Implémentation d'un filtrage d'accès innovant sur un triple-store RDF


Dossier POST Sprint 1

Encadrants:

- E. Coquery
- R. Thion
- T. Sayah

Auteurs:

- Ben Hmiden Nihel
- Cazenave-Lévêque Raphaël
- Hdidar Rami
- Sapunaru Patricia

Projet:

`svn+ssh://scm.gforge.liris.cnrs.fr/svnroot/acrdf/branches/acrdf-fuseki-ti5/`

Lundi 2 novembre 2015

Travail effectué :

Découverte et compréhension du projet

Lors de nombreuses réunions avec T.Sayah, nous avons découvert le contexte du projet, et obtenu des explications sur l'implémentation existante.

Provenant d'horizon différents, il nous est paru évident qu'une rapide remise à niveau au sujet de RDF, SPARQL, était nécessaire. Aussi, nous n'étions pas familiers avec le framework Apache Jena ainsi qu'avec son serveur et endpoint, Fuseki.

Préparation de l'environnement de travail

Lorsque nous avons commencé à souhaiter utiliser le code existant pour tester la version modifiée de fuseki-server, nous avons été confronté à quelques problèmes liés à nos différents systèmes d'exploitations, et aux configurations de MySQL du pilote jdbc qui étaient présentes dans l'implémentation actuelle. Cette étape, bien utile pour nous pour nous approprier le projet, n'était pourtant pas strictement nécessaire, comme T.Sayah nous l'a expliqué.

Souhaitant disposer d'étapes reproductibles de "build" sur toutes nos machines, nous avons retravaillé le descripteur maven, la génération de code Java avec JavaCC, et écrit un document qui nous a servi de tutorial (document "réunion-19-10-2015" <https://docs.google.com/document/d/1-qL2Ri5z8IndIaVy6ktwh185cetXv-qkBz3AOUY1kDI> sur Google Drive).

Mise en place d'une API REST en Spring

Notre objectif était d'établir une représentation coté serveur des ressources nécessaires à l'administration.

Nous avons commencé par exposer les opérations CRUD sur "RawTDB" et "Policy" afin de pouvoir les valider, y avoir accès via des URL, et par la suite les persister en base de données.

Puis nous nous somme formés pour utiliser les parseurs de Policy de T.Sayah, E.Coquery et l'API de manipulation de graphe fournie par Jena.

La personne affectée à la conception de cette API étant plus familière des environnement .NET, à donc du prendre un petit temps d'adaptation avec l'environnement Java et Spring pour mettre en oeuvre cette architecture. Afin de faciliter cette démarche, nous avons utilisé le projet Spring Boot qui permet de s'abstraire de la configuration initiale du projet. Hélas au moment où nous avons déployé l'artefact en tant que WAR sur le serveur d'application GlassFish 4.1 intégré à Netbeans 8.0.1 (dernière release stable), nous nous sommes rendu compte qu'un bug <https://java.net/jira/browse/GLASSFISH-21265> de GlassFish empêchait le déployement. Donc nous avons du utiliser une version plus récente de GlassFish (la

dernière release stable, 4.1.1), qui ne n'intégrait pas nativement dans Netbeans.

Étant donné que nous travaillions de concert sur la partie backend et frontend, que nos besoin étaient plus du prototypage que de l'implémentation définitive, et que nous n'avions que peu de code métier, nous avons fait le choix d'ignorer temporairement les "Services" et "Repositories" Spring, et de n'utiliser que des Contrôleurs Spring en stockant les représentations des entités en mémoire dans des collections Java.

Pour résumer, notre implémentation se compose de trois Contrôleurs Spring:

- Ressource "RawTDB"
- Ressource "Policy"
- Ressource "NewTDB" , disposant d'une opération de création d'une TDB protégée, qui est utilisable par la version modifiée de fuseki-server

Réalisation d'un "frontend" administrateur

Nous avons commencé par créer une interface minimale pour l'administrateur, pour saisir les autorisations et le graphe à protéger. Nous avons choisi le framework MV* Backbone.js enrichi avec Marionettejs afin d'utiliser les mêmes technologies que celles du frontend de Fuseki. Aussi parce que ces deux frameworks, lorsqu'ils sont utilisés conformément aux bonnes pratiques (routage, communications, templating, arborescence de vues), et associés à système de module comme AMD, permettent un très bon découplage de la logiques métier et des problématiques d'interface homme-machine. Ainsi le surplus de temps que nous avons passé à travailler en ce sens nous permettra d'anticiper l'ajout de fonctionnalités.

L'objectif indiqué par T.Sayah était d'offrir une assistance à l'écriture des autorisations, des règles d'inférences, et des graphes qui seront utilisés pour la construction de la TDB protégée. Il nous était demandé aussi de pouvoir saisir les utilisateurs, associer les utilisateurs avec un rôle, et associer les rôles avec les niveaux d'autorisation. Étant donné le manque de temps, et le soin apporté sur l'architecture, nous avons reporté ces tâches au prochain sprint. L'interface facilitée pour l'écriture des autorisations et des graphes pourra être prototypée avant le Sprint 2 afin d'éviter de nouveaux imprévus.

Entre le Sprint 1 et le Sprint 2

- Construire un sous-projet/module maven nommé "model" qui servira à mutualiser toutes les classes entités.
- Etablir et valider un schema relationnel via JPA
- Vérifier les dépendances sur Spring dans Jena/Fuseki, si elles existent déjà, on s'autorisera l'utilisation de Repositories Spring comme DAO dans le module "model", sinon on codera des DAO simples sans Spring.
- On devra utiliser des DTO pour éviter de polluer les modèles avec du code code métier.
- Recherche d'une méthode efficace pour automatiser le redéploiement de la partie frontend sur GlassFish, ou utiliser un wrapper grunt autoreload via maven,

ou gérer le CORS sur le backend. Le but étant de rapidement propager les modifications du frontend dans le navigateur pour faciliter la visualisation des modifications apportées.

Prochain Sprint

- Interface assistée pour la saisie
  - des autorisations
  - des règles d'inférences
  - des graphes
  - des utilisateurs
  - des rôles
- Rendre l'interface plus attractive au niveau design (Bootstrap ou SemanticUI)
- Modulariser la partie backend selon les principes de Spring
- Permettre le lancement du fuseki-server modifié depuis le backend, le backend offrira une route pour que le frontend puisse en faire la requête
- Rendre plus flexible/dynamiser la configuration de fuseki-server, depuis le backend (programmatiquement, ou via des variables d'environnement)
- Gérer la suppression d'une règle (voir la faisabilité)