FROM RAW PUBLISHED DATA TO INTERLINKED SEMANTIC DATA

Authors	Laurent BIHANIC & Gabriel KEPEKLIAN
Reviewer	
Title	Architecture and integration
Date	29 mai 2012
Reference	D5.2
Version	1.0
Destination	Consortium members only

Projet DataLift

De la donnée brute publiée à la donnée sémantique interconnectée

Appel ANR CONTINT 2010 ANR-10-CORD-009

RAPPORT DE RECHERCHE

Sommaire

Informations sur le document

Projet ANR	ANR Contint - ANR-10-CORD-009
Nom du projet	Datalift
Nom complet	Un ascenseur pour les données
URL du projet	www.datalift.org

Lot	**Numéro**	WP5	**Titre**	Architecture and integration
Tâche	**Numéro**	T5.1	**Titre**	Datalift Platform architecture
Délivrable	**Numéro**	D5.2	**Titre**	Datalift Integration and modularity

Date de livraison	Contractuelle	M12 ~~M31~~	Actuelle	29.05.2012
Statut	~~Préliminaire~~ Final			
Nature	~~Prototype~~ Rapport ~~Dissémination~~			
Niveau de dissémination	Public Interne au consortium			

Rédacteurs	Laurent Bihanic, Gabriel Képéklian (Atos)		
Resp. de la rédaction	Nom	Partenaire	Mail
	Laurent Bihanic	AtoS	laurent.bihanic@atos.net

Résumé	Ce document est décrit l'intégration des modules de la plateforme Datalift.
Mots clés	Module, Modularité, Intégration, Architecture, Plateforme

Versions		
Date	Rév.	Auteur
06/06/2012	1.0	Laurent Bihanic & Gabriel Képéklian

1. Introduction

Le présent document constitue le livrable D5.2 de la tâche T5.1 du lot 5 (WP5).

Sous le titre « Plateform architecture », cette tâche est consacrée à la conception de l'architecture de la plate-forme et à son développement. Cette architecture doit intégrer les différents outils développés dans le projet ; par outil, on entend des modules implémentant des fonctions essentielles et « visibles » de Datalift comme par exemple la conversion d'un document XML en son équivalent RDF ou le catalogue d'ontologies qui sert à la sélection d'ontologie, mais un module peut aussi être un composant nécessaire à plusieurs fonctions.

L'architecture sous-tend le processus global d'élévation de données comprenant un certain nombre d'étapes à exécuter en vue de promouvoir des données brutes au rang de données liftées. Ces dernières sont délivrées valides, un dispositif de validation est donc présent dans Datalift et constitue module. Le processus de validation assure l'intégrité, le respect de la licence, et la disponibilité des données liées.

Par intégration, il faut comprendre que Datalift doit pour s'intégrer dans un système d'information à la façon d'un composant, en l'occurrence d'élévation de données, ou que Datalift doit pouvoir être intégrer, c'est-à-dire que tout ou partie de ses modules doivent pouvoir être intégrés, ou que Datalift doit pouvoir intégrer de nouveaux modules.

Le développement d'une interface de programmation au-dessus de la plate-forme DataLift donne un accès unifié à certains modules de plate-forme. L'API DataLift permet ainsi d'inclure le processus d'élévation dans les outils externes. Il permet également de développer de nouveaux modules qui étendent ou améliorent le processus de publication. Par exemple, il permet d'ajouter le support d'un nouveau type de source de données ou de développer une nouvelle interface graphique au-dessus du module de sélection d'ontologies.

2. Principes d'intégration d'un module générique

Les principes d'intégration, qui sont l'objet du présent chapitre, constituent à minima des bonnes pratiques. Certains sont impératifs et d'autres des recommandations.

Quelques définitions préalables

Définition d'un module

Un module, en programmation, désigne un espace de noms. Chaque module peut exporter ou importer certains symboles comme des variables, des fonctions ou des classes. Les modules peuvent se regrouper en package éventuellement hiérarchique.

Nommage des modules

Le nommage des sources techniques peut être du nommage « marketing » des modules. Dans ce document, il n'est question que du nommage technique. En effet, pour des raisons « marketing », il peut arriver que sous un même nom « marketing » on retrouve plusieurs modules techniques.

Principes impératifs

Versionnement (impératif)

Tout module doit être dans un état de version clairement défini. La règle en matière Open source est de distinguer à minima cinq niveaux de version : prototype, alpha, béta (publique ou privée), release candidate (RC) et version stable.

Il faut garder à l'esprit que c'est le niveau du module intégré le plus bas qui détermine le niveau de l'ensemble.

⇨ **Attention** : il reste à définir comment identifier les versions.

Séparation (impératif)

Chaque module doit pouvoir s'exécuter avec ses propres dépendances (bibliothèques dans des versions propres qui ne doivent pas interférer avec celles utilisées par d'autres modules). Chaque module s'exécute dans un environnement clos (classloader).

Ainsi les échanges entre les modules sont limités à ceux qui s'opèrent par le poste client (appel de Web Services) et/ou le triplestore (échange de données RDF).

Environnement d'implémentation (impératif)

Le développement des modules doit s'appuyer sur les règles d'architecture définies dans le dossier d'architecture de Datalift (livrable D5.1) et les bibliothèques Datalift.

Chargement (impératif)

Les modules développés sont chargés dynamiquement par la plateforme. Pour cela leur artefact de déploiement doit respecter la structure attendue par la plateforme. Cette structure est décrite ci-après.

Open source (impératif)

La plateforme Datalift est Open source. Les modules qui lui sont ajoutés, doivent se conformer à des règles précises pour que le caractère Open source de la plateforme ne soit pas compromis.

Licence (impératif)

La licence choisie pour la plateforme Datalift autorise le développement de modules privés, non open source. Elle autorise aussi la création de versions modifiées de la plateforme ou de son framework sans obligation de reversement à la communauté. De même, certains modules pourraient ainsi être à usage limité, par exemple s'ils s'appuient sur des logiciels tiers commerciaux.

Documentabilité (impératif)

Le développement des modules doit respecter l'état de l'art en matière de documentation du code en fonction du langage d'implémentation (commentaires Javadoc, Scaladoc …)

Testabilité (impératif)

Les risques liés au développement de modules standardisés sont limités par rapport aux développements spécifiques. L'intégration est facilitée par des tests d'intégration prêts à l'emploi. Le module est livré avec ses tests.

Test d'intégration (impératif)

Il est impossible de réaliser des tests d'intégration avec l'ensemble des modules potentiellement intégrés. Aussi, chaque module doit embarquer et gérer ses procédures de test (unitaire, non-régression, intégration, robustesse …)

Comportements prédéfinis (impératifs)

Les modules de Datalift doivent posséder quelques comportements prédéfinis pour permettre à la plateforme d'agir sur tous ses modules. Pour les modules préexistants, il suffit de les encapsuler dans une surcouche qui implémente ces comportements. Les comportements prédéfinis sont les principes impératifs énumérés dans ce chapitre.

Etapes de projet

Chaque module de workspace est défini au niveau d'un unique étage de Datalift.

Principes recommandés

Vocation (recommandation)

Il est recommandé que chaque module applicatif soit défini au niveau d'un unique étage Datalift.

3. Modalité d'intégration des modules existants

Il y a deux types de modules dans Datalift. Les premiers sont des modules préexistants que Datalift intègre par nécessité. Les seconds sont des modules développés dans le cadre du projet et destinés à augmenter les fonctionnalités de la plateforme.

Chaque module intégré à la plateforme en constitue une extension. Aucune ne doit être indispensable. Toutefois, certaines fonctionnalités communes étant requises, des modules les implémentant sont requis ; la mise en œuvre de ces fonctionnalités sous forme de modules répond aux objectifs de modularité de la plate-forme et offre la possibilité de remplacer les implémentations par défaut par d'autres qui répondent à des besoins ou des contraintes particulières.

Le modèle d'élévation de données proposé par la plate-forme Datalift est formé de 5 étages distincts : sélection, conversion, publication, interconnexion, exploitation. Tout module est défini pour correspondre à un étage et un seul.

Points d'intégration

Le workspace

Le workspace est le module dans lequel sont implémentées les fonctions d'élévation de données du framework. Il est intégré à la plateforme Datalift sous la forme d'une IHM HTML utilisant les services web des modules de conversion et d'interconnexion.

Il est possible d'intégrer un module externe au sein du processus d'élévation de données un implémentant l'interface *ProjectModule* du framework Datalift qui permet d'interagir avec le cycle de vie des projets d'élévation de données et d'intégrer l'IHM des modules au sein du workspace.

Le SPARQL endpoint

La plate-forme Datalift requiert la présence d'un SPARQL endpoint implémentant l'accès HTTP aux RDF stores et en fournit une implémentation par défaut. Déployée en tant que module, celle-ci peut être étendue ou remplacée si besoin.

Par rapport aux SPARQL endpoints natifs des triplestores RDF, le SPARQL endpoint de Datalift offre les fonctionnalités suivantes :

- Interface publique limitée au requêtage des données : les accès en modification (SPARQL 1.1 Update) sont bloqués, même pour les utilisateurs authentifiés. Les modifications de données sont autorisées aux seuls modules applicatifs.

- Routage entre les différents RDF stores configurés dans la plate-forme. Pour les utilisateurs anonymes (non authentifiés), seuls les RDF stores marqués comme publics dans la configuration sont accessibles.
- Support du protocole JSONP (http://en.wikipedia.org/wiki/JSONP) pour permettre l'interrogation SPARQL à partir de pages chargées hors du domaine Datalift (mashups).

Une couche d'abstraction a été introduite dans l'architecture de la plate-forme pour permettre l'utilisation de plusieurs triple stores RDF (voir ci-dessous), sur laquelle s'appuie le SPARQL endpoint pour exécuter les requêtes.

Les connecteurs de RDF stores

Une couche d'abstraction a été introduite dans l'architecture de la plate-forme pour permettre l'utilisation de plusieurs triple stores RDF.

Actuellement, les triple stores supportés sont :

Nom	Version	URL	Licence(s)
Sesame	2.6+	http://www.openrdf.org/	Licence BSD-3clause
OWLIM	4.3+	http://www.ontotext.com/owlim	Version light : LGPL
AllegroGraph	4.x	http://www.franz.com/agraph/allegrograph/	propriétaire
Virtuoso	6.1+	https://github.com/openlink/virtuoso-opensource	GPLv2 et propriétaire

Nouveaux modules

Pour que l'intégration des nouveaux modules se réalisent conformément au cadre de l'architecture de Datalift, leur développement doit respecter les principes décrits plus haut dans ce document.

Signalétique de module

Les caractéristiques d'un module sont :

- Etage(s) Datalift d'appartenance
- Nom et descriptif
- Version, sous version

- Cartouche habituel (auteur, compagnie/organisation, dates création et dernière modification)
- Documentation du code (JavaDoc)

Cas d'un module applicatif

L'interface *Module* définie par le framework Datalift permet la réalisation de modules applicatifs, i.e. de modules hébergés et intégrés à la plate-forme Datalift et offrant des services complémentaires, non prévus par celle-ci.

Ces services sont rendus disponibles sous forme de ressources JAX-RS, i.e. sous la forme de web services REST (*REpresentational State Transfer*).

Les modules applicatifs, étant intégrés à la plate-forme, ont accès à tous les services Datalift :

- Configuration
- Accès en modification aux triple stores RDF
- reroutage de requêtes vers le SPARQL endpoint pour affichage direct des résultats dans le navigateur de l'utilisateur
- Accès au stockage de fichiers privé

Pour la réalisation d'IHM web, la plate-forme Datalift inclut certaines facilités (moteur de templates apache Velocity, librairie Javascript JQuery) mais chaque module est libre d'utiliser les technologies de son choix (e.g. pages JSP, GWT...).

Le workspace, composant en charge de la gestion des projets d'élévation de données, propose une extension de l'interface *Module* : *ProjectModule*. Cette interface permet de réaliser des modules interagissant avec les projets d'élévation de données. En fonction de l'étape à laquelle se situe un projet, ces modules peuvent apparaître, intégrés à l'IHM du workspace. A partir de là il peuvent proposer leur propre IHM.

Cas d'un module d'extension du framework

La plate-forme Datalift offre plusieurs possibilités d'extensions du framework :

- Les connecteurs de RDF stores : RepositoryFactory
- Les politiques d'URI : UriPolicy

Les connecteurs de RDF stores permettent d'intégrer l'accès à différents types de triple stores RDF à la plate-forme Datalift. Celle-ci intègre de base un connecteur pour les triple stores supportant nativement l'interface OpenRDF Sesame 2 (comme Sesame 2.6 lui-même ou OWLIM).

Deux connecteurs optionnels sont aussi mis à disposition pour AllegroGraph et Virtuoso. Ces deux triple stores, bien que supportant l'API Sesame 2, requièrent l'utilisation de classes spécifiques pour la connexion avec le triple store.

Les politiques d'URI autorisent la mise en œuvre de politiques de résolution d'URI spécifiques. De telles politiques permettent :

- L'utilisation d'URL différentes pour les objets RDF et leurs représentations (RDF, HTML), par exemple en faisant varier un des éléments constitutifs de l'URL : nom hôte, chemin, extension
- L'adaptation des données retournées : ajout de triples complémentaires (droits d'utilisation, licence) ou filtrage des données en fonction du profil de l'utilisateur connecté.

Intégration de code existant

Un code existant est un code applicatif qui n'a au départ pas été conçu pour Datalift. Son intégration peut donc être potentiellement plus difficile que pour un nouveau module réalisé selon les principes architecturaux de Datalift. Mais il n'est pas envisageable de modifier la plateforme Datalift pour intégrer du code existant.

Datalift étant basé sur la plate-forme Java, l'intégration de code préexistant, quel que soit son implémentation, peut nécessiter l'implémentation d'une ou plusieurs interfaces Java définies par le framework Datalift. A partir de ces implémentations, toutes les possibilités de la plate-forme Java sont disponibles pour accéder aux fonctionnalités du module préexistant : appel de code Java ou natif (C/C++), accès TCP ou UDP.

Dans le cas où le code existant intègre une IHM web, l'intégration peut être réalisée en déployant celui-ci derrière le même frontal web que la plate-forme Datalift (module Python, PHP…) ou dans le conteneur JEE dans lequel s'exécute la plate-forme.

4. Intégration des nouveaux modules

Datalift utilise la fonctionnalité Service Provider de Java pour identifier et charger les modules (applicatifs et extensions).

Ce mécanisme requiert, pour chaque interface implémentée (*Module*, *ProjectModule*, *UriPolicy*...) d'inclure dans la distribution du module un fichier nommé `META-INF/services/<nom interface>`, avec `<nom interface>` le nom complet (avec le package) de la classe d'interface implémentée (par exemple `org.datalift.fwk.Module`). Ce fichier contient la liste des noms (complets) des classes du module implémentant l'interface, une classe par ligne.

Une distribution peut donc inclure N modules de P types (*Module*, *ProjectModule*, *UriPolicy*...). Cette distribution peut prendre la forme d'un fichier JAR ou d'un répertoire. Dans les deux cas, la distribution inclut les classes des modules ainsi que les fichiers JAR des librairies tierces utilisées.

Si les modules souhaitent mettre à disposition des ressources statiques publiques (pages HTML, scripts Javascript, images, feuilles de styles...), celles-ci doivent se trouver dans un répertoire `public`, situé à la racine de la distribution (JAR ou répertoire).

Datalift utilise par défaut l'outil Ant pour la compilation et la génération des distributions. Chaque module est libre d'utiliser l'outil de son choix : Ant, Maven, Gradle, SBT...

5. Howto, création par l'exemple d'un nouveau module

La création d'un nouveau module peut se faire selon les recommandations et le scénario décrits ici.

Outillage

Pour créer un module, les technologies dont l'usage est recommandé sont :

Un codage en Java ou dans un langage dérivé (Scala, Groovy…) et s'exécutant nécessairement dans une machine virtuelle Java 1.6+.

- Un environnement de développement intégré (IDE), par exemple Eclipse, IntelliJ…
- Un outil de génération automatisé : Ant, Maven…

La plate-forme Datalift intègre nativement les bibliothèques suivantes pour la gestion des interfaces Web :

- JQuery et JQuery UI pour l'utilisation de Javascript sur le poste utilisateur
- Le moteur de rendu Velocity (fondation Apache) pour le génération des pages HTML coté serveur.

Il est aussi possible d'utiliser des pages JSP si le serveur d'application le supporte.

Squelettes

La plate-forme Datalift propose deux squelettes de base en standard. Ils servent de base au développement de tout nouveau module. Ils se trouvent dans le répertoire `sample-ui` des sources. Ils sont écrits en Java.

Squelette de module applicatif

La classe SampleUI est le prototype de module applicatif qui affiche l'ensemble des widgets JQuery UI disponibles. Il peut être réutilisé comme base à la réalisation de modules présentant une interface utilisateur riche.

L'annotation JAX-RS `@Path` définit le chemin d'URL associé au module :

```
@Path("sample-ui")
public class SampleUI extends BaseModule
{
    ...
```

La réception d'une requête HTTP GET sur `/sample-ui` déclenchera l'exécution de la méthod `getIndex()`, grâce à la présence de l'annotation JAX-RS `@GET`. La classe `UriInfo` permet d'accéder aux données de la requête HTTP relatives à l'URL demandée ; leur injection automatique est demandée via l'annotation JAX-RS `@Context`. La méthode retourne un objet demandant d'activer un moteur de rendu, dans le cas présent, Velocity (suffixe `.vm`).

```
@GET
public Viewable getIndex(@Context UriInfo uriInfo) {
    return new Viewable("/" + uriInfo.getPath() + "/index.vm");
}
```

Squelette de module du workspace

Un module du workspace est une fonctionnalité qui s'intègre aux projets d'élévation de données que composent les utilisateurs de la plate-forme.

La classe SampleProjectModule est le prototype de module du workspace. Dans la mesure où ce module est paradigmatique, il appartient de facto à tous les étages de transformation Datalift sans distinction puisqu'il indique être applicable à toutes les étapes.

Ainsi, son utilisation requiert impérativement de le spécialiser pour un unique étage, ce qui s'effectue par la modification de la méthode canHandle(). Cette méthode analyse le projet courant pour déterminer si le module est applicable ou non selon le contexte. Si c'est le cas, elle retourne un objet UriDesc décrivant les paramètres d'affichage et le web service à appeler si l'utilisateur choisi d'utiliser le module.

```
public UriDesc canHandle(Project p) {
    try {
        // Ce module, quand sélectionné, affiche une simple image.
        UriDesc desc = new UriDesc(this.getName() + "/java-guy.jpg",
                                   "Sample Project Module");
        // Toujours afficher tout en bas de la liste des modules.
        desc.setPosition(1000000);
        return desc;
    }
    catch (Exception e) {
        log.fatal("Uh?", e);
        throw new RuntimeException(e);
    }
}
```