Version 1.4

ORI-OAI-harvester : Moisson d'autres entrepôts via le protocole OAI-PMH

> ⚠ **Module optionnel**

Voir l'architecture du système

Dans quels cas l'utiliser

Si vous voulez construire un index à partir de fiches de métadonnées distantes récupérées via le protocole OAI-PMH

Composants obligatoires

- **ORI-OAI-indexing** pour indexer les fiches de métadonnées moissonnées
- **Base de données SQL** pour le stockage des fiches de métadonnées et des données de gestion

Description

Ce composant du système correspond au moissonneur OAI-PMH. Utilisant le logiciel OAIHarvester2, il permet le moissonnage de fiches de métadonnées sur tout entrepôt OAI. Les fiches moissonnées sont alors stockées localement dans une base de données SQL.

Tout comme le moteur de workflow, ce module fournit toutes les fiches de métadonnées moissonnées au moteur d'indexation dans le but d'être indexées. L'index de recherche Lucene est alors composé des documents locaux, mais aussi des fiches de métadonnées moissonnées par ce module.

Via une interface graphique conviviale, l'administrateur du système peut programmer les différentes moissons qui seront lancées par le gestionnaire de tâches Quartz.

Voir la documentation technique

Spécifications

Objectifs, Rôle et contrats

Ce document présente le rôle du module moissoneur au sein du système ORI, ses interactions avec les autres modules, et un ensemble de documents de modélisation comportant les cas d'utilisations, une description des solutions techniques envisagés, un modèle de donnée, les diagrammes de classes et de séquences principaux.

L'**objectif** du système ORI dévolu à ce module est de constituer un corpus unique, cohérent et exploitable de fiches, à partir de différentes sources éparses, fiches concernant la description d'un document ou constituant ce document en tant que tel.

Le choix technique pour réaliser cet objectif est l'utilisation du protocole OAI-PMH, qui permet à diverses sources dépositaires d'un ensemble de documents, de disséminer sous forme de fiches au format contrôlé les méta-données descriptives de chaque document discret. Chaque format contrôlé détermine un type de ressource particulier, et chaque ensemble d'un même type peut faire l'objet d'un traitement particulier.

Le **rôle** de ce module est de moissonner des fiches OAI dans les multiples formats utilisés dans l'enseignement supérieur (LOM,TEF, CDM, Dublin Core...) via le protocole OAI, à partir d'un **ensemble d'entrepôts** OAI paramétrable.

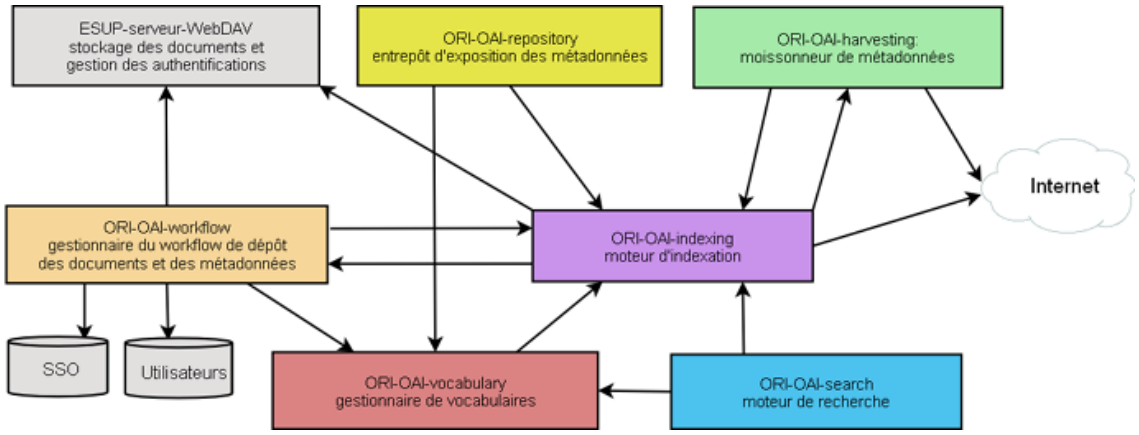
A partir d'un entrepôt OAI, qui peut exposer plusieurs formats et comporter plusieurs ensembles OAI, la granularité d'une moisson telle que définie par ce module doit s'entendre comme une **séléction de fiches**, basée sur un ensemble de paramètres : format, ensembles, dates.

Le module permet donc de gérer un *ensemble de

moissons*, en offrant la possibilité d'en **programmer l'exécution de manière répétée**, et d'adopter une stratégie de **mise à jour** entre chaque occurence d'une même moisson, visant à maintenir l'intégrité de l'ensemble de fiches issues de la moisson (fiches détruites, modifiées).

L' **unité d'opération principale** de ce module sera définie comme la **moisson**. Les autres objets pertinents du module sont **l'enregistrement** et la **fiche**.

Schéma global ORI



La moisson

La moisson désigne en même temps l'unité d'opération principale du module, et par extension, l'ensemble de fiches résultant de cette unité d'opération.

Caractéristiques

La moisson, au niveau le plus fin de sa granularité, est caractérisée par un ensemble d'attributs sélectifs, qui déterminent comme résultat un corpus de fiches : * l'entrepôt moissonné

- le format - sélection minimale obligatoire.
- le ou les ensembles - optionnel
- la ou les dates, définissant soit une limite inférieure, supérieure, ou un intervalle - utilisées pour les mises à jour.
  Le module maintient de façon persistente la relation entre toute fiche moissonnée et ses caractéristiques de moisson.

Contrats

L'exécution de la moisson se décompose en deux phases, qui remplissent deux contrats : * Le stockage de fiches dans une base de données -> un point d'accès à ces fiches au module d'indexation ORI-OAI-indexing remplit le contrat "stockage des fiches".

- l'indexation d'un ensemble paramétrable de champ pour chaque type de fiche par le module d'indexation
  -> assure le contrat "référencement d'une fiche moissonnée au sein du système par indexation".

Les modalités d'éxécution de ces contrats comprennent deux aspects : * la programmation de l'exécution répétée à intervalles paramétrables de la moisson

- la relation entre deux répétitions d'une même moisson, à savoir **un mode de mise à jour**, dépendant des caractéristiques de l'entrepôt moissonné.

La mise à jour consiste à répercuter les changements possible dans l'ensemble de fiches représenté par la moisson : **création**, **modification**, **suppression**.
Ce dernier type de changement, la **suppression**, est tributaire du mode de gestion des fiches supprimées de l'entrepôt moissonné, parmis les trois qui sont définies par le protocole OAI-PMH à travers l'attribut "deletedRecord"..
Il en découle deux modes de mise à jour possibles : *** gestion persitente (deletedRecord="persistent") : une *mise à jour incrémentale* peut être exécutée pour tous les types de changement, où le module traite chaque enregistrement dont la date est postérieure à la date de mise à jour précédente : remplacement des enregistrements marqués comme "deleted" dans la base de donnée et supression de leur entrée dans l'index, et insertion des autres enregistrements.

- La suppression s'entend du retrait de la fiche de la base de donnée,(et de son remplacement par un enregistrement "deleted"), et du retrait de l'entrée correspondante dans l'index.
- L'insertion s'entend comme l'ajout de la fiche dans la base de donnée et son indexation par le module ORI-OAI-indexing.

- pas de gestion (deletedRecord="no" ou "transient") : une **mise à jour mixte,** incrémentale pour les fiches créées et modifiées, et une **mise à jour comparative** pour les enregistrements supprimés.
  Cette dernière est réalisée par comparaison entre l'ensemble des fiches stockées issues de la moisson précédente et l'ensemble des fiches d'une nouvelle moisson limitée à la date de la moisson précédente. Le delta de ces ensembles permet de définir une liste de fiches à supprimer.
  Note : Si le mode est "transient", on recoupe cette liste avec les éventuels enregistrements marqués "deleted" d'une moisson incrémentale, en gardant priorité à ces derniers pour conserver l'information de la date de suppression. En absence d'enregistrement "deleted" d'une moisson incrémentale, la mise à jour comparative aura à charge de créer cet enregistrement pour chaque enregistrement supprimé, avec la date de la moisson.

L'enregistrement

On appellera "enregistrement" l'ensemble de la structure XML "record" définit par le protocole OAI, auquel le module applique son traitement, ou encore la structure XML "ori-record" qui encapsule cette dernière.

La fiche

Lorsque l'on parle de "fiche", cela désigne plus précisément la section "metadata" de la structure "record", qui est récupérée isolément de l'enregistrement par les autres modules.

Note : Le module, pour chaque fiche, stocke l'ensemble de l'enregistrement OAI, donc la fiche ne se distingue pas en tant qu'élément autonome. Au sein du module, la fiche est une partie de l'enregistrement.

Cas d'utilisation

Cette section décrit les cas d'utilisation concrets qui impliquent ce module. Ces cas d'utilisation concernent aussi bien l'interaction de l'utilisateur avec le système, que les interactions entre les différentes parties du système, issues de la programmation faite par l'utilisateur.

Définition et paramètrage d'une moisson

L'utilisateur peut définir et paramétrer le sous-système "moissonneur" pour effectuer un ensemble de moissons, telles que définies précédemment. Une fois ce paramétrage injecté dans le module, ce dernier gère les différents moments de son exécution.

Stockage des caractéristiques de la moisson

Les caractéristiques de chaque moisson définie par l'utilisateur sont stockées dans une collection de la base de données, en correlation avec un identifiant de moisson généré par le système.

Stockage des enregistrements OAI

Dans un premier temps, le moissonneur accède aux entrepôts selon les caractéristiques définies pour la moisson, et récupère un ensemble d'enregistrements, qui sont stockées dans une base de données. Le stockage inclut une relation constante entre les fiches et les caractéristiques de la moisson, à fins de gestion des mises à jour et d'exposition sélectives des fiches moissonnées.

Le module **modifie** , soit au niveau stockage, soit au niveau accès, le nom (setSpec) des **ensembles natifs** associés aux enregistrements OAI, en leur ajoutant un préfixe "ori-import:[", ceci afin de distinguer ces ensembles natifs des ensembles gérés par le module ORI-OAI-Repository, lors de la ré-exposition des métadonnées.

Elements de l'enregistrement stockés

Cas nominal : l'enregistrements moissonné complet est stocké (header, metadata et about)

Cas dégradé : quand il s'agit d'un enregistrement supprimé, seul le header est stocké.

Identifiant unique de l'enregistrement

L'identifiant utilisé est l'identifiant de l'enregistrement OAI moissonné, combiné au prefixe du format de la fiche.

Relation de l'enregistrement à une moisson

caractérisée
Chaque fiche est associée à l'identifiant de la moisson dont elle est issue, par l'affectation de cet identifiant à un attribut stocké en même temps que l'enregistrement.

Emplacement du stockage de la fiche

Cas nominal : par défaut, l'enregistrement est stocké dans la collection nommé avec le préfix OAI de son format, sous la racine de la collection des moissons.

Cas dégradé : un emplacement peut être spécifié par l'utilisateur dans le paramétrage de la moisson.

Programmation d'une moisson

L'utilisateur peut définir une programmation pour chaque moisson ou pour un ensemble de moisson.

Moisson ponctuelle

L'utilisateur peut choisir de définir une moisson comme ponctuelle, dont l'exécution est immédiate et unique.

Moisson avec mise à jour périodique

L'utilisateur peut définir, pour une moisson ou un ensemble de moisson, une exécution répétée, définie par les paramètres suivants : * Date de première exécution

- Fréquence de répetition

Exécution d'une moisson

Moisson initiale

Le moissonneur, à la première moisson, traite l'ensemble des fiches resultant de ses caractéristiques. Ce traitement comporte le stockage dans une base de données et l'indexation.

Stockage dans la base de donnée

Le stockage de chaque fiche se déroule selon le paramétrage de l'utilisateur décrit dans la section précédente.

Indexation des fiches

L'indexation d'une fiche comporte, outre les métadonnées de son format, son **identifiant**, le **préfixe** de son format, et le nom de son **entrepôt** d'origine. Le moissonneur transmet ces informations au module ORI-OAI-indexing.

Mise à jour des enregistrements OAI

Aprés la moisson initiale, toute réitération se comporte comme une mise à jour. Cette mise à jour consiste à traiter la création, modification et suppression de fiches depuis la moisson précédente.

Création de fiches

Chaque nouvelle fiche est stockée et indexée comme lors d'une moisson initiale.

Modification de fiches

Chaque fiche modifiée est remplacée dans la base de donnée en utilisant l'identifiant de l'enregistrement et le préfixe de son format, et ré-indexée.

Suppression de fiches

- Cas nominal : l'entrepôt OAI à un mode de gestion des fiches supprimées persistant.
  Chaque fiche indiquée comme "deleted" remplace la fiche du même identifiant et du même format dans la base. L'entrée dans l'index est supprimée.
- Cas dégradé : l'entrepot OAI n'a pas de mode de gestion des fiches supprimées persistant.
  Par comparaison des fiches stockées et des fiches fraichement moissonnées, une liste des fiches à supprimer est constituée, et pour chaque fiche concernée, le système remplace la fiche dans la base de donnée par une fiche "deleted" du même identifiant et du même format. L'entrée dans l'index est supprimée.

Supression d'une moisson

L'utilisateur peut supprimer une moisson selon plusieurs acceptions :

Suppression des fiches d'une moisson

L'utilisateur peut supprimer l'ensemble des fiches issues d'une moisson, sans supprimer ses caractéristiques du système.

Suppression des caractéristiques d'une moisson

L'utilisateur peut supprimer la définition d'une moisson injectée dans le système.

Suppression d'une programmation

L'utilisateur peut supprimer la programmation affectée à une moisson ou à un ensemble de moisson. Le module cessera les mises à jour définies par cette programmation.

Recherche d'enregistrements par un utilisateur via ORI-OAI-search

et ORI-OAI-indexing
L'utilisateur interagit avec le système pour rechercher un ensemble d'enregistrement correpondant à sa requête. Le sous-système intervient pour délivrer chaque fiche concernée par le résultat.

Récupération des fiches concernés par une recherche par leur

identifiant
Le module d'indexation demande à ce module une fiche ou un ensemble de fiches en lui transmettant le ou les identifiants de celles-ci, ainsi que le préfixe du format requis.

Moissonnage des fiches moissonnées

Un utilisateur moissonne les fiches moissonnées par le système ORI, via une requête OAI. Le sous-système intervient pour délivrer l'ensemble

des fiches correspondant aux critères OAI retransmis par ORI-OAI-repository.

Interactions avec les autres modules

Des interactions avec les autres modules se définissent d'aprés les cas d'utilisations précédents : * Exposer les fiches moissonnées au protocole OAI-PMH
Il est assuré par le module ORI-OAI-repository, lequel accède aux fiches stockées via un service fourni par le module : *** un service getRecord(recordID, metadataPrefix)

- Exposer des ensembles de fiches selon les caractéristiques de moisson(dates, format, entrepôt, ensembles).
  La constitution d'ensembles OAI (set) est à la charge du module ORI-OAI-repository. Cependant, le moissonneur doit prévoir de garder la trace des ensembles utilisés pour la moisson, que l'on nommera "ensemble natifs", ainsi que de l'entrepôt d'origine de la fiche. Ces informations pourront être exploitables via un service fournis par le module : *** un service GetRecordsID(from, until, SetSpec, metadataPrefix, storageID)

- Retrouver une fiche associée à un résultat de recherche
  Il est assuré par le module ORI-OAI-indexing, lequel accède aux fiches stockées via un service fourni par le module : *** un service getMetadata(recordID, metadataPrefix)

- Indexer une fiche OAI
  Il est assuré à l'initiative du moisonneur par appel d'un service requis du module ORI-OAI-indexing : *** un service indexRecord

Les sections suivantes recensent l'ensemble des services que le module ORI-OAI-harvester doit fournir aux autres modules dans le cadres de ces interactions, suivi de l'ensemble des services que ce module requiert des autres modules.

Services fournis par l'interface "Frontal moisson"

Service getMetadata

- cas d'utilisation : recherche de fiches
- rôle : Retrouver une fiche associée à un résultat de recherche

- utilisateurs : ORI-OAI-indexing
- paramètres : *** recordID : l'identifiant de l'enregistrement
  - metadataPrefix : le préfixe du format requis pour la fiche

- retour : une chaine comprenant la structure XML des métadonnées propres au format.

Service getRecord

- cas d'utilisation : Exposer les fiches moissonnées au protocole OAI-PMH
- rôle : Retrouver un enregistrement OAI complet
- utilisateurs : ORI-OAI-repository
- paramètres : *** recordID : l'identifiant de l'enregistrement
  - metadataPrefix : le préfixe du format requis pour la fiche

- retour : une chaine comprenant la structure XML de l'enregistrement OAI.

Remarque : en fonction de considération sur les performances, ont peut prévoir l'ajout ou la suppresion de paramètres pour ces services, afin d'optimiser les temps de réponse. Le passage de metadataPrefix est, dans cette optique, optionnel.

Service getRecordsID

Ce service renvoie une liste d'identifiants des enregistrements sélectionnés par les paramètres transmis. Il est appelé par ORI-OAI-repository pour exposer les enregistrements moissonnés.

- cas d'utilisation : Exposer les fiches moissonnées au protocole OAI-PMH
- rôle : Retrouver un ensemble d'enregistrement (leurs identifiants) selon les caractéristiques de moisson(dates, format, entrepôt, ensembles).
- utilisateurs : ORI-OAI-repository
- paramètres : **** metadataPrefix : le préfixe du format des fiches requises. *Obligatoire**
  - from, until : dates au format UTC définit par le protocole OAI-PMH. Sélectionne les enregistrement d'après leur

dateStamp. Peut être "null" (optionnels)
- setSpec : identifiant d'un ensemble. Sélectionne les fiches appartenant à cet ensemble.
- storageID : identifiant d'un entrepôt moissonné

- retour : une liste d'identifiants d'enregistrements stockés, implicitement pour le format défini par le préfixe.

Services requis par le module

Services requis du module ORI-OAI-indexing

Service indexRecord

- cas d'utilisation : moisson d'un entrepôt OAI
- rôle : indexer une fiche de métadonnée
- utilisateurs : ORI-OAI-harvester
- paramètres : ** **metadata: section XML contenant les métadonnées d'un enregistrement. *Obligatoire**
  - prefix : préfixe du format de la fiche
  - recordID : identifiant de l'enregistrement contenant les métadonnées indexées.
  - storageID : identifiant de l'entrepôt moissonné duquel est issue la fiche.

- retour : un code déterminant le résultat de l'indexation

Remarque : toutes ces données sont transmises à l'indexation en tant qu'élements à indexer, à des fins d'utilisation concréte par le module de recherche. Seul le recordID, qui est la clé qui relie les deux modules, est utilisé pour retrouver la fiche.

Service deleteRecord

- cas d'utilisation : moisson d'un entrepôt OAI (mise à jour)
- rôle : supprimer l'entrée dans l'index correspondant à une fiche "deleted"
- utilisateurs : ORI-OAI-harvester
- paramètres : ** recordID : identifiant de l'enregistrement contenant les métadonnées indexées.

Choix techniques

Langages et environnement

Le module devra s'exécuter dans un environnement multiplate-forme J2EE, dans un machine virtuelle compatible Java 1.5. Outre l'aspect multi-plate-forme de cet environnement, l'importante collection d'API et de frameworks disponibles en Open Source renforce la détermination de ce choix.

Persistence des données

Le stockage des moissons se fera dans une base XML dont le choix s'est arrêté sur eXist, qui offre différentes modalités d'utilisation (XML-RPC, SOAP, WEBDAV...), et utilise des standards XML comme XPath, XQuery, XInclude.

Framework, API

OAIHarvester2

Cette application est basée sur le projet OAIHarvester2 d'OCLC Online Computer Library Center, qui fournit les fonctions de requêtes OAI-PMH.

Notes : * limites et adaptation de l'API OAICat : *** org.oclc.oai.server.verb.ListRecords :
Les champs from et until sont validés pour une longueur entre 0 et 10, ce qui force à utiliser une granularité YYYY-MM-DD.

- autres

Spring

Le framework Spring sera utilisé partout où il est pertinent de l'utiliser, et plus généralement, l'utilisations de POJOS sera le choix par défaut pour définir les couches d'accès aux données, ainsi que les autres éléments de configuration.

Spring XML Database Framework

Ce framework établira la liaison entre les classes métiers du moissonneur, et la couche persistence dans la base de donnée XML

Quartz scheduler

La gestion des tâches programmées est réalisée grâce au projet open source Quartz d'OpenSymphony.

XFire

Pour l'interaction entre les modules, des Web Services sont mis en oeuvre à l'aide du framework Codehaus XFire.

Modèle de donnée

Un modèle de donnée s'appuyant sur une base relationnelle repose sur un modèle entité-relation, qui se concrétise par des tables avec des clés primaires et secondaires, ou encore des tables d'association. Pour une base XML, les entités sont concrétisées par des structures XML(documents ou noeuds), et les relations entre ces entités seront définies à travers des attributs servant de clés. Dans ce contexte, les requêtes XQuery remplacent les requêtes SQL.

Unable to render embedded object: File (ERD%20moissonneur.jpeg) not found.

- La relation entre **entrepôt** et **fiche** est assurée par l'attribut **ori-record/@repositoryName** de la fiche et l'attribut **oaistore/@repositoryName** de la définition de la moisson.
- La relation entre **moisson** et **fiche** est assurée par l'attribut **ori-record/@rharvestID** de la fiche et l'attribut **harvest/@id** de la définition de la moisson.

Structures XML

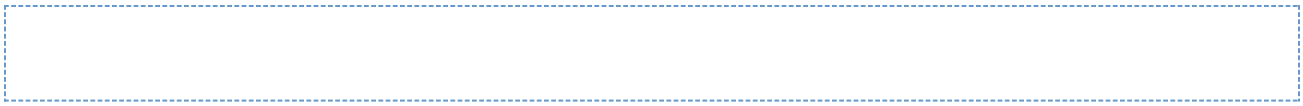
Entités de configuration

Suit la description des entités servant à la configuration. Chaque description est de la forme : Nom - Description. * HarvestConfig - Définition d'une moisson

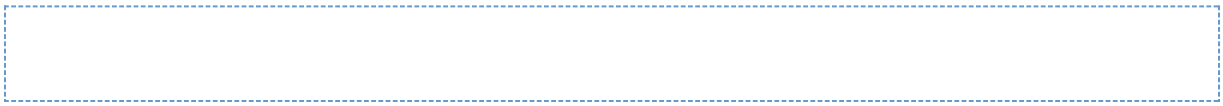
```
<harvest id="harvest1" onLaunch="false" metadataPrefix="lom" collection="lom" from="2006-11-08">
   <oaistore url="http://cas.enseeiht.fr/injac-oai/OAIHandler"/>
<schedule cron="0 30 23 * * ?"/>
</harvest>
```

Entités d'information

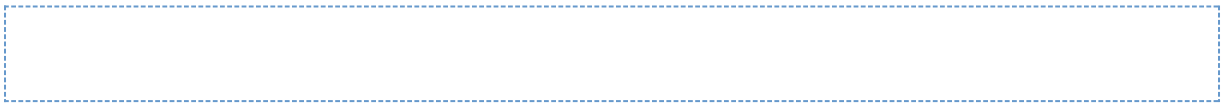
Les entités d'information sont générées aucours des différentes actions. Chaque description est de la forme : Nom - Description - Action(s) impliquées. * OAIStoreInfos (FormatInfos*, SetInfos*) - Informations sur l'entrepôt OAI - Création ou enregistrement d'une définition :



- HarvestReport (CollectReport*) - Rapport de moisson - Moisson :



- CollectInfos (CollectStoreInfos*) - Etat des récoltes - Administration :



Entités de production

Ce sont les entités produites par le module. Chaque description est de la forme : Nom - Description - Action(s) productrice(s).

- ORIRecord - fiche OR,I représentant un enrobage de la fiche OAI - Moisson :

```
<ori:ori-record xmlns:ori="http://ori-oai.org/ORI/1.0/" repositoryName="INP ENSEEIHT" harvestID=
"harvest1" ORI_id="xxx">
<oai:record/>
</ori:ori-record>
```

Requêtes XQuery

Voci quelques modèles de requêtes XQuery utilisées par le module, qui définissent les relations entre les entités :

- liste des identiants ORI pour un entrepôt et un préfixe donné :

```
declare namespace oai="http://www.openarchives.org/OAI/2.0/";
declare namespace ori="http://ori-oai.org/ORI/1.0/";


for $item in //ori:ori-record[@repositoryName ='INP ENSEEIHT' and
contains(name(./oai:record/oai:metadata/*[1]),'lom')] return
<identifier>
{$item/@ori_ID}
</identifier>
```

- liste des identiants ORI pour une moisson et un entrepôt donnés : relation entre moisson et fiche

```
for $item in //ori:ori-record[@repositoryName ='INP ENSEEIHT' and @harvestID='harvest1']
```

L'identifiant ORI_id premet de retrouver une fiche d'un format précis pour un document donné. Il sert également de nom de ressource à l'intérieur de la base XML. Cependant, ce nom de resources interdit l'usage de certains caractères, aussi une conversion à lieu pour faire l'aggrégation préfixe + OAI id (classe org.ori.harvesting.IDConverter).

Algorythme de mise à jour

persistent : * on distingue les fiches modifiées et crées des fiches supprimées par l'élément status=deleted (from)

- pour chaque ensemble on applique les traitement d'ajout mise à jour ou de suppression

transient : * on distingue les fiches modifiées et crées des fiches supprimées par l'élément status=deleted (from=last)

- on ajoute/met à jour les fiches du premier ensemble, et on supprimes celles qui sont indiquées comme telles
- on compare les fiches until=last de l'entrepôt avec les fiches until=last précédemment moissonnées, et on supprime les fiches appartenant au delta

no : * on ajoute/met à jour les fiches de l'ensemble from=last sans distinction

- on compare les fiches until=from de l'entrepôt avec les fiches until=from précédemment moissonnées, et on supprime les fiches appartenant au delta

Scenarii de test

Tests unitaires

Moisson initiale

Une première moisson est lancée, pour laquelle il faut vérifier ques les fiches moissonnées ont bien été stockées.

Mise à jour de moisson d'un entrepôt non-persistent

Le test nécessite les phases suivantes :

- Une moisson initiale est réalisée avec n document
- Une mise à jour est opérée a n documents
- Une mise à jour est opérée a n+x documents
- Une mise à jour est opérée a n-x documents

Diagramme de package

Ce diagramme présente les différents packages du module, et leurs interactions :

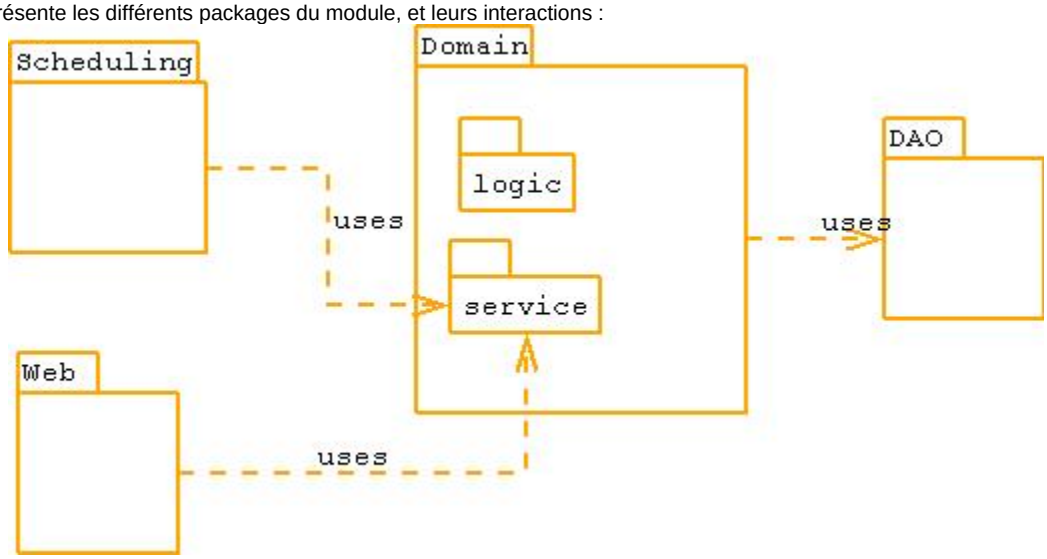


Diagramme de classe

- Diagramme général :
  Dans ce diagramme, les trois classes de **services** qui sont au centre assurent tous les services du module : ** **ProviderService** expose des Web Services utiles aux autres modules
    - **HarvestService** présente des services internes, masquant la logique métier au contrôleur.
    - **SchedulerService** présente des services internes pour l'exécution des tâches programmées.

- Réalisations d'interfaces :
  
- Ce diagramme présente les implémentations fournies pour les interfaces utilisées : *** IRecordStorage et IConfigStorage sont les interfaces de la DAO. Les impléme,ntations fournies concernent une base XML et utilisent Spring XML DB.
    - ISchedulerService est une interface pour gérer les tâches programmées. L'implémentation fournie utilise Quartz.
    - IIndexerInvoker est une interface pour l'indexation des fiches. L'implémentation fournie utilise Codehaus XFire.

- Vues et controleurs
  Trois types de vues sont prévues pour l'IHM du moissonneur, correspondant à trois controleurs : ** **les vues concernant les *définitions** des moissons utiliseront le controleur HarvestController
    - les vues concernant les **récoltes** produites par les moissons utiliseront le controleur CollectController
    - les vues concernant les **rapports** des moissons utiliseront le controleur ReportController.

Diagrammes de séquences

Voici les diagrammes de séquences principaux du module :

- Création du HarvesterManager :
  
- Moisson OnLaunch :

!Moisson_OnLaunch_SD.jpg|width=32,height=32!* Moisson programmée :

Idées d'évolutions futures

- Intégrité/validation
    - tester si la tâche programmée est compatible avec la granularité de l'entrepôt. (1 tache par jour max pour les granularité YYYY-MM-DD)

- rapports
  Exportation des rapports, diagrammes

Changements de version

Changements de la version 1.4.0

- ajout d'un tri alphabétique et hiérarchique dans l'affichage des ensembles OAI d'un entrepôt (feature 4658).
- améliorations de l'ergonomie(feature 4649)
- définition multi set : union, intersection, exclusion d'ensembles(feature 3446)
- suppression du support pour persistance XML DB (utiliser une version précédente si vous n'avez pas déjà migrée votre base eXist)
- correction de bugs

Installation

Il existe plusieurs modes d'installation de ce module. Le mode recommandé est l'utilision ori-oai-commons-quick-install. Ceci vous permettra de déployer la suite ori-oai avec un minimum de personnalisation tout ceci en utilisant un seul fichier de configuration.

L'installation manuelle vous fera éditer manuellement différents fichiers afin de configurer au mieux votre application.

Il est préférable d'utiliser la première solution. En effet, celle-ci vous apportera un déploiement rapide de ORI-OAI sur un serveur de production avec une configuration de base. Vous pourrez toutefois après cette installation apporter toutes les configurations avancées que vous souhaitez à vos modules.

Reportez-vous à la documentation en ligne d'installation de ORI-OAI.

Installation manuelle

Pré-requis

JDK

Le moissonneur ORI-OAI est une application Java fonctionnant avec un **JDK 1.5** ou supérieur. La variable d'environnement JAVA_HOME doit exister et pointer sur le répertoire d'installation du JDK.

Toutefois, si cette variable pointe sur un autre JDK, les scripts **ant.bat** (pour Windows) et **ant.sh** (pour Linux) sont fournis, dans lesquels vous pouvez définir l'emplacement d'une JDK 1.5 utilisée de façon alternative pour le moissonneur.

ANT

Les tâches de compilation, de déploiement et certaines actions utilisent ANT 1.6. La variable d'environnement ANT_HOME doit exister et pointer sur le répertoire d'installation de ANT.

Tomcat

Une version de Tomcat 1.5.* doit être disponible sur la machine de déploiement, et la variable d'environnement CATALINA_HOME doit être définie pour pointer son emplacement.

Pour assurer l'application de fonctionner avec l'encodage UTF-8, il ajouter cette ligne dans le fichier startup.sh ou catalina.sh (répertoire tomcat-xxx/bin) : export CATALINA_OPTS="**-Dfile.encoding=UTF-8** $CATALINA_OPTS"

Stockage et indexation des moissons

La version 1.4 intègre une seule forme de stockage SQL, et délaisse l'implémentation XML trop longue à maintneir en parallèle et non utilisée. Pour une migration depuis la base XML, utiliser la version 1.1.3. L'outil de migration se limite toutefois à exporter les définitions de moissons, l'utilisateur devant ensuite lancer chaque moisson une par une afin de reconstituer les récoltes et de les ré-indexer.

Le moissonneur intègre deux formes de traitement des moissons :

- Le stockage se fait dans une base relationnelle, configurable dans le fichier ori.properties avec les paramètres de connexion adéquats (voir section plus bas).

Une base doit donc être prélablement crée*. Pour MySQL, on s'assurera qu'elle soit de type InnoDB afin de gérer les transactions. Pour ce faire, le fichier my.cnf doit comporter, selon la version, une des deux lignes suivantes :
Versions récentes de MySQL :

```
default-storage_engine = innodb
```

Versions anciennes de MySQL :

```
default-table-type = innodb
```

Ligne de création pour MySQL :

```
create database `ori-harvester` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

- L'indexation des métadonnées dans un index Lucene : une installation préalable du module d'indexation ORI-OAI-INDEXING, doit donc être dispônible en accès HTTP depuis le moissonneur. La [documentation technique|http://www.ori-oai.org/documentation/doc+technique/] des modules ORI-OAI est disponible sur Internet.
  Voir la section Configuration du service d'indexation pour plus de détails.

Migration

Pour effectuer une migration depuis une installation antérieure vers la version 1.1, il faut simplement configurer le partie eXist du fichier ori.properties et la partie SQL, et ensuite lancer les target :

```
ant init

ant upgrade
```

ant upgrade produit les modifications nécessaires à la base de données entre deux versions : elle marche conjointement avec l'attribut **upgrade.fromVersion** du fichier init-build.properties, que vous devez renseigner AVANT de lancer cette target.

Dans le cas d'une migration de la version 1.0 à 1.1.0, cette dernière target ANT transfert les définitions de moissons de votre base eXist vers la nouvelle base SQL, initialisée par ant init. Dans les autres cas elle fait les modifications de tables éventuelles de la base SQL nécessaires à la nouvelle version.

Configuration et installation manuelle du moissonneur

Une possibilité de configuration centralisée permet de changer automatiquement la valeur de certains paramètres entre crochets et en majuscules, grâce à la configuration du module **quick-install**. Les fichiers **build.properties** et **ori.properties** sont concernés. Ainsi, vous avez le choix entre une installation "par module" où vous devez **créer et editer** ces fichiers à partir de init-build.properties et conf/properties/ori.example.properties, et une installation "centralisée" où ces fichier seront automatiquement générés lors de la compilation et du déploiement. Les sections suivantes décrivent l'installation "par module", pour l'installation centralisée, se reporter en plus à la documentation du quick-install. Les paramètres ne figurant pas entre crochets peuvent de façon facultative être changés à la main.

L'installation se fait en 5 étapes :

1. décompression de l'archive ori-harvester-x-x.zip

2. configuration du fichier build.properties

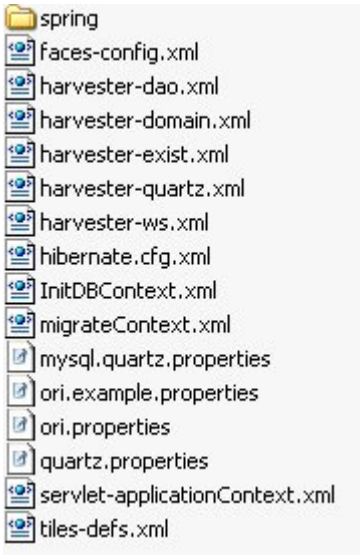
Un fichier build.properties doit être crée par recopie de init-build.properties.
Il faut indiquer l'endroit ou l'on veut installer l'application, l'emplacement du Tomcat, ainsi que l'emplacement d'un JDK 1.5+. Les paramètres **entre crochet** sont à modifier, ou seront modifés par la configuration centrale du quick-install :

```
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
## 2) Installation manuelle du module
# Dans ce cas, il est nécessaire de commenter
# le paramètre commons.parameters.central.file.url
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#

# Cette partie du fichier doit être mise à jour avant
# la première utilisation de votre application dans votre environnement

#JDK 1.5 directory
java.home=[JAVA_HOME]
#Tomcat directory
tomcat.home=[PATH_TOMCAT_HARVESTER]
#Deployment directory
deploy.home=[PATH_TOMCAT_HARVESTER]/webapps
#Nom de ditribution de l'application - distribution name
app.name.deploy=[CONTEXT_HARVESTER]

#Log Directory (optionnal - default : ${tomcat.home}/logs )
#log.dir=E:/logs
#Log mode : prod or debug
log.mode=prod
# optional - needed for upgrading from a previously installed version
upgrade.fromVersion=[HARVESTER_UPGRADE_PREVIOUS_VERSION]
```

1. Les chemins utilisant des anti-slashes comme sous Windows, doivent s'écrire soit avec un double anti-slash soit avec un slash, comme sous Linux.
2. si le JAVA_HOME de la machine n'est pas par défaut un JDK 1.5, il faut indiquer le chemin d'un jdk 1.5+ dans le fichier **ant.bat** (pour Windows) ou **ant.sh** (pour Linux).
Mise à jour de la base de données : indiquer la version depuis la quelle vous faites la mise à jour. La tâche **ant upgrade** fera les modifications en fonction de cette valeur

3. Adapter les fichiers du répertoire properties

Les fichiers de propriétés sont situés dans le répertoire conf/properties à la racine du projet :



- Configuration rapide :
Le fichier définissant les traces de l'application (log4j.properties) est produit automatiquement lors du déploiement de l'application.
Un fichier ori.properties*doit être créé* à partir de l'exemple ori.example.properties, dans lequel les valeurs **entre crochet** doivent être adaptées à la main si le quick-install n'est pas utilisé :

```
# propertie file for ORI Harvester

############### Database Section ################
hibernate.connection.driver_class=[HARVESTER_SQL_DRIVER_CLASS]
hibernate.connection.url=[HARVESTER_SQL_CONNECTION_URL]
hibernate.connection.username=[HARVESTER_SQL_USERNAME]
hibernate.connection.password=[HARVESTER_SQL_PASSWORD]
hibernate.properties.dialect=[HARVESTER_SQL_DIALECT]


#harvester global options

# set to true each time you want to update database with XML config file
harvester.reloadconfig=false
# optional - if reloadConfig = true
harvester.initconfig=[PATH_TOMCAT_HARVESTER]/[CONTEXT_HARVESTER]/WEB-INF/config/harversterConfig.x
//[HOST_INDEXING]:[PORT_INDEXING]/[CONTEXT_INDEXING]/xfire/IndexingService?WSDL
indexing.ws.lookupServiceOnStartup=false

#enable filtering local records (must match ORIRecordFactory.repositoryIdentifier property of
your repository)
local.repositoryIdentifier=[REPOSITORY_IDENTIFIER]

# set to true ONLY if you want to harvest in spite of unreachable indexing Web Service
indexing.ignore.error=false

#xml database - optional - for migration from version 1.0 only (must be previously created)
exist.url=[EXIST_URL]
exist.login=[EXIST_USERNAME]
exist.password=[EXIST_PASSWORD]
exist.collection=[HARVESTER_EXIST_COLLECTION]

# if true, persist deleted OAI records in database
storage.persistDeleted=true

# Define proxy parameters here (for external  WS access to indexer or stores) or leave blank if
no proxy
# example :
# proxy.host=proxy-rech.my-univ.fr
# proxy.port=3128
proxy.host=[PROXY_HOST]
proxy.port=[PROXY_PORT]
```

La signification de ces valeurs :
harvester.reloadconfig : si vous choisissez de définir des moissons par le biais d'un fichier XML, il faut positionner cette valeur à TRUE au premier lancement, et à FALSE ensuite. Si il est à FALSE , le fichier défini par le paramètre **harvester.initconfig** ne sera pas lu.
harvester.initconfig : emplacment du fichier de pré-configuration de moissons.
indexing.ws.url : URL du Web Service du moteur d'indexation
local.repositoryIdentifier : identifiant du repository local (doit correspondre au paramètre **ORIRecordFactory.repositoryIdentifier** du fichier ori-oaicat.properties)
*exist.url * : url de la base eXist servant au stockage des fiches (jusqu'à version 1.0)
exist.login et **exist.password** : login et mot de passe d'un utilisateur autorisé à se connecter sur la base définie par le paramètre **exist.collection**
exist.collection : nom de la collection racine utilisée par le moissonneur dans la base XML.
storage.persistDeleted : à TRUE, les enregistrements supprimés seront conservés dans la base XML, à FALSE, effacés
proxy.host et **proxy.port** : paramètres du proxy le cas échéant pour l'accès aux services extérieurs.

- Configuration avancée :
  Les fichiers de configuration Spring et JSF (dont la liste suit) permettent de configurer l'application d'une façon plus poussée et plus personnalisée, si le besoin s'en fait sentir.
  - faces-config.xml permet de modifier les régles de navigation et de déclarer des composant JSF personnalisés (avancé)
  - harvester-dao.xml définit la couche d'accès aux données (ici base XML eXist)
  - harvester-domain.xml contient les objets des services métiers
  - harvester-quartz.xml régle la partie "tâche programmée" de l'application (avec le fichier quartz.propeties qu'il référencie)
  - harvester-ws.xml gère les accès aux Web Services
  - servlet-applicationContext.xml est le fichier principal Spring
  - tiles-def.xml sont les templates de pages TILES utilisés dans l'IHM.

- faces-config.xml permet de modifier les régles de navigation et de déclarer des composant JSF personnalisés (avancé)
  Enfin, le fichier de log peut être personnalisé en modifiant le fichier log4j.prod.properties en mode production ou log4j.debug.properties en mode développement/ déboggage.

4. Configurations avancées (optionnel)

Définir un fichier de pré-configuration pour les moissons. Cette étape est optionnelle, les moissons pouvant plus confortablement être définies par l'IHM une fois l'application lancée.
Se reporter à la page suivante.

5. Déploiement

Lancer les target *ant * suivantes :

```
ant init

ant all
```

Vous pouvez alors accéder à l'interface Web avec le contexte **ori-harvester**, par exemple : http://localhost:8080/ori-oai-harvester

Configurations avancées

Filtrer le traitement des fiches par leurs identifiants

La propriété **local.repositoryIdentifier** désigne l'identifiant du repository local, qui permet d'exposer les fiches locales du workflow.

Cette propriété permet de filtrer le traitement des fiches locales afin qu'elles ne soient pas ré-indexées avec leur identifiant OAI, en doublon avec leur indexation depuis le workflow.

Plus généralement, pour empecher l'indexation de certaines fiches, on peut définir d'autres filtres en ajoutant des entrées dans la propriété **idFilters** du fichier **conf/properties/harvester-domain.xml** , dans le bean **harvestServiceNoTx** :

```
<property name="idFilters">
 <list>
  <value>${local.repositoryIdentifier}</value>
               <value>[filtre identifiant fiches]</value>
 </list>
</property>
```

- **[filtre identifiant fiches]** représente la valeur avec laquelle doivent être filtrées les fiches.

Définir des moissons par fichier

Les moissons OAI peuvent être caractérisées par deux ensembles d'éléments disincts, les **sources** des moissons, définies par les éléments suivants :

- les **adresses** des entrepôts OAI à moissonner
- le **type de format** à moissonner pour chacun de ces entrepôts (LOM, Dublin Core, ETDMS...)
- et optionnellement les **ensembles** (sets) OAI spécifiques
  et les **destinations** ou lieux de stockage de ces moissons, incluant ou non un traitement (base de données, indexation...).

Pour charger ce fichier dans la base du moissonneur, il faut renseigner les propriétés harvester.reloadconfig à true et harvester.initconfig avec le chemin complet vers le fichier XML, et lancer la target :

```
ant init-config
```

Un exemple de fichier de configuration est fourni , dans le répertoire WEB-INF/config de l'installation : harvesterConfig.example.xml.

Configuration des sections moissons

Les moissons proprement dites se configurent dans l'élément <harvests>, incluant une ou plusiers balises <harvest>:

```
<?xml version="1.0" encoding="UTF-8"?>
<harvests>
    <harvest id ="injacs" onLaunch="no" metadataPrefix="lom" collection="lom">
        <oaistore url="http://cas.enseeiht.fr/injac-oai/OAIHandler" />
        <schedule cron="0 0 23 * * ?"/>
    </harvest>
    <harvest id ="theseINP" onLaunch="no" metadataPrefix="oai_dc" collection="oai_dc">
        <oaistore url="http://ethesis.inp-toulouse.fr/perl/oai2" />
        <schedule cron="0 30 23 * * ?"/>
    </harvest>
    <harvest id ="docinsa" onLaunch="no" metadataPrefix="oai_dc" collection="oai_dc">
        <oaistore url="http://docinsa.insa-lyon.fr/oai/oai2.php" />
        <schedule cron="0 30 0 * * ?"/>
    </harvest>
    <harvest id ="MIT" onLaunch="no" metadataPrefix="oai_dc" collection="oai_dc">
        <oaistore url="http://dspace.mit.edu/dspace-oai/request" />
        <schedule cron="0 30 1 * * ?"/>
    </harvest>
</harvests>
```

La structure de cette balise est la suivante :

- attributs de l'élément <harvest>
    - id : identifiant de la moisson
    - metadataPrefix : préfixe des metadonnées moissonnées
    - collection : permet un sous-classement à l'intérieur d'une moisson (optionnel)
- sous-éléments <oaistore> de l'élément <harvest>
    - **attributs de l'élément <oaistore>**
        - url : adresse d'un entrepôt OAI.
- sous-éléments <schedule> de l'élément <harvest>
    - **attributs de l'élément <schedule>**
        - cron : une chaine utilisant la syntaxe CRONTAB

Niveau de déboggage

Le niveau de débogage peut être modifier en éditant le fichier properties\log4j.*.properties. Deux modèles de fichiers sont fournis, log4j.prod.propeties en mode production, et log4j.debug.properties en mode débogage.

.Les niveaux disponibles sont : **debug**, **info**, **warn**, **error** et **fatal**, du plus prolixe au plus concis. Le niveau de déboggage influe sur les performances de l'application.

Note : il faut adapter les chemins des fichiers de log, par exemple à la ligne :

```
log4j.appender.R.File=E:\Java\jakarta-tomcat-5.0.28\logs\ori-harvester.log
```

Test

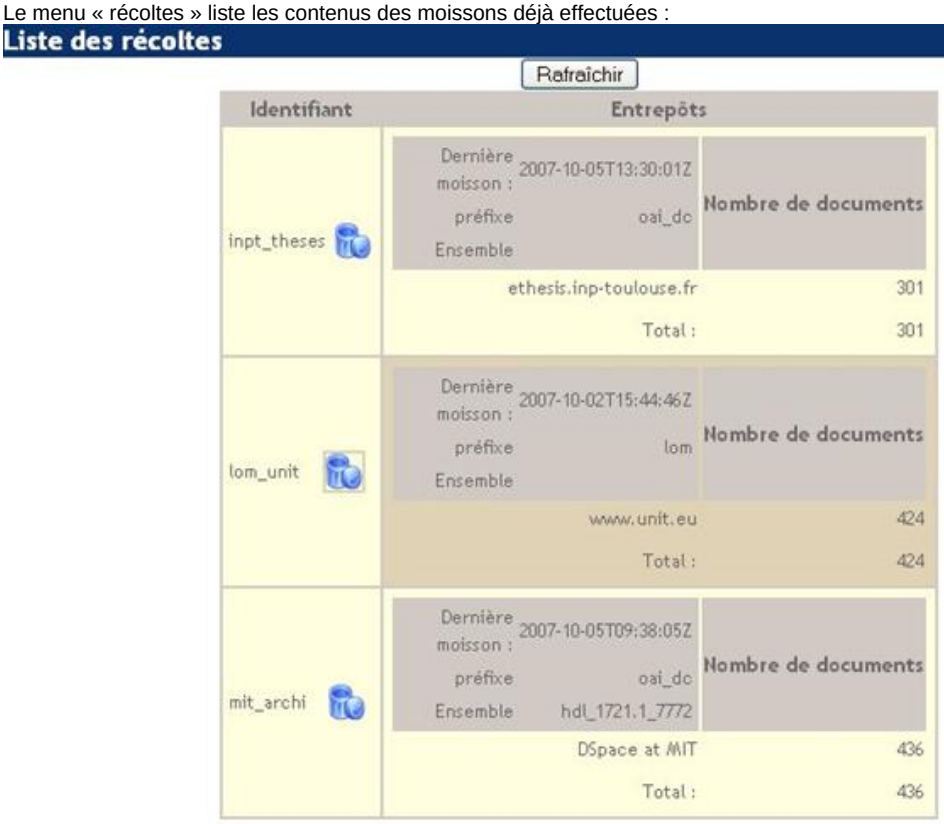
L'accès à l'interface du moissonneur se fait par l'URL :

```
http://[HOST_INSTALL]:8181/ori-oai-harvester
```

Pour lancer une moisson, cliquez sur la flèche verte correspondante, et vérifiez l'état de la récolte dans l'onglet « Récolte ». Dés qu'une date de « dernière moisson » apparaît, c'est que la récolte est terminée. Appuyez sur le bouton « rafraîchir » si ce n'est pas le cas, jusqu'à obtention de la date.

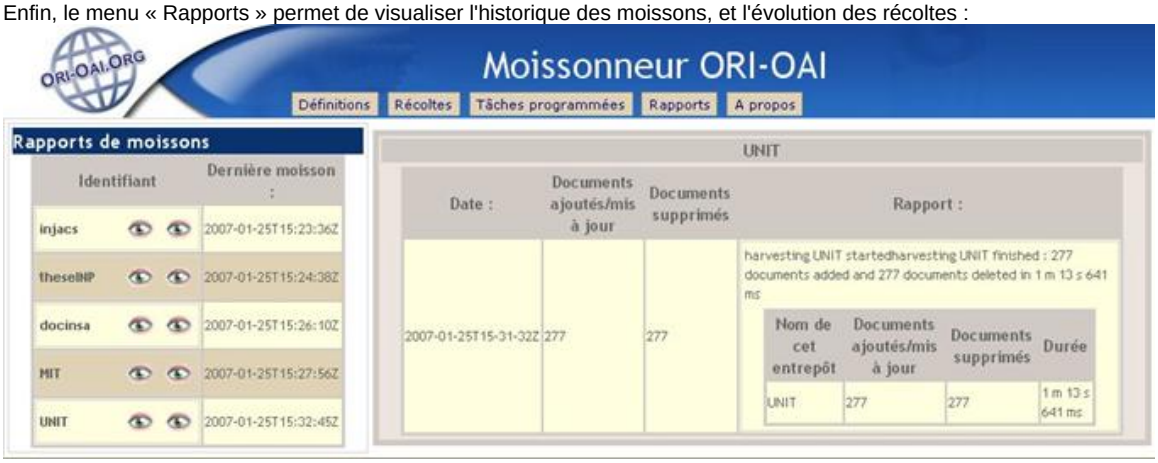
Quand la moisson est terminée, consultez le rapport dans l'onglet « rapports ». Cette page liste les rapports des moissons effectuées pour chaque définition.

Le menu « récoltes » liste les contenus des moissons déjà effectuées :



Le menu « Tâche programmées » liste des informations sur toutes les programmations dont font l'objet les définitions de moissons :

Enfin, le menu « Rapports » permet de visualiser l'historique des moissons, et l'évolution des récoltes :



Si tout s'est bien passé, vous pouvez rechercher les documents moissonnés dans l'interface de recherche (ori-oai-search) à l'URL :

```
http://[HOST_INSTALL]:8184/ori-oai-search
```

Exemple « non contractuel » :

Aspects pratiques

Encodage et fichier properties Java

Les modules ORI-OAI utilisent un mécanisme de placeholder permettant de définir des paramètres dans des fichiers *.properties uniques pour chaque module (voire touts les modules avec le commons-parameter.properties du quick-inistall), afin d'éviter de toucher aux différents fichier XML qui définissent les Beans Spring.

Il existe une possibilité de paramétrage de l'encodage du placeholder qui peut resoudre certains soucis :

```
<property name="fileEncoding">
      <value>ISO-8859-1</value>
 </property>
```

Cet encodage est fonction de celui utilisé par l'editeur avec lequel on modifie le fichier properties...
Si on a le choix, rappelons que le standard pour les fichiers properties est ISO-8859-1.

Utiliser les Web services

Utiliser les Web services

Le moissonneur expose des services SOAP pour récupérer les fiches qui ont été integrées.

URL du service et du fichier descripteur

L'URL du service est le suivant :
{url de déploiement]**/ws/xfire/HarvesterWebService** _Exemple : http://localhost:8080/ori-oai-harvester/ws/xfire/HarvesterWebService_ L'URL du fichier descripteur WSDL est la suivante :
{url de déploiement]**/ws/xfire/HarvesterWebService?WSDL** _Exemple :
http://localhost:8080/ori-oai-harvester/ws/xfire/HarvesterWebService?WSDL_ *Note : ceci est paramétré dans le fichier WEB-INF/xfire-servlet.xml* h2. Configuration d'un client avec Spring et XFire
Pour configurer un client de ce service avec Spring et Xfire, suivre les étapes suivantes : * Inclure le jar Xfire dans votre librairie, disponible sur [la page de téléchargement|http://xfire.codehaus.org/Download] (ou reprendre le jar inclus dans le moissonneur xfire-all-1.2.4.jar) * Ajouter le fichier Java de l'interface dans les sources : /org/orioai/harvesting/domain/service/ProviderService * Enfin, déclarer un bean dans le contexte d'application :

```
<bean id="indexingWebServiceClient"
class="org.codehaus.xfire.spring.remoting.XFireClientFactoryBean">
<property name="serviceClass">
<value>
org.orioai.harvesting.domain.service.ProviderService
</value>
</property>
<property name="wsdlDocumentUrl">
<value>[http://localhost:8080/ori-oai-harvester/ws/xfire/HarvesterWebService?WSDL]</value>
</property>
</bean>
```

Ensuite, utiliser ce bean avec les méthodes implémentées par l'interface ProviderService :

```
public interface ProviderService {
  /**
   * Returns a metadata record for a given id and prefix
   * @param id
   * @param prefix
   * @return string containing xml record
   */
 String getRecordFromPrefix(String id, String prefix);
 /**
   * Returns a metadata record for a given id and namespace
   * @param id
   * @param namespace
   * @return string containing xml record
   */
 String getRecordFromNamespace(String id, String namespace);
}
```

Utilisation

Menus de l'interface Web

Définitions

L'onglet "définitions" liste les moissons qui ont été définies, et permet de mener les actions suivantes :

- Définir une nouvelle moisson : ouvre la page d'édition d'une définition en mode "création" (voir image plus bas)
- Editer une définition : ouvre la page d'édition d'une définition en mode "édition". Ce mode est plus restricitif que le mode "création" dans le sens où l'on ne peut pas modifier la liste des entrepôts.
- Supprimer une définition : supprime la définition (icone poubelle).
  Attention, cela ne supprime pas la récolte, c'est-à-dire l'ensemble des fiches moissonnées et indexées : il faut la supprimer AVANT de supprimer la définition, dans l'onglet "Récoltes".

- Lancer une moisson : lance la moisson correspondant à la définition (icone flèche verte).
- Récupération : cette fonction permet de ré-indexer les fiches déjà moissonnée, dans le cas où l'index a été altéré ou endommagé. La marche à suivre est la suivante :
  - restauration complète :
    - supprimer l'index du module ori-oai-indexing
    - lancer la restauration (bouton "Tout réindexer")
  - restauration partielle :
    - choisir une date au format AAAA-MM-JJ
    - lancer la restauration (bouton "Tout réindexer depuis cette date")

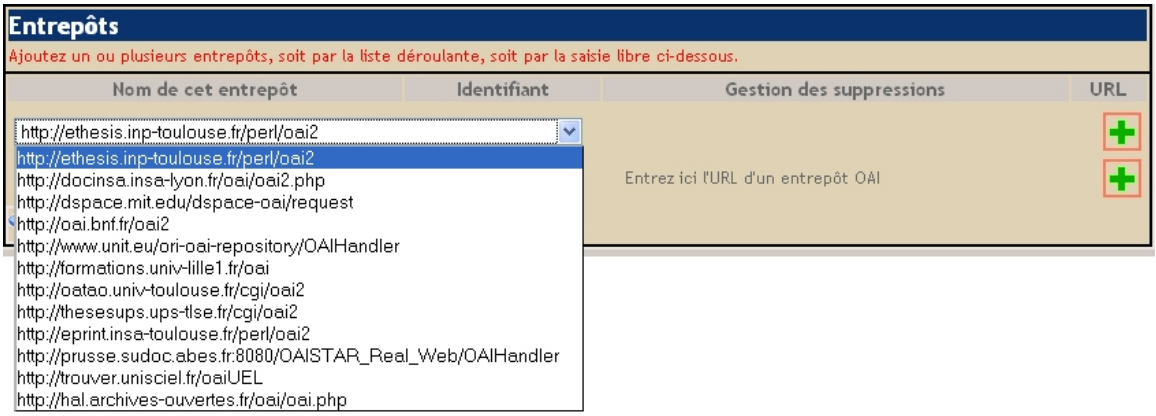
Création et modification d'une définition

En mode modification, on ne peut pas ajouter ou supprimer d'entrepôts ni changer l'identifiant de la moisson. Pour le reste, les deux modes ont une interface identique :

- Identifiant : nom unique de la définition
- Préfixe OAI : menu déroulant indiquant les préfixes disponibles pour les entrepôts. N'affiche que les préfixes communs à tous les entrepôts.
- Ensemble : menu déroulant indiquant les ensembles (sets OAI) disponibles pour les entrepôts. N'affiche que les ensembles communs à tous les entrepôts.
- Tâches programmées : permet de définir la ou les occurences des moissons.
- Entrepôts : ajouter un entrepôt soit par le menu déroulant, soit en entrant une nouvelle URL pour un entrepôt OAI. Cliquer sur l'icone en forme de "plus" vert pour valider le choix.

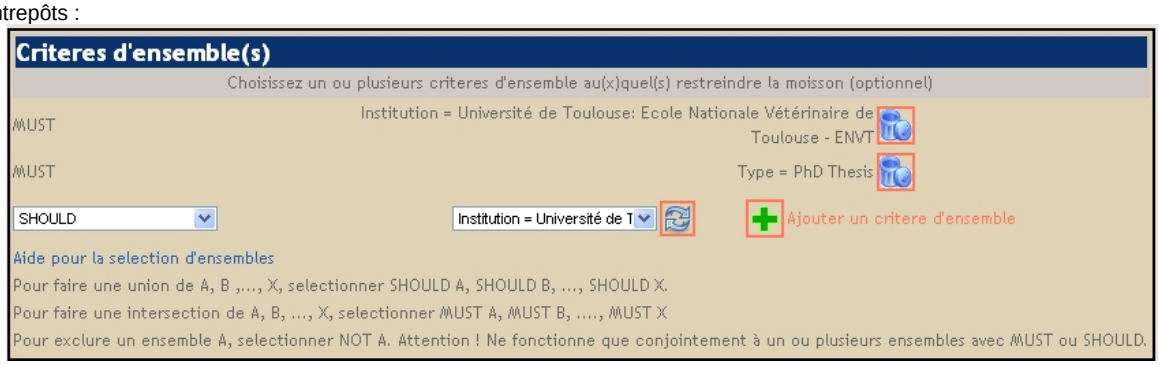
1. Initialisation d'une nouvelle définition de moisson

Pour créer une nouvelle moisson, vous devez déjà chosir un entrepôt :



2. Critères de moissons

Une fois un ou plusisuers entrepôts choisis, vous pouvez (**optionnel**) restreindre la moisson en séléectionnant des ensembles disponibles dans le ou les entrepôts :



Dans cet exemple, on définit une intersection entre l'ensemble "Toulouse - ENVT" et "Type = PhD Thesis" afin de ne moissonner que les thèses de l'Ecole de Véterinaires.
Si aucun ensemble n'est choisi, **tout l'entrepôt sera moissonné**.

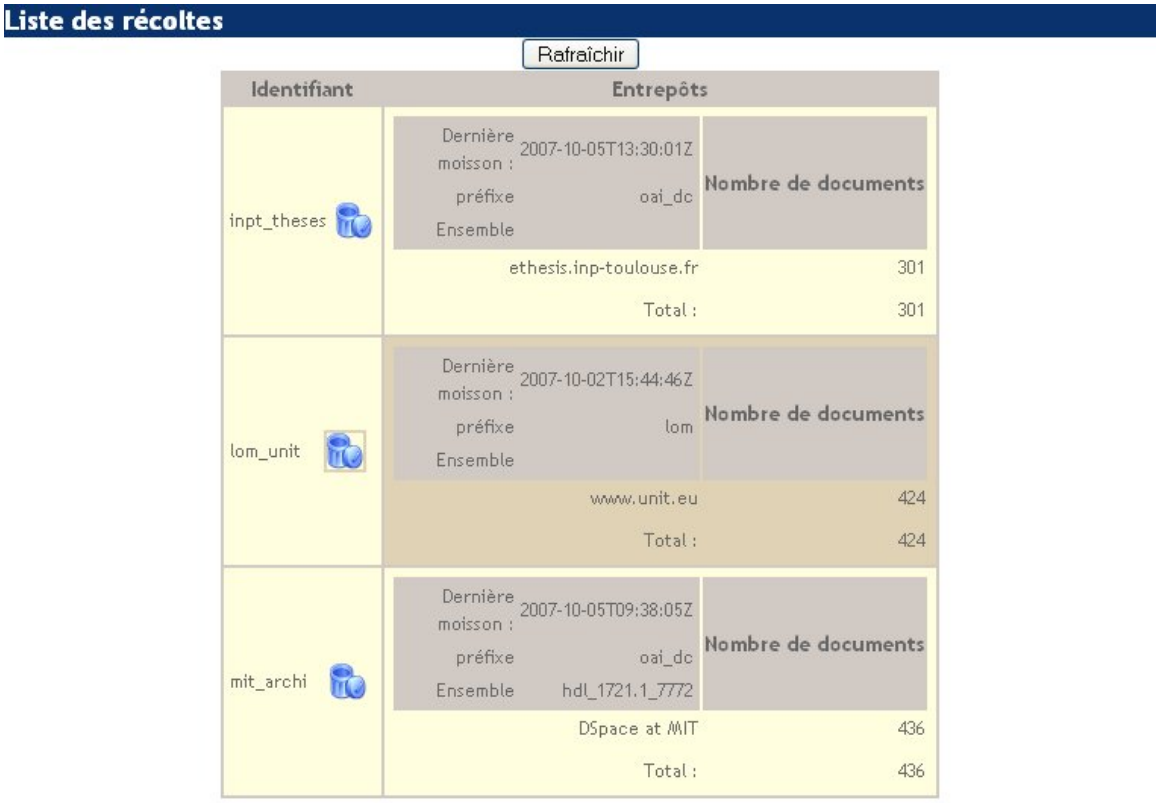
3. Autres réglages : tâches programmées

Récoltes

L'onglet "récoltes" permet de visualiser le nombre de documents qui ont été récoltés et qui sont stockés dans la base XML. Pour rafraîchir le calcul des fiches, appuyer sur le bouton "Rafraichir".

La seule action possible sur cette page est la suprresion d'une récolte, c'est-à-dire de l'ensemble des fiches correspondant à la moisson.

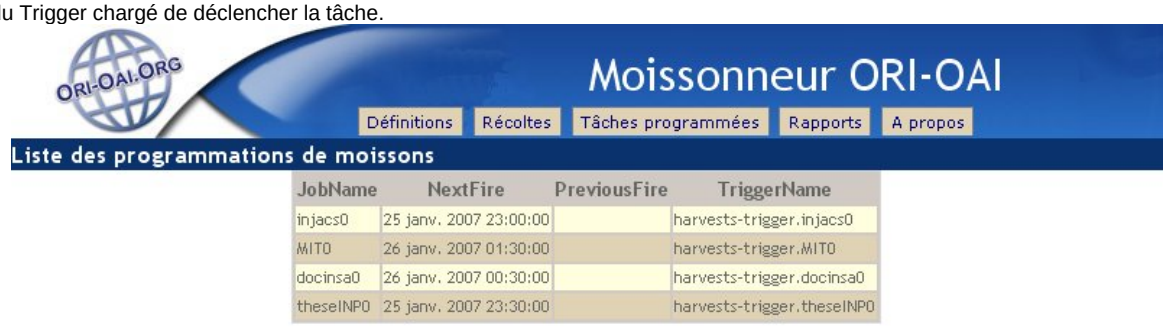
Cette action peut-être utile dans le cas où on ne veut plus des fiches de cette définition, ou dans le cas où l'on veut recommencer une moisson depuis le début. Dans ce dernier cas, il faut supprimer la récolte, puis effacer le champ "from" dans les paramètres avancés de la définition de la moisson, et relancer la moisson.



Tâches programmées

Cet onglet liste certaines informations relatives aux tâches programmées en cours, issues des différentes définitions de moisson. On y trouve les renseignements suivants : * JobName : nom de la tâche, qui reprend l'identifiant de la moisson suivi d'un chiffre.

- NextFire : prochain lancement prévu de la tâche
- PreviousFire : date d'éxecution de la dernière fois où a étée lancée la tâche.
- Nom du Trigger chargé de déclencher la tâche.

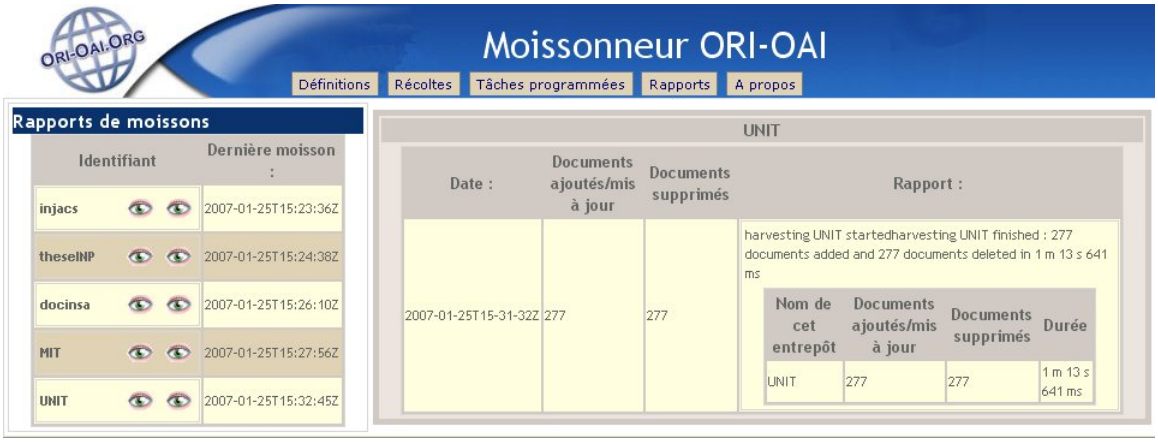


Rapports

Cet onglet permet de consulter les rapports concernant les moissons qui ont été exécutées. Pour chaque définition, on peut voir simplement le dernier rapport (icône oeil de gauche), ou l'ensemble des rapports depuis la première moisson (icône oeil de droite).

Chaque rapport contient les informations suivantes : * La date d'exécution de la moisson

- le nombre de documents ajoutés ou mis à jour (total et par entrepôt)
- le nombre de documents supprimés (total et par entrepôt)


ORI-OAI.ORG
Moissonneur ORI-OAI
Définitions
Récoltes
Tâches programmées
Rapports
A propos
Rapports de moissons
Identifiant
Dernière moisson :
injacs
2007-01-25T15:23:36Z
theseINP
2007-01-25T15:24:38Z
docinsa
2007-01-25T15:26:10Z
MIT
2007-01-25T15:27:56Z
UNIT
2007-01-25T15:32:45Z
UNIT
Date :
Documents ajoutés/mis à jour
Documents supprimés
Rapport :
2007-01-25T15-31-32Z
277
277
harvesting UNIT startedharvesting UNIT finished : 277 documents added and 277 documents deleted in 1 m 13 s 641 ms
Nom de cet entrepôt
Documents ajoutés/mis à jour
Documents supprimés
Durée
UNIT
277
277
1 m 13 s 641 ms