flibuste.net/moin.py/Les_20exceptions
- http://codemark.tuxfamily.org/tutoriel-soap-en-python
- http://www.ibm.com/developerworks/library/ws-pyth5
- http://codemark.tuxfamily.org/tutoriel-soap-en-python
  Divers site proposant des exemples de code Python qui m'ont aidé dans la réalisation du Curator

- http://www.modpython.org/live/current/doc-html
  Toute la documentation sur le « mod_python » qui m'a permit de créer les scripts Python générant des pages web.

- http://docs.python.org
  La documentation officielle de Python

7 ANNEXES

Les annexes suivantes se trouvent à la suite de ce document

1. Spécification du projet Curator
2. Installation d'Invenio
3. Synthèse sur les flux de communication entre Invenio et Curator
4. API

[1] **Réforme de Bologne**

L'objectif de la réforme de Bologne consiste à réaliser un espace européen de l'enseignement supérieur et de la recherche qui soit concurrentiel et dynamique. Les principaux axes de la réforme sont le système de formation en deux cycles (bachelor et master) et l'introduction d'un système de crédits favorisant la transparence et la mobilité.

Plus d'informations : http://www.bbt.admin.ch/themen/internationales/00163/index.html?lang=fr

[2] **CDSware**

CDSware est l'ancienne appelation de CDS Invenio lequel est décrit au point 1.2.1.

Plus d'informations : http://cdsware.cern.ch/invenio/index.html

[3] **Google-like**

L'expression « Google-like » est utilisé pour designer une interface semblable à celle du célèbre moteur de recherche Google disponible à l'adresse http://www.google.com

[4] **Google Scholar**

Google Scholar permet d'effectuer facilement une recherche étendue portant sur des travaux universitaires.

Plus d'informations : http://scholar.google.com

[5] **XML MARC 21**

XML MARC désigne un format de données permettant d'informatiser les catalogues de bibliothèques dans un format XML défini. Il se présente à l'écran comme une succession de champs de données, appelée grille MARC, de longueur variable portant chacun une étiquette (un nombre de 3 chiffres)

Plus d'informations : http://www.loc.gov/marc/marcxml.html

[6] **BibTeX**

BibTeX est un logiciel et un format de fichier conçu par Oren Patashnik et Leslie Lamport en 1985 pour LaTeX. Il sert à gérer et traiter des bases bibliographiques.

Plus d'informations : http://www.bibtex.org

[7] **Python**

Python est un langage de programmation interprété. Il autorise la programmation impérative structurée, orientée objet, et fonctionnelle. Le langage Python est placé sous une licence libre et fonctionne sur la plupart des plates-formes informatiques.

Plus d'informations : http://www.python.org

[8] **Apache**

Le logiciel **Apache HTTP Server**, souvent appelé **Apache**, est un serveur HTTP produit par la Apache Software Foundation. C'est le serveur HTTP le plus populaire du Web. C'est un logiciel libre avec un type spécifique de licence, nommée licence Apache.

Plus d'informations : http://www.apache.org

[9] **mod_python**

Mod_python est un module Apache qui embarque un interpréteur Python. Avec « mod_python », il est aisé d'écrire des applications web en Python, il fournit également des outils d'accès aux bases de données et aux librairies internes d'Apache.

Plus d'informations : http://www.modpython.org/

[10] **MySQL**

MySQL est un serveur de bases de données relationnelles SQL développé dans un souci de performances élevées. Il est multi-thread, multi-utilisateurs. C'est un logiciel libre développé sous double licence en fonction de l'utilisation qui en est faite : dans un produit libre (open-source) ou dans un produit propriétaire.

Plus d'informations : http://www.mysql.org

[11] **Unix**

UNIX™ est le nom d'un système d'exploitation créé en 1969, à usage principalement professionnel, conceptuellement ouvert et fondé sur une approche par laquelle il offre de nombreux petits outils chacun dotés d'une mission spécifique, multitâche et multiutilisateur. Il a donné naissance à une famille de systèmes, dont les plus populaires en 2007 sont GNU/Linux, *BSD, Mac OS X et Solaris. On nomme « famille Unix » l'ensemble de ces systèmes. On dit encore qu'ils sont de « *type Unix* »

Plus d'informations : http://fr.wikipedia.org/wiki/UNIX

[12] **Stand-alone**

Caractérise une application à part entière. Ce n'est donc pas un add-on, ou un greffon, et elle peut fonctionner indépendamment de toute autre application.

[13] **Service web**

Un Service Web est un programme informatique permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur Internet ou sur un Intranet, par et pour des applications ou machines.

Les Services Web désignent l'implémentation logicielle des spécifications et reposent tous sur un ensemble de protocoles et de standards de base utilisés pour l'échange de données entre applications dans des environnements hétérogènes :

- Le protocole SOAP pour l'échange de messages.
- Le fichier WSDL pour la description des messages, de leur type de données...
- Une entrée éventuelle dans un annuaire UDDI

Plus d'informations : http://fr.wikipedia.org/wiki/Service_Web

[14] **Pybliographer**

Pybliographer est un outil qui facilite la gestion et la création des bases de données BibTeX et XML MARC. Pybliographer fournit des outils tel que :

- la création et l'édition des références bibliographiques
- le triage des références
- des recherches dans la base de données
- l'exportation des références en différents formats : HTML, TXT, TeX, XML MARC

Pybliographer fournit également une API Python qui permet d'intégrer ses outils dans notre propre application, le Curator, afin d'effectuer des traitements sur nos données bibliographiques, autrement dit, les notices d'Invenio.
Plus d'informations : http://pybliographer.org/

[15] **GET**

La méthode GET est un moyen de passer des paramètres d'une requête HTTP depuis le navigateur au serveur. Cette méthode place les paramètres, généralement séparés par un caractère spécial tel que « & », dans l'URL même, qui est visible pour la personne qui utilise le navigateur. L'autre méthode se nomme POST, et est utilisée lorsque le site ne souhaite pas transmettre les paramètres dans l'URL. Cette méthode est préférable lorsque le texte à envoyer au serveur est volumineux ou si les données sont sensibles. Les formulaires utilisent la méthode POST par défaut.

[16] **SOAPpy**

Le SOAP (Simple Object Access Protocol) est un protocole d'échange d'information entre deux objets distants en format xml, il permet d'appeler une fonction distante et de récupérer son résultat. SOAPpy est une librairie Python permettant l'utilisation de ce protocole.
Plus d'informations : http://pywebsvcs.sourceforge.net/

[17] **XML MARC 21**

XML MARC désigne un format de données permettant d'informatiser les catalogues de bibliothèques dans un format XML défini. Il se présente à l'écran comme une succession de champs de données, appelée grille MARC, de longueur variable portant chacun une étiquette (un nombre de 3 chiffres)

Plus d'informations : http://www.loc.gov/marc/marcxml.html

[18] **BibTeX**

BibTeX est un logiciel et un format de fichier conçu par Oren Patashnik et Leslie Lamport en 1985 pour LaTeX. Il sert à gérer et traiter des bases bibliographiques.

Plus d'informations : http://www.bibtex.org