Projet WSFS

Semantic Web

File System

Par Valérie Bourla

Fanny Linel

Matthieu Hahn

Julien Serra

Danny Seng

Et Jean-Pierre Vuong

M2 MIAGE 2008/2009

Table des matières

Présentation du projet

A.Objectif

On observe que si l'accès à des ressources quelconques sur un réseau informatique se fonde sur des modèles de graphes plus ou moins ramifiés, l'accès aux ressources propres d'un ordinateur via les systèmes d'exploitation les plus courants se fonde sur des arborescences de répertoires. Les artifices que constituent les « raccourcis » et autres « alias » que l'utilisateur peut créer et placer à sa convenance visent à pallier la rigidité de ces moyens d'accès, de même d'une certaine façon que le principe des « variables d'environnement ».

A l'évidence il n'existe pas de sémantique d'organisation arborescente de ressources qui satisfasse à elle seule l'ensemble souhaitable des modalités d'accès. A défaut de pouvoir le faire, l'organisation des systèmes de fichiers est laissée à la libre initiative des administrateurs, des architectes d'applications et enfin aux utilisateurs d'ordinateurs, sans moyen d'en faire ressortir une logique d'organisation : celle-ci supposerait l'implémentation de graphes de relations entre les ressources, établi selon un modèle sémantique de l'organisation souhaitée.

Aujourd'hui, avec le Web sémantique, sont apparus des langages formels fortement répandus, RDF, RDFS, OWL pour exprimer ces sortes d'organisations. On se propose de les exploiter pour organiser l'accès aux ressources propres d'un ordinateur autrement que par la métaphore de répertoires emboîtés. L'ergonomie d'utilisation d'un tel ordinateur, comme sa gestion devrait ainsi s'en trouver fortement améliorée.

B.Organisation et calendrier

Tout au long du projet, nous avons réalisé un journal retraçant toute l'analyse et les recherches que nous avons effectuées dans ce projet. Ce journal est situé en annexe de ce document.

Au niveau de la répartition des parties, voici les différentes équipes qui ont travaillées sur le projet :

- **L'interface graphique** : Julien Serra / Danny Seng,
- **Le classifieur** : Valérie Bourla / Matthieu Hahn,
- **L'URI – URL** : Fanny Linel / Jean-Pierre Vuong.

Description des besoins fonctionnels

Le projet est partitionné en trois grandes parties distinctes :

- WSFS GUI : Interface Homme Machine qui permettra l'utilisation de l'application,

- WSFS URI-URL : Réalisation des web services de gestion des ressources et de leur association à une ou plusieurs situations effective,

- WSFS Classifier : Service de classement de ressources en fonction de leurs métadonnées.

L'interface graphique

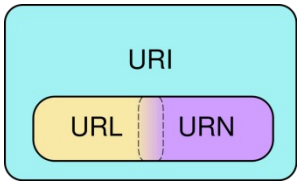
URI - URL

I.Explication

« Il s'agit de réaliser les (web) services de gestion des ressources identifiées par des URI et de leur association à une ou plusieurs situations effective (URL). Parmi ces web services on prévoit celui qui gère le tracé des versions et des variantes d'une ressource, en exploitant éventuellement des solutions de compression des versions précédentes d'une ressource.

La préoccupation de l'archivage et des sauvegardes est prise en compte par un service d'infrastructure générant des répertoires de rangements couvrant des périodes temporelles, cette notion étant intégrée comme un concept de base dans l'ontologie d'organisation »

La première phase consiste d'abord à comprendre puis à analyser les différentes notions. Nous avons compris dans un premier temps qu'il faut mettre en place un web service qui pourra gérer des ressources. L'URI d'une ressource est son identifiant sémantique, contrairement à l'URL qui est sa localisation physique dans un réseau. Un URI peut être associé à plusieurs URL.



La deuxième phase est comprendre le fonctionnement d'un outil de gestion de version. En effet, l'un des objectifs qu'on doit atteindre est de gérer les versions d'une ressource. Afin de faciliter notre travail, nous avons choisi d'utiliser un logiciel libre remplissant nos fonctionnalités. Puis de l'interfacer avec un web service qu'on aura développé. Dans la suite de ce rapport, une explication sur nos choix du logiciel et sur le fonctionnement d'outil de version seront présentées.

Rappel des fonctionnalités principales :

*A partir d'un URI :

Déterminer sa localisation physique pour une nouvelle ressource (le déplacer du dépôt vers une localisation). Transmettre l'URL dans la base.

Déplacer une ressource du dépôt vers une localisation implique forcément des règles d'orientation (orientation du fichier vers sa localisation physique). Ces règles dépendent de l'URI.

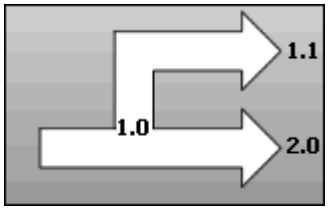
*Création d'une version :

Dans le cas d'une nouvelle ressource, on crée juste une nouvelle version de la ressource.

*Dans le cas d'une ressource existante :

Une petite modification crée une sous-version

Pour une modification importante on crée une nouvelle version



*Consulter :

Donne les informations sur la version de la ressource

*Supprimer:

La ressource est logiquement supprimée mais pour des raisons d'archivage la ressource reste sur le dépôt physiquement.

*Verrouillage :

La ressource est logiquement supprimée mais pour des raisons d'archivage la ressource reste sur le dépôt physiquement.

Voici un exemple d'un URI et de son URL :

URI	URL
/ Documents/Photos/Vacances/photo1	C:/Documents and settings/users/St-Malo/photo1.jpg
/Site/Vente/index	http://www.ventepascher.com

Recherche de l'existant

Dans cette partie seront présentés les différents outils de gestion de version existant :

ClearCase.

ClearCase est l'un des outils de gestion le plus puissant sur le marché, recommandé pour la gestion d'un grand nombre de projet.

Avantages

- possibilités étendues
- très complet (avec des fonctionnalités inexistantes dans les autres outils de gestion)

Inconvénients

- complexe
- demande une formation pour l'utiliser

CVS.

CVS est un gestionnaire de version très répandu.

Avantages

- Nombreuses interfaces graphiques

Inconvénients

- Incrémente le numéro de version de chaque fichier
- Problèmes d'ajout de suppression de fichier et de répertoire

Subversion

Subversion est un système de contrôle de version open Source, créé pour remplacer CVS et combler ses lacunes.

Avantages

- système léger
- rapide d'utilisation
- simplicité d'utilisation grâce à des interfaces graphiques clients (tels tortoiseSVN, rapideSVN)
- utilisation possible d'Apache pour le serveur
- gestion des fichiers binaires
- gestion des permissions utilisateurs
- incrémente le numéro de version du projet

Inconvénients

- orienté pour des petits projets

D'autres outils

- VSS (Visual Source Safe) créé par Microsoft

Découverte et manipulation de subversion

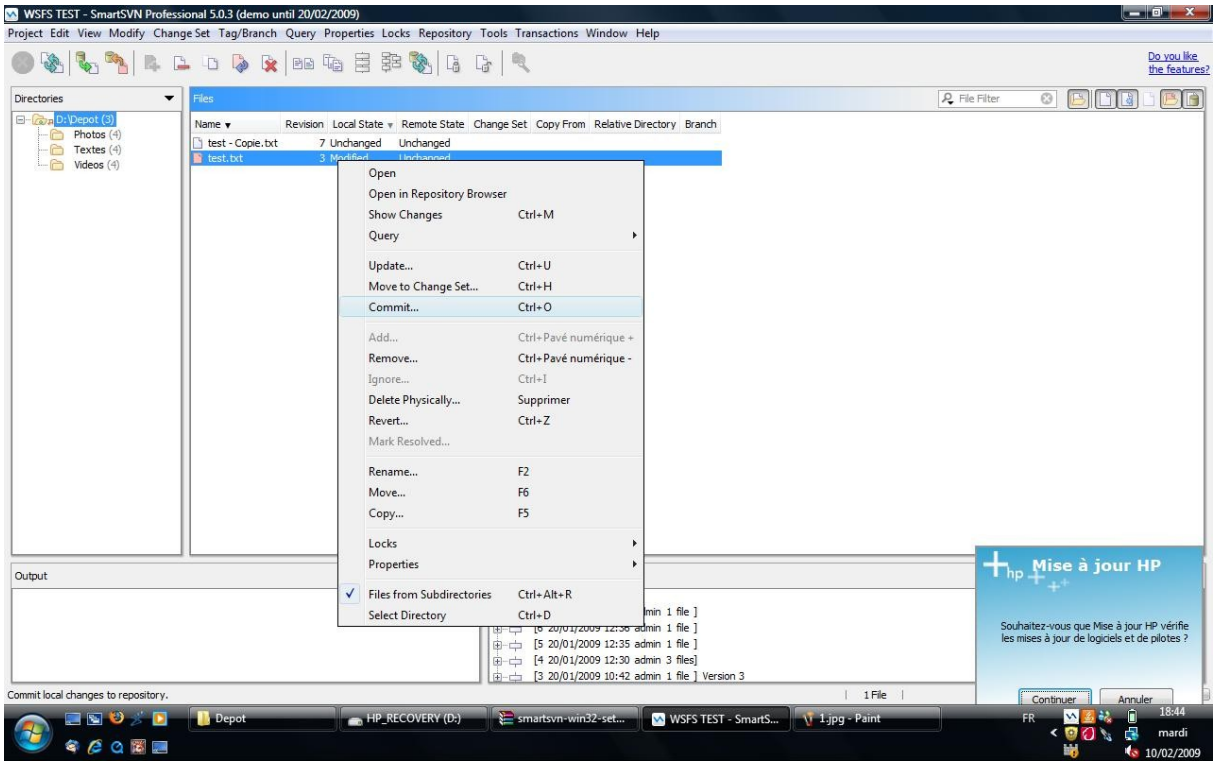
Subversion est un logiciel libre qui nous était inconnu. Nous avons donc décidé de consacrer du temps sur la découverte de son fonctionnement et nous avons découvert son rapprochement avec clearcase.

Nous avons tout de même pris la décision d'installer un serveur local de subversion sur nos postes.

Pour commencer, nous avons cherché des tutoriaux décrivant le processus d'installation. Malgré plusieurs recherches, l'objectif n'a pas été atteint. Nous avons rencontré des problèmes dont nous n'avions pas su résoudre. Cependant, un dépôt a été mis en place sur : svn://silpion.dyndns.org/home/vince/public_html/telw/svn/wsfs

Ensuite, nous avons installé un client afin d'accéder à une interface graphique pour comprendre le fonctionnement. L'accès vers le dépôt mis en place n'a pas fonctionné, mais nous avons finalement réussi à effectuer quelques tests sur un dépôt défini localement.

Modification d'un fichier dans le dépôt :



Lors de cette phase de découverte, nous avons constaté un problème. Nous n'arrivons pas à effectuer des sous-versions. Cette idée de sous-version est temporairement mise à l'écart.

Dans subversion, la notion de révision correspond à la version de la dernière ressource intégrée + 1 dans le dépôt svn. C'est-à-dire que la dernière ressource mise à jour dans le dépôt à forcément la version la plus récente parmi les autres ressources. Exemple : Ressource A version 4 va être mise à jour. Ressource B a une version 9. Après la mise à jour, la ressource A aura une version 10 (Version B + 1).

L'API : entre Netbeans et Subversion

Nous avons trouvé deux APIs réalisées par des particuliers

- la première fournit les méthodes pour se connecter au serveur, mais peu de méthodes pour la gestion des fichiers,
- la seconde fournit plus de méthodes pour la gestion des objets (fichiers, répertoires) et des index, de plus elle prend en compte du nombre d'utilisateur travaillant sur le même fichier.

Les éléments à mettre en place

- Création du Web service.
- Créer une hiérarchie au sein du serveur svn pour ranger les ressources déposées.
- Créer le mapping permettant de classer la ressource dans le serveur svn (par une analyse de l'URI).

Sources

Pourquoi utiliser Subversion ?

http://www.sosign.com/Subversion.html

http://www.travailleursduweb.com/2008/03/10/subversion-un-outil-indispensable.html

Pourquoi utiliser ClearCase ?

ftp://ftp-developpez.com/loic-mathieu/clearcase.pdf

Installation de SubVersion ?

ftp://ftp-developpez.com/hugo/tutoriels/outils/subversion/subversion.pdf

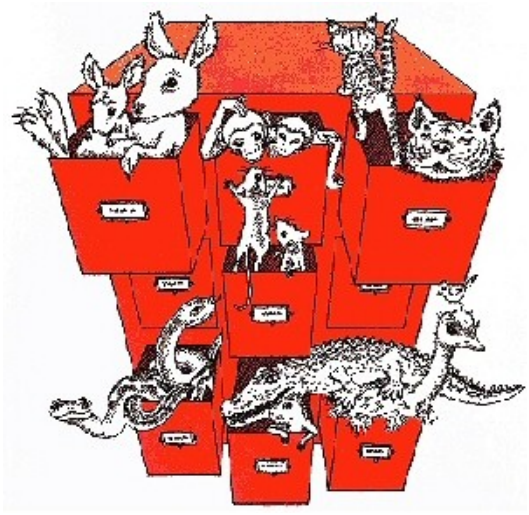
http://iusti.polytech.univ-mrs.fr/~meister/documents/presentation_subversion.pdf

API

http://kickjava.com/src/org/netbeans/modules/subversion/OutputLogger.java.htm

https://forge.process-one.net/browse/~raw,r=139/Erlybird/trunk/erlybird/erlybird-erlang-platform/src/org/netbeans/modules/languages/erlang/platform/index/ErlangIndexer.java

Le classifieur

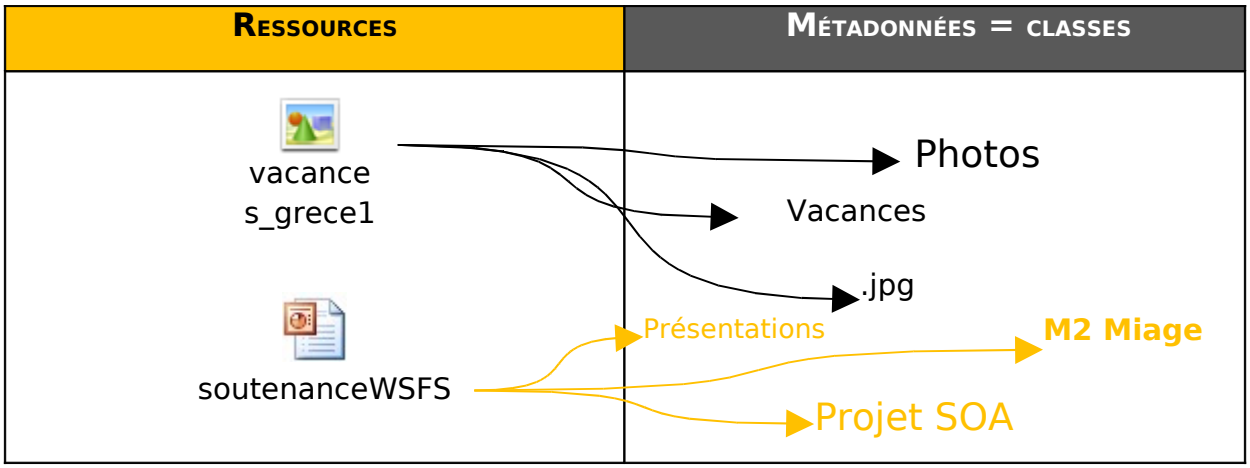


Le classifieur est un service qui classe les ressources en fonction de leurs métadonnées. Ces métadonnées sont des données propres à la ressource qui la caractérisent (par ex : fichier Word, photo, extension du fichier, etc.).

Grâce à un certain nombre de règles, le classifieur range les ressources par catégories, que nous appellerons classes, en comparant leurs métadonnées avec celles des classes.

Une ressource peut être :

- Un fichier,
- L'adresse d'un site,
- L'adresse d'une base de données,
- Le nom d'une machine sur un réseau,
- ...



Des mots clés sont associés à la ressource et ceux-ci correspondent à des classes de l'ontologie. Le travail du classifieur est de lier la ressource aux classes correspondantes à cette ressource à l'aide des mots clés.

Pour cela, il va se fier à une ontologie. Elle va décrire les concepts que le classifieur va utiliser pour classer les ressources.

II.L'ontologie

C.Nepomuk : un projet européen de PC sémantique

Un ordinateur qui indexe les informations textuelles du système à la manière d'un cerveau humain: c'est le principe de Nepomuk, dont l'objectif est de créer des applications open source qui faciliteront la recherche de documents.

Seize entreprises européennes du secteur IT, dont Thales, IBM, SAP ou Mandriva, sont à l'origine du coup d'envoi de ce vaste projet de développement logiciel : Nepomuk[1]. Derrière cet acronyme , se profile un ambitieux projet appuyé par la Commission européenne et doté du budget conséquent de 17 millions d'euros.

Selon Stéphane Lauriere, chargé de projet chez Linux Mandriva France, l'objectif est de développer une série d'applications dotée d'un coeur commun, permettant d'organiser d'une nouvelle manière les informations présentes sur l'ordinateur, des e-mails aux documents Word.

L'ambition est de faciliter la recherche de données sur le système en "allant plus loin" que les outils tels que Google Desktop Search (GDS) ou Spotlight d'Apple. Le coeur des logiciels Nepomuk intégrera un index de toutes les informations textuelles: mails, documents bureautiques, agenda, carnet d'adresses, bookmarks, liste des tâches...

Mais cet index ira au-delà de la prise en compte des mots inscrits dans ces documents, comme c'est le cas des technologies actuelles. Sur le modèle de la déduction humaine, il cherchera les liens existants entre les mots, à commencer par le sens des phrases. D'où l'appellation de projet de "poste de travail sémantique".

Cette appellation nous a interpellés étant donné que nous recherchions s'il n'existait pas une ontologie existante décrivant les ressources d'un ordinateur. C'est donc cet aspect du projet Nepomuk que nous avons épluché, histoire de s'en inspirer pour notre projet.

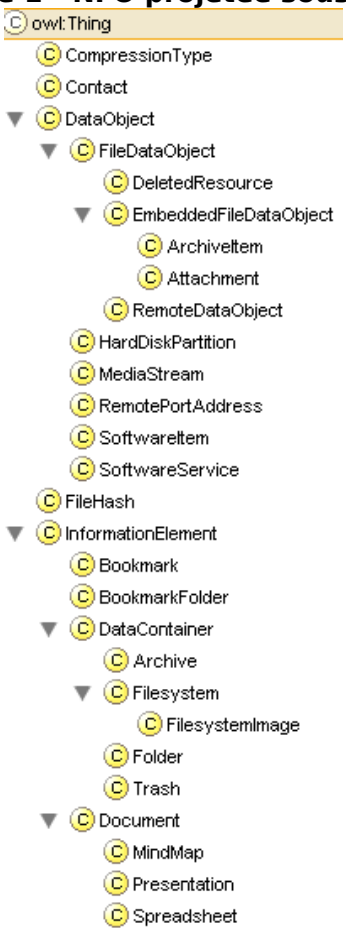
Bien entendu, Nepomuk est un projet de longue haleine s'étalant sur 3 ans et il en va de soi que nous n'avons pas réutilisé toute leur ontologie. La partie qui suit explique pourquoi.

[1] Nepomuk signifie: Networked environment for personalized ontology-based management of unified knowledge

D.Création d'une ontologie spécifique au projet WSFS

L'ontologie à laquelle nous nous sommes intéressées est Nepomuk File Ontology (NFO). Elle fait partie d'une série de sept autres ontologies. La particularité de NFO est qu'elle fournit le vocabulaire exprimant les informations issues de différentes sources.

Figure 1 - NFO projetée sous Protégé



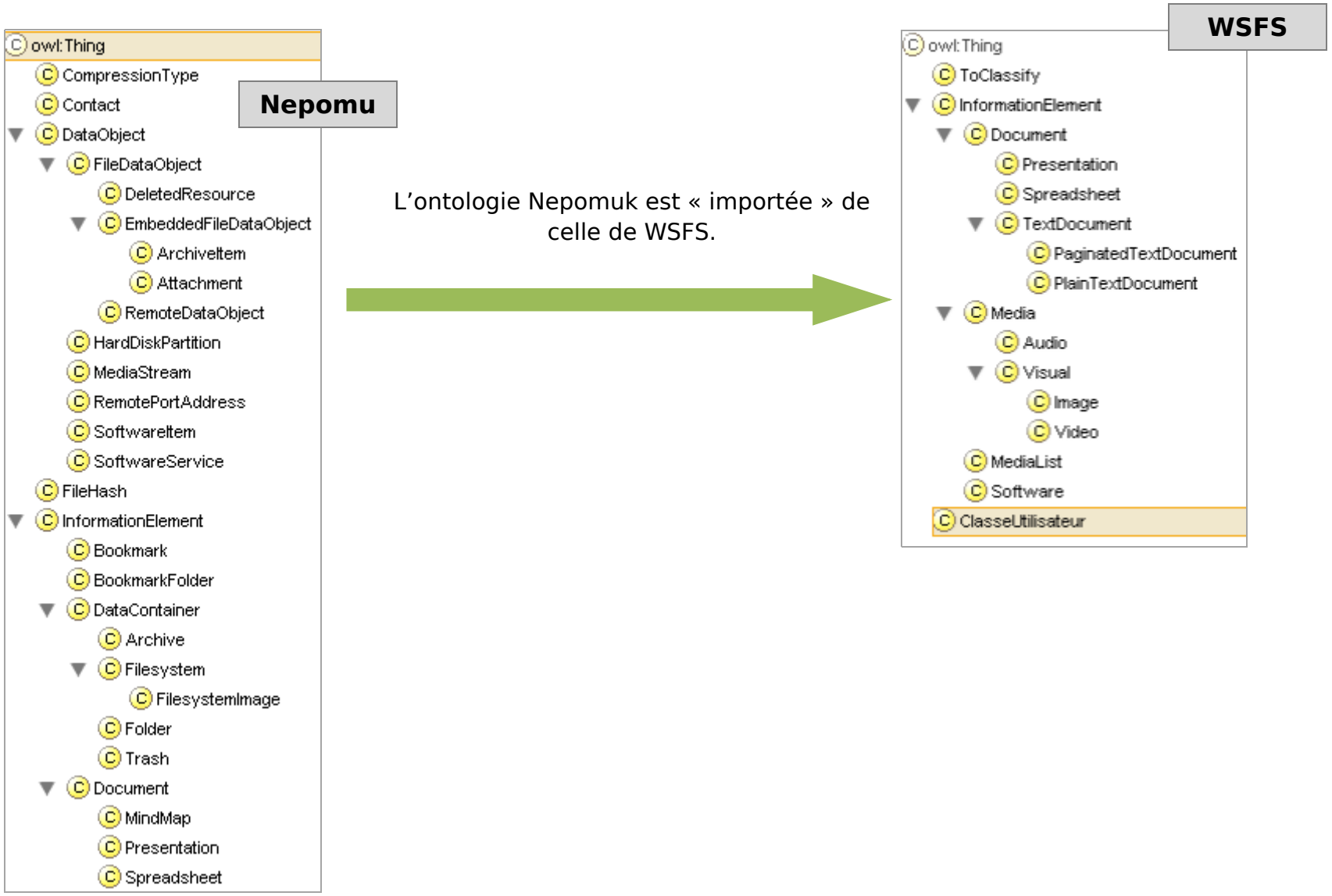
Manifestement les informaticiens qui ont produit ce travail ont un usage informatique plus que sémantique des espaces de noms, puisqu'ils s'en servent de conteneurs en lieu et place de champs sémantiques.

De plus, nous sommes assez critiques quant à son approche ontologique : elle nous semble tomber exactement dans le travers dans lequel tombent - après quelques philosophes Grecs - bien des informaticiens, consiste à vouloir catégoriser a priori toutes choses.

En ce qui nous concerne, pour être effectivement intéressant WSFS doit laisser aussi libres les utilisateurs qu'ils le sont actuellement lorsqu'ils créent et nomment leurs répertoires. En quelque sorte nous souhaitions que l'ontologie de

WSFS soit vivante, évolutive dans ses classes et ses propriétés comme dans son peuplement.

L'approche du projet Nepomuk construit une infrastructure, dont la sémantique semble davantage destinée au système d'exploitation sous jacent qu'à l'utilisateur. Elle semble, selon nous, nécessaire pour le premier aspect, insuffisante pour l'autre. Elle peut donc être exploitée comme une ressource importée de notre ontologie.


Nepomu
owl:Thing
CompressionType
Contact
DataObject
FileDataObject
DeletedResource
EmbeddedFileDataObject
ArchiveItem
Attachment
RemoteDataObject
HardDiskPartition
MediaStream
RemotePortAddress
SoftwareItem
SoftwareService
FileHash
InformationElement
Bookmark
BookmarkFolder
DataContainer
Archive
Filesystem
FilesystemImage
Folder
Trash
Document
MindMap
Presentation
Spreadsheet
L'ontologie Nepomuk est « importée » de celle de WSFS.
WSFS
owl:Thing
ToClassify
InformationElement
Document
Presentation
Spreadsheet
TextDocument
PaginatedTextDocument
PlainTextDocument
Media
Audio
Visual
Image
Video
MediaList
Software
ClasseUtilisateur

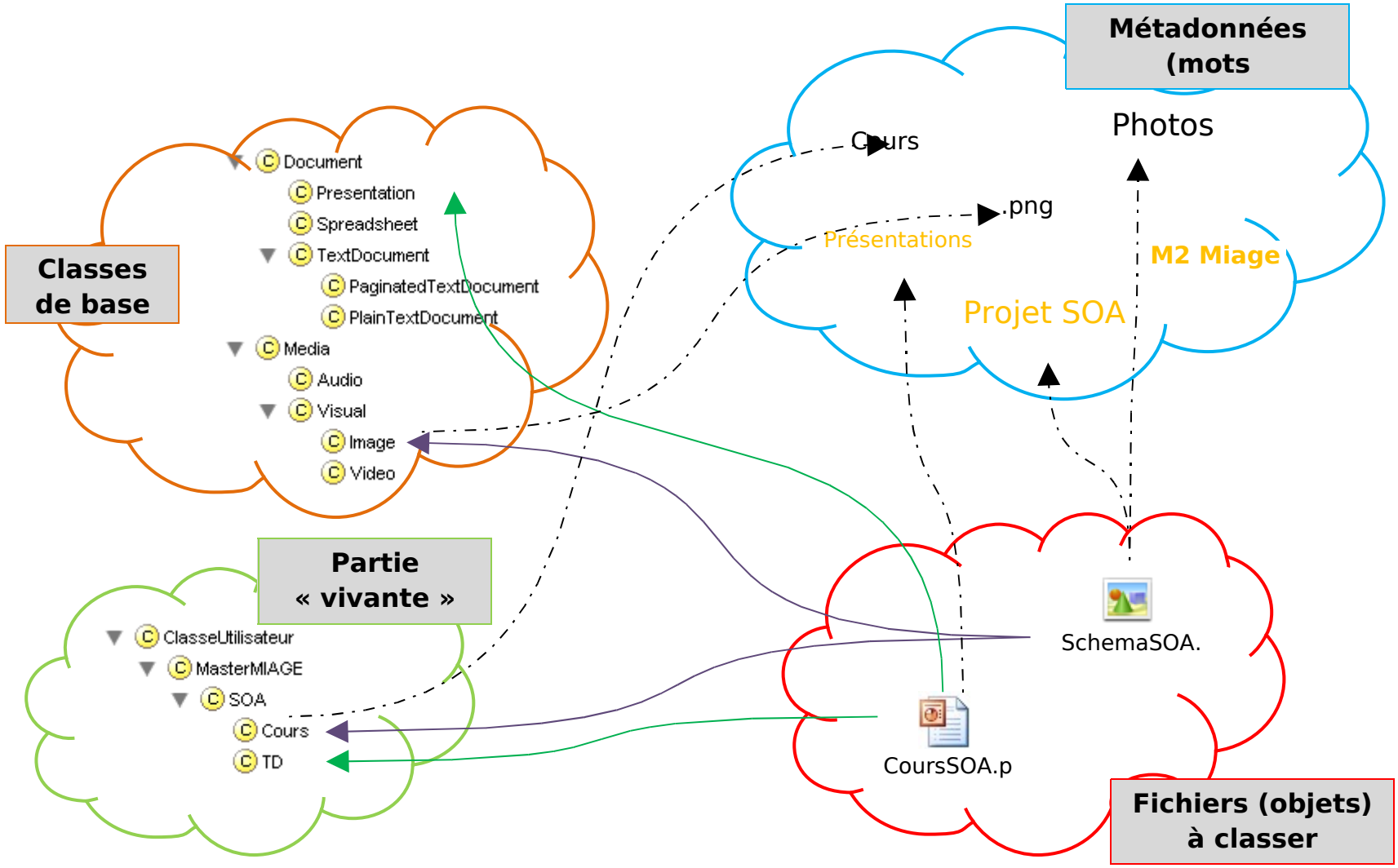
Notre ontologie comporte quatre classes principales :

- *InformationElement* : C'est que nous appelons la partie "fixe" de notre ontologie. Cette classe décrit les éléments de base du système comme les documents textes paginés, non paginés, les images, les fichiers média, etc. Cette partie a été inspirée par l'ontologie NFO,

- *ClasseUtilisateur* : C'est la partie "vivante" de l'ontologie. Elle permet de faire des croisements entre les classes de bases et celles créent par l'utilisateur (que nous nommerons classes utilisateurs). Par exemple, l'utilisateur décide de créer un dossier (une classe) « Master MIAGE » dans lequel il souhaite classer tous ces cours. Il y aura à la fois des documents Word, PDF, des présentations PowerPoint, des images, etc. Il y a donc des objets (fichiers) qui sont classés dans les classes de bases telles que *PaginatedTextDocument* mais aussi dans les classes utilisateurs telles que CoursSOA. D'où la classe *ClasseUtilisateur* que nous avons insérée dans notre ontologie. Cette classe comportera toutes les classes créées par l'utilisateur,

- *ToClassify* : C'est la classe où sont stockées toutes les ressources que le classifieur doit classer,

- *Metadonnees* : Classe qui répertorie toutes les caractéristiques, ou tous les mots clés, qui différencient les classes des unes des autres. Nous vous expliquerons dans le mécanisme du classifieur, un peu plus loin, pourquoi avons-nous eu besoin de cette classe et comment est-ce qu'elle fonctionne.

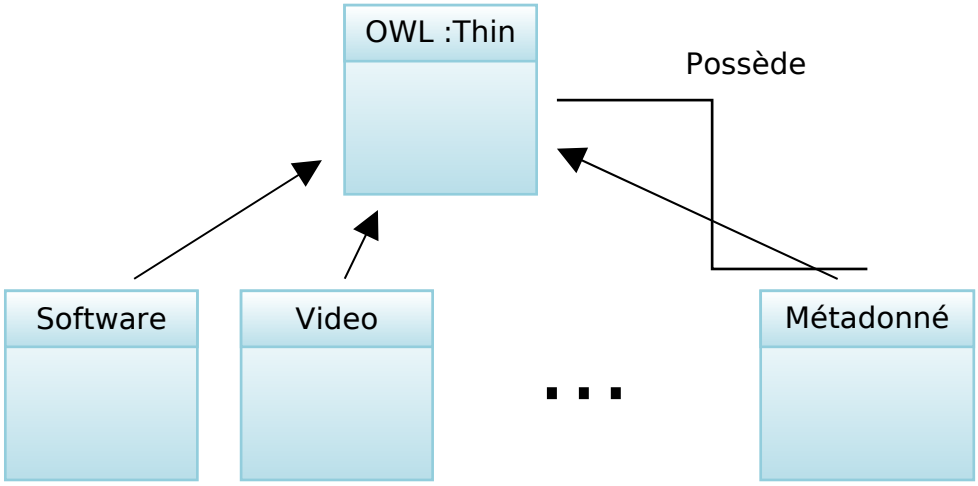

Métadonnées (mots
Photos
Cours
.png
Présentations
M2 Miage
Projet SOA
Classes de base
Document
Presentation
Spreadsheet
TextDocument
PaginatedTextDocument
PlainTextDocument
Media
Audio
Visual
Image
Video
Partie « vivante »
ClasseUtilisateur
MasterMIAGE
SOA
Cours
TD
SchemaSOA.
CoursSOA.p
Fichiers (objets) à classer

Le fonctionnement du « classifier »

E.La récupération des métadonnées

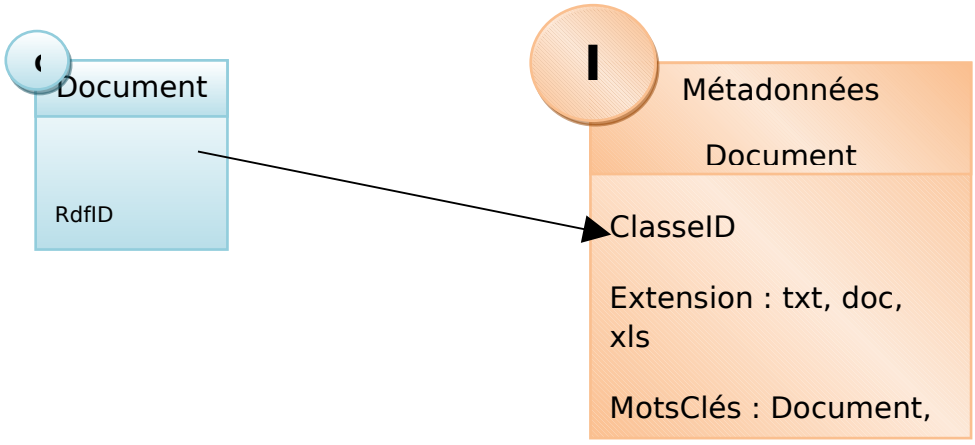
Lors de la création d'une nouvelle ressource, l'utilisateur saisie les métadonnées correspondantes (Nom, Description, Date, etc.). Afin de classer cette nouvelle ressource, nous avons trouvé une astuce permettant de comparer les métadonnées de celle-ci avec les métadonnées « stockées » dans l'ontologie. Seulement, les métadonnées stockées dans l'ontologie doivent être liées aux classes de l'ontologie de façon à pouvoir effectuer le classement de nouvelles ressources, comme dans le diagramme suivant :

Figure 2 : Diagramme UML Métadonnées



La solution consiste donc à utiliser une classe *Metadonnees*. Elle permet de stocker les métadonnées des ressources. Cette classe possède des propriétés qui correspondent aux diverses métadonnées qui sont entrées par l'utilisateur. Ensuite, chaque classe de l'ontologie possède un individu. Ce dernier contient, en statique, les différentes métadonnées de cette classe, comme le montre l'exemple suivant :

Figure 3 : Exemple Métadonnées



L'individu est lié à la classe par le *RdfID* de la classe qu'il stocke en statique dans une variable *ClasseID*. Cela permet de classer les ressources selon leurs métadonnées.

F.La classification

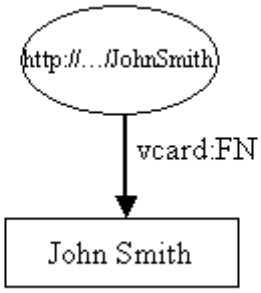
Une fois les métadonnées récupérées, il convient de classer la ressource. Cette ressource peut être liée à plusieurs classes à la fois, ainsi, un document Excel pourra se trouver lié à la classe *Document*, *Spreadsheet* et *ClasseUtilisateur*. Certains de ces liens auront donc été créés automatiquement et d'autres manuellement par l'utilisateur lui-même. En effet, une fois les métadonnées comparées, une liste des classes correspondantes aux métadonnées est affichée et l'utilisateur choisit la ou les classe(s) qu'il veut faire correspondre à cet individu (Ici, la classe document et la classe *ClasseUtilisateur* pour un fichier Excel par exemple). Si l'utilisateur a choisi la classe *Document*, la ressource est automatiquement liée aux classes parentes de la classe *Document*.

La persistance des données

G.Qu'est-ce que Jena ?

Jena est un API Java qui permet de créer et de manipuler des ontologies, donc des structures RDF (Resource Description Framework). RDF est une recommandation du W3C qui permet de décrire des ressources. Qu'est-ce qu'une ressource ? Cette question fait encore l'objet de nombreux débats mais pour simplifier, une ressource est finalement tout ce qui peut être identifié.

Figure 4 - Un exemple pour comprendre RDF



Si on prend comme exemple la figure ci-dessus, John Smith est notre ressource. Elle est représenté par une elipse dont l'identifiant est une URI (Uniform Resource Identifier) "http://.../JohnSmith".

Les ressources ont des propriétés. Dans notre exemple, nous allons nous intéresser aux propriétés que peut avoir la carte de visite de John Smith. Figure 4 ne montre qu'une seule propriété, le nom complet de John Smith. Une propriété est représentée par un arc, labélisée avec le nom de cette propriété. Le nom d'une propriété est aussi une URI. Le diagramme ci-dessus représente cette propriété sous forme XML appelée "qname". La partie précédent le ':' est appelée le préfixe (namespace prefix) et représente un espace de nommage (namespace). La partie suivant le ':' est appelée 'local name' et représente le nom de la propriété dans l'espace de nommage. Les propriétés sont généralement représentées sous forme "qname" pour faciliter leurs transpositions en RDF. C'est aussi une manière simple et claire de les représenter dans les diagrammes et les textes. A chaque propriété, lui est associée une valeur. Cette valeur est nommée "littéral". Les littéraux sont tout simplement des chaînes de caractères et représentés par des rectangles dans les diagrammes.

Jena va nous permettre de créer et de manipuler des graphes RDF comme celui que nous venons de présenter. Jena possède des objets (classes) pour représenter des graphes, des ressources, des propriétés et des littéraux. Les interfaces permettant cette représentation sont respectivement nommées Resource, Property et Literal. Dans Jena, un graphe est appelé un modèle et est représenté par l'interface Model.

H.Utilisation de Jena

Voici les étapes qui nous ont été nécessaires afin de faire fonctionner Jena avec notre ontologie :

- Etape 1 : Créer une instance du driver MySQL afin de pouvoir se connecter à notre base Jena
- Etape 2 : Créer une connexion à la base Jena à l'aide de cette instance
- Etape 3 : Créer un « ModelMaker » qui va nous permettre de gérer les modèles

- Etape 4 : Créer ou ouvrir un modele

Voici le code de ces différentes étapes :

`String className ="com.mysql.jdbc.Driver";//path of driver class`
`Class.forName (className);  //Load the Driver`
	ETAPE 1

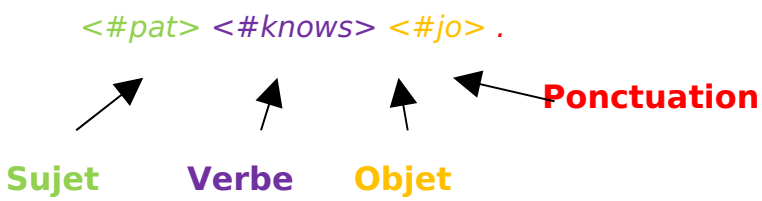
`String DB_URL = "jdbc:mysql://localhost/test";// URL of database`
`String DB_USER =  "????";  // database user id`
`String DB_PASSWD = "????";  // database password`
`String DB =  "MySQL";  // database type`
	INSTANCIATION DES VARIABLES

`IDBConnection conn = new DBConnection ( DB_URL, DB_USER, DB_PASSWD, DB );`	**ETAPE 2**
`ModelMaker maker = ModelFactory.createModelRDBMaker(conn) ;`	**ETAPE 3**
`Model model = maker.createDefaultModel();`	**ETAPE 4**

Apprentissage et inférence

Afin de faire vivre notre ontologie, nous pensions nous appuyer sur le langage N3 ou Notation 3. Il s'agit d'une alternative compacte et claire à la syntaxe XML de RDF ayant par ailleurs pour but d'en élargir les possibilités d'expressions.

En N3, on peut écrire un « triple » de cette façon :



Chacun des éléments, qu'il soit sujet verbe ou encore objet, est identifié par une URI (ex : <http://www.w3.org/> or http://www.w3.org/2000/10/swap/test/s1.n3#includes). Il existe tout de même une exception : l'objet, peut être un littéral (un mot, un chiffre, etc.) :

```
<#pat> <#knows> <#jo> .

<#pat> <#age> 24 .
```

Le verbe (ici, « knows ») est pour notre ontologie, une propriété.

Pour le projet WSFS, l'utilisation du langage N3 permet d'exprimer les règles comme :

- Si l'extension de la ressource est .doc => classe = Document,
- Si la ressource a comme métadonnées Photo => classe = Media,
- Etc.

Grâce à cet ensemble de règle, le classifieur est capable de classer de façon plus efficace les ressources. Jena est capable de prendre en entrée des fichiers N3 et de les interpréter.