

Agence ou Service : ISTAS

Projet : ULISSE/SITTOOLS2



Dossier de conception SITools2 V2

Rédigé par : David ARPIN Jean-Pascal BOIGNARD Bastien FIORITO Mathieu GOND Matthieu MARSEILLE	Diffusé à : Jean-Christophe MALAPERT
Approuvé par : Jean-Pascal BOIGNARD	

LISTE DES MODIFICATIONS DU DOCUMENT

Vers.	Date	Paragraphe	Description de la modification
01.0	12/11/12		Création du document

■ Table des matières

1	Introduction	6
2	Conception préliminaire	7
2.1	Description générale	7
2.1.1	Partie Serveur	7
2.1.2	Partie cliente	7
2.2	Architecture de système	7
2.3	Description de chaque module	7
3	Conception détaillée	8
3.1	Description détaillée de chaque module	9
3.1.1	Structure générale d'un module	9
3.1.2	Structure générale des stores	11
3.1.3	Applications	12
3.1.4	Gestion des autorisations	14
3.1.5	Gestion des collections de jeux de données	15
3.1.6	Gestion des jeux de données	16
3.1.7	Gestion des mappings entre un jeu de données et un dictionnaire	18
3.1.8	Gestion des convertisseurs	19
3.1.9	Gestion des filtres	20
3.1.10	Gestion des vues de jeux de données	21
3.1.11	Gestion des sources de données JDBC	22
3.1.12	Gestion des sources de données MongoDB	23
3.1.13	Gestion des espaces de stockages de fichiers	24
3.1.14	Gestion des dictionnaires	25
3.1.15	Gestion des templates de concepts	26
3.1.16	Gestion des dimensions	27
3.1.17	Gestion des flux RSS ou ATOM	28
3.1.18	Gestion des composants de formulaire	29
3.1.19	Gestion des formulaires multi-jeux de données	30
3.1.20	Gestion des formulaires de jeu de données	31
3.1.21	Gestion des graphes de projet	32
3.1.22	Gestion des inscriptions	33
3.1.23	Gestion des notifications	34
3.1.24	Gestion des OpenSearch	35
3.1.25	Gestion des commandes hors ligne	38
3.1.26	Gestion des applications plugins	39
3.1.27	Gestion des filtres de sécurité plugin	40
3.1.28	Plugins de ressources	42
3.1.29	Gestion du portail	43
3.1.30	Gestion des projets	45
3.1.31	Gestion des modules IHM de projet	47
3.1.32	Gestion des roles	48
3.1.33	Gestion SolR	49
3.1.34	Gestion des tâches	51
3.1.35	Gestion des espaces de stockage utilisateur	52
3.1.36	Administration des utilisateurs et des groupes	54
3.1.37	Recherches Multi-Datasets avec Opensearch	55
3.1.38	Interrogation des jeux de données (API records)	57
3.1.39	Validation des Plugins	64
3.2	Description détaillée des données	65

3.2.1	Common.model	65
3.2.2	Collections	66
3.2.3	Jeux de données	67
3.2.4	Mappings dictionnaire / concept	68
3.2.5	Convertisseurs en sortie	69
3.2.6	Filtres	71
3.2.7	Vues de jeu de données	72
3.2.8	Sources de données	73
3.2.9	Espaces de stockages de fichiers	74
3.2.10	Dictionnaires	75
3.2.11	Concepts de dictionnaire	76
3.2.12	Dimensions	77
3.2.13	Flux RSS ou ATOM	78
3.2.14	Composants de formulaire	79
3.2.15	Formulaires de recherche multi-jeux de données	80
3.2.16	Formulaires de recherche sur un jeu de données	81
3.2.17	Graphes de projet	82
3.2.18	Inscriptions	83
3.2.19	Notifications	84
3.2.20	Opensearchs	85
3.2.21	Commandes hors ligne	86
3.2.22	Applications plugins	87
3.2.23	Filtres de sécurité plugins	88
3.2.24	Ressources plugins	89
3.2.25	Portail	90
3.2.26	Projets	91
3.2.27	Modules IHM de projet	92
3.2.28	Rôles	93
3.2.29	Solr	94
3.2.30	Tâches	96
3.2.31	Espaces de stockage utilisateur	97
3.2.32	Utilisateurs et Groupes	98
3.3	Description des messages d'erreur	99
3.4	Recherche multi-dataset	99
3.4.1	Le modèle de données	99
3.4.2	Les services	99
3.5	Gestion des dates	100
3.5.1	Types de date pour les bases de données	100
3.5.2	Format d'échange des dates	102
4	Architecture OSGI	103
4.1	Bundles SITools	104
4.2	Bundle service admin	105
4.3	Bundle service storage	105
4.4	Etat d'implémentation actuel	106
5	Dépendances d'appel entre ressources et applications	107
5.1	Catégories d'applications	107
6	ANNEXES	108
6.1	Export HTML des classes	108
7	Documents applicables et de référence (A/R)	109
8	Glossaire et abréviations	110
8.1	Glossaire	110

8.2 Abréviations.....	110
------------------------------	------------

1 Introduction

Dans la mise en place des méthodes agiles sur le projet, il a été convenu de pratiquer une conception émergente. C'est-à-dire, à chaque sprint, l'équipe fait évoluer la conception logicielle en rapport avec les fonctionnalités à mettre en place dans le sprint.

2 Conception préliminaire

2.1 Description générale

2.1.1 Partie Serveur

La conception générale repose sur les concepts Restlet de Composant, Application, Resource et Representation.

Ainsi, chaque « Service métier » est en général représenté par une classe Application, des classes de Resource attachées à l'application, et des classes de Representation fournies par les Resource.

Le modèle métier est représenté par des objets simples POJO.

Le modèle de service est représenté par les classes de Resource exploitant la couche persistance et les objets du modèle métier.

Le modèle de persistance repose sur l'implémentation d'interfaces dédiées pour chaque entité.

Ainsi, derrière chaque « Service métier » s'occupant de la gestion des entités X, il existe une interface XStore, et une implémentation XStoreImpl. Cela laisse ainsi libre choix de changer l'implémentation de la persistance pour un service.

Le modèle de sécurité repose sur l'utilisation des classes Restlet de sécurité, Authenticator et Authorizer qui ont été étendues et combinées pour gérer la sécurité propre à chaque application.

Ainsi chaque application est protégée par un filtre de sécurité combinant un Authenticator et une combinaison de RoleAuthorizer et MethodAuthorizer paramétrables dans l'interface d'administration.

2.1.2 Partie cliente

La partie cliente est développée en Ext.js et découpée en 3 modules :

- **le client user**

Il s'agit du module d'accès utilisateur, il contient l'ensemble des ressources (js, css, images, html) de l'interface publique de SITools2.

- **le client admin**

Il s'agit du module d'administration de SITools2, il contient l'ensemble des ressources (js, css, images, html) de l'interface d'administration.

- **le client public**

Cette partie contient des ressources communes (js, css, images) aux deux autres modules

2.2 Architecture de système

Cf. DAR-SITOOLS2-V2-1.0

2.3 Description de chaque module

Cf. DAR-ULISSE-V2-1.0

3 Conception détaillée

La conception détaillée présente dans un premier temps une description de chaque module de SITools2. Pour chacun de ces modules on présentera un diagramme de classe et quelques explications sur ce diagramme.

La deuxième partie de cette conception contient le détail des modèles de l'application.

La conception ayant été réalisée de façon émergente et le document rédigé au fur et à mesure des développements, les diagrammes ne sont pas forcément à jour, principalement dans les détails de chacune des classes. La hiérarchie entre les classes et leurs relations sont, elles, bien à jour.

Une version plus à jour des classes et des diagrammes de classes de chacun des packages est disponible via un export HTML accessible avec un navigateur web (de préférence Internet Explorer, ne marche pas sous Google Chrome). Pour voir cette interface, il suffit d'ouvrir le fichier `index.html` dans le dossier `export_html_classes`.

3.1 Description détaillée de chaque module

3.1.1 Structure générale d'un module

class Structure générale module SITools2

La gestion des sources de données est assez représentative de la conception de chacun des modules d'administration de Sitools.

Chaque module suit cette conception, à savoir :

Une implémentation de Store, **DataSourceStoreXML** : Implémentation de la persistance des données dans un format XML

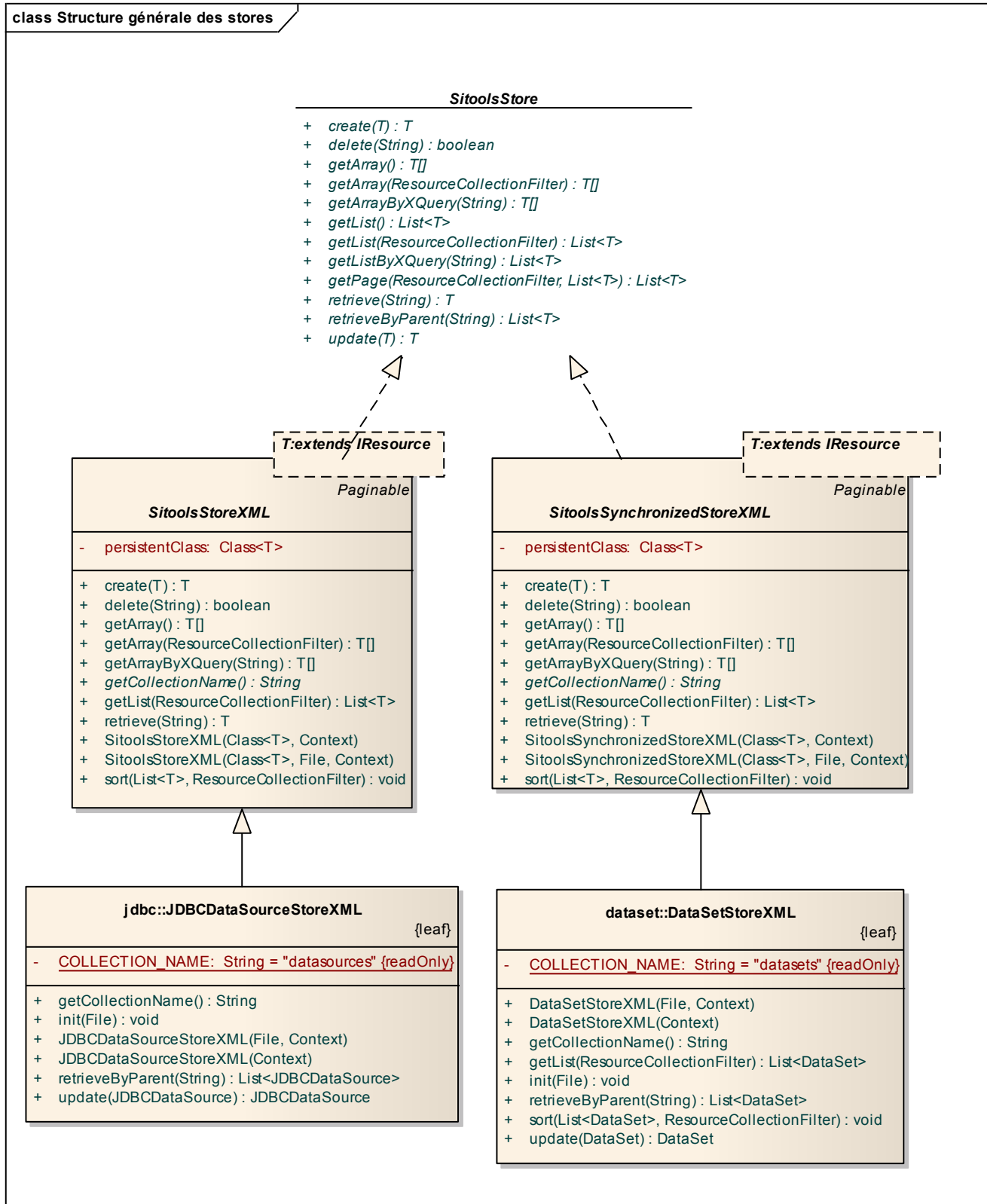
Une application d'administration, **DataSourceAdministration** : Définit les ressources disponibles sur ce module

Une classe abstraite de Resource, **AbstractDataSourceResource** : Définit les méthodes communes aux différentes ressources.

Une ressource « collection », **DataSourceCollectionResource** : Permet la manipulation de l'ensemble des DataSources, permet d'ajouter une DataSource et liste l'ensemble des DataSources.

Une ressource, **DataSourceResource** : Permet la manipulation d'une DataSource en particulier, de la récupérer, de la modifier et de la supprimer.

3.1.2 Structure générale des stores



L'interface **SitoolsStore<T>** est une interface générique qui représente un store dans SITools2. Il en existe 2 implémentations pour stocker les modèles en format XML, une synchronisée pour éviter les problèmes d'écritures concurrentes **SitoolsSynchronizedStoreXML** et une non synchronisée, plus légère, **SitoolsStoreXML**.

Dans chacun des modules, on surcharge l'une ou l'autre de ces classes pour gérer les spécificités des modèles à traiter.

3.1.3 Applications

class applicationsWeb

Sitools Application : Classe abstraite racine de toutes les applications de Sitools

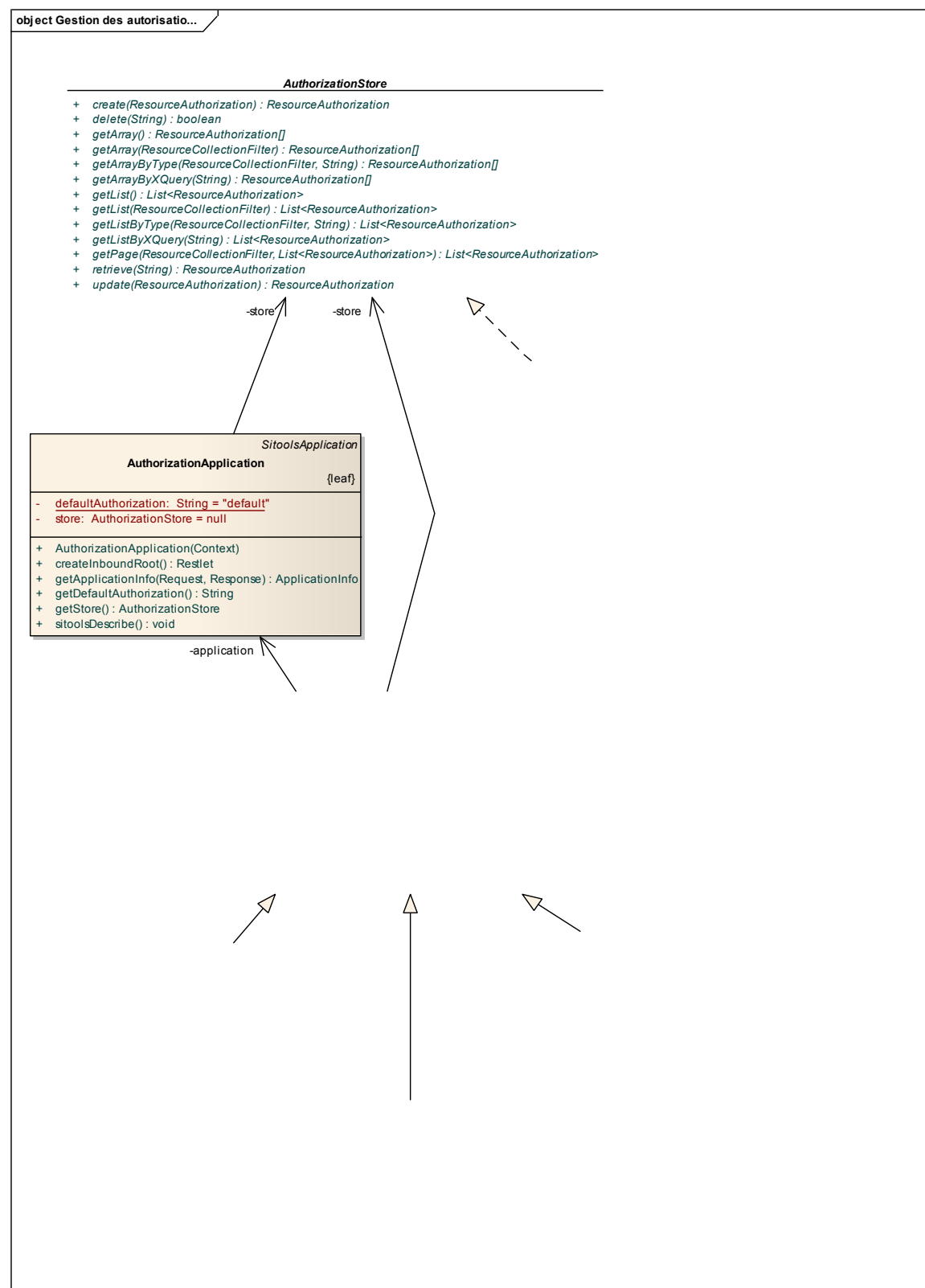
StaticWebApplication : Classe abstraite dont le but est de publier un répertoire sur internet à la manière d'un serveur Apache

ClientAdminApplication : Classe qui publie le répertoire de l'interface client-admin

ClientUserApplication : Classe qui publie le répertoire de l'interface client-user

PublicApplication : Classe qui publie le répertoire de l'interface client-public

3.1.4 Gestion des autorisations



3.1.5 Gestion des collections de jeux de données

object Gestion des collection de jeux de donné...

La classe **CollectionStoreXML** hérite de la classe **SitoolsStoreXML** qui est elle-même une implémentation de l'interface **SitoolsStore**.

La classe **PropertyListResource** permet de retourner l'ensemble des propriétés de jeux de données de la collection.

La classe **CommonConceptsResource**, retourne l'ensemble des concepts communs de la collection pour un dictionnaire donné.

3.1.6 Gestion des jeux de données

3.1.6.1 Administration

class Administration des jeux de donn...

3.1.6.2 Consultation

class Consultation des jeux de donné...

3.1.7 Gestion des mappings entre un jeu de données et un dictionnaire

class DatasetDictionaryMappingAPI

Cette API enrichie l'API d'administration des jeux de données. On attache 3 nouvelles ressources à l'application « **DatasetAdministration** » :

- **DataSetDictionaryMappingCollectionResource** : Permet de récupérer la liste des mappings de dictionnaire pour un dataset donnée
- **ConceptTemplateResource** : Gère un mapping en particulier (suppression, modification, lecture)
- **DataSetNotificationResource** : Gère les notifications sur les Datasets, dans notre cas, cela permet de mettre de supprimer un mapping si le dictionnaire est supprimé ou de supprimer un concept s'il est supprimé.

3.1.8 Gestion des convertisseurs

class Gestion des convertisseu...

3.1.9 Gestion des filtres

class Gestion des filtres

La classe « **FilterChainResource** » gère une liste ordonnée de « **FilterModel** ». Elle permet de consulter l'ensemble des filtres, l'ajout d'un filtre, la modification de l'ordre des filtres ainsi que la suppression de l'ensemble des filtres pour un jeu de données donné.

La classe « **FilterResource** » gère un « **FilterModel** » en particulier. Elle permet sa consultation, sa modification et sa suppression.

3.1.10 Gestion des vues de jeux de données

object Gestion des vues de jeux de donn...

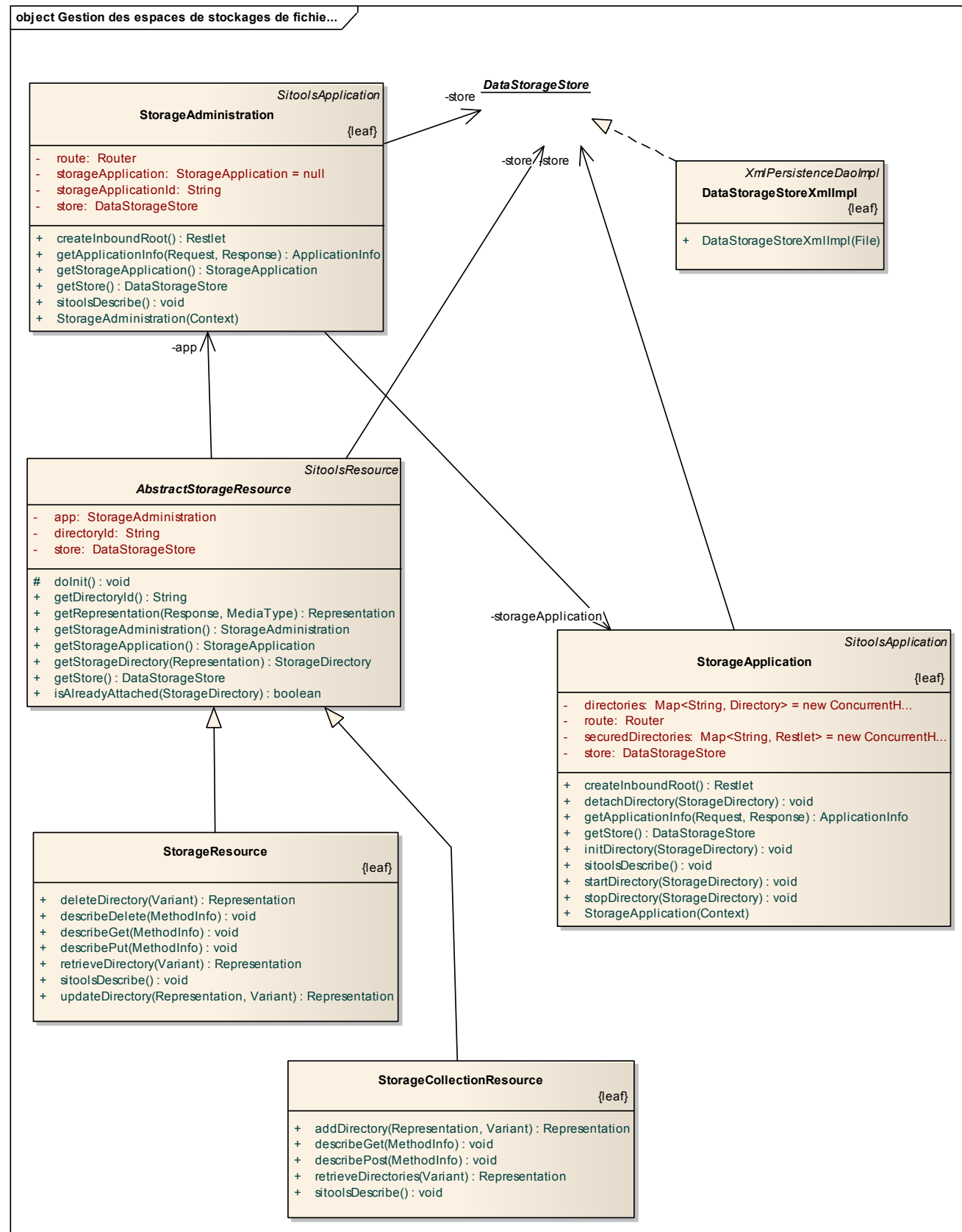
3.1.11 Gestion des sources de données JDBC

class jd...

3.1.12 Gestion des sources de données MongoDB

class mongodb

3.1.13 Gestion des espaces de stockages de fichiers



3.1.14 Gestion des dictionnaires

class dictionary

Cette API contient une Application « **DictionaryAdministration** » qui donne accès à 3 ressources :

- **DictionaryCollectionResource** : Gère les collections de **Dictionary** ainsi que l'ajout d'un nouveau Dictionary
- **ConceptTemplateResource** : Gère un **Dictionary** en particulier (suppression, modification, lecture)
- **PropertyResource** : Permet de récupérer les détails d'un concept donné dans un dictionnaire donné.

3.1.15 Gestion des templates de concepts

class TemplateConceptAPI

Cette API contient une Application « **ConceptTemplateAdministration** » qui donne accès à 3 ressources :

- **ConceptTemplateCollectionResource** : Gère les collections de **ConceptTemplate** ainsi que l'ajout d'un nouveau **ConceptTemplate**
- **ConceptTemplateResource** : Gère un **ConceptTemplate** en particulier (suppression, modification, lecture)
- **PropertyResource** : Permet de récupérer les détails d'une propriété donnée dans un **ConceptTemplate** donné.

3.1.16 Gestion des dimensions

object Gestion des dimensio...

3.1.17 Gestion des flux RSS ou ATOM

object Gestion des flux RSS ou ATOM

La classe **FeedsClientResource** permet l'accès au flux dans un format RSS ou ATOM.

Les autres classes de ressource, **FeedsAdminResource** et **FeedsCollectionResource** permettent l'administration des flux.

3.1.18 Gestion des composants de formulaire

class Gestion des composants de formulaire

3.1.19 Gestion des formulaires multi-jeux de données

class Gestion des formulaires multi-jeux de donn...

3.1.20 Gestion des formulaires de jeu de données

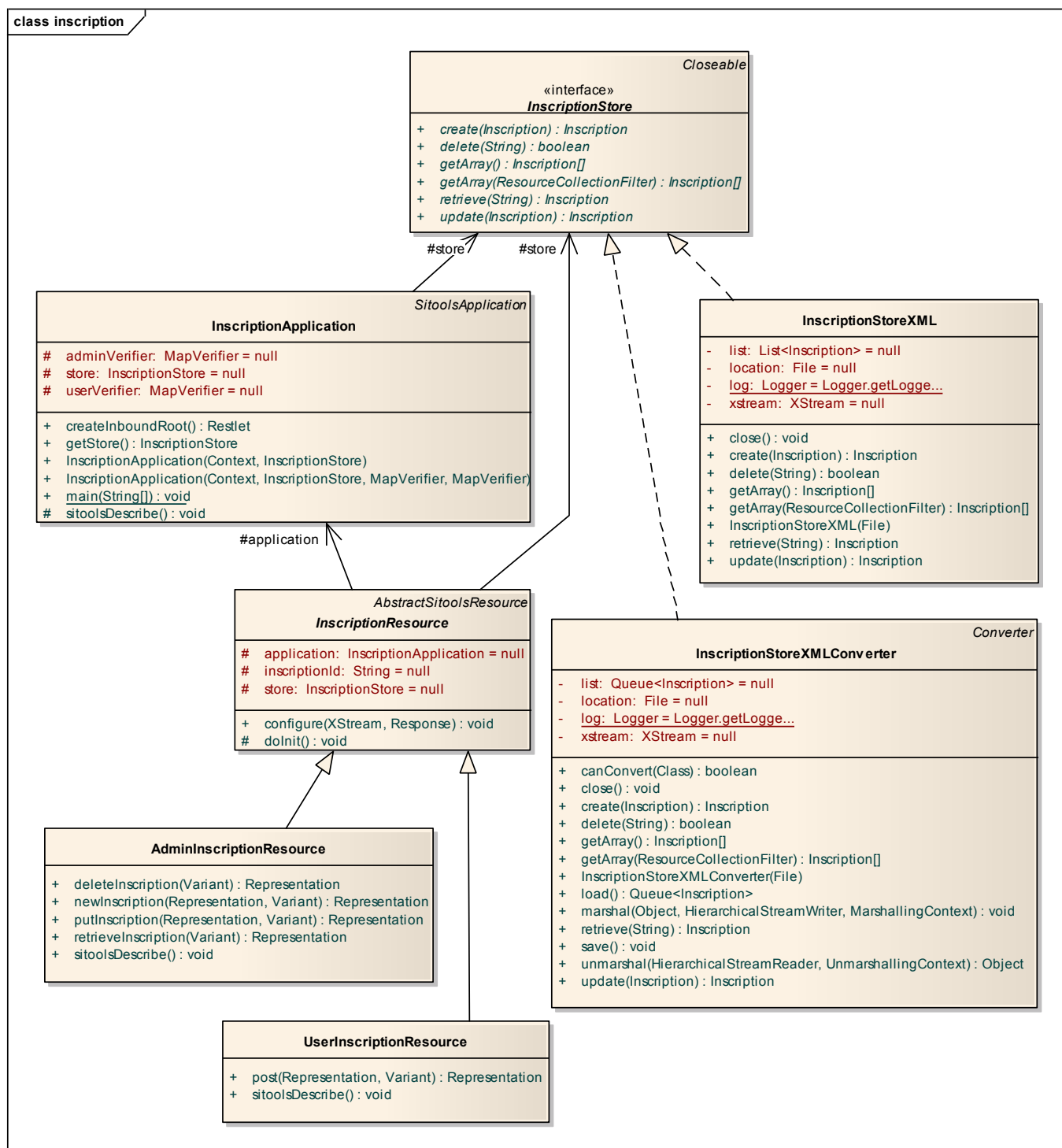
class Gestion des formulaires de jeu de donn...

3.1.21 Gestion des graphes de projet

object Gestion des graphes de pro...

Ces ressources sont attachées directement à l'application **ProjectAdministration**, il n'y a donc pas d'application spécifique pour les graphes de projet.

3.1.22 Gestion des inscriptions



3.1.23 Gestion des notifications

class notification

3.1.24 Gestion des OpenSearch

class opensearch

3.1.24.1 Gestion asynchrone de l'indexation



L'indexation peut prendre beaucoup de temps, il est donc nécessaire de réaliser ce traitement de manière asynchrone. La classe **OpensearchRefreshRunnable** permet donc de gérer cette indexation de cette manière, au sein d'un Thread différent des ressources.

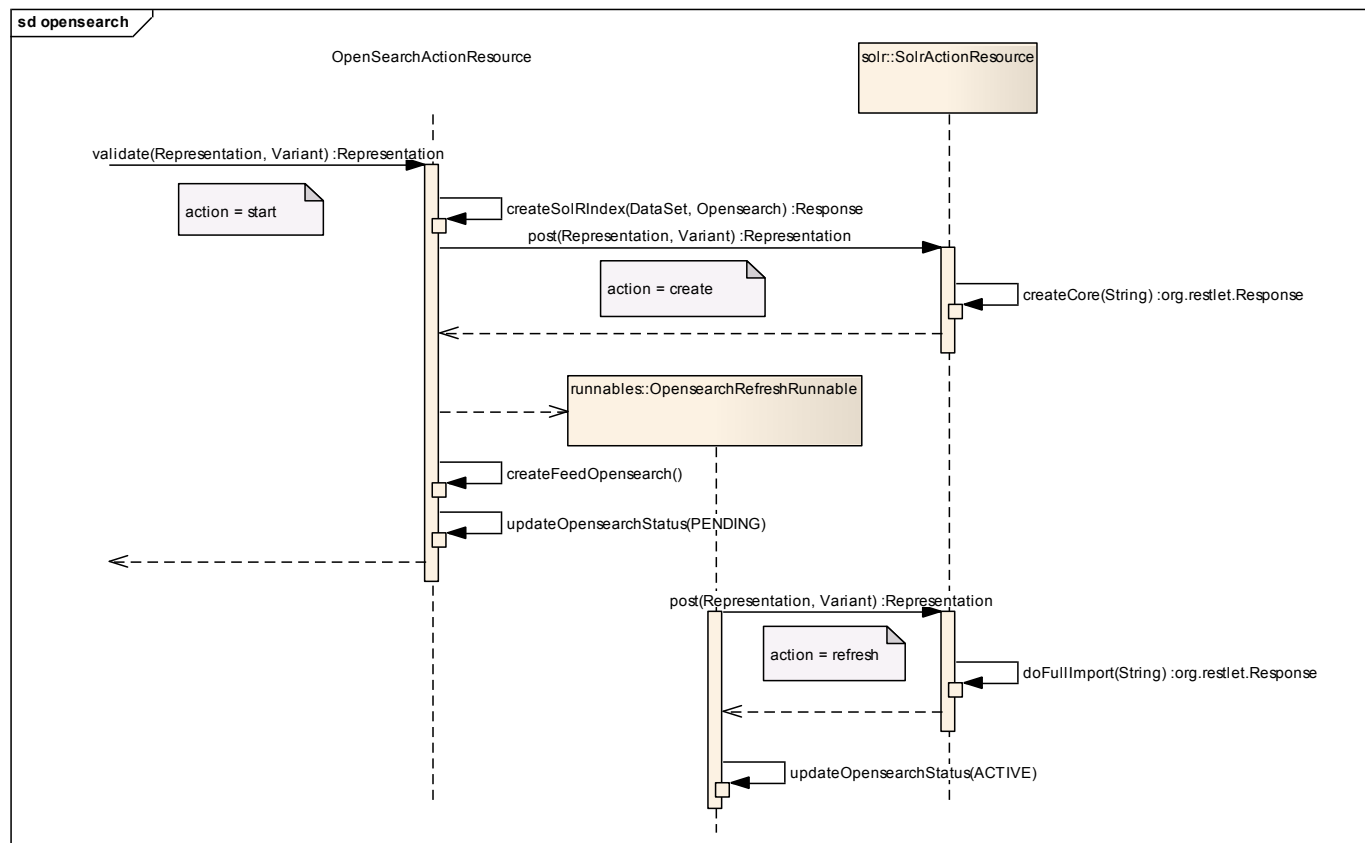
3.1.24.2 Détails de OpensearchActionResource

OpensearchActionResource est une classe qui permet d'agir sur un index Opensearch. Les actions disponibles sont :

- start : Créer un index et index les données
- stop : Stop un index et supprime les données indexées
- refresh : réindexe les données
- cancel : annule l'opération en cours

Les actions de « start » et de « refresh » utilisent la classe **OpensearchRefreshRunnable**. Cela permet de répondre rapidement au client, en lui spécifiant que l'indexation est en cours.

L'action « create » se décompose donc comme ceci :



L'action de « refresh » est sensiblement la même sans la première partie de création de l'index. Il y a uniquement la partie avec création de l'OpensearchRefreshRunnable.

L'action de « stop » est un appel à SolrActionResource, avec l'action « delete »

L'action de « cancel » est un appel à SolrActionResource, avec l'action « cancel »

3.1.25 Gestion des commandes hors ligne

class Gestion des commandes hors ligne

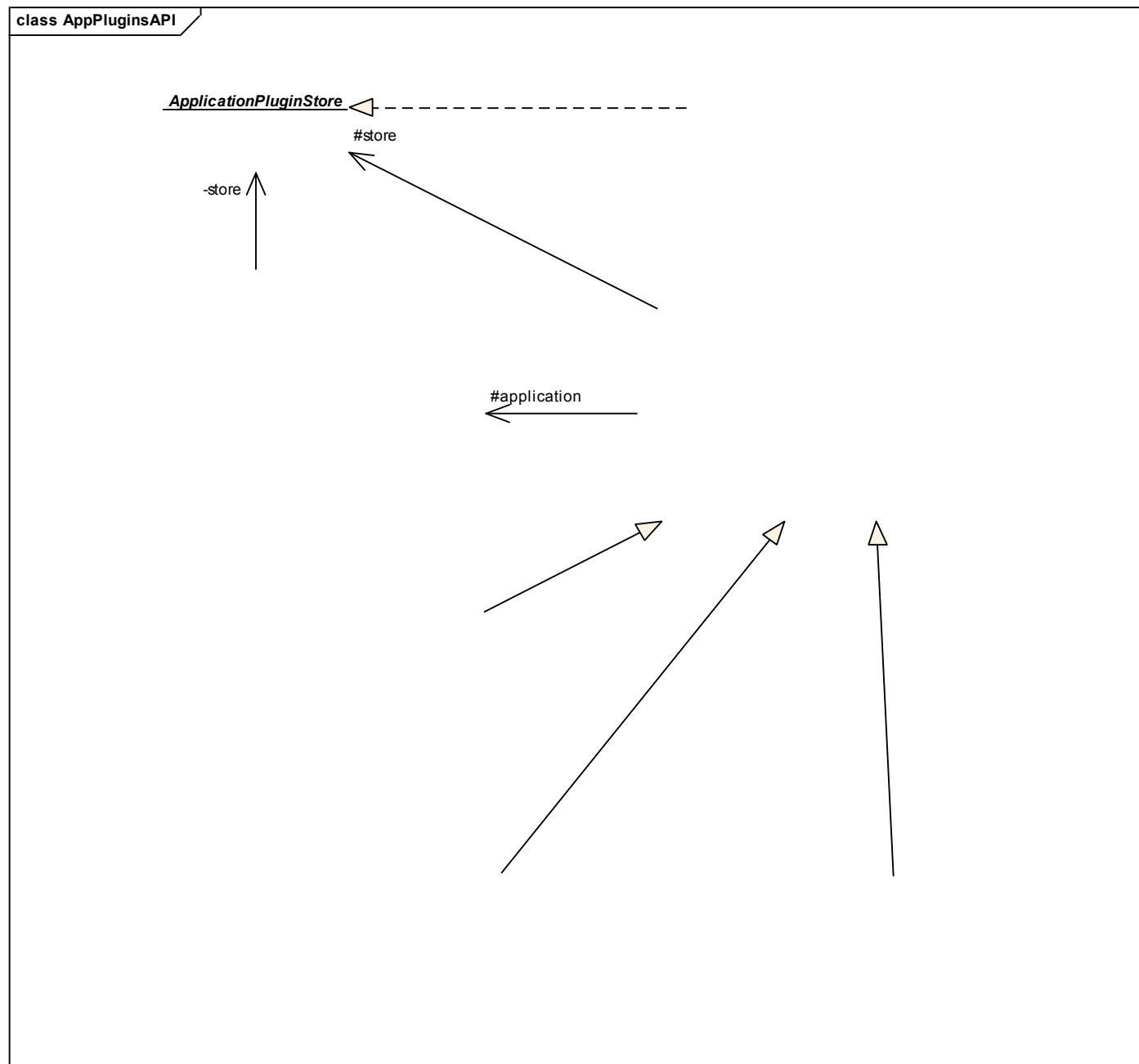
Pour la gestion des commandes hors ligne, il y a 2 applications, une pour l'administrateur et une pour l'utilisateur. L'administrateur a plus de possibilité de modification que l'utilisateur. De plus en ayant 2 applications différentes, il est possible de définir plus finement la sécurité de la gestion de ces commandes.

3.1.26 Gestion des applications plugins

3.1.26.1 Récupération de la liste des applications plugins disponibles

class AppPluginsListingAPI

3.1.26.2 Administration des applications plugins



3.1.27 Gestion des filtres de sécurité plugin

Il s'agit de filtres que l'on peut ajouter aux espaces de stockage de fichier afin de personnaliser la gestion de la sécurité.

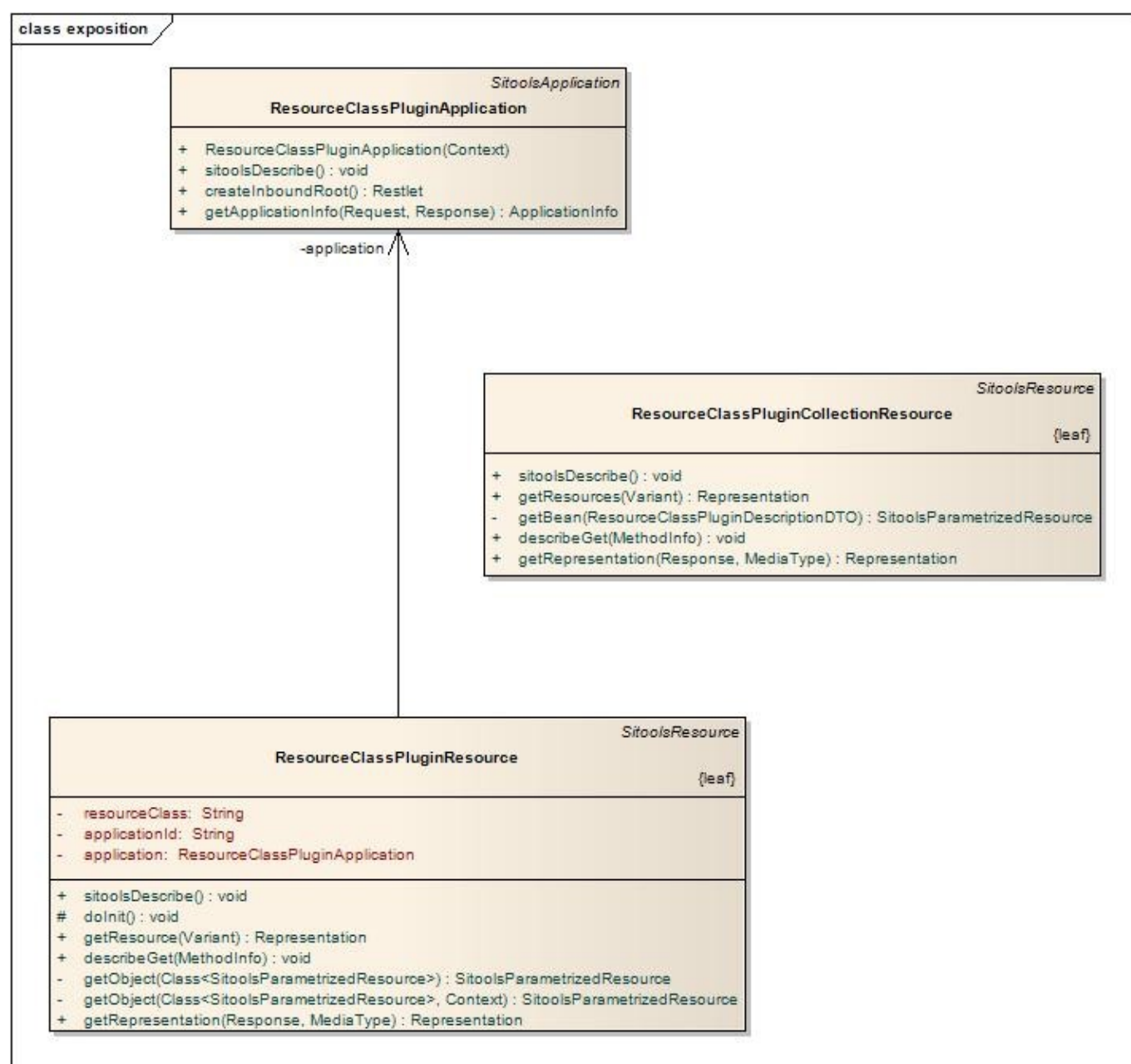
class Gestion des filtres de sécurité

3.1.28 Plugins de ressources

NB : à l'heure actuelle, seules les applications projets, dataset, les administrations de projets et datasets ainsi que les applications plugin sont conçues pour accueillir des plugins de ressource.

3.1.28.1 Exposition des ressources disponibles

Les ressources disponibles sont développées dans le module fr.cnes.sitools.extensions à partir de d'une classe abstraite de ressource (**SitoolsParametrizedResource**) qui doit faire référence à une extension de la classe abstraite de modèle **ParametrizedResourceModel** (cf. classe d'exemple **BasicParametrizedResource**).

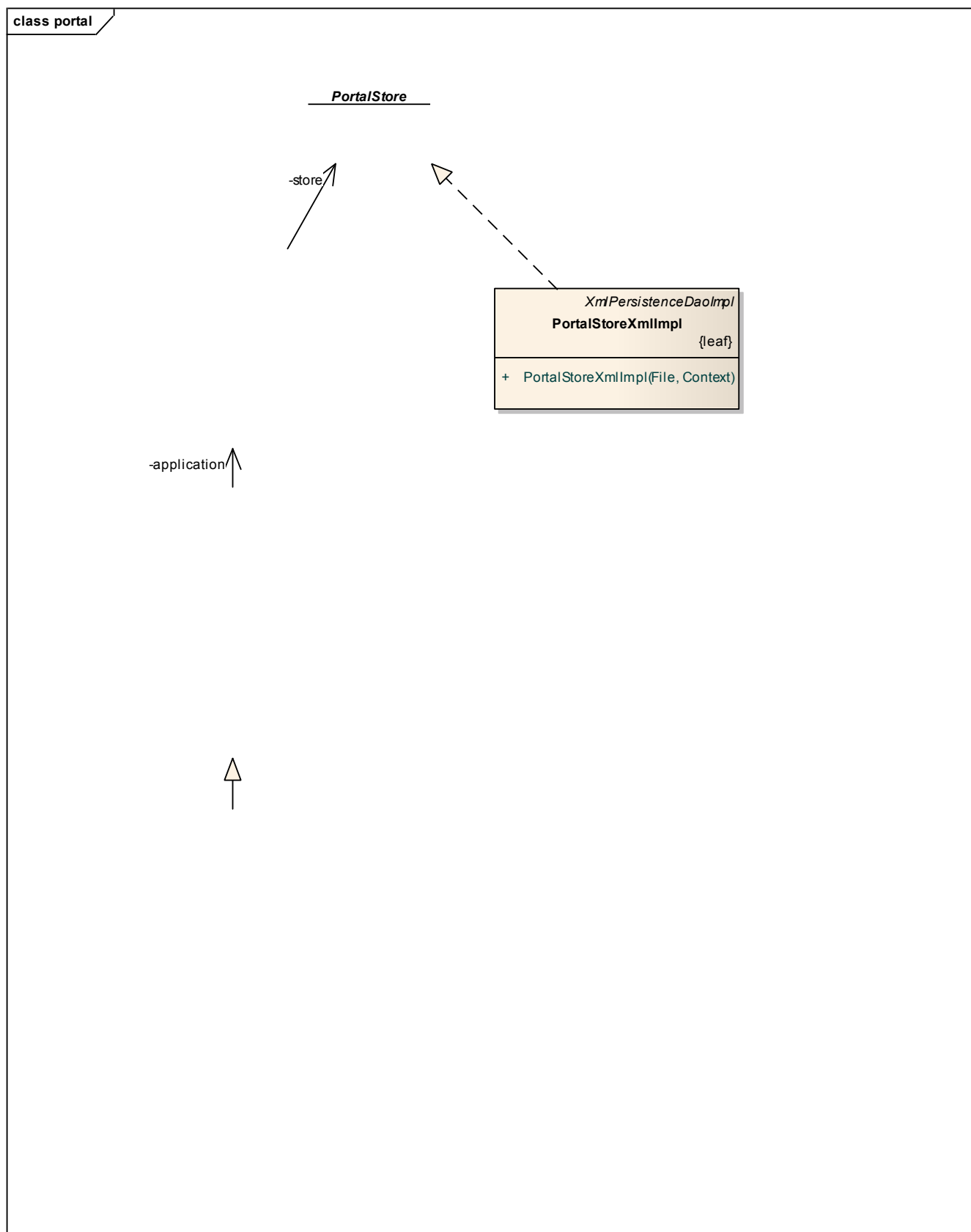


3.1.28.2 **Ajout/Lecture/Modification/Suppression d'un plugin de ressource**

class Gestion des ressources plugins

3.1.29 Gestion du portail

Il s'agit de la gestion du portail utilisateur. Un seul portail est ajouté par défaut et un seul sera utilisé par la suite. Ce module permet cependant d'avoir la liste des projets, des flux et les détails d'un projet donné au niveau portail.

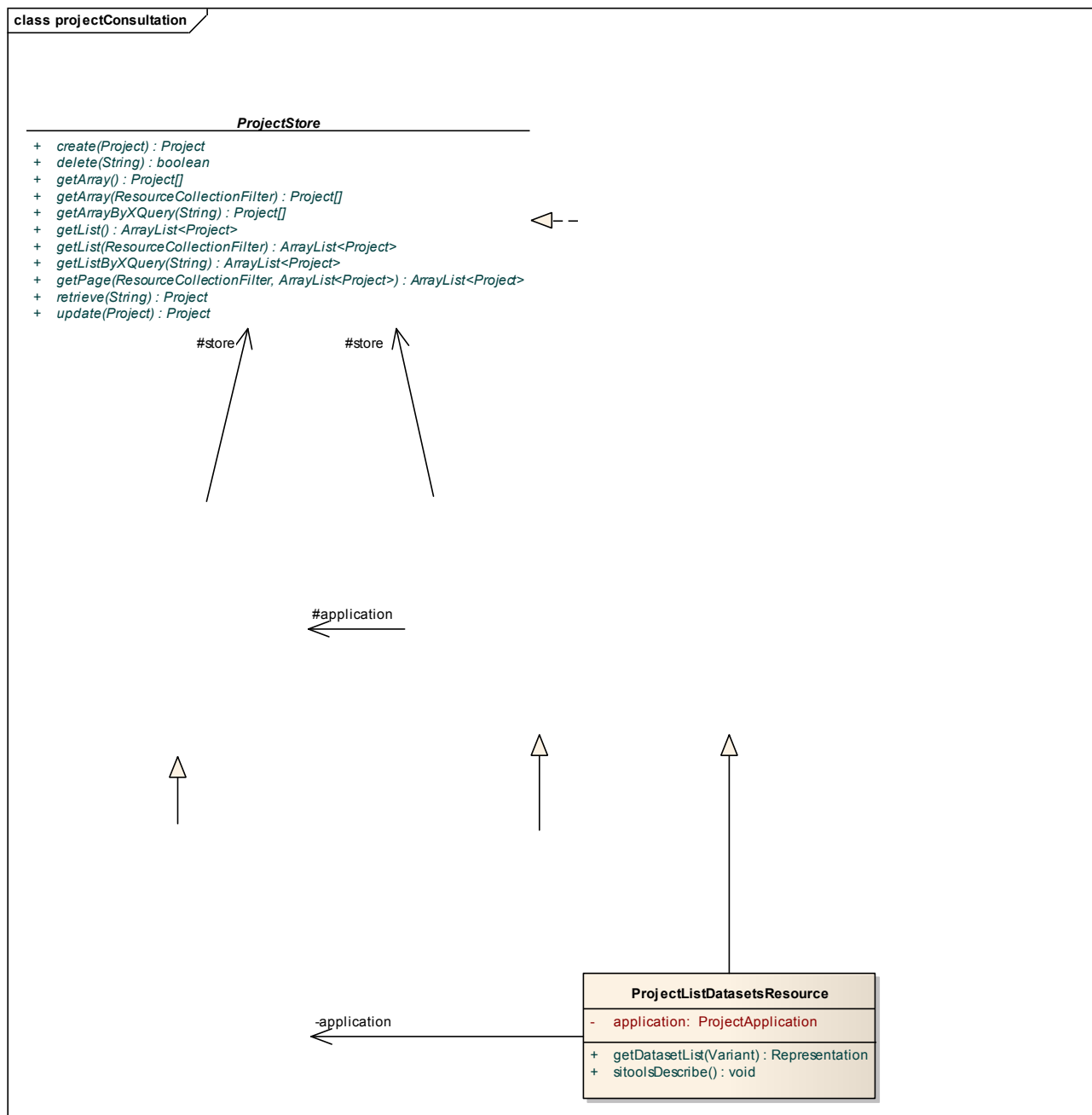


3.1.30 Gestion des projets

3.1.30.1 Administration

object Administration des proj...

3.1.30.2 Consultation



3.1.31 Gestion des modules IHM de projet

object Administration des proj...

3.1.32 Gestion des roles

class role

3.1.33 Gestion Solr

class Class Solr

La classe **SolrActionResource** est en charge des actions sur les indexes Solr (création, suppression, rafraichissement). La classe **SolrResource** est en charge de l'interrogation des indexes solr et la classe **SolrForward** offre une interface via le protocole http à Solr.

La classe **SolrResource** permet également d'effectuer des requêtes « d'autocompétion ». Ceci retourne du JSON formaté pour les navigateurs web (exemple : le widget de moteur de recherche de Mozilla Firefox) ou pour le client Sitools.

3.1.33.1 Détails de SolrActionResource

La classe SolrActionResource est en charge des action sur les indexes Solr, plus précisément sur les cœurs Solr.

Les actions disponibles sont :

- create : créer un cœur Solr

Requête SolR :

Requête GET :

```
"solr://default/admin/cores?action=CREATE&name="+indexName+"&instanceDir="+instanceDir
```

« indexName » est le nom du cœur SolR et « instanceDir » est le dossier du cœur SolR.

- update : modifie un cœur SolR. non implémenté, il faut recréer le coeur SolR. Effectue le même traitement que « create »

« indexName » est le nom du cœur SolR

- refresh : Effectue l'indexation des données

Requête SolR :

Requête GET :

```
"solr://" + indexName + "/dataimport?command=full-import"
```

- delete : supprime un cœur SolR. Concretement, cette action supprime l'ensemble des données indexés et réalise un « UNLOAD » sur le cœur. Les fichiers de l'index ne sont donc pas supprimés.

Requêtes Solr :

Suppression des données :

Requête Post :

```
"solr://" + indexName + "/update?commit=true&optimize=true&waitFlush=true&waitSearcher=true"
```

de :

```
"<delete><query>*:*/</query></delete>"
```

Déchargement (UNLOAD) du cœur :

Requête GET :

```
"solr://default/admin/cores?action=UNLOAD&core="+indexName
```

- cancel : annule l'action en cours

Requête Solr :

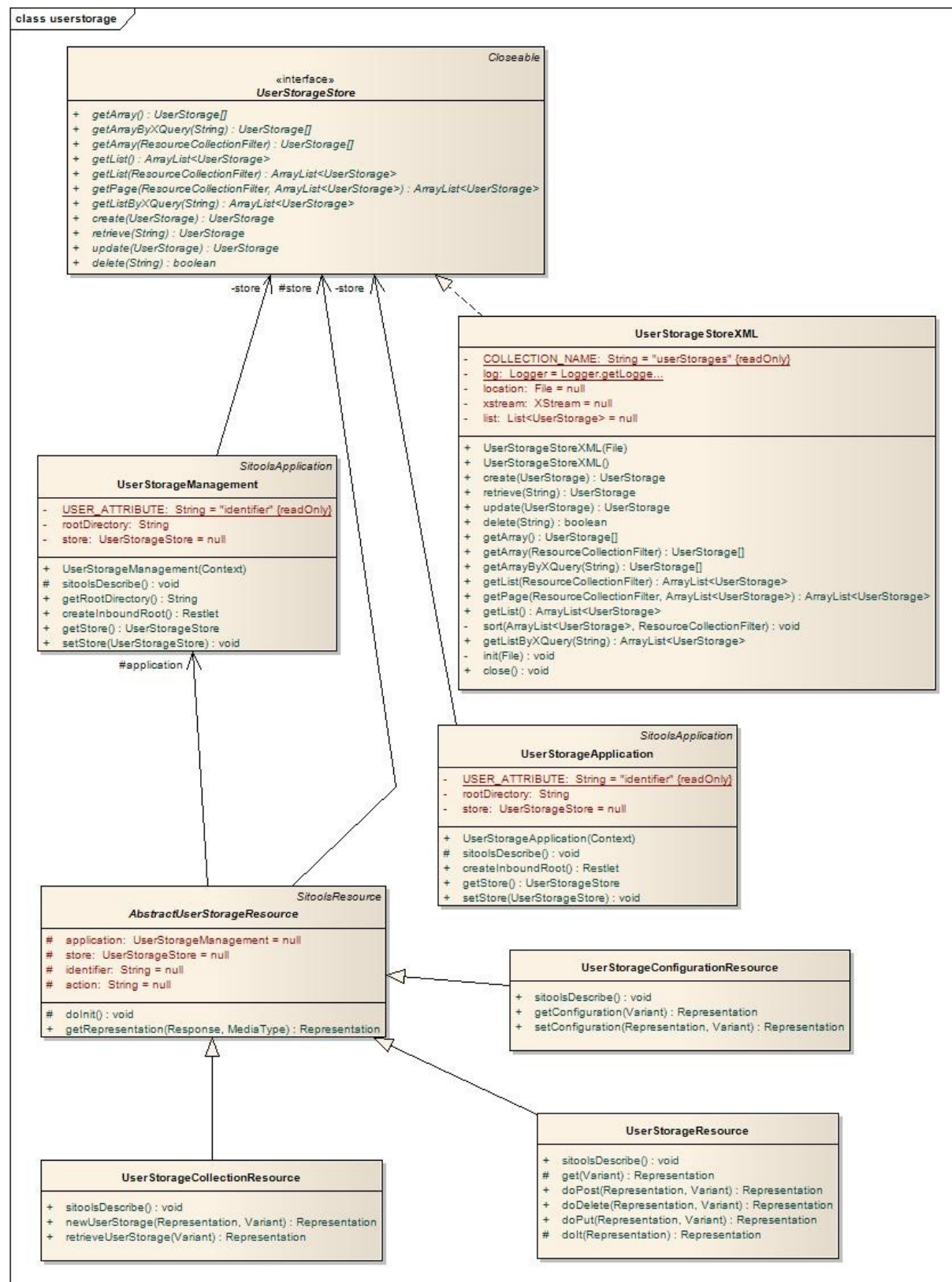
Requête GET :

```
solr://" + indexName + "/dataimport?command=abort
```

3.1.34 Gestion des tâches

class Gestion des taches

3.1.35 Gestion des espaces de stockage utilisateur

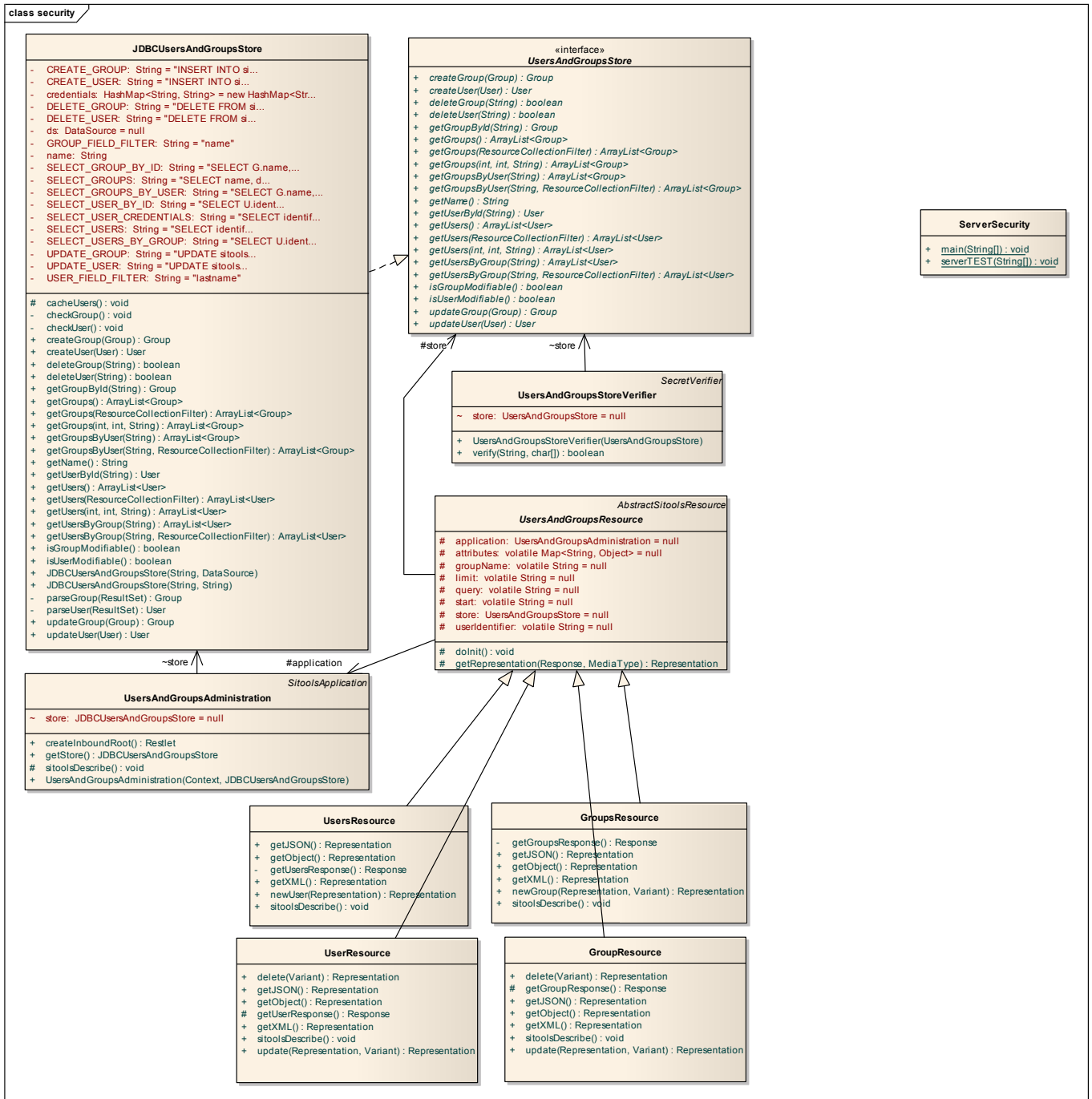


La gestion des espaces de stockage utilisateurs consiste en 2 applications :

- **UserStorageManagement** avec les ressources permettant de définir un espace de stockage utilisateur et de l'activer / désactiver.
- **UserStorageApplication** avec les ressources permettant aux utilisateurs d'accéder réellement au répertoires de fichiers, incluant les méthodes GET, POST, PUT, DELETE de fichiers selon les droits configurés pour chacun sur son répertoire.

La politique de sécurité d'accès aux répertoires utilisateurs est gérée par une classe d'autorization particulière pour ce cas de figure. La **SitoolsUserAuthorization** se base sur l'identifiant de l'utilisateur en tant que paramètre de la requête pour ensuite vérifier si l'utilisateur authentifié / non possède les droits d'accès à ce répertoire.

3.1.36 Administration des utilisateurs et des groupes



3.1.37 Recherches Multi-Datasets avec Opensearch

Pour rechercher sur plusieurs jeux de données en même temps, on utilise la recherche Opensearch définie sur chacun des jeux de données. Cette recherche est effectuée sur le champ par défaut de chacun des Opensearch. Cette ressource permet également de retourner l'ensemble des Opensearch disponibles pour un utilisateur donné.

class opensearch

La ressource **MultiDsOsResource** récupère la liste des Opensearch accessibles. Elle instancie ensuite l'une ou l'autre des deux représentations en fonction de la demande :

- **MultiDsOsSearchRepresentation** pour une recherche
- **MultiDsOsSuggestRepresentation** pour une suggestion (utilisé pour l'auto complétion)

Les représentations interrogent donc chacun des Opensearch avec les mêmes paramètres et réalise une agrégation des résultats.

Pour une recherche, on recrée un flux RSS à partir des « items » des différents résultats. Il n'y a pas de traitement et les « items » sont ajoutés les uns à la suite des autres dans l'ordre d'arrivée. Cette agrégation est effectuée en streaming pour plus de fluidité.

Pour une suggestion, on recrée un flux JSON en y mettant l'ensemble des suggestions des différents résultats. Comme pour la recherche ils sont ajoutés les uns à la suite des autres ce qui peut occasionner de la redondance. Le flux est également créé en streaming pour plus de fluidité.

3.1.38 Interrogation des jeux de données (API records)

3.1.38.1 Partie Ressources et Représentations

class Interrogation des jeux de donn...

DataSetExplorerResource expose l'API permettant de requêter les données d'un jeu de données. On lui associe deux Représentations qui réalisent le formatage d'un enregistrement (**DBRecordRepresentation**) ou de plusieurs enregistrements (**DBRecordSetRepresentation**) en JSON ou en XML.

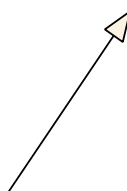
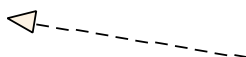
La classe **DataSetExplorerUtil** contient l'ensemble des méthodes pour parser la requête cliente et en faire une requête pour la base de données.

3.1.38.2 Requêtes sur le jeu de données avec SQL

class Requêtes sur le jeu de données avec S...

database::DatabaseRequest

```
+ buildURI() : String
+ calculateTotalCountFromBase() : int
+ checkRequest() : void
+ close() : void
+ createDistinctRequest() : void
+ createRequest() : void
+ getCount() : int
+ getDistinctRequestAsString() : String
+ getMaxResultsToSend() : int
+ getPrimaryKeys() : List<String>
+ getRecord() : Record
+ getRequestAsString() : String
+ getSelectedPrimaryKey() : List<String>
+ getStartIndex() : int
+ getTotalCount() : int
+ isCountDone() : boolean
+ isLastResult() : boolean
+ nextResult() : boolean
```



Ces classes contiennent tous les paramètres d'une requête sur un jeu de données. Elles permettent également de requêter ce même jeu de données. Afin d'avoir un mécanisme extensible, l'interface « **DatabaseRequest** » définit l'API pour requêter un jeu de données.

SQLDatabaseRequest est l'implémentation par défaut, fonctionnant avec des bases de données.

SQLListIdDatabaseRequest est une implémentation différente, effectuant une requête pour chacun des enregistrements.

SQLRangeDatabaseRequest est une implémentation qui permet de faire des requêtes sur des plages d'enregistrements. Ces plages sont spécifiées par des intervalles d'index d'enregistrements dans la base de données. Une requête est effectuée pour chacune des plages.

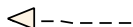
Les classes **WildcardSQL** et **OperatorSQL** permettent de transformer certaines constantes en chaîne de caractères correspondantes à une source de données SQL.

3.1.38.1 Requêtes sur le jeu de données avec MongoDB

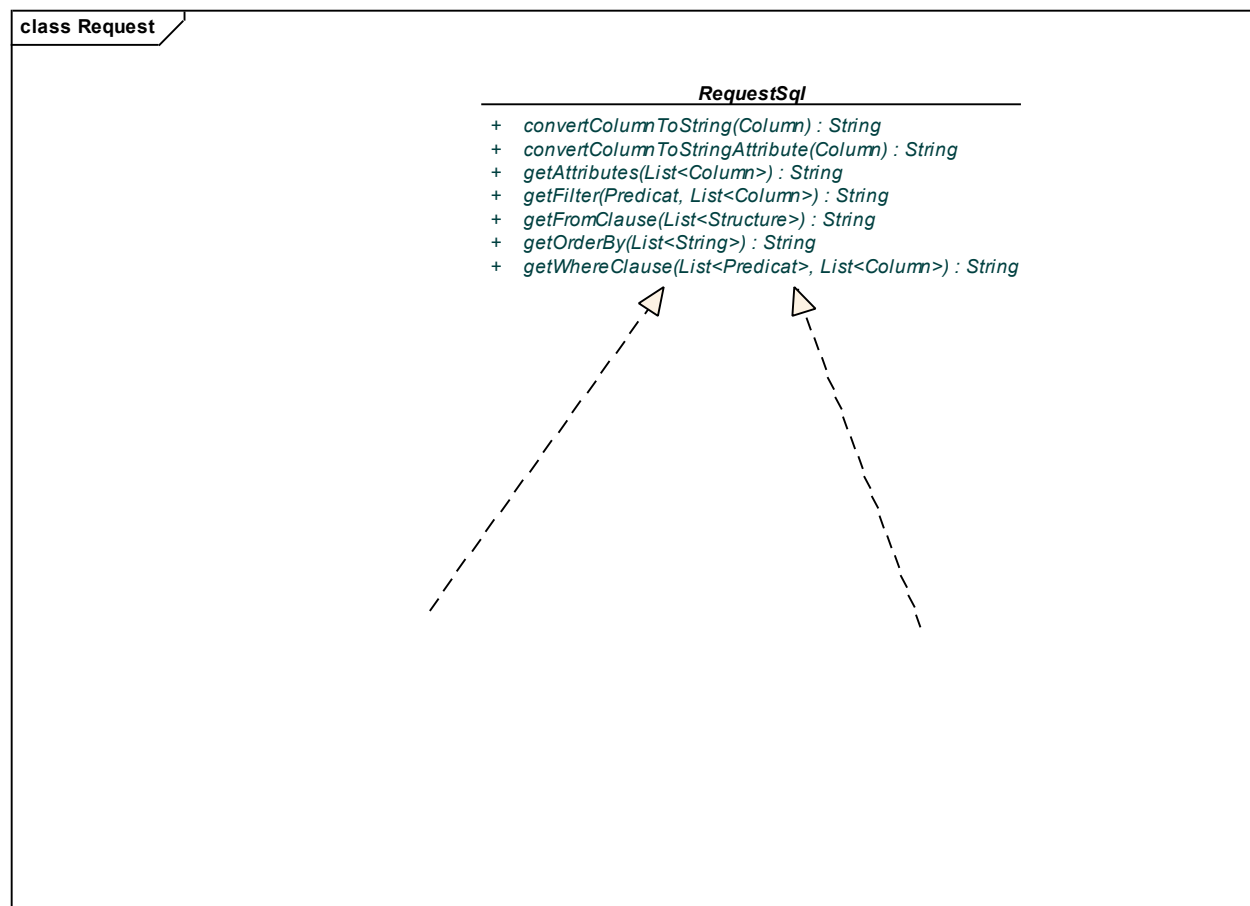
class Requêtes sur le jeu de données avec Mongo...

database::DatabaseRequest

```
+ buildURI() : String
+ calculateTotalCountFromBase() : int
+ checkRequest() : void
+ close() : void
+ createDistinctRequest() : void
+ createRequest() : void
+ getCount() : int
+ getDistinctRequestAsString() : String
+ getMaxResultsToSend() : int
+ getPrimaryKeys() : List<String>
+ getRecord() : Record
+ getRequestAsString() : String
+ getSelectedPrimaryKey() : List<String>
+ getStartIndex() : int
+ getTotalCount() : int
+ isCountDone() : boolean
+ isLastResult() : boolean
+ nextResult() : boolean
```



3.1.38.2 Requête sur la source de données SQL



Ces classes produisent les requêtes aux sources de données (base de données SQL dans notre cas). Une factory (**RequestFactory**) est disponible pour récupérer le bon type de requête en fonction du type de base de données. Afin de rendre le mécanisme extensible, l'interface **RequestSQL** définit l'API d'interrogation d'une source de données et il existe 2 implémentations, une pour MySQL et une pour **PostgreSQL**.

3.1.38.3 Limitations des types utilisables pour une base SQL

Les types de données utilisables pour la clé primaire d'un jeu de données sont :

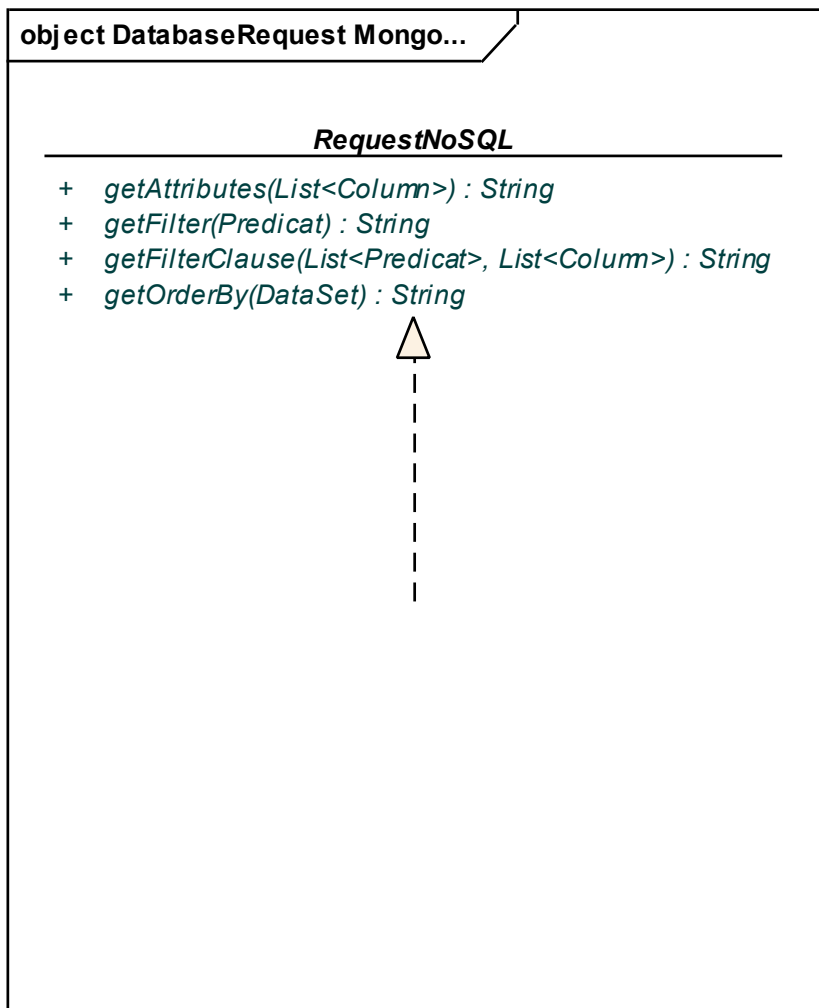
Pour MySQL :

- Varchar
- Tinyint
- Mediumint
- Int
- Bigint
- Decimal
- Timestamp
- Datetime
- Date
- Year
- Time
- Char

Pour PostgreSQL

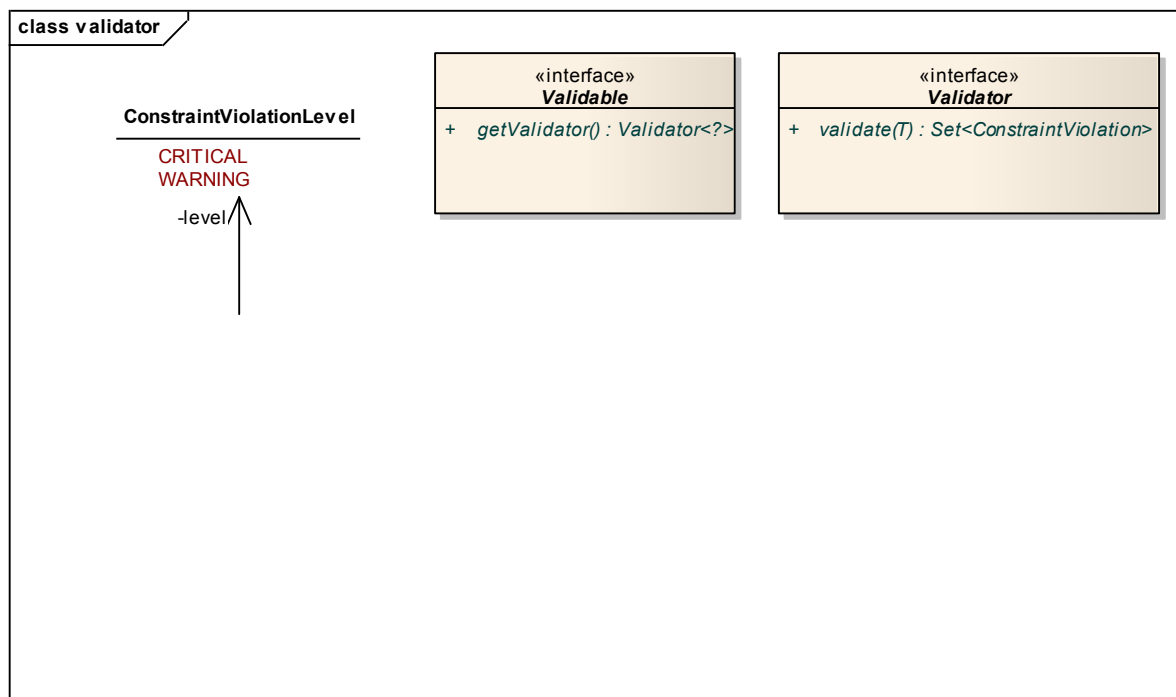
- int
- smallint
- bigint
- float
- double
- numeric
- varchar
- char
- timestamp
- date
- time (limité à une précision à la milliseconde)
- timestamp_with_time_zone (limité à une précision à la milliseconde)
- time_without_time_zone (limité à une précision à la milliseconde)
- serial
- bigserial

3.1.38.4 Requête sur la source de données MongoDB



3.1.39 Validation des Plugins

Par plugin on entend tous les composants extensibles de SITools2, c'est-à-dire, à l'heure actuelle, les convertisseurs, les filtres, les applications plugins et les ressources plugins.



Il s'agit d'un mécanisme permettant la validation des plugins au sein de Sitools. Ce mécanisme est assez simple et permet une description détaillée des problèmes rencontrés lors de la validation.

Ce mécanisme comprend

- Une interface **Validable**

Cette interface permet de dire qu'un objet peut être validé. Elle contient la définition d'une méthode GetValidator() qui retourne un objet de validation.

- Une interface Validator :

Elle comprend une méthode validate<T> qui doit être implémentée pour effectuer la validation d'un objet de type <T> (<T> doit implémenter « Validable »). Cette méthode doit retourner un ensemble d'anomalies détectées sur l'objet

- Une classe ConstraintViolation :

Elle définit ce qu'est une anomalie. Elle permet une définition assez précise car elle contient un message, un niveau de criticité ainsi que la valeur qui a été saisie

- Une énumération ConstraintViolationLevel :

Elle définit 2 niveaux de criticités :

- CRITICAL

Il s'agit d'une anomalie critique entraînant l'erreur de la création du plugin

- WARNING :

Il s'agit d'une anomalie de type avertissement. Il existe un paramètre (suppressWarning) qui permet de spécifier si ce type d'anomalie entraîne une erreur ou non.

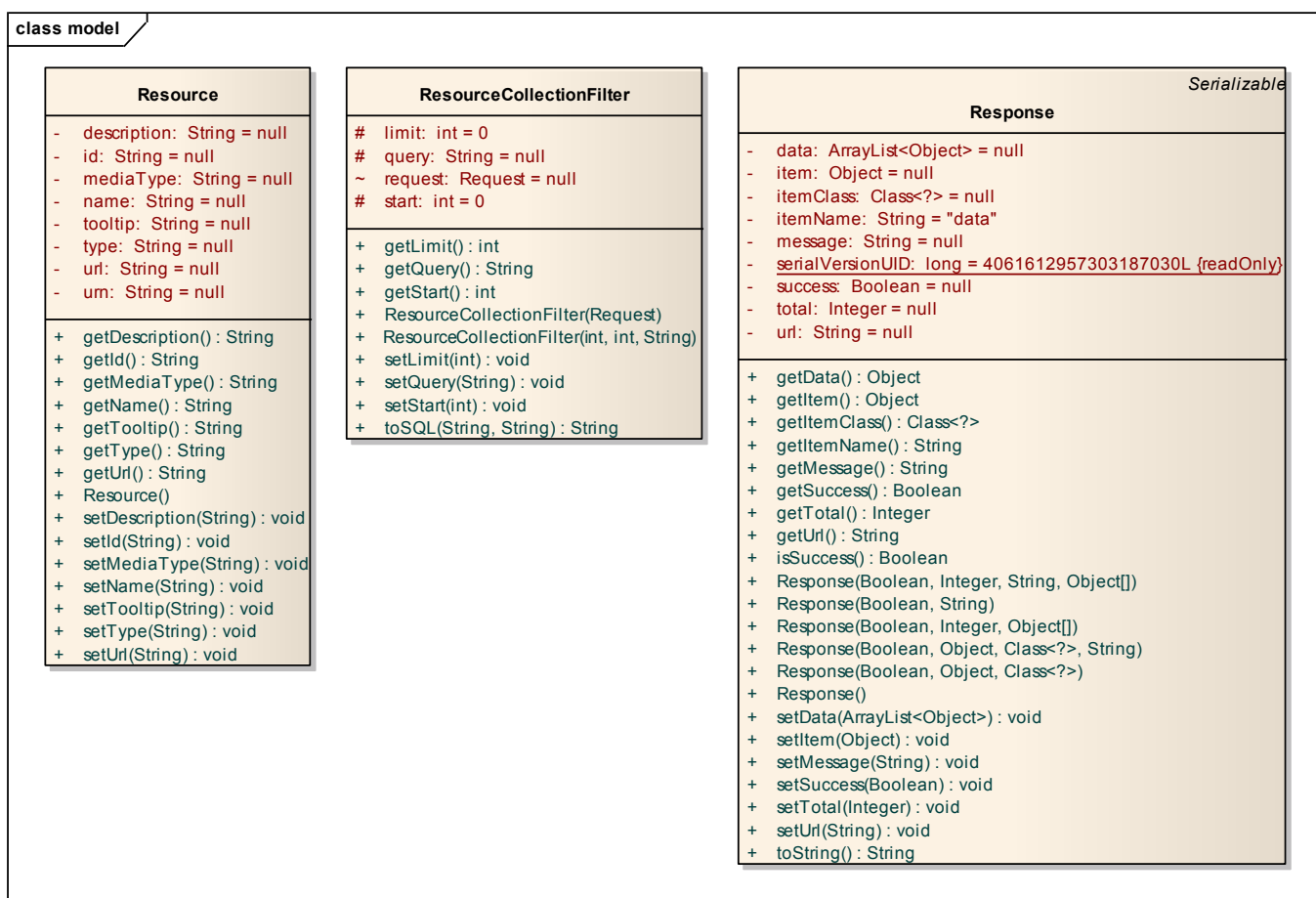
3.2 Description détaillée des données

NB : Les données de l'application évoluant assez fréquemment, il est très difficile de garder une documentation précise à jour. Les diagrammes présentés ci-dessous représente donc une vue globale des objets et les attributs ou méthodes présents dans ces objets ne représente pas forcément l'état actuel de l'application.

Certains diagrammes sont même simplifiés au maximum, c'est dire sans les méthodes, à des fins de présentation dans ce document.

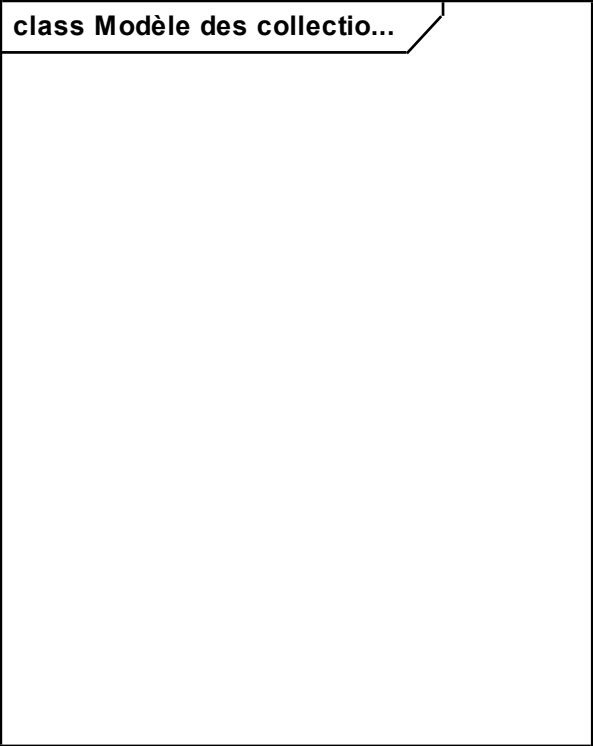
Pour une représentation plus fidèle des modèles, se référer à l'interface HTML qui contient l'ensemble des classes et des diagrammes pour la version 2.0 de SITools2.

3.2.1 Common.model



3.2.2 Collections

class Modèle des collectio...



3.2.3 Jeux de données

class Modèle simplifié jeu de donnés...

Cf : package fr.cnes.sitools.dataset.model dans l'export HTML.

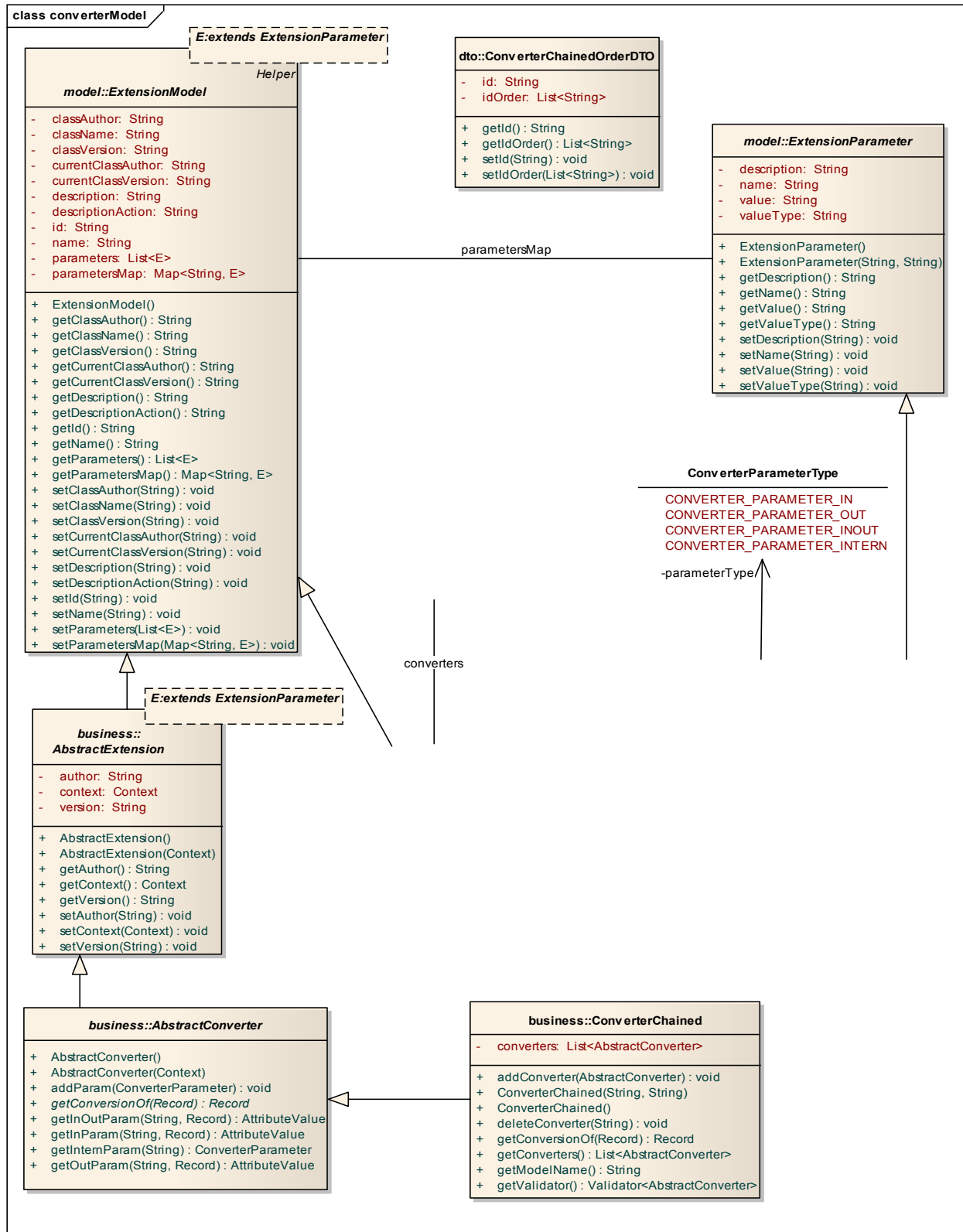
3.2.4 Mappings dictionnaire / concept

class DictionaryMappingModel

Un **DataSet** contient un ensemble de **DictionaryMapping**. Chaque **DictionaryMapping** contient l'identifiant d'un dictionnaire, si ce dictionnaire est le dictionnaire par défaut ainsi qu'une liste de **ColumnConceptMapping**. Ce dernier représente un mapping entre une colonne et un concept. On ne stocke que les identifiants des colonnes et des concepts.

Remarque : Dans ce diagramme de classe, on ne représente que les informations qui nous intéressent dans la classe **DataSet**. Il s'agit bien de la même classe que présentée précédemment.

3.2.5 Convertisseurs en sortie



Cf : package `fr.cnes.sitools.dataset.converter.model` dans l'export HTML.

Afin de bien comprendre les convertisseurs en sortie, il est nécessaire de les repositionner dans la chaîne de traitement des requêtes.

Une requête envoyée au serveur

- Passe dans un filtre en entrée.
- Se voit rajouter des prédicats s'il s'agit d'un appel depuis un formulaire.
- Est traitée par la source de données, celle-ci produit une liste de résultats
- Les résultats sont transformés en enregistrements génériques
- Les enregistrements génériques passent dans un ensemble de convertisseurs en sortie
- Les enregistrements génériques sont transformés en XML ou en JSON

Il y a donc un ensemble de convertisseurs en sortie, exécutés dans un ordre donné et spécifique à un jeu de données. Un convertisseur agit sur un enregistrement à la fois, il n'a pas connaissance de l'ensemble des enregistrements mais seulement du courant.

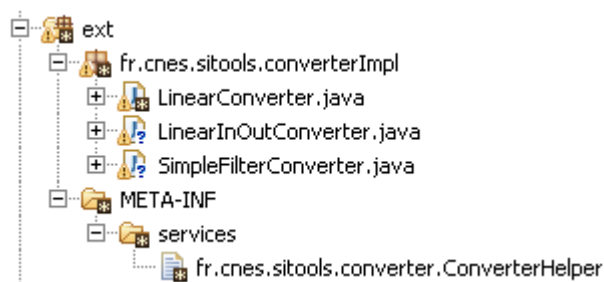
Les convertisseurs en sortie sont répartis dans quatre parties du projet.

- La partie persistance (package model) contient les objets qui permettent de sauvegarder les paramètres des convertisseurs
- La partie métier (package business) définit ce qu'est un convertisseur (par l'intermédiaire de la classe abstraite `AbstractConverter`) et offre les méthodes d'exécution des convertisseurs
- La partie implémentation contient les implémentations des convertisseurs
- Une autre partie (package `converterPlugin`) contient les ressources Restlet pour accéder à l'ensemble des convertisseurs ainsi que le mécanisme de découverte des convertisseurs.

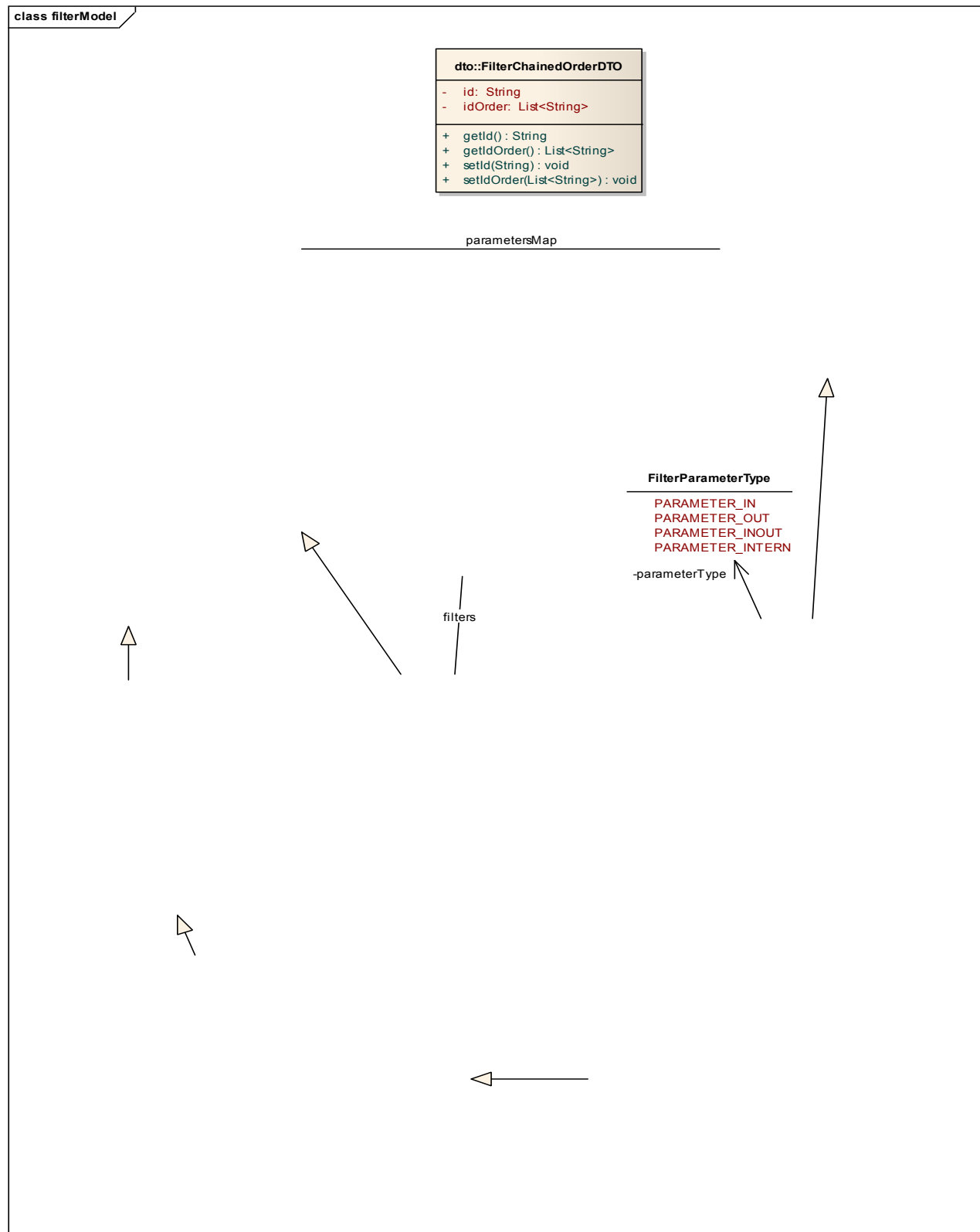
Les paramètres du convertisseur sont de 4 types différents :

- `CONVERTER_PARAMETER_IN` : Colonne à prendre en entrée du convertisseur
- `CONVERTER_PARAMETER_OUT` : Colonne en sortie du convertisseur, peut être une colonne virtuelle créée par le convertisseur
- `CONVERTER_PARAMETER_INOUT` : Colonne à prendre en entrée et en sortie
- `CONVERTER_PARAMETER_INTERN` : Constante interne au convertisseur.

La partie implémentation contient l'ensemble des convertisseurs. Les classes de convertisseur peuvent être situées dans des packages différents car ils sont découverts automatiquement au démarrage du serveur. Il faut simplement que la classe hérite d'`AbstractConverter` et que le nom compte de la classe soit ajouté au fichier `fr.cnes.sitools.converter.ConverterHelper`. Ce fichier doit être situé dans le dossier `META-INF/services`.

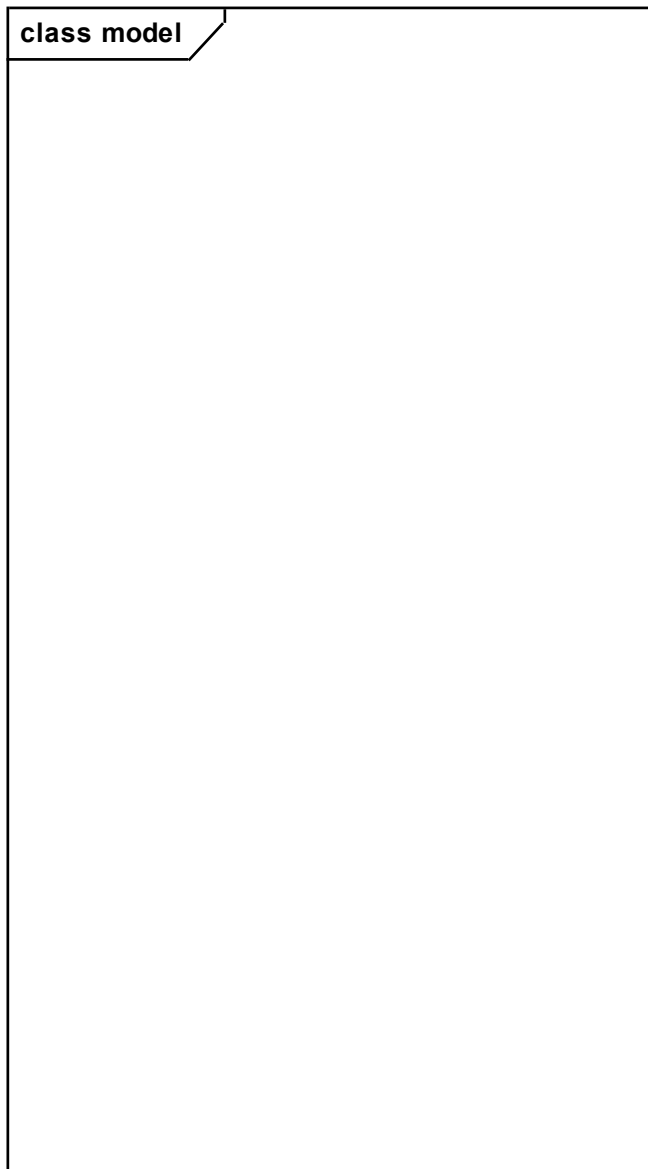


3.2.6 Filtres



Cf : package fr.cnes.sitools.dataset.filter.model dans l'export HTML.

3.2.7 Vues de jeu de données

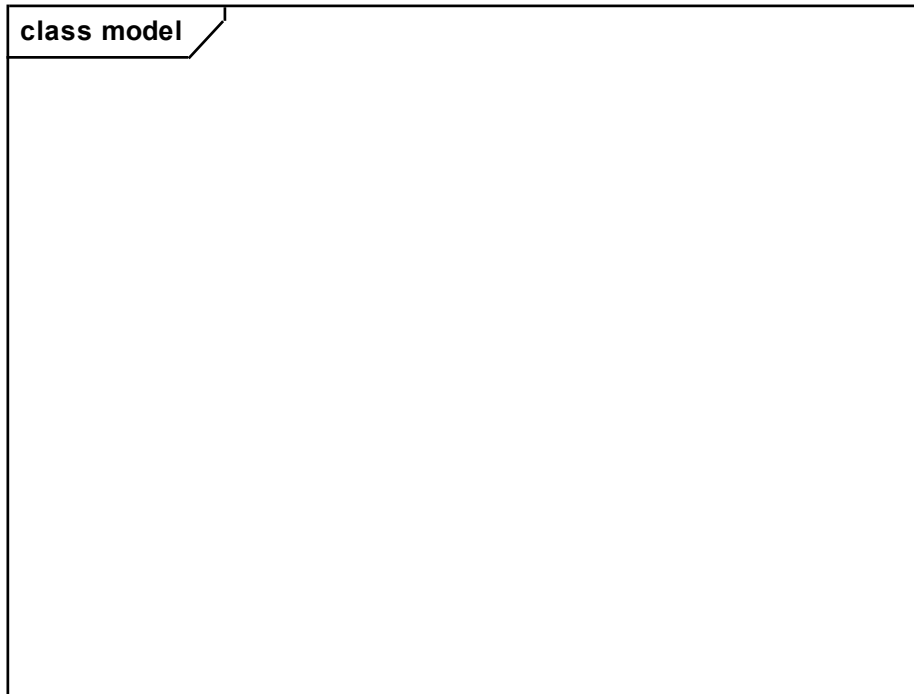


3.2.8 Sources de données

class Modèle simplifié source de donn...

Cf : packages fr.cnes.sitools.datasources.jdbc.model et fr.cnes.sitools.datasources.mongodb.model dans l'export HTML.

3.2.9 Espaces de stockages de fichiers



Cf : package fr.cnes.sitools.userstorage.model dans l'export HTML.

3.2.10 Dictionnaires

class model

Un dictionnaire est composé d'un ensemble de concepts correspondant à un **ConceptTemplate**.

Cf : package fr.cnes.sitools.dictionary.model dans l'export HTML.

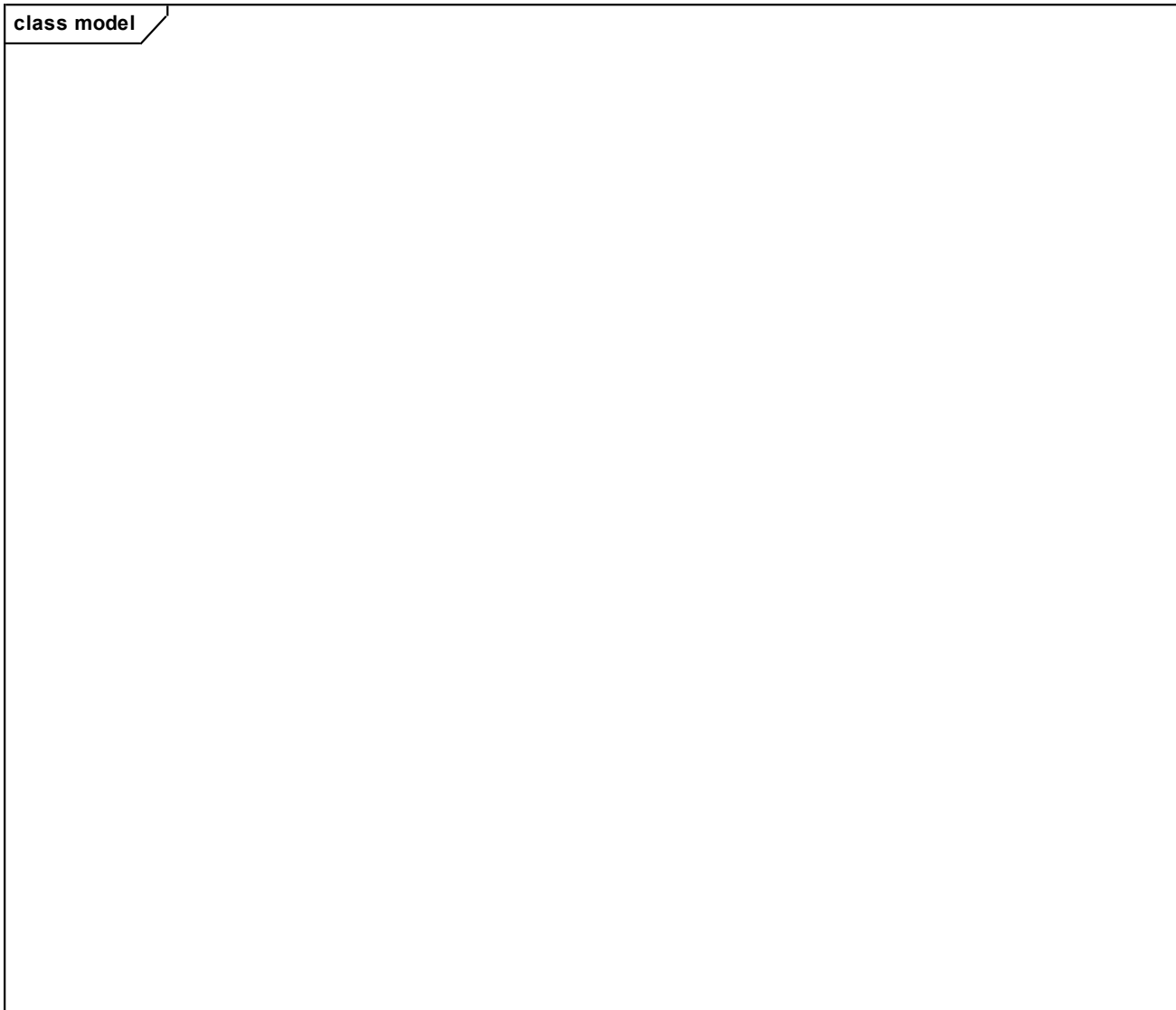
3.2.11 Concepts de dictionnaire

class TemplateConceptModel

Cf : package fr.cnes.sitools.dictionary.model dans l'export HTML.

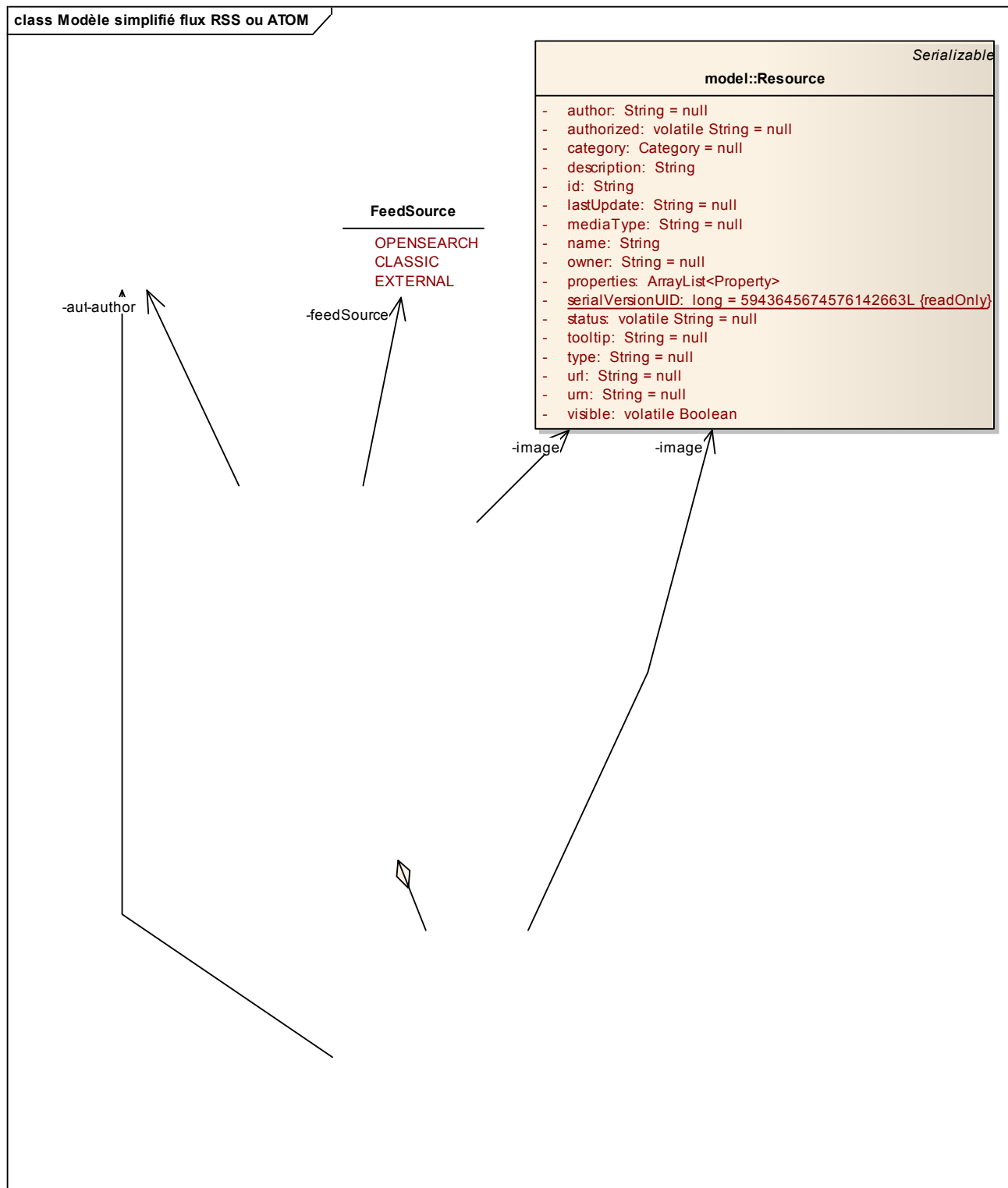
La classe **ConceptTemplate** stocke les paramètres simples d'un Template ainsi qu'une liste de **Property** permettant de stocker un ensemble de clés valeurs.

3.2.12 Dimensions



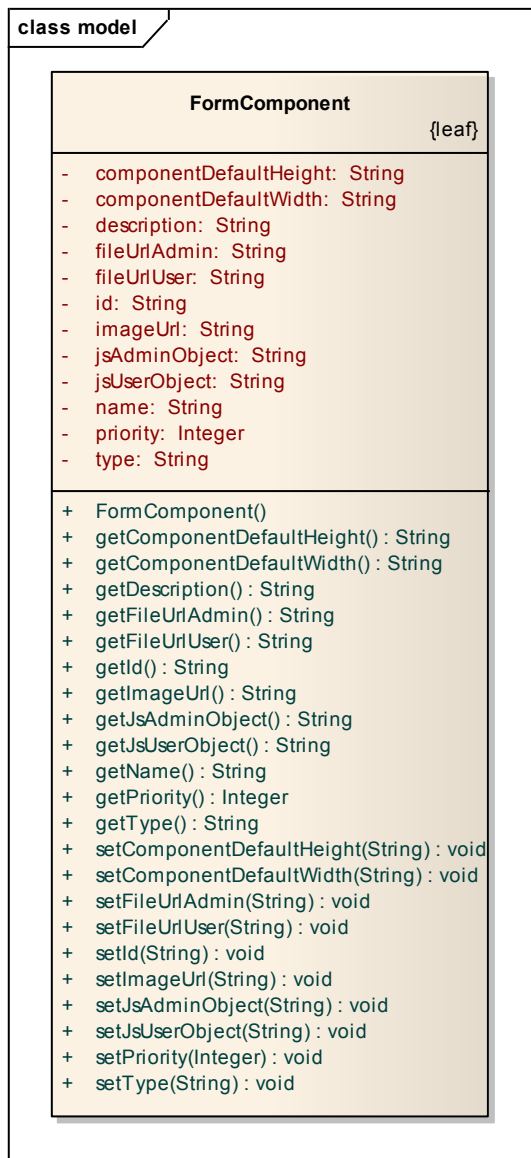
Cf : package fr.cnes.sitools.units.dimension.model dans l'export HTML.

3.2.13 Flux RSS ou ATOM

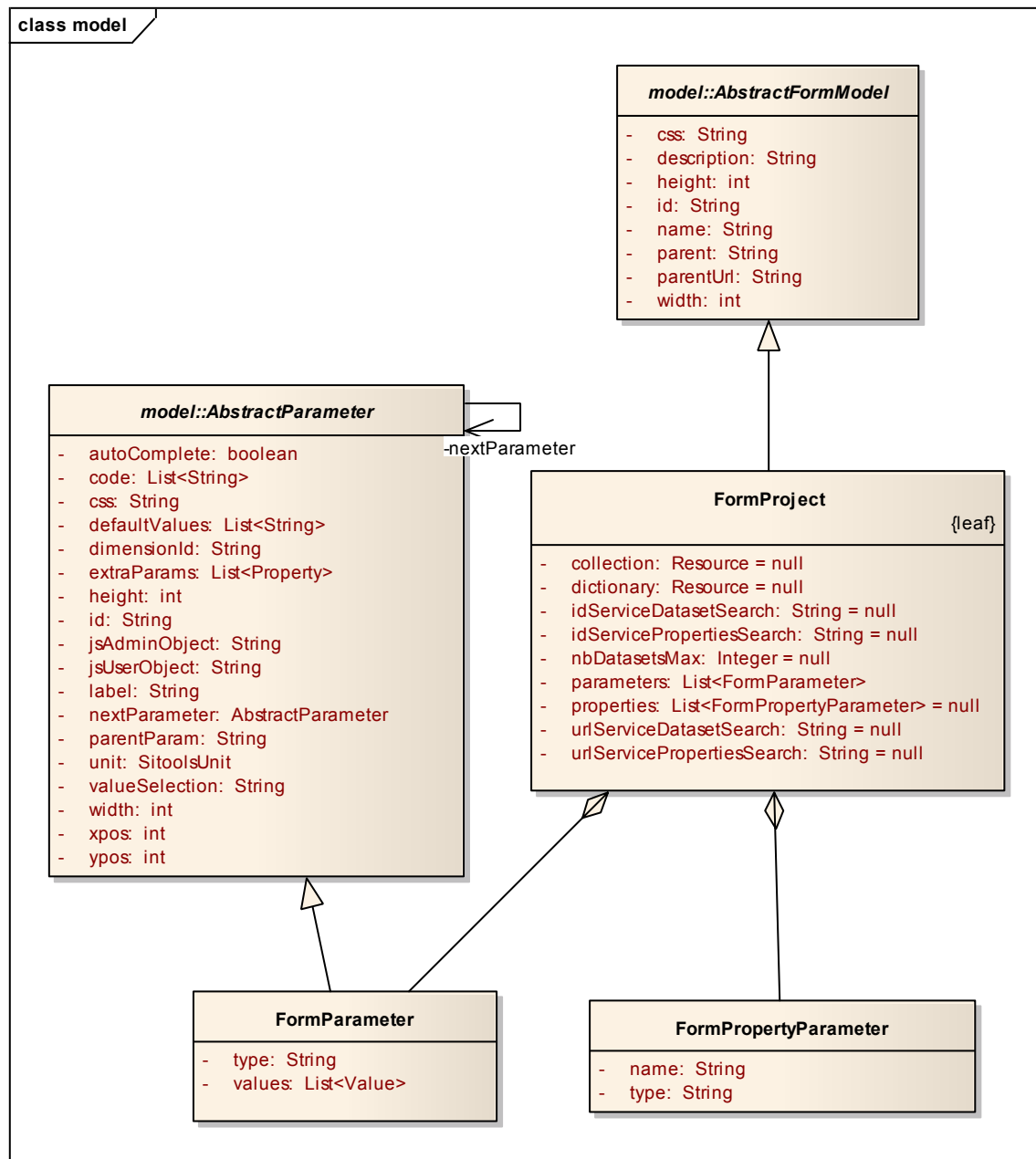


Cf : package fr.cnes.sitools.feeds.model dans l'export HTML.

3.2.14 Composants de formulaire

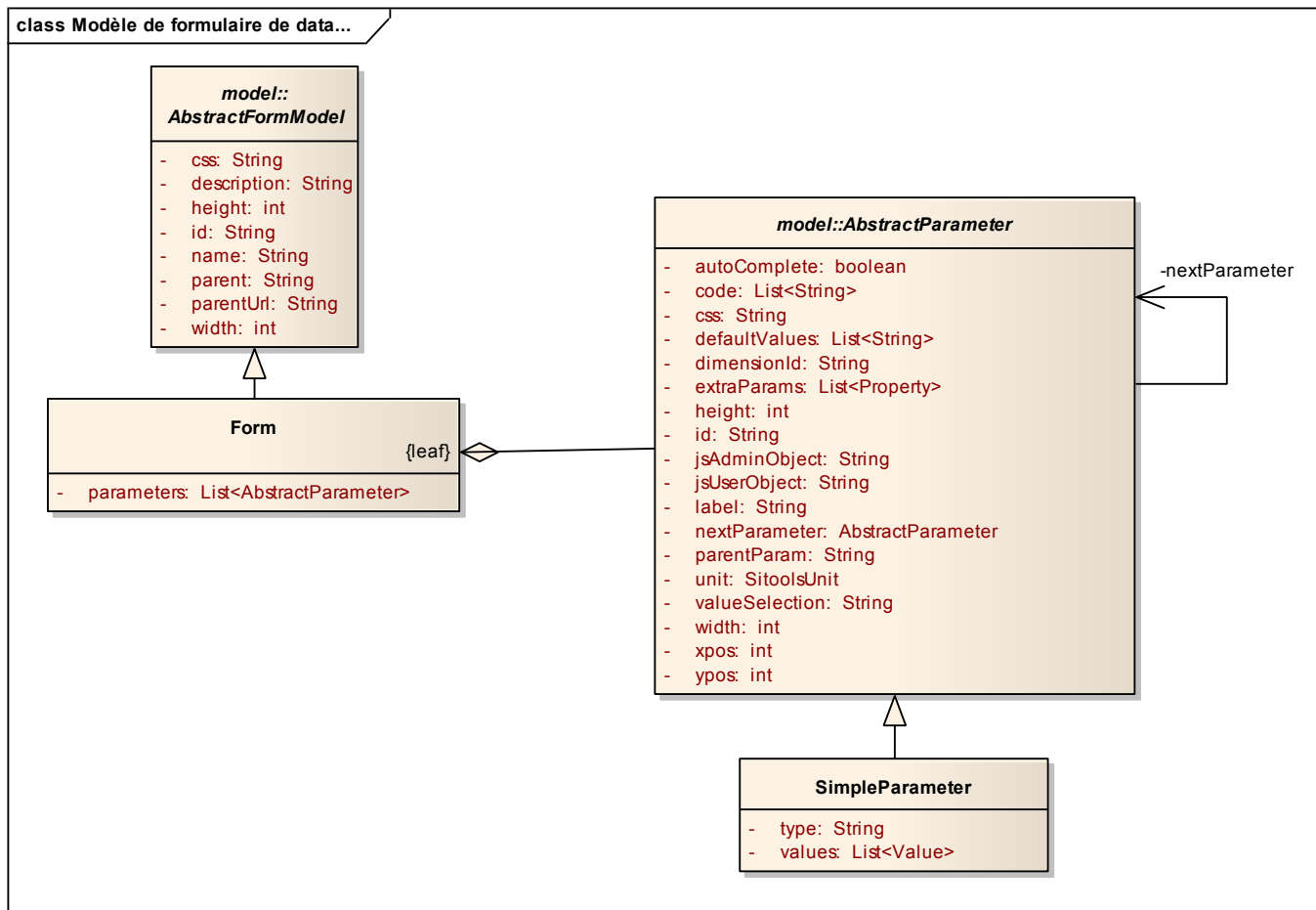


3.2.15 Formulaires de recherche multi-jeux de données



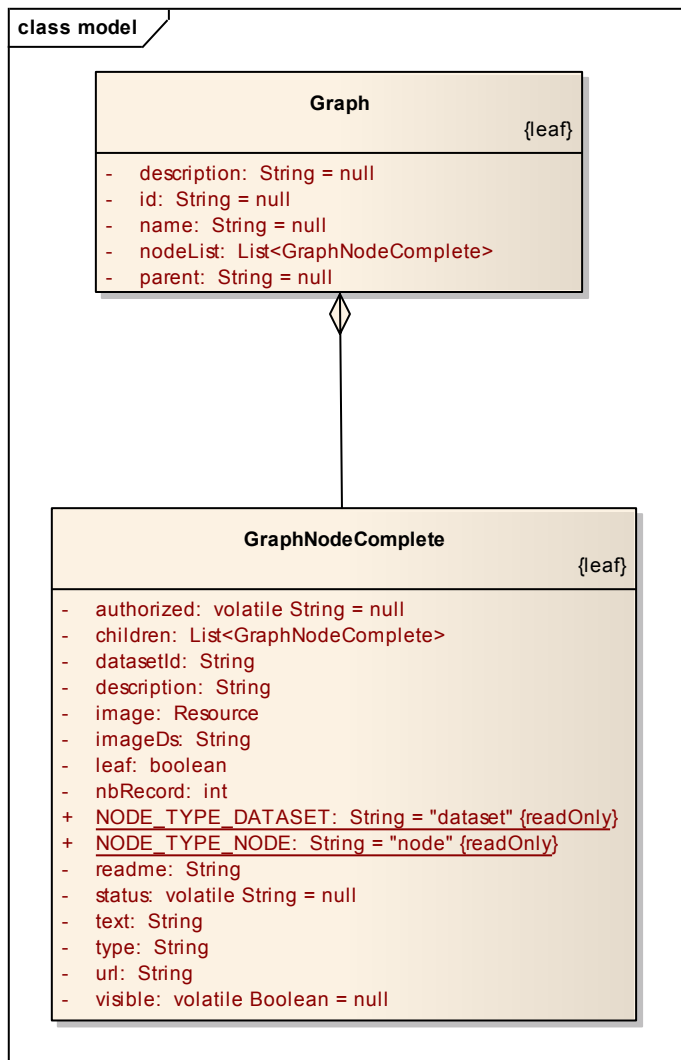
Cf : package fr.cnes.sitools.form.project.model dans l'export HTML.

3.2.16 Formulaires de recherche sur un jeu de données



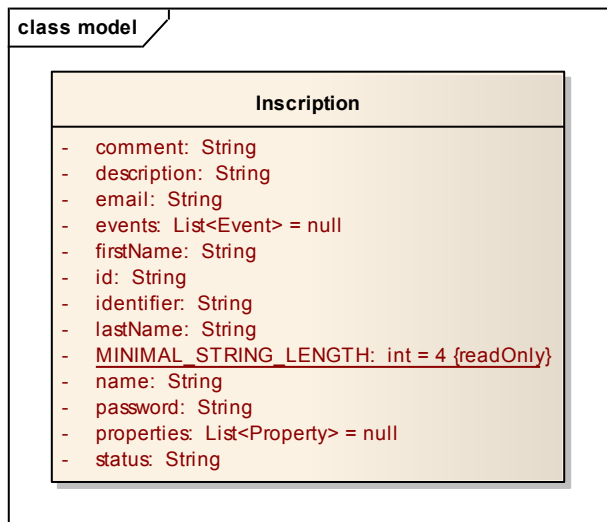
Cf : package fr.cnes.sitools.form.dataset.model dans l'export HTML.

3.2.17 Graphes de projet



Cf : package fr.cnes.sitools.project.graph.model dans l'export HTML.

3.2.18 Inscriptions



Cf : package fr.cnes.sitools.inscription.model dans l'export HTML.

3.2.19 Notifications

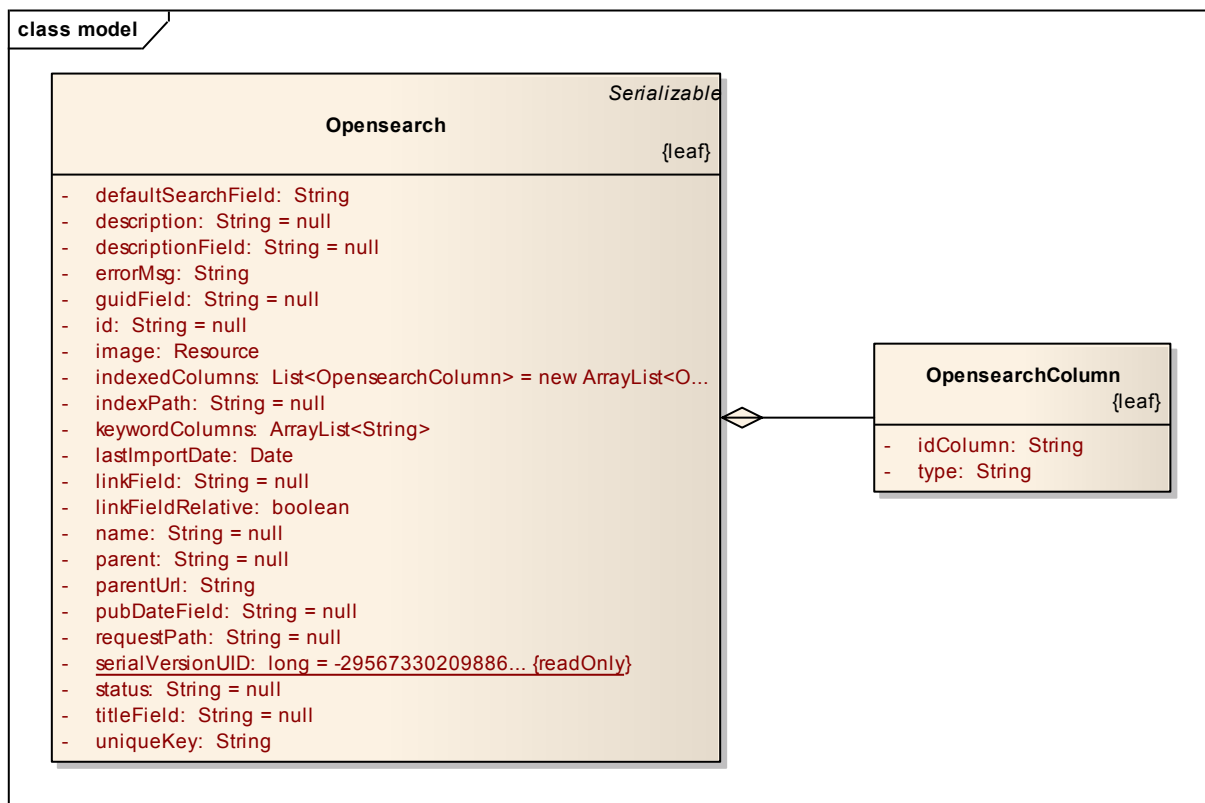
class model

Notification	Serializable
	{leaf}
<ul style="list-style-type: none"> + <u>ATTRIBUTE: String = "sitools.notifi..." {readOnly}</u> - event: String - eventSource: Object - message: String - observable: String - <u>serialVersionUID: long = 1L {readOnly}</u> - status: String 	
<ul style="list-style-type: none"> + getEvent() : String + getEventSource() : Object + getMessage() : String + getObservable() : String + getStatus() : String + Notification() + setEvent(String) : void + setEventSource(Object) : void + setMessage(String) : void + setObservable(String) : void + setStatus(String) : void 	

RestletObservable	Serializable
	{leaf}
<ul style="list-style-type: none"> - observers: List<RestletObserver> = new ArrayList<R... - <u>serialVersionUID: long = 3374364584167082902L {readOnly}</u> - store: volatile NotificationStore = null - uri: String 	
<ul style="list-style-type: none"> + addObserver(RestletObserver) : void + getObservers() : List<RestletObserver> + getStore() : NotificationStore + getUri() : String + notifyObservers(Context, Notification) : void + removeObserver(String) : void + RestletObservable() + setObservers(List<RestletObserver>) : void + setStore(NotificationStore) : void + setUri(String) : void 	

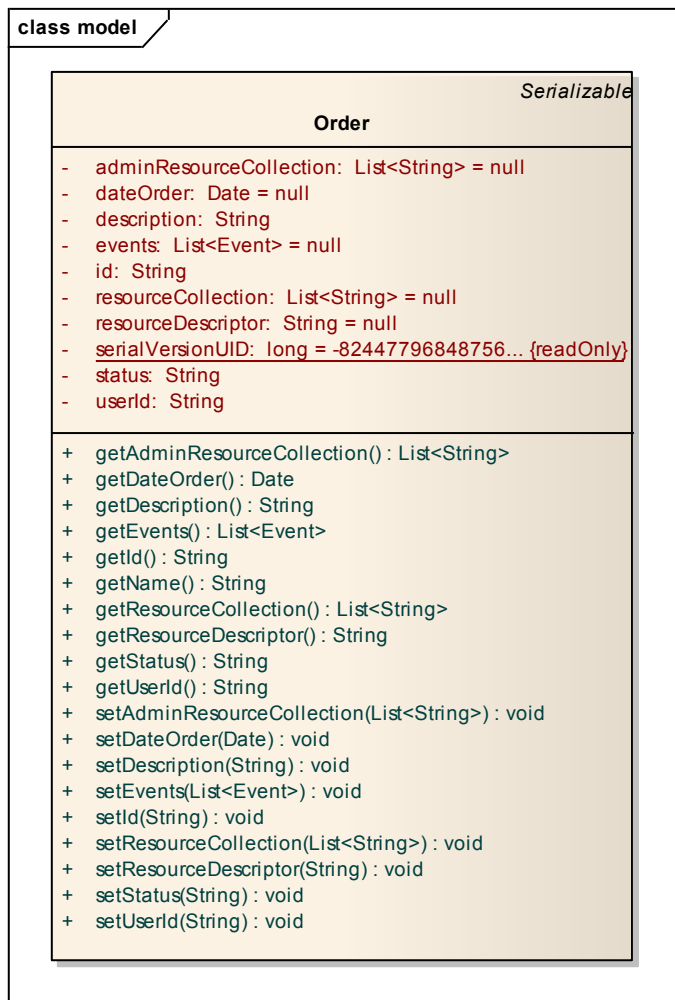
RestletObserver	Serializable
	{leaf}
<ul style="list-style-type: none"> - <u>DEFAULT_METHOD_TO_NOTIFY: Method = Method.POST {readOnly}</u> - mediaTypeToNotify: String = null - methodToNotify: String = null - <u>serialVersionUID: long = 6121649214628657269L {readOnly}</u> - uriToNotify: String - uuid: String 	
<ul style="list-style-type: none"> + getMediaTypeToNotify() : String + getMethodToNotify() : String + getUriToNotify() : String + getUuid() : String + RestletObserver() + setMediaTypeToNotify(String) : void + setMethodToNotify(String) : void + setUriToNotify(String) : void + setUuid(String) : void + update(RestletObservable, Object) : void 	

3.2.20 Opensearchs



Cf : package fr.cnes.sitools.dataset.opensearch.model dans l'export HTML.

3.2.21 Commandes hors ligne

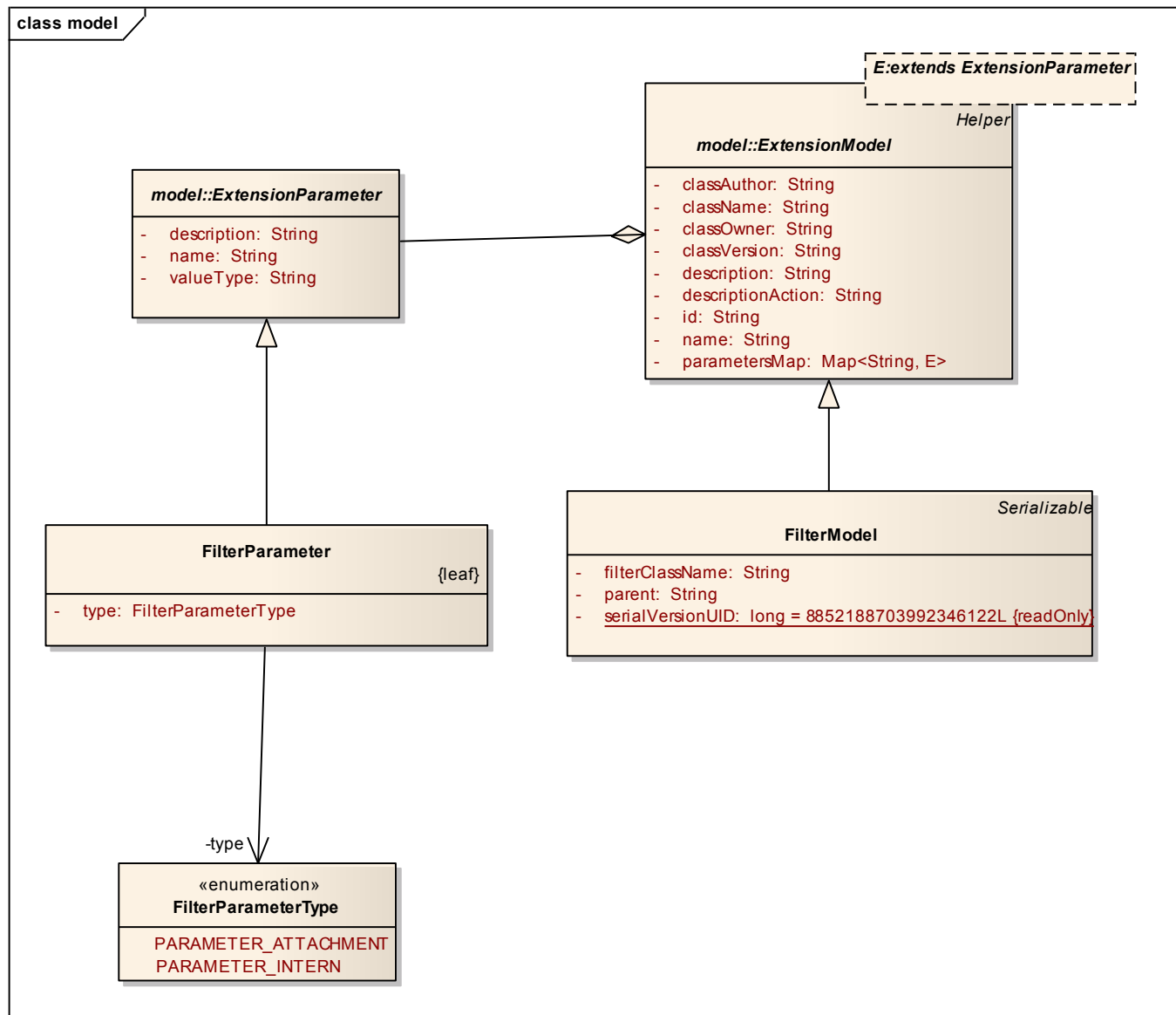


3.2.22 Applications plugins

class applicationPluginModel

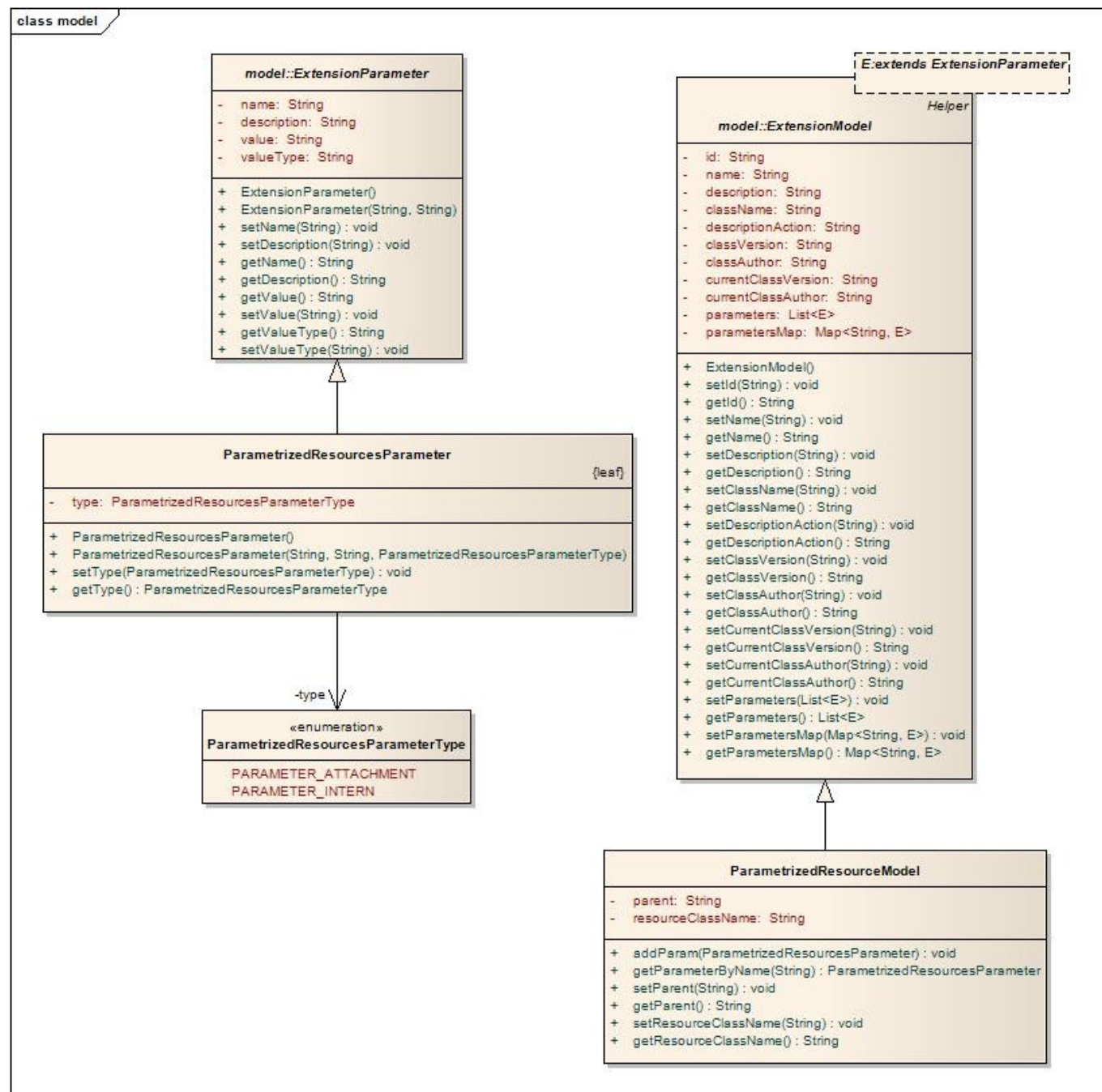
Cf : package fr.cnes.sitools.plugins.applications.model dans l'export HTML.

3.2.23 Filtres de sécurité plugins



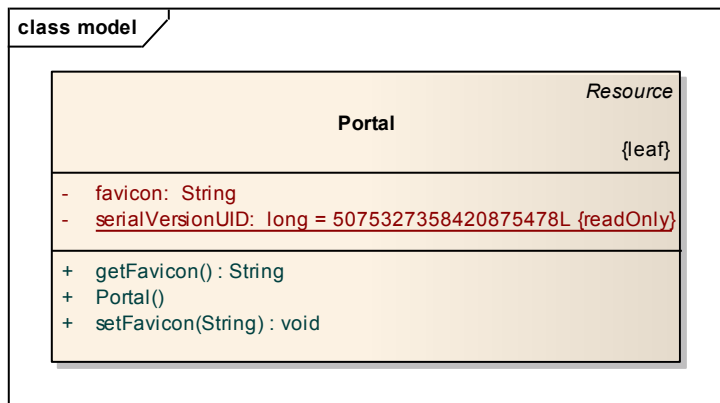
Cf : package `fr.cnes.sitools.plugins.filters.model` dans l'export HTML.

3.2.24 Ressources plugins



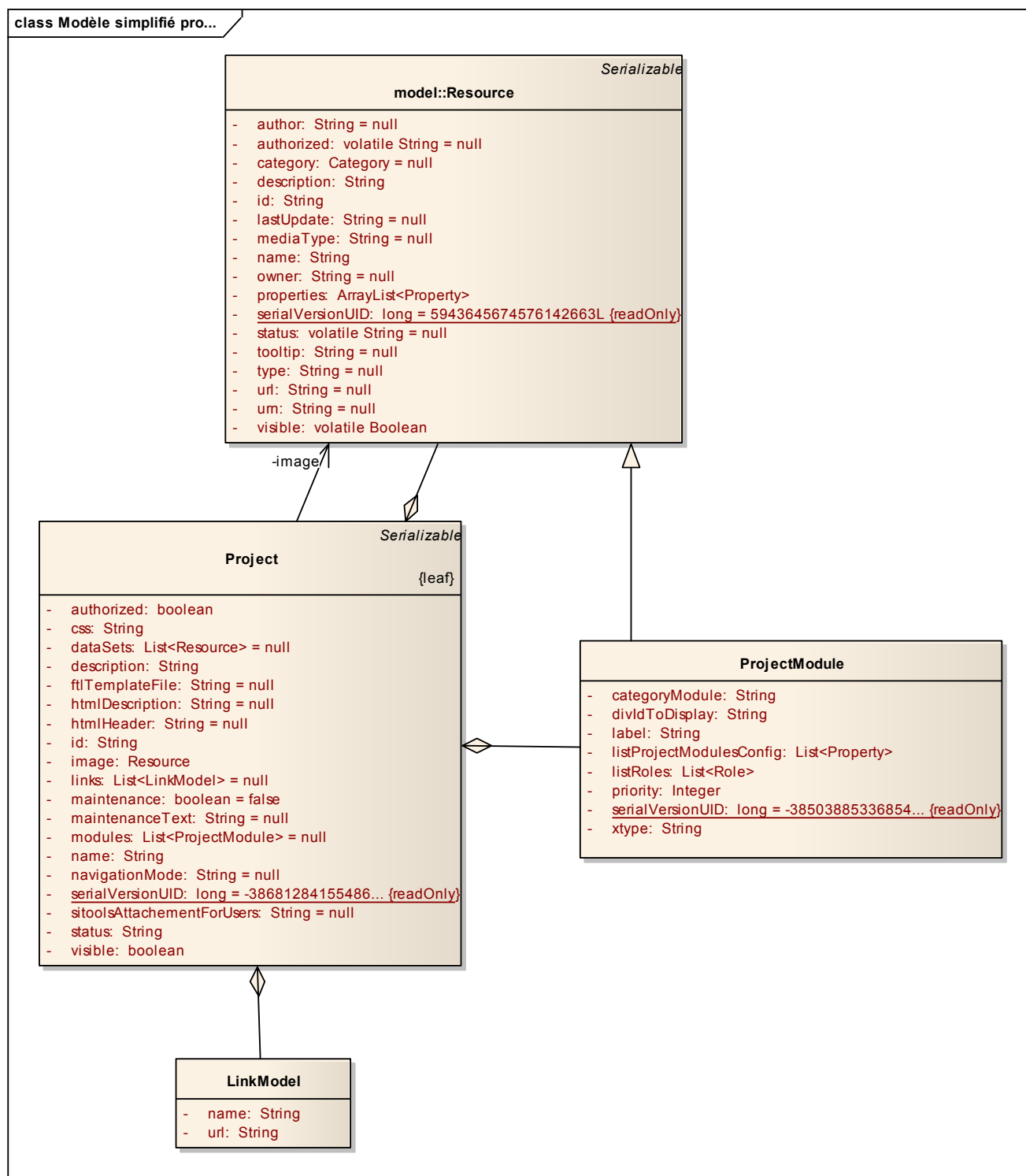
Cf : package fr.cnes.sitools.plugins.resources.model dans l'export HTML.

3.2.25 Portail



3.2.26 Projets

3.2.26.1 Modèle de classe



Cf : package fr.cnes.sitools.project.model dans l'export HTML.

3.2.26.2 Détails sur l'accès aux projets

Les projets définis par l'utilisateur de SITools2 sont accessibles de deux manières via l'API.

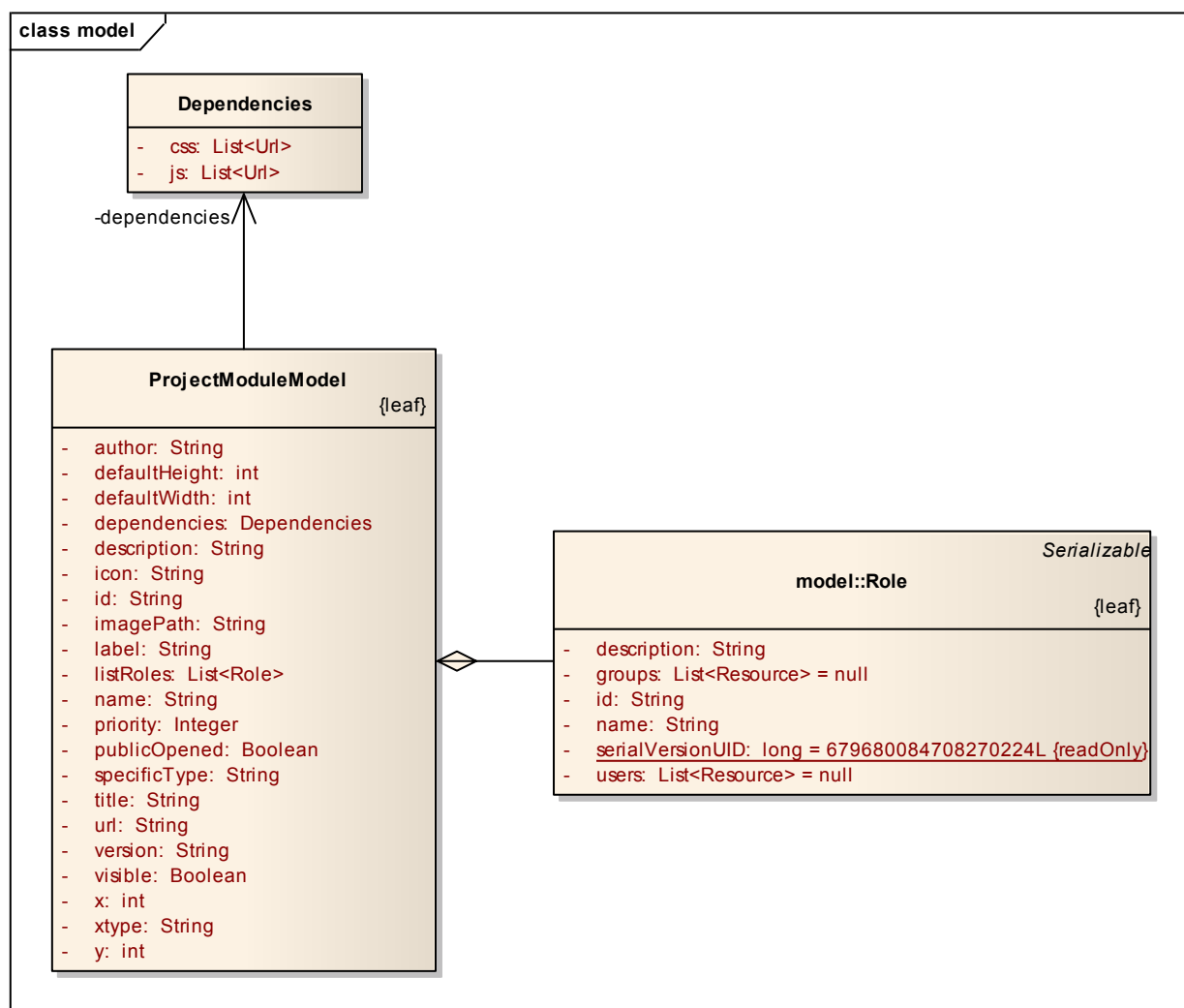
- La première consiste à obtenir la description du projet à partir de son URL d'attachement défini lors de sa création.
- La seconde permet l'accès au portail à partir de l'API de l'application client-user. Le projet est alors rattaché à l'URL de cette application par son nom.

Exemples :

<http://localhost:8182/project-attachment> = accès à la description du projet

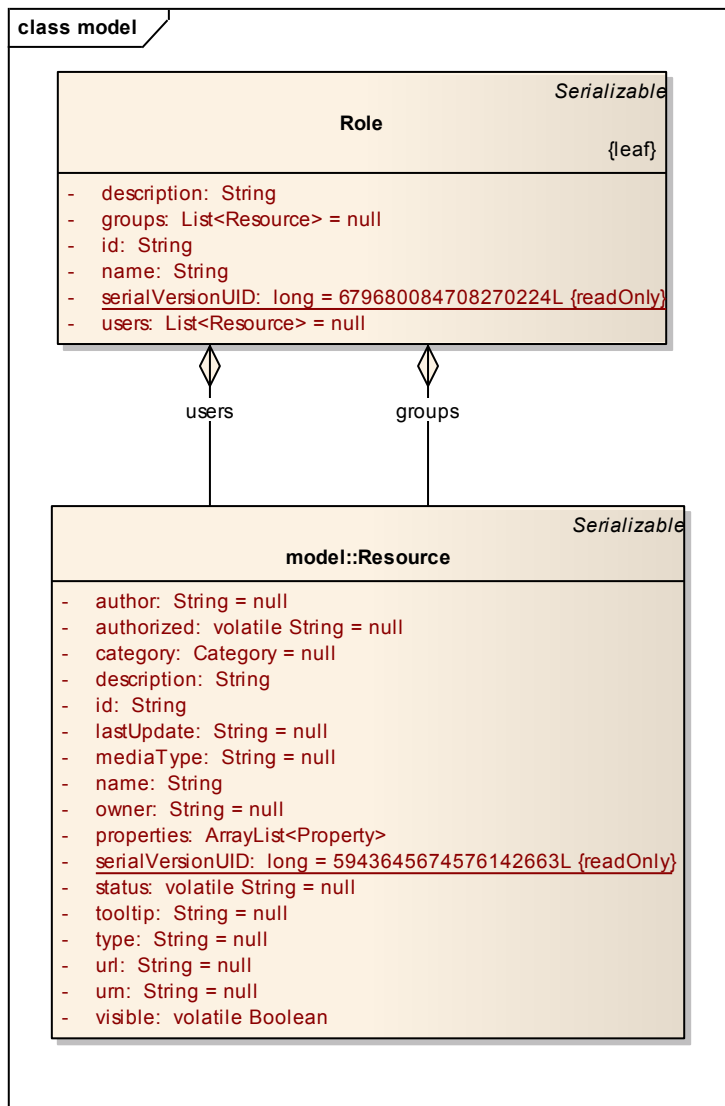
<http://localhost:8182/sitools/client-user/project-name/index.html> = accès au bureau du projet

3.2.27 Modules IHM de projet



Cf : package fr.cnes.sitools.project.modules.model dans l'export HTML.

3.2.28 Rôles



Cf : package fr.cnes.sitools.role.model dans l'export HTML.

3.2.29 Solr

class Solr class model

Cf : package fr.cnes.sitools.solr.model dans l'export HTML.

A chaque dataset est associé un cœur SolR.

Les ressources associées à SolR permettent une configuration minimale d'un cœur solr. Il est possible de configurer :

- Les colonnes indexées
- La colonne clé primaire
- Les colonnes utilisées pour l'auto complétion,
- La colonne de recherche par défaut
- Le type d'indexation pour chaque colonne.

Pour chaque colonne indexée on peut également lui associer un champ du flux RSS de retour.

A noter qu'une colonne définie comme clé primaire doit forcément être de type « string » (ou tout autre type ayant pour classe « solr.StrField »). Cette colonne sera indexée, stockée et pourra être retournée mais ne pourra pas être requêtée.

Dans le cas où cette configuration est insuffisante, il est possible de modifier à la main certains fichiers générés.

Il est possible de modifier le fichier rss.xsl, qui réalise la transformation en RSS des résultat SolR (fichier présent dans le dossier conf/xslt d'un cœur SolR).

Il est possible de modifier le fichier SolrConfig.xml, qui contient la configuration du cœur SolR de manière générale. Une modification dans ce fichier demande un redémarrage serveur pour que SolR prenne en compte les modifications.

Les fichiers schema.xml et db-data-config.xml sont les deux autres fichiers utilisés principalement dans un cœur solr. Il s'agit de la configuration des colonnes indexés et de la source de données utilisée. Ces fichiers sont générés à chaque création de cœur SolR, pour les modifier, il faut donc éditer les templates permettant de générer ces fichiers (attention les modifications ne seront pas répercutées sur les cœurs déjà créé, ils devront être recréés). Ces templates se trouvent dans le package solr, les templates sont schema.ftl et db-data-config.ftl.

3.2.29.1 Le mécanisme d'auto complétion :

SolR permet d'offrir un mécanisme d'auto complétion. Cette auto complétion est utilisée dans Sitools2 ainsi que dans le widget du moteur de recherche de Firefox.

Cette auto complétion utilise le mécanisme de « TermComponents » qui permet de lister tous les termes pour une colonne donnée. Ce mécanisme est très rapide car il utilise la fonction « TermEnum » de Lucene pour itérer sur les termes. Le « TermComponents » permet également de donner le nombre d'occurrence du terme dans l'index et peut être utilisé sur plusieurs colonnes simultanément.

La requête utilisé est donc la suivante :

```
"solr://" + osId + "/terms?" + fieldStr + "&terms.prefix=" + query + "&terms.sort=index"
```

osId : L'identifiant du cœur SolR

query : Requête saisie par l'utilisateur, elle sert de préfix pour sélectionner uniquement les termes qui commencent par cette requête.

fieldStr : La liste des colonnes utilisées pour l'auto complétion. Il s'agit d'une liste de paramètres de la forme : "terms.fl=" + fieldName où fieldName est le nom de la colonne.

3.2.29.2 Limitations de Solr et proposition de solution

Les fichiers du cœur Solr ne sont pas supprimés lors de la suppression d'un index

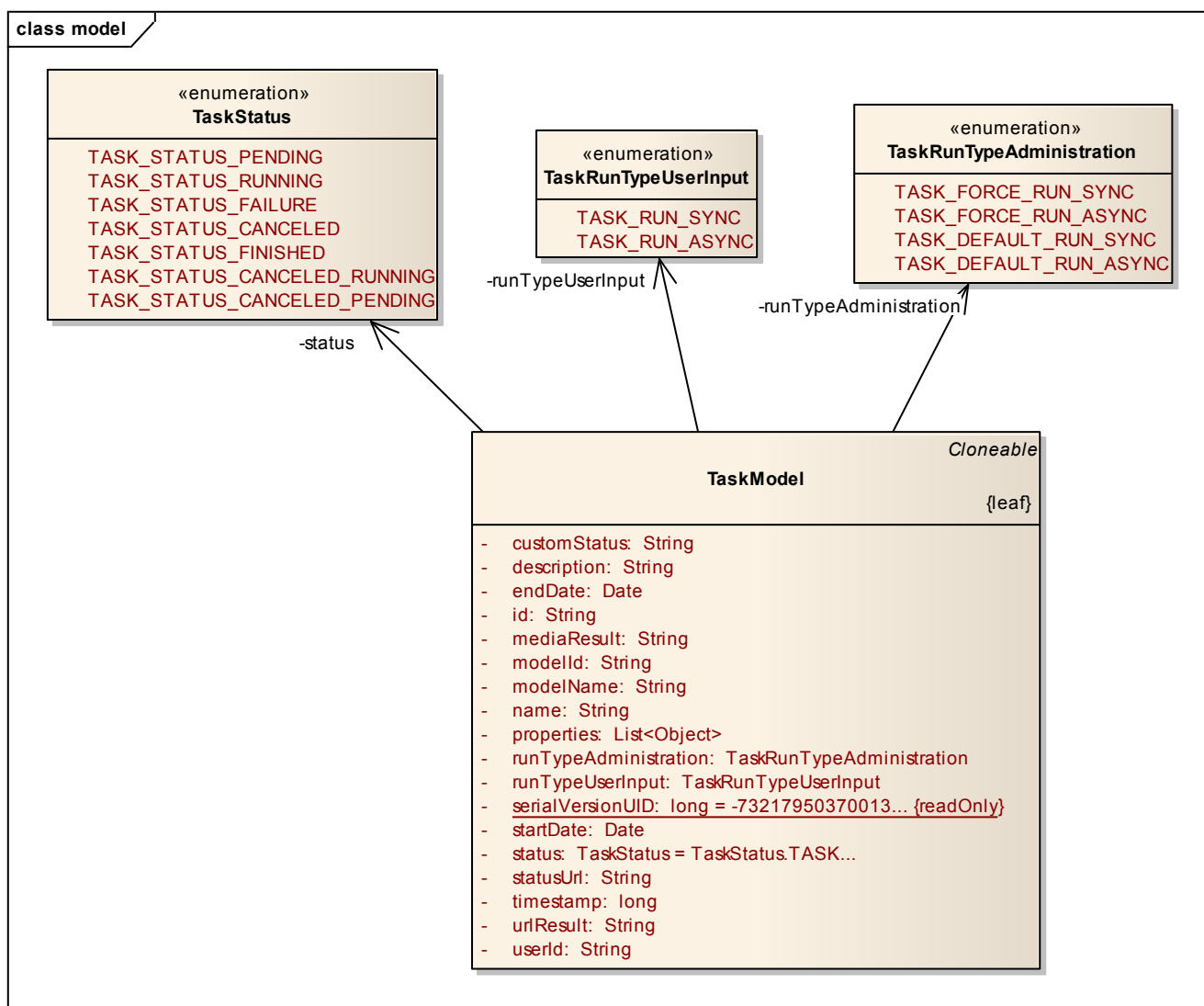
En effet, il semble que ces fichiers soient encore utilisés par le système après le déchargement (commande UNLOAD) d'un cœur. Il est donc impossible de les supprimer.

Proposition de solution :

Une solution consiste à supprimer ces fichiers au démarrage du serveur. Il faudrait donc garder la liste des cœurs ayant été supprimé et les supprimer avant le démarrage de Solr.

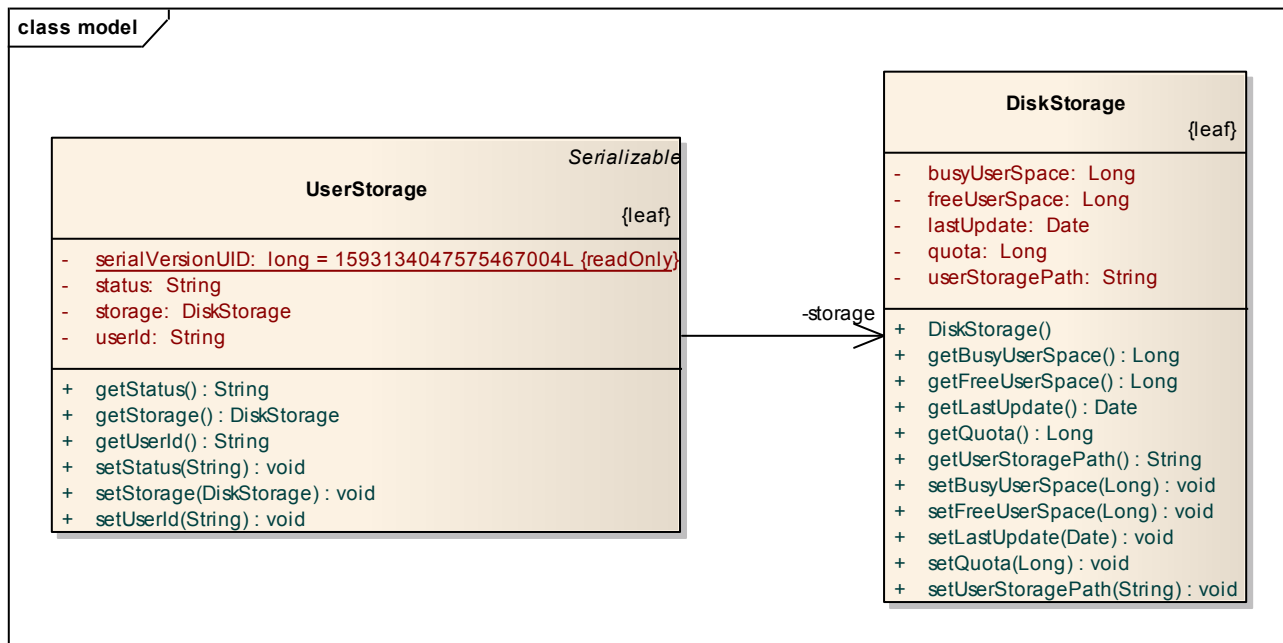
Une autre solution serait d'arrêter Solr, de supprimer les fichiers de cœurs supprimés et redémarrer Solr. Cela permettrait de n'arrêter que Solr et non le serveur en entier (Il faut tout d'abord voir si l'on peut supprimer uniquement Solr).

3.2.30 Tâches



Cf : package fr.cnes.sitools.tasks.model dans l'export HTML.

3.2.31 Espaces de stockage utilisateur



3.2.32 Utilisateurs et Groupes

class model

3.3 Description des messages d'erreur

La gestion des messages d'erreur s'effectue par défaut par la remontée d'exceptions sur le client avec pour message d'erreur un code correspondant à un message dans les fichiers i18n d'internationalisation.

Pour un appel classique, une page HTML d'erreur est retournée avec le code de l'exception dans le texte.

Dans les échanges client-serveur en json une réponse d'erreur se traduit par une réponse http 200 et avec un contenu :

```
{  
  success : false,  
  message : 'error.user.create'  
}
```

3.4 Recherche multi-dataset

Sitools2 propose une fonctionnalité de recherche multi-dataset

3.4.1 Le modèle de données

Cf : partie 3.2.15, Formulaire de recherche multi-jeux de données

3.4.2 Les services

La recherche multidataset se décompose en 2 services, le service de recherche par propriété de dataset et le service de recherche sur les données du dataset.

Le service de recherche par propriété sert à filtrer la liste des datasets candidats à la recherche par rapport à leurs propriétés.

Le service de recherche sur les données du dataset permet de compter le nombre d'enregistrements sur chacun des datasets pour une requête donnée.

3.4.2.1 Service de recherche par propriété

Un dataset contient une liste de propriété. Une propriété est composée d'un nom, d'une valeur et d'un type. Le service est implémenté par une ResourcePlugin ajoutée au projet automatiquement lorsque l'on l'ajoute d'un formulaire multidataset. Et il répond à une signature particulière, inspirée des filtres sur les données des datasets :

`k[#]=FILTER_TYPE|property|value1`

Où :

- # est le numéro du filtre dans la requête à partir de 0

- *FILTER_TYPE* est le type de recherche sur les propriétés
- *property* est le nom de la propriété
- *value1* est la valeur à rechercher. Il peut y avoir plusieurs valeurs en fonction du type de recherche, chacune des valeurs est séparée par un |

3.4.2.2 Service de recherche sur les données du dataset

Ce service est implémenté par une TaskResourcePlugin attachée automatiquement au projet lorsque l'on crée un formulaire multidataset. Il effectue un comptage du nombre d'enregistrements de la requête donnée par l'utilisateur. La signature du service est la même que pour la recherche sur un dataset mais au lieu de donner des colonnes en paramètre, l'utilisateur saisit un nom de dictionnaire et un nom de concept. On a donc la signature suivante :

c[#]=FILTER_TYPE|dictionary_name|conceptName|value1

Où :

- # est le numéro du filtre dans la requête à partir de 0
- *FILTER_TYPE* est le type de filtre à appliquer
- *dictionary_name* est le nom du dictionnaire
- *concept_name* est le nom du concept
- *value1* est la valeur à rechercher. Il peut y avoir plusieurs valeurs en fonction du type de recherche, chacune des valeurs est séparée par un |

Il s'agit d'un service asynchrone et afin de l'optimiser, chaque requête est lancée dans un Thread particulier. Cela permet de paralléliser les requêtes et d'avoir un résultat plus rapidement. Pour cela on utilise un Pool de Thread avec une taille fixe et on le remplit avec l'ensemble des requêtes à effectuer (classe DataSetCounterCallable). Ensuite on stocke les résultats au fur et mesure dans le statut de la tâche en cours. Le client peut donc interroger le statut de la tâche et récupérer les résultats au fur et à mesure.

3.5 Gestion des dates

3.5.1 Types de date pour les bases de données

Les tableaux suivants présentent les différents types de date disponible dans les bases de données. Pour chacun de ces types, s'ils contiennent une information de timezone et si leur utilisation est possible dans Sitools2.

3.5.1.1 Postgresql:

Type de colonne	Information de TimeZone	Géré par Sitools2
Timestamp	NON	OUI

Timestamp with timezone	OUI	NON
Date	NON	OUI
Time	NON	OUI
Time with timezone	OUI	NON

Le type « Time with timezone » ne devrait pas être utilisé, en effet une timezone sans date ne permet pas de connaître les changements d'heure d'été, d'hiver. Postgresql semble le prendre en compte car il est dans la spécification SQL mais il est déconseillé de l'utiliser.

« The type time with time zone is defined by the SQL standard, but the definition exhibits properties which lead to questionable usefulness. In most cases, a combination of date, time, timestamp without time zone, and timestamp with time zone should provide a complete range of date/time functionality required by any application. »¹

3.5.1.2 MySQL

Type de colonne	Information de TimeZone	Géré par Sitools2
Date	NON	OUI
Datetime	NON	OUI
Timestamp	OUI	OUI et NON
Time	NON	OUI
Year	NON	NON

Les valeurs d'une colonne de type timestamp sont converties en UTC depuis la timezone (celle du serveur, ou configuré dans MySQL) lorsqu'elles sont stockées. La conversion inverse est faite lors de l'affichage. La date affichée sera donc correcte uniquement si la TimeZone du serveur est la même lors de l'insertion en base que lorsque l'on effectue une requête. (Ce qui explique le OUI et NON). Il est cependant conseillé d'utiliser un type DateTime, non sensible à la TimeZone.

Le type Year ne pourra pas être pris en compte car il n'est pas disponible en Postgresql, présente peu d'intérêt et sera compliqué à gérer au niveau de l'affichage et des requêtes.

¹ Extrait de la documentation Postgresql : <http://www.postgresql.org/docs/9.1/static/datatype-datetime.html>

3.5.2 Format d'échange des dates

Ce schéma présente la façon dont les dates sont véhiculées dans les différentes couches de Sitools2.

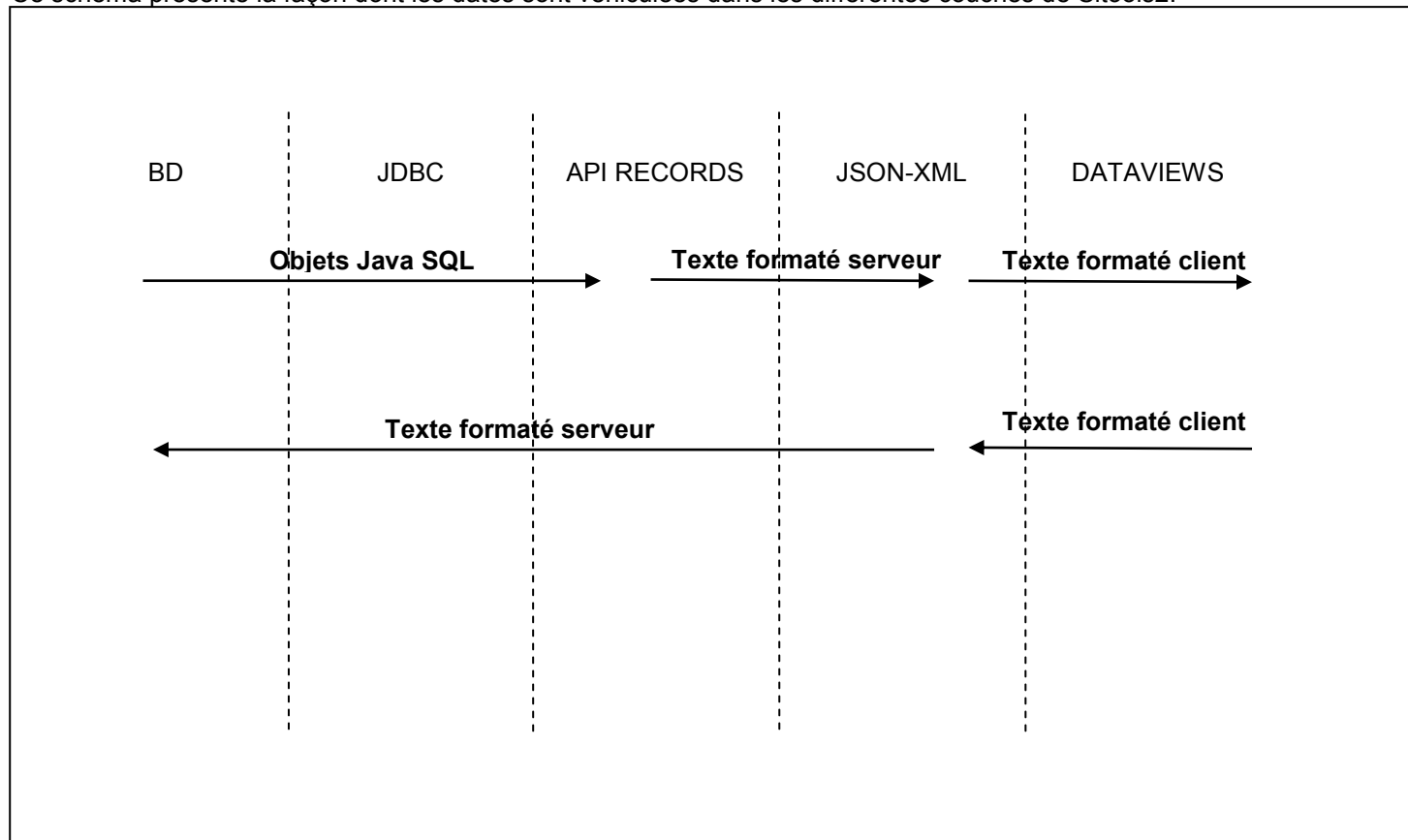


Figure 1: Format des dates dans les différentes couches de Sitools

Objets Java SQL : Un objet Java de l'API JDBC correspondant au type de données de la base de données.

Texte formaté serveur : Une date formatée selon un format standard.

- Dans le cas d'une date ou d'un timestamp : « yyyy-MM-dd'T'HH:mm:ss.S »
- Dans le cas d'un temps (type time) « HH:mm:ss.S »

Il s'agit de template de date Java.

Texte formaté client : Une date formatée selon un format défini par l'administrateur pour chacune des colonnes de dataset ou des composants de formulaire. Il s'agit d'un template de date ExtJs.

Cela permet d'avoir un format d'échange standard entre le client et le serveur (Texte formaté serveur) et un affichage spécifique pour chaque vue du dataset. Un format d'affichage par défaut est également défini.

4 Architecture OSGi

Sitools2 repose sur une architecture modulaire calquée sur le modèle OSGi de Restlet.

Cependant, Sitools2 est actuellement testé et validé dans un mode de fonctionnement hors plateforme OSGi.

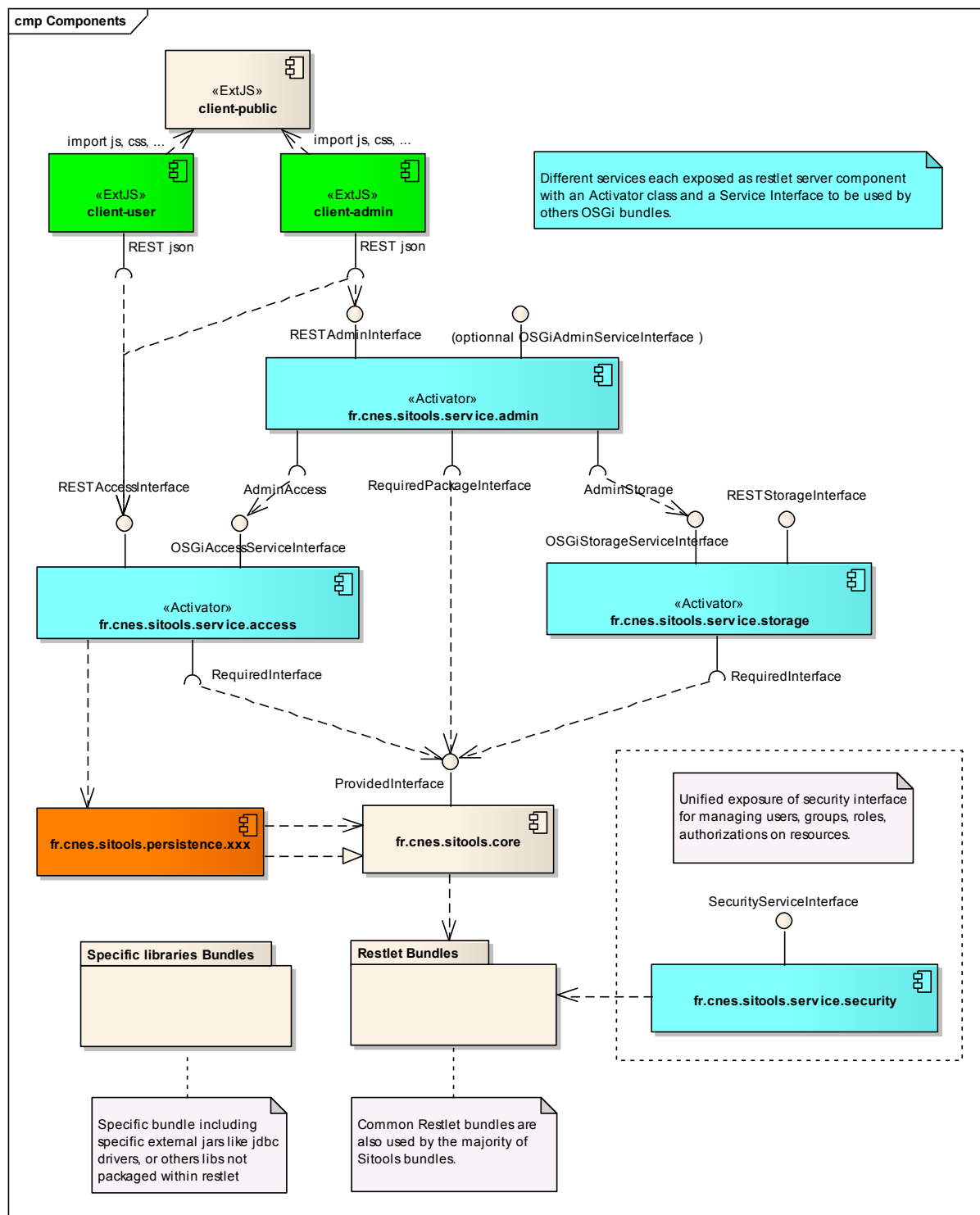
Le lancement de Sitools2 est fait de manière classique par un batch avec un classpath incluant tous les jar nécessaires.

Dans un mode de fonctionnement OSGi, il s'agirait de lancer une plateforme osgi, qui s'occuperait automatiquement de découvrir et charger les jars/classes des bundles présents dans un répertoire.

L'architecture Sitools2 pour ce mode de fonctionnement est décrite ci-après, avec un découpage du Sitools2 actuel en 3 bundles activables : bundle sitools administration, sitools accès et sitools storage.

Ce mode de fonctionnement OSGi a été expérimenté et doit encore être validé.

4.1 Bundles SITools



Dans notre conception actuelle, le service de stockage consiste à exposer des répertoires de fichiers en REST soit en intranet soit en extranet. Il est constitué d'un serveur Restlet avec 2 applications : une application d'administration destinée à la définition des répertoires et de leur mode d'accès, et d'une application gérant l'ensemble des accès sur ces répertoires.

Des extensions peuvent être envisagées pour exposer des contenus de répertoires distants (accédés par FTP par le composant storage) de la même manière que des répertoires locaux.

Ce bundle peut ainsi être conçu de façon très isolée du reste de Sitools. Les seules dépendances sont une dépendance vers des packages core de Sitools (fr.restlet.common, ...) et une dépendance vers un service de sécurité mutualisé.

4.2 Bundle service admin

Le service d'administration consiste à exposer des ressources propres à l'administration de la plateforme et de ses autres services.

La première étape a consisté à découvrir via une API REST les autres services, ainsi que les autres bundles osgi par le biais de simples ressources.

Dans de futures versions, nous pouvons envisager de piloter les services arrêt/démarrer, gérer l'installation de nouveaux plugins, de nouvelles versions de jar le tout dynamiquement via une ihm et une api d'administration de la plateforme.

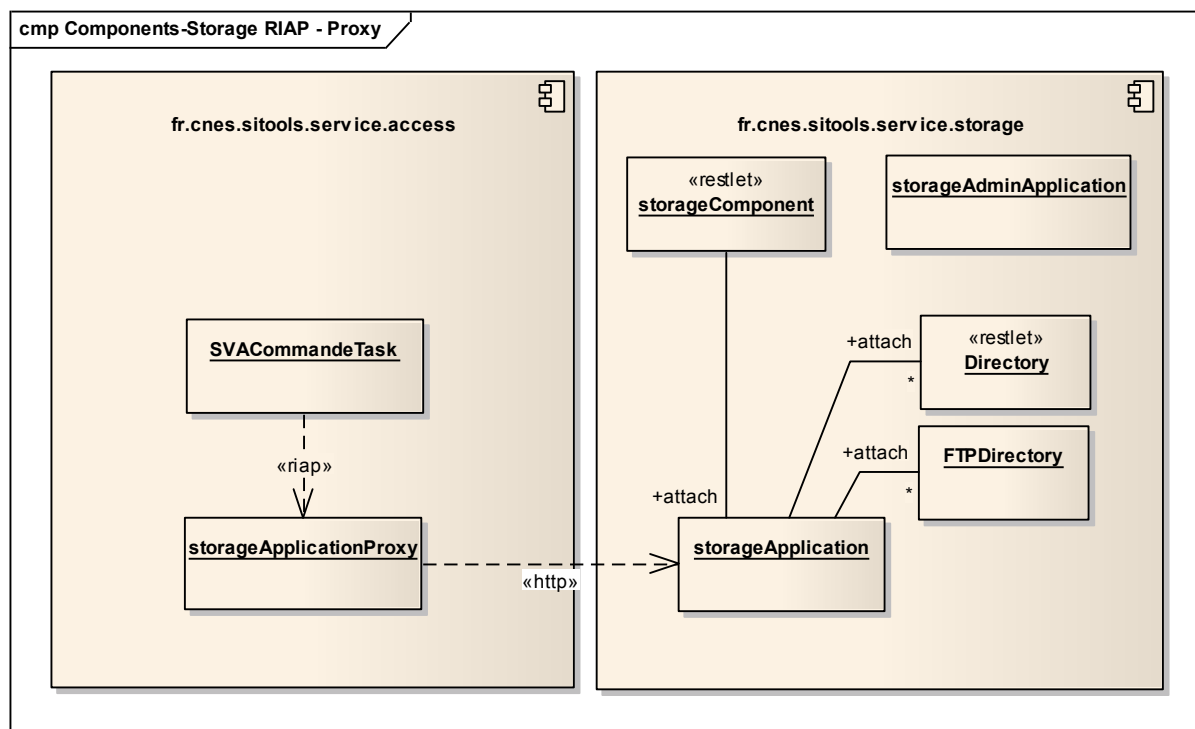
4.3 Bundle service storage

Interaction entre la fonction d'accès de commande et la fonction de storage.

Dans une première étape, la fonction de stockage est réalisée par une application intégrée dans le serveur d'accès. L'appel depuis un SVA de commande par exemple s'effectue en RIAP pour limiter le couplage à l'API rest de l'application.

Si l'on devrait comme décrit plus haut, isoler la fonction de stockage dans un service spécifique, avec son propre serveur, qui d'ailleurs pourrait être distant du service d'accès, la solution que nous proposons de mettre en place consiste à développer une StorageApplicationProxy, simplement chargée d'effectuer la redirection vers le service approprié.

Le schéma est le suivant :



Lorsqu'une application/ressource du service d'accès a besoin des fonctions du service de stockage, elle sollicite une application proxy en riap.

La `storageApplicationProxy` effectue une redirection des requêtes vers le service Storage et son application.

De cette façon, on peut assembler un serveur d'accès soit avec les applications réelles soit avec des applications proxy, sans modifier les applications/ressources qui en dépendent.

L'assemblage consiste à attacher en dynamique à un serveur les applications au `component.getInternalRouter().attach(...)` pour que celle-ci soit accessible via riap.

4.4 Etat d'implémentation actuel

Le mode de fonctionnement décrit précédemment est en cours d'implémentation.

Le couplage lâche entre le service access et le service storage n'est pas encore réalisé. Le service d'accès utilise via riap une application storage directement instanciée et attachée au même serveur que le service d'accès.

5 Dépendances d'appel entre ressources et applications

La désactivation d'applications de Sitools2 peut amener des dysfonctionnements majeurs dus à la dépendance des applications entre elles. En particulier, les appels en RIAP entre ressources peuvent être rompus. Deux actions ont été menées pour prévenir les risques liés à cette problématique :

- Les applications de Sitools2 ont été classées par catégories, afin d'indiquer à l'administrateur le niveau de criticité quant à l'arrêt ou l'ajout de limitations d'accès à certaines applications.
- Une carte des dépendances a été réalisée, afin d'anticiper les éventuels dysfonctionnements liés à l'arrêt, ou l'impossibilité d'accès, d'une application donnée. Cf : DCP-ULISSE-1.0.doc

5.1 Catégories d'applications

Les applications de Sitools2 ont été classées suivant trois catégories distinctes :

- **SYSTEM** : ces applications ne doivent en aucun cas être arrêtées ou limitées en accès, auquel cas le fonctionnement du serveur est gravement atteint (impossibilité de démarrage, de configuration). Elles constituent l'ensemble minimal permettant l'arrêt/ le démarrage d'autres applications.
- **ADMIN** : ces applications servent à l'administration des éléments fonctionnels de Sitools2. La limitation d'accès ou leur arrêt ne stoppe pas de manière critique Sitools2, mais les services d'administration concernés seront inaccessibles et engendrer de potentiels dysfonctionnements.
- **USER** : ces applications permettent à l'utilisateur de Sitools2 de bénéficier de ses fonctionnalités, sans toucher de manière critique à sa configuration interne. Elles sont pour vocation d'être limitée en accès suivant les groupes et utilisateurs de l'instance.

3 nouvelles catégories permettent de classer les applications créées par configuration :

- **ADMIN_DYNAMIC** : Cela concerne actuellement uniquement les applications d'exploration de base de données associées à une datasource
- **USER_DYNAMIC** : il s'agit des applications créées par configuration : DataSetApplication, ProjectApplication, applications plugins.
- **SYSTEM_DYNAMIC** : il s'agit des applications créées par configuration qui serait très critique. Pour l'instant il n'y a aucune application dans cette catégorie.

Ces catégories sont disponibles en interrogeant l'API des applications (GET <HOST>/sitools/applications), ou grâce à l'IHM d'administration de Sitools2 (menu Applications).

6 ANNEXES

6.1 Export HTML des classes

Est joint à ce document, un export HTML des classes contenant l'ensemble des classes du cœur de SITools2 ainsi qu'un diagramme de classe de chacun des packages.

Certains chapitres de la partie 3.2 y font référence via le nom du package.

7 Documents applicables et de référence (A/R)

A/R	Référence	Titre

8 Glossaire et abréviations

8.1 Glossaire

Terme	Définition

8.2 Abréviations

Abréviation	Nom détaillé