Prototype Linked Data
de la Chancellerie fédérale

Data Architect

Rapport

Novembre 2024

Auteur : Fabian Cretton

Chef de projet: Nicolas Debons

Data Semantics Lab de la HES-SO // Valais – Wallis – Sierre.

1 Table des matières

2 Révisions

Date	Version	Statut	Commentaire
30.10.2024	0.1	Draft	Premier jet
13.11.2024	0.2	Draft	Proposition de version finale, avec des remarques sur les ajouts et questionnements
28.11.2024	1.0		Version finale

3 Objectif

3.1 Objectif général du projet

L'objectif de ce projet a été défini par la Chancellerie fédérale dans leur document « Offertanfrage - Freihändiges Verfahren_Linked Data_DataArchitect » dont nous avons tiré les informations ci-dessous.

Ce travail s'inscrit dans le cadre de la numérisation pour façonner l'administration publique suisse de demain. Les échanges entre les autorités et la population suisse doivent être simplifiés. Les données sont à la base des services gouvernementaux numériques. À cette fin, nous faisons progresser la mise à disposition d'attributs de données de base distribués (décentralisés) par le biais d'organisations fédérales (fédérales, cantonales et communales). De cette manière, les données de base peuvent être utilisées dans le respect de la protection des données et les lois et processus applicables peuvent être mieux automatisés.

Aujourd'hui, les registres ne sont pas suffisamment connectés les uns aux autres et il est long et coûteux pour les autorités qui les utilisent de collecter les informations nécessaires et contraignantes sur les entités (par exemple sur une personne physique). La solution technique la plus simple serait de centraliser les données. Pour ce faire, il faut la volonté de la Confédération, des cantons et des communes, ainsi que d'importantes modifications de la loi, y compris une éventuelle modification constitutionnelle. Cependant, nous comprenons que la solution technique la plus simple n'est pas nécessairement toujours la plus convenable.

La centralisation des données de base nécessite une adaptation importante et chronophage d'un grand nombre de lois et d'ordonnances (y compris cantonales) dans notre système législatif fédéral. D'un point de vue technique, la commune, les cantons et l'administration fédérale utilisent une grande variété de solutions pour la gestion de leurs données de base. Ces systèmes sont basés sur une grande variété de concepts technologiques. Les utilisateurs de données de base doivent effectuer de nombreuses intégrations d'interfaces différentes afin d'obtenir les données nécessaires. Pour chaque interface, les applications nécessitent des informations d'identification différentes (credentials).

Afin d'explorer d'autres possibilités, l'objectif de ce prototype est d'ajouter parallèlement aux données et systèmes existants une couche Linked Data. Les technologies et concepts du Linked Data pourraient avoir un impact majeur, en particulier pour l'administration publique suisse avec la répartition fédérale des responsabilités : Le Linked Data permet à différentes autorités de gérer les données (des attributs aux entités) de manière décentralisée avec leur infrastructure existante et de les relier entre elles au moyen de liens mutuels du Linked Data. De cette manière, les données peuvent rester sur le lieu de leur création avec la répartition des responsabilités applicable, tandis que dans le même temps, ces données peuvent être mises à la disposition d'autres autorités pour l'exécution de leurs tâches statutaires de manière efficace et économe en ressources. Une adaptation de la loi peut être plus simple qu'avec un registre central, car l'approche des données liées est basée sur l'interopérabilité des données et non sur la redistribution des responsabilités.

Dans ce prototype, les attributs sont transférés vers un Triple Store local, associés à une identification des données sources, et mis à disposition via un URI (Uniform Resource Identifier).

3.2 Vision d'ensemble et rôles

Dans le prototype Linked Data, différents rôles sont définis, tels que le développement de la base technique (BFH), les fournisseurs de données et les utilisateurs de données ou encore les architectes de données (HES-SO Valais/Wallis).

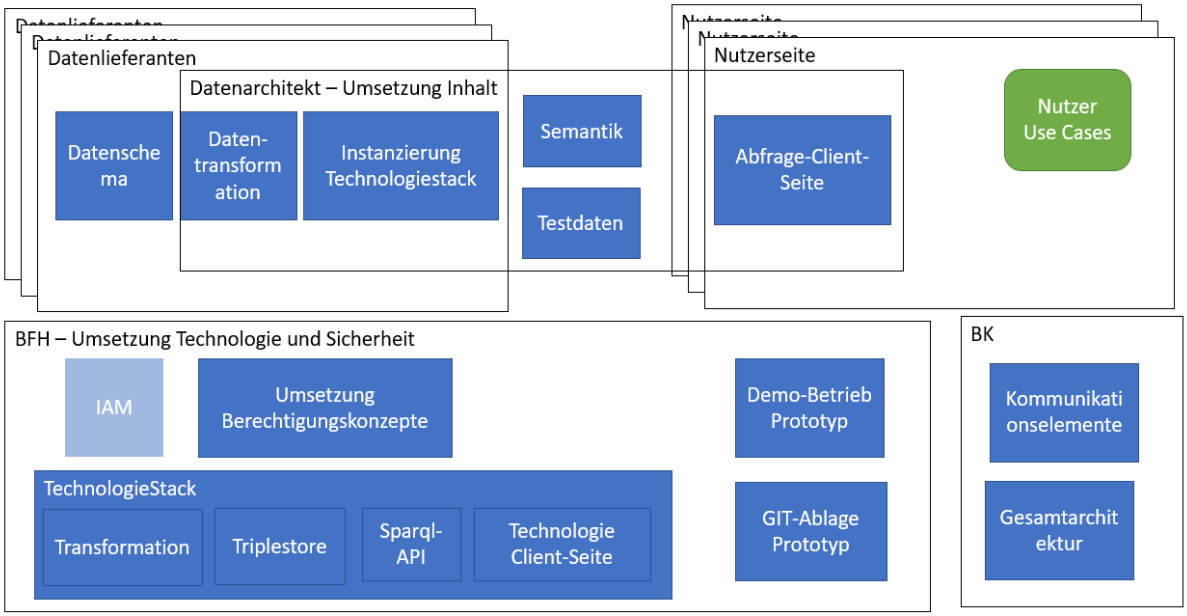

Figure 1 - Architecture du projet

3.3 Rôle Data Architecte

Ce rapport concerne la gestion des données et du contenu pour le rôle Data Architect, et notamment :

- Analyse des données source selon le choix d'un cas d'utilisation
- Analyse et choix des schémas permettant la représentation de ces données en Linked Data (ontologies)
- Conversion des données sources en Linked Data
- Validation de l'intégrité des données par le standard SHACL
- Hébergement des données Linked Data dans plusieurs bases de données (Triple Stores)
- Requêtage des données par le langage SPARQL (langage d'interrogation des données Linked Data), sur plusieurs Triple Stores
- Intégration de ce travail avec celui de la BFH pour la gestion des droits d'accès

4 Linked Data : Introduction

Le *Linked Data*, ou *données liées*, est une méthode de publication de données structurées qui permet de les interconnecter et de les interroger entre différents ensembles de données et sites web. En utilisant des normes spécifiques définies par le W3C, le *Linked Data* permet de relier et de combiner des informations de façon significative, rendant le web plus utile et intégré. C'est un composant essentiel de l'évolution du Web, où les données provenant de sources différentes peuvent être exploitées ensemble d'une manière non prévue initialement par les éditeurs de données.

Même si ce n'est pas le but principal de ce projet-ci qui s'intéresse aux données fermées et protégées, le Linked Data est aussi une manière de publier de données ouvertes 5 étoiles, comme décrit par Tim Berners-Lee[1], l'inventeur du Web.

4.1 Principes Clés du Linked Data

Le concept de *Linked Data* repose sur quatre principes[2], proposés par Tim Berners-Lee, :

1. **Utiliser des URI pour Identifier les Données** : Chaque entité (comme une personne, un lieu ou un objet) possède un URI (Uniform Resource Identifier) unique, qui sert d'identifiant sur le web.

2. **Utiliser des URI HTTP pour l'Accessibilité** : En utilisant des URI HTTP, ces entités peuvent être accessibles en ligne. Lorsqu'une personne (ou un système) consulte un URI, elle devrait obtenir des informations utiles.

3. **Fournir des Informations Utiles** : Les données fournies à l'URI doivent être structurées dans un format standard (comme RDF - Resource Description Framework) et décrire les relations avec d'autres entités.

4. **Lier à d'Autres URI pour Construire un Réseau de Données** : En reliant les entités à d'autres URI, chaque point de donnée se connecte à d'autres, créant un réseau de données liées.

4.2 Technologies pour le Linked Data (et utilisées pour ce projet)

- **RDF (Resource Description Framework)**[3] : Le modèle standard pour l'échange de données sur le web. C'est un modèle simple basé sur des *triplets*, qui décrivent des relations entre des entités sous la forme suivante : Sujet - Prédicat – Objet.

  Par exemple : Jean - est l'auteur de - Livre1

- **RDFs** (RDF Schéma)[4] et **OWL** (Web Ontology Language)[5] : Des langages pour définir des schémas de données (ontologies) qui permettent de décrire des relations complexes entre les éléments de données.

[1] https://5stardata.info/fr/
[2] https://www.w3.org/DesignIssues/LinkedData
[3] https://www.w3.org/RDF/
[4] https://www.w3.org/TR/rdf11-schema/
[5] https://www.w3.org/TR/owl2-overview/

- **Ontologie** : schéma de donnée, représentation formelle et structurée d'un domaine de connaissances. Elle définit un ensemble de concepts (ou classes), ainsi que les relations entre ces concepts (ou propriétés), permettant ainsi de modéliser les données de manière compréhensible et exploitable par des machines. Suivant le langage utilisé (RDFs, OWL, etc.), l'ontologie peut contenir plus ou moins d'informations sémantiques (axiomes logiques, règles d'inférence, etc.).

- **RDFizer** : un logiciel qui convertit des données issues de formats variés (comme des bases de données relationnelles, des feuilles de calcul, des fichiers CSV, XML, JSON, etc.) en format RDF.

- **SHACL** (Shapes Constraint Language)[6] : un langage de validation pour les données RDF. Il permet de définir des **règles et des contraintes** pour vérifier la conformité et la qualité des données RDF en fonction d'un modèle prédéfini, souvent appelé « forme » (*shape*).

- **Triple Store** : un type de base de données conçu spécifiquement pour stocker et gérer des triplets RDF.

- **SPARQL**[7] : Un langage de requête spécifiquement conçu pour interroger les données RDF.

- **SPARQL endpoint** : un point d'accès web permettant aux utilisateurs d'interroger des bases de données RDF en utilisant le langage SPARQL conçu pour extraire et manipuler des données du Web sémantique. Il fonctionne comme une API REST[8], où les utilisateurs envoient des requêtes SPARQL à un serveur via une URL, et reçoivent les résultats au format JSON, XML ou RDF.

[6] https://www.w3.org/TR/shacl/
[7] https://www.w3.org/TR/sparql11-query/
[8] https://fr.wikipedia.org/wiki/Representational_state_transfer

5 Linked Data : Mise en œuvre dans le projet

Voici un schéma de la mise en œuvre des principes et de la technologie Linked Data dans ce projet sur la « Figure 2 - Implémentation Linked Data pour le projet »

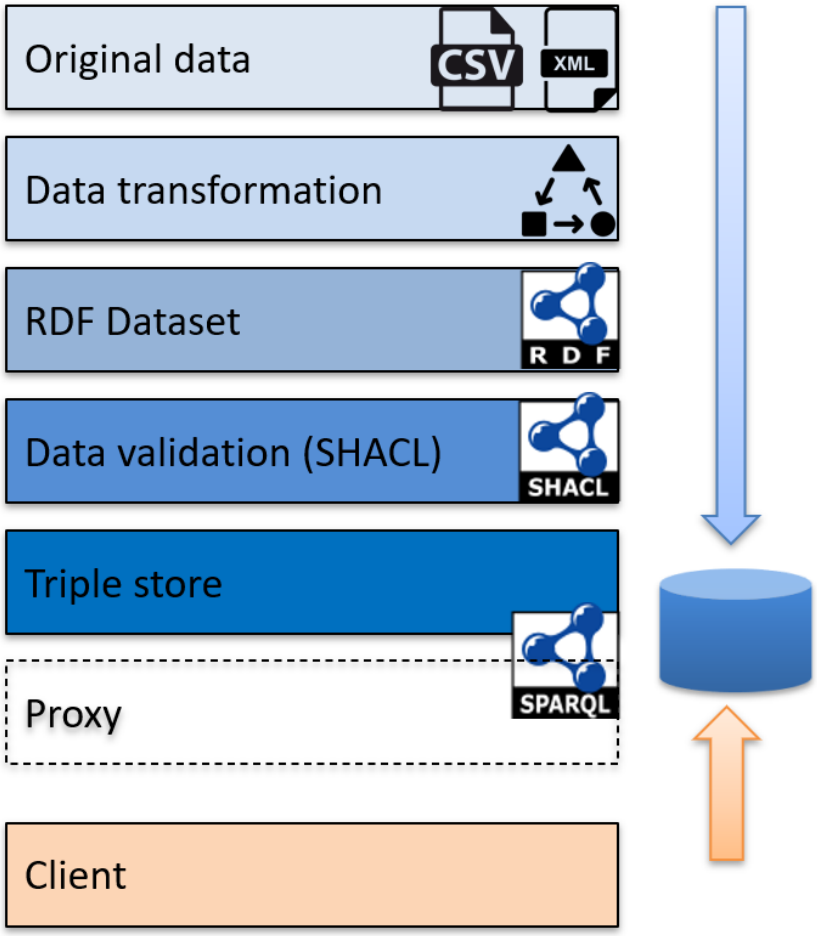

Figure 2 - Implémentation Linked Data pour le projet

En résumé : des données sources (CSV, XML, etc.) sont transformées en RDF à l'aide d'un RDFizer. Elles sont validées avec SHACL et stockées dans un Triple Store. Un SPARQL endpoint permet de poser des requêtes sur ces données à l'aide d'appels web REST standards.

Ces différentes étapes sont décrites en détail ci-dessous, et le code est à disposition sur le GitHub du projet[9].

[9] https://github.com/swiss/ld-prototype-data

6 Données source - Cas d'utilisation : Serafe

Pour réaliser ce projet sur un cas d'utilisation pratique et répondant à nos besoins de données distribuées, c'est la redevance radio-télévision (Serafe[10]) qui nous as servi de base. Merci de vous référer au rapport de la BFH qui donne tous les détails sur Serafe et sa mise en œuvre, notamment le registre UPI pour la description des personnes et le registre EWR des habitants (Einwohnerregister) pour les résidences principales de ces personnes.

6.1 UPI - Unique Person Identification

Dans le cadre du cas d'utilisation Serafe, il a été décidé de représenter les personnes selon les données de base de l'UPI[11], et de créer un petit jeu de données basé sur des caractères fictifs de Disney.

Les données UPI sont décrites dans la FAQ[12], sous « Questions générales UPI / Quelles données sont disponibles dans UPI ? »

Les données suivantes sont disponibles :

- Numéro AVS
- Nom
- Prénom
- Date de naissance : la date de naissance peut être renseignée uniquement avec une année dans certains cas (par ex. 00.00.1995) ou le mois et l'année (00.05.1995).
- Sexe

D'autres données peuvent être renseignées par les registres sources (nationalité, nom de célibataire, etc.), mais elles n'ont pas été modélisées pour cet exemple.

Nous avons créé un jeu de données fictif, en CSV[13], en se basant sur cette liste de la FAQ. A noter que pour ces données de personnages Disney, nous avons aussi ajouté les identifiants des personnages dans Wikidata[14], ce qui permet la mise en œuvre d'autres exemples de données liées et de déréférencement d'URL (voir le Prototype).

Pour une représentation XML de ces données, la norme eCH-0044[15] « Norme concernant les données Echange d'identifications de personnes » fournit le fichier .xsd qui définit la structure des données XML. Le GitHub contient également un petit jeu de données fictif en XML[16] basé sur cette norme.

Le eCH-0044 est référencé par les différentes normes eCH qui décrivent des échanges de données qui contiennent des identifications de personnes. C'est le cas notamment du eCH-0201[17] « Norme

[10] https://www.serafe.ch
[11] https://www.zas.admin.ch/zas/fr/home/partenaires-et-institutions-/unique-person-identification--upi-.html
[12] https://www.zas.admin.ch/zas/fr/home/partenaires-et-institutions-/unique-person-identification--upi-/faq.html
[13] https://github.com/swiss/ld-prototype-data/blob/main/UC-Serafe/UPI-Dataset-fiction/UPI_Personnes_fiction.csv
[14] https://www.wikidata.org
[15] https://ech.ch/fr/ech/ech-0044/4.1
[16] https://github.com/swiss/ld-prototype-data/blob/main/XML2RDF/sparql-generate/persons-eCH0044.xml
[17] https://www.ech.ch/fr/ech/ech-0201/1.0

d'interface Livraison de données concernant les personnes pour la redevance des ménages » et du eCH-0099[18] « Livraison données de l'EWR à la statistique »

Les données sont aussi reportées dans l'annexe de ce rapport.

6.2 Identifiants des personnes basés sur le numéro AVS

Conformément aux lois en vigueur, il a été décidé d'utiliser le numéro AVS comme identifiant des personnes, ce qui permet de faire le lien entre les personnes des jeux de données UPI et EWR. Ce numéro AVS est attribué et géré par l'Office fédéral des assurances sociales OFAS[19].

Pour ne pas utiliser de numéros AVS existants, nous avons été conseillés sur l'utilisation de 24 numéros AVS fictifs qui sont prévus à cet effet (ceux utilisés dans les données en CSV), voire des numéros dans la série 979XXXXXXXXX qui correspond à une zone de données de test selon la norme ISO des codes des noms de pays[20]. Ceci n'est pas un problème dans notre système de test, vu que la structure du numéro AVS ne doit pas forcément être respectée (ceux utilisés dans les données en XML).

6.3 EWR - Registre des résidents et relation d'annonce

Le cas d'utilisation Serafe repose donc sur les personnes, ainsi que sur leurs adresses. Celles-ci sont définies officiellement dans les registres de résidents par les relations d'annonce[21] (Meldeverhältnis). Nous avons tenu compte des deux relations d'annonce possibles : résidence principale (établissement) et résidence secondaire (séjour).

D'autre part une relation d'annonce est un « événement » avec une historisation (date début/fin), ce qui permet aussi d'illustrer une manière de gérer une historisation des données.

Dans la réalité, les adresses sont un lien vers le registre des bâtiments et logements[22], mais pour simplifier notre exemple nous représentons directement les adresses dans la relation d'annonce.

Le document « Recommandations pour l'adressage des bâtiments et l'orthographe des noms de rues »[23] nous indique les données principales pour l'adressage :

> Structure de l'adresse des bâtiments
> L'adresse d'un bâtiment se compose du nom de la rue, du numéro de maison (aussi appelé numéro d'entrée ou numéro de police) et de l'indication de la localité avec le numéro postal d'acheminement (NPA). La combinaison nom de rue/numéro de maison doit être univoque dans la localité, de sorte que l'adresse soit unique en Suisse.

[18] https://www.ech.ch/fr/ech/ech-0099/2.1
[19] https://www.bsv.admin.ch/bsv/fr/home.html
[20] Kodifikation des ausgebenden Landes (Schweiz) nach der Norm ISO 3166-numeric
AHV-Nummer https://www.zas.admin.ch/zas/de/home/partenaires-et-institutions-/navs13.html
[21] https://www.bfs.admin.ch/bfs/fr/home/registres/registre-personnes/harmonisation-registres/relation-annonce.html
[22] https://www.bfs.admin.ch/bfs/fr/home/registres/registre-batiments-logements/adresses-batiments.html
[23] https://www.bfs.admin.ch/bfs/fr/home/registres/registre-batiments-logements/adresses-batiments.assetdetail.5566190.html

A partir de ces informations nous avons créé un jeu de données fictif, en CSV[24], pour représenter les résidences principales nécessaires au cas d'utilisation. Les données sont aussi reportées dans l'annexe de ce rapport.

7 Choix des schémas de données – Ontologies

7.1 Réutilisation d'ontologies existantes

Pour représenter des données en Linked Data, il est de principe de réutiliser des ontologies existantes et connues de préférence.

Quelques avantages :

- Des experts d'un domaine passent du temps à concevoir une ontologie, et partagent le résultat de leur travail avec la communauté
- L'accès et la compréhension des données est facilité par ce partage de schéma entre différent jeux de données

Au niveau des inconvénients, il n'est pas forcément simple de rechercher une ontologie qui convienne, l'analyser et la comprendre, la réutiliser correctement.

7.2 Souplesse des ontologies

Voici quelques précisions complémentaires :

- Le choix n'est pas exclusif
  Un jeu de données peut se baser sur plusieurs ontologies (utilisant des classes/propriétés d'ontologies différentes)
- Il est fréquent en Linked data de dédoubler quelques informations afin de faciliter la réutilisation des données :
  - Une ressource peut être une instance de différentes classes (d'ontologies différentes)
  - L'attribut d'une ressource peut être exprimé/dupliqué avec différentes propriétés (d'ontologies différentes)
- Il est tout à fait possible de créer une nouvelle ontologie, indépendante ou qui vienne compléter une/des ontologies existantes.
  Si une ontologie existante répond partiellement à un besoin, il est ainsi possible de la compléter avec de nouvelles classes/propriétés selon le cas spécifique

7.3 Critères de sélection

Voici différents critères permettant d'orienter le choix d'une ou plusieurs ontologies :

- Interopérabilité :
  - schema.org[25] idéal pour s'ouvrir à l'ensemble du Web
  - Core vocabularies[26] pour une interopérabilité dans l'Administration, et même une conformité avec l'Union européenne

[24] https://github.com/swiss/ld-prototype-data/blob/main/UC-Serafe/EWR-Dataset-fiction/EWR_ResidencesPrincipales.csv
[25] https://schema.org

- Domaine/portée de la modélisation :
    - Ontologies génériques vs spécialisées
    - Génériques : schema.org, Wikidata[27], DBpedia[28], etc.
    - Spécialisées : domaine médical, smartCities, etc.
- Expressivité : RDFS (RDF Schema) et/ou OWL (Web Ontology Language)

7.4 Recherche d'ontologies

La recherche d'ontologies peut s'effectuer par les moteurs de recherche classiques, ou sur des sites spécialisés comme :

- Site de catalogue d'ontologies : LOV[29], etc.
- Ontologies spécialisées : sites de domaine spécialisé, Industrial Domain Ontologies (H2020)[30], BIOPortal[31], etc.

7.5 Choix pour le projet : Core vocabularies de la Commission européenne

Dans le cadre de ce projet, les critères suivants ont guidé notre recherche :

- Interopérabilité :
  Les Core vocabularies de l'Union européenne semblent une évidence
- Domaine/ portée de la modélisation :
  Ontologie générique complétée d'ontologies spécifiques pour certains jeux de données
- Expressivité : RDFs suffit largement dans le contexte du Linked Data où le but est la mise à disposition de données et le lien entre les jeux de données. Il est tout à fait possible d'utiliser des ontologies plus sophistiquées, basées sur OWL par exemple, mais l'utilisation d'axiomes logiques ou de règles d'inférence n'est pas primordial pour le Linked Data.

Ces ontologies sont maintenues, et de nouvelles versions ont été publiées au printemps 2024 notamment.

Il a donc été décidé d'analyser les Core vocabularies de la Commission européenne pour vérifier s'ils permettent de modéliser les données UPI et EWR du projet.

Il est à noter que ces Core vocabularies reposent sur des ontologies existantes et reconnues, notamment :

- foaf[32] friend of a friend
- dct[33] Dublin Core
- skos[34] Simple Knowledge Organization System

---

[26] https://joinup.ec.europa.eu/collection/semic-support-centre/core-vocabularies
[27] https://www.wikidata.org/
[28] https://www.dbpedia.org/
[29] https://lov.linkeddata.es/dataset/lov/
[30] https://ontocommons.eu/node/146
[31] https://bioportal.bioontology.org/ontologies
[32] http://xmlns.com/foaf/0.1/
[33] http://purl.org/dc/terms/
[34] http://www.w3.org/2004/02/skos/core

- adms[35] Asset Description Metadata Schema
- locn[36] ISA Programme Location Core Vocabulary

7.5.1 UPI - Unique Person Identification

Nous avons validé que les données UPI peuvent être modélisée avec le Core Person Vocabulary[37] dont voici le schéma dans la « Figure 3 – Vue d'ensemble du Core Person Vocabulary ».

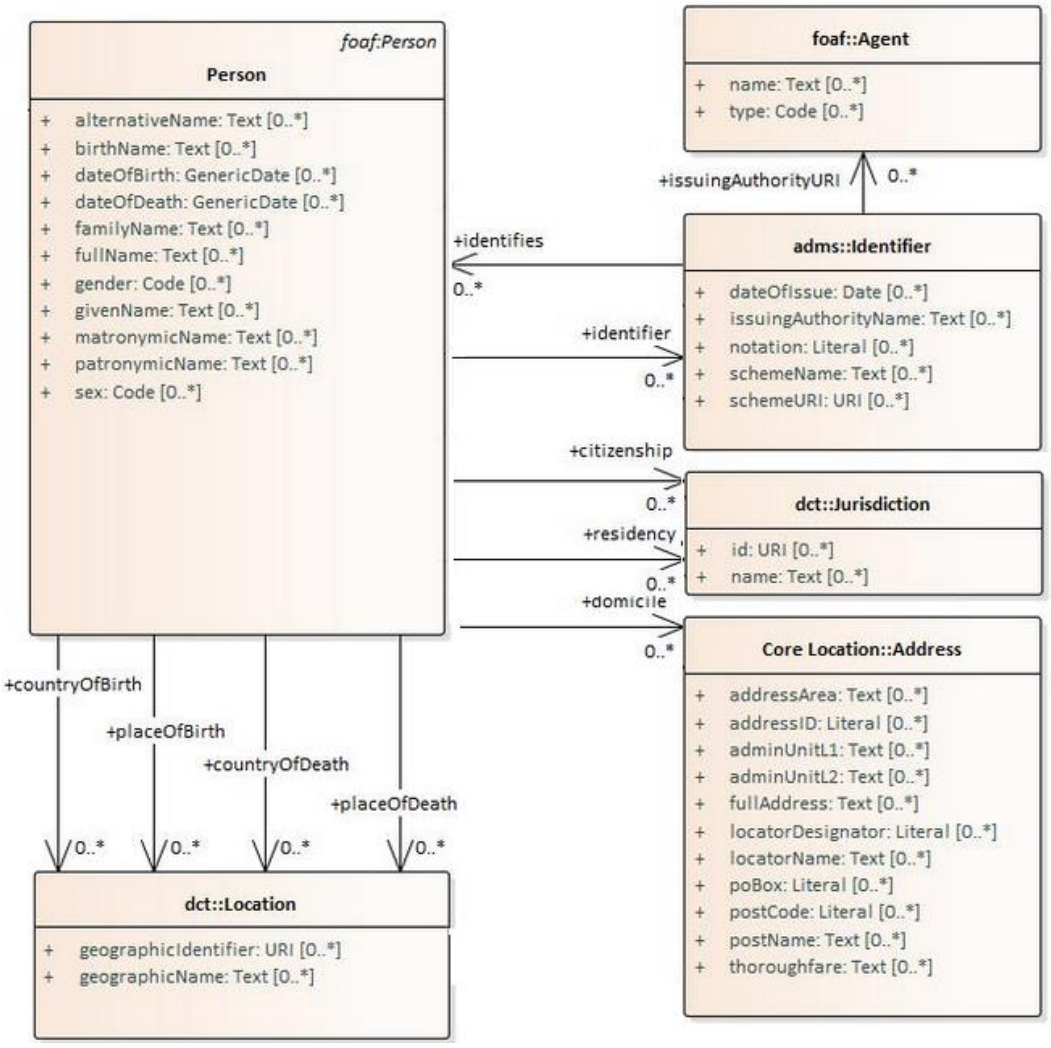

Figure 3 – Vue d'ensemble du Core Person Vocabulary

[35] http://www.w3.org/ns/adms
[36] https://www.w3.org/ns/legacy_locn
[37] https://semiceu.github.io/Core-Person-Vocabulary/releases/2.1.1/

La « Figure 4 - Données UPI Mickey Mouse en RDF » représente le personnage « Mickey Mouse » modélisé selon cette ontologie.

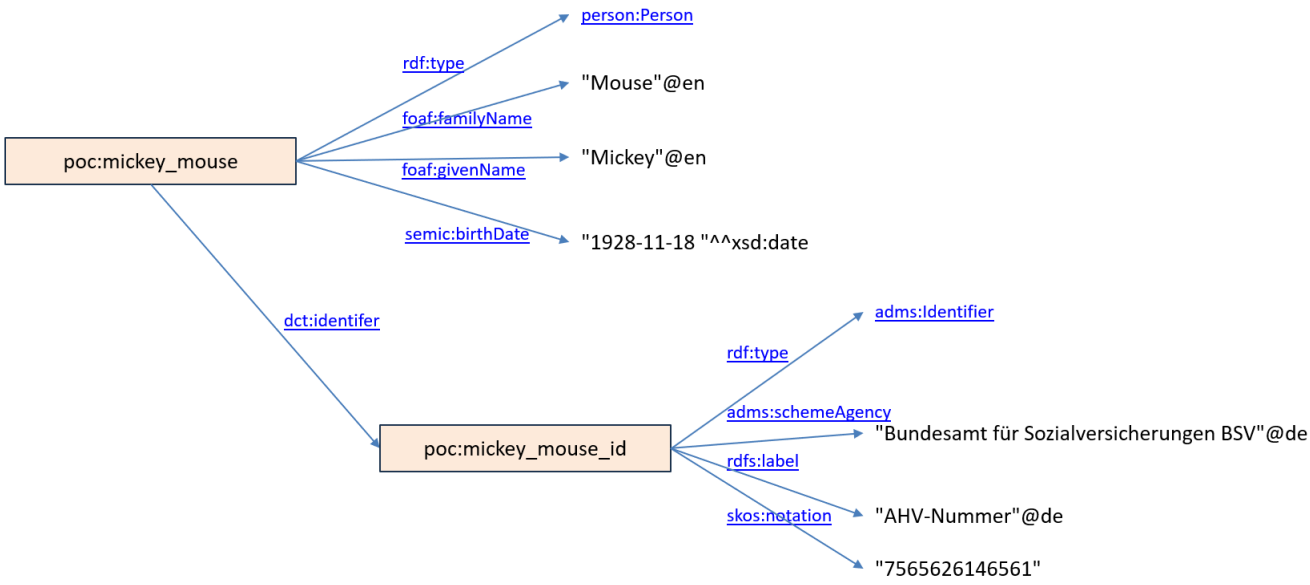

Figure 4 - Données UPI Mickey Mouse en RDF

7.5.2 EWR - Registre des résidents et relation d'annonce

Le Core Criterion and Core Evidence Vocabulary (CCCEV) [38] permet la modélisation de tout type de documents officiels, dont les relations d'annonce (Meldeverhältnis) qui nous intéressent.

Le schéma présenté dans « Figure 5 - Vue d'ensemble du CCCEV » peut-être visible de manière plus complète sur la page Web[39].

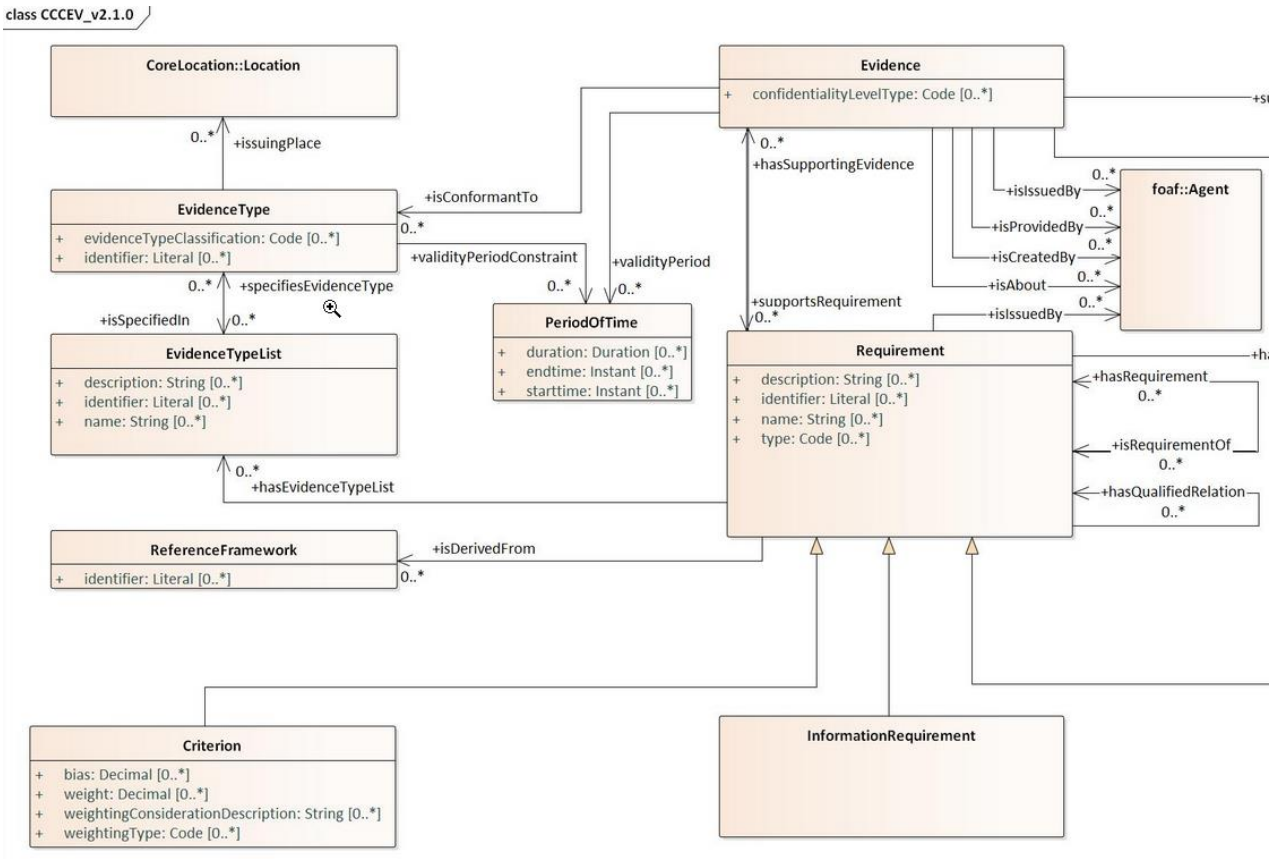

Figure 5 - Vue d'ensemble du CCCEV

Un exemple est fourni en ligne[40] pour démontrer l'utilisation de cette ontologie pour modéliser une carte d'identité.

D'autre part les adresses que nous ajoutons dans notre projet aux données EWR peuvent être modélisées avec le Core Location Vocabulary[41].

[38] https://semiceu.github.io/CCCEV/releases/2.1.0/
[39] https://semiceu.github.io/CCCEV/releases/2.1.0/#specoverview
[40] https://semiceu.github.io/CCCEV/releases/2.1.0/CCCEV_AnExplanatoryExample.pdf
[41] https://semiceu.github.io/Core-Location-Vocabulary/releases/2.1.0/

Le schéma présenté dans « Figure 6 - Vue d'ensemble du Core Location Vocabulary » peut-être visible de manière plus complète sur la page Web[42].

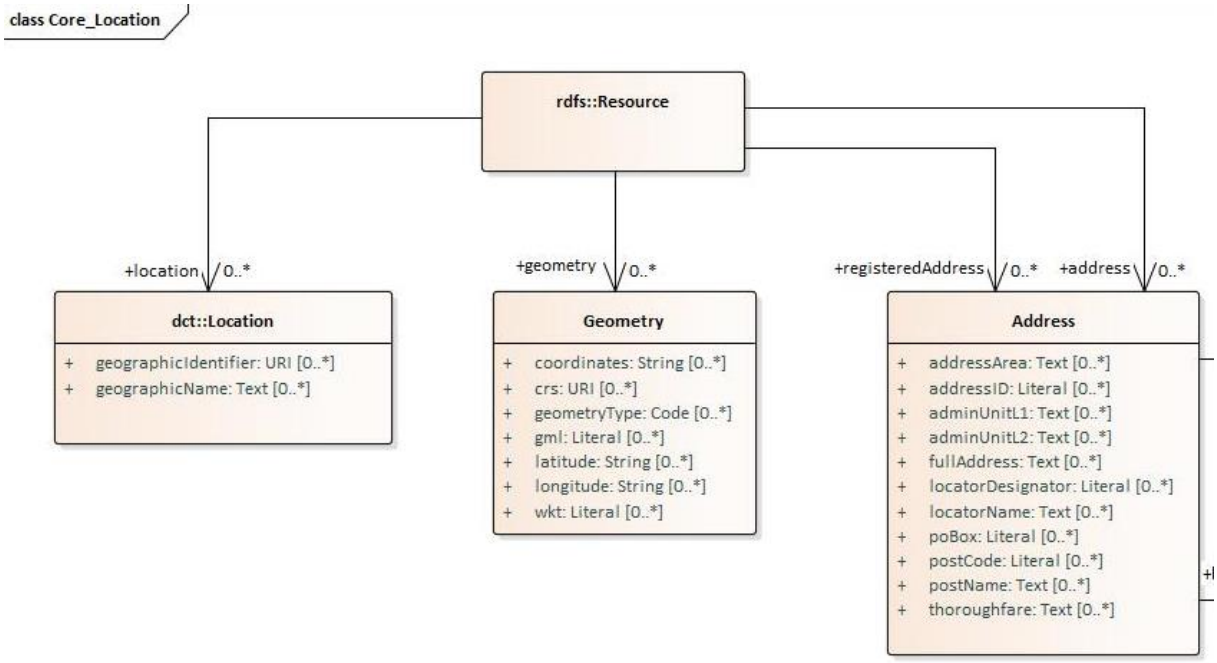

Figure 6 - Vue d'ensemble du Core Location Vocabulary

La « Figure 7 - Résidence principale de Mickey Mouse en RDF » représente les données EWR de la résidence principale du personnage « Mickey Mouse » modélisé selon cette ontologie.

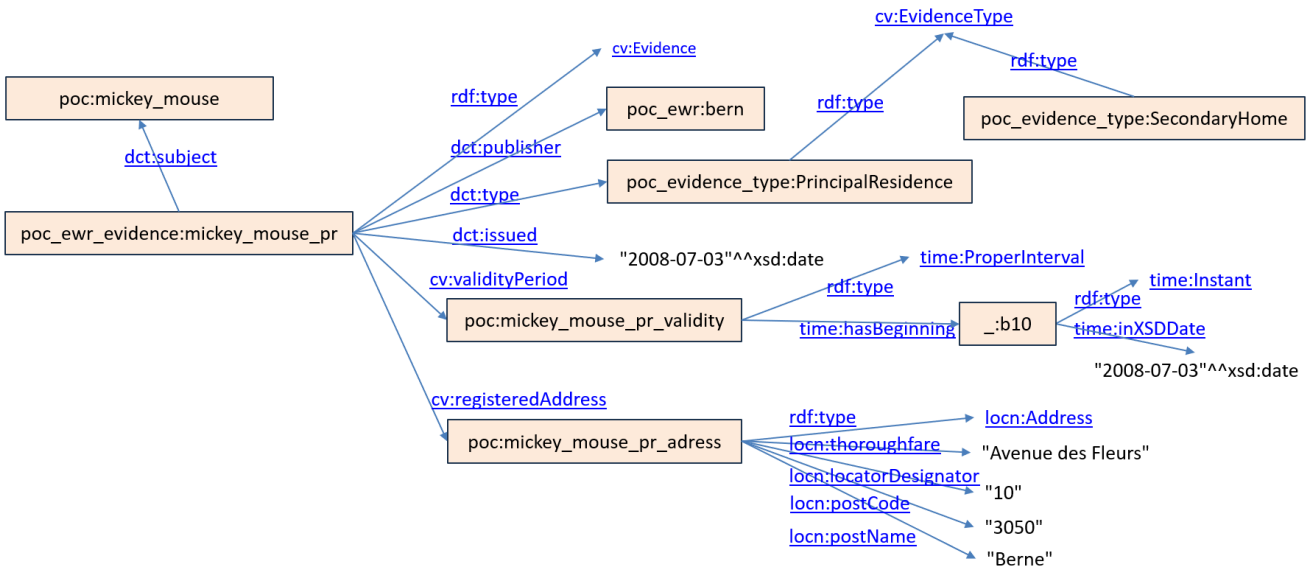

Figure 7 - Résidence principale de Mickey Mouse en RDF

[42] https://semiceu.github.io/Core-Location-Vocabulary/releases/2.1.0/#specoverview

8 Identifiants des ressources

Lorsque l'on génère des données RDF, il est nécessaire d'attribuer des identifiants à chaque nouvelle ressource/instance (une personne, une relation d'annonce par exemple). Lorsque l'on met à jour ou complète des données existantes, il faut s'assurer que les mêmes identifiants sont conservés.

Dans le monde du Linked Data, ces identifiants sont donc des Uniform Resource Identifier (URI)[43], et plus spécifiquement des Uniform Resource Locator (URL)[44] de préférence, donc des adresses Web telles qu'on les connait.

En choisissant des URL, il faut veiller à leur stabilité, leur lisibilité, leur conformité aux standards, et à ce qu'elles facilitent les interconnexions au sein de l'écosystème Linked Data. Ces identifiants permettant de lier des données entre différents jeux de données, leur choix et surtout leur stabilité sont bien plus importants que dans un écosystème fermé et isolé.

Pour la stabilité et donc la pérennité de ces identifiants, il faut donc utiliser un nom de domaine que l'on prévoit pour le long terme. On peut ensuite construire des identifiants propres à chaque type de ressource (personnes, relation d'annonce, adresse, etc.) en jouant avec les sous-domaines et les chemins de l'URL. Et finalement il faut compléter l'URL avec un identifiant unique à chaque ressource, pour chaque type de ressource (chaque personne a un identifiant unique parmi les personnes, chaque relation d'annonce a un identifiant unique parmi les relations d'annonce). Ce dernier choix repose en général sur des identifiants existants déjà dans les données sources et les registres, comme le numéro AVS utilisé dans notre cas pour identifier les personnes.

Pour garder une cohérence au sein de l'administration fédérale, et ne pas réinventer la roue non plus au niveau de la stratégie à adopter pour l'attribution des identifiants, il sera intéressant d'avoir des discussions avec les Archives fédérales (AFS) et l'Office fédéral de l'environnement (OFEN) par rapport au travail réalisé dans le cadre de la plateforme LINDAS[45] et de la gouvernance[46] qui contient déjà de la documentation pour la gestion des Namespaces (sous-domaines)[47] et des URI[48].

Voici un exemple d'identifiant pour la municipalité de Lausanne dans LINDAS :

https://ld.admin.ch/municipality/5586

Protocole : https
Nom de domaine réservé : admin.ch
Sous-domaine propre aux données linked data : ld.admin.ch
Chemin propre aux municipalités : /municipality
Identifiant de la municipalité Lausanne : 5586

[43] https://fr.wikipedia.org/wiki/Uniform_Resource_Identifier
[44] https://fr.wikipedia.org/wiki/Uniform_Resource_Identifier#Uniform_Resource_Locator_(URL)
[45] https://lindas.admin.ch/
[46] https://ld.admin.ch/governance/
[47] https://ld.admin.ch/governance/namespaces/
[48] https://ld.admin.ch/governance/URI-templates/

Voici quelques exemples d'identifiants que nous avons générés dans nos jeux de données fictifs :

Les personnes de l'UPI :

http://example.com/person/{numéro AVS}

Les identifiants AVS :

http://example.com/avs_id/{numéro AVS}

Une relation d'annonce de résidence principale pour le canton de Berne

http://example.com/ewr/mv/principal_residence/be/{id_relation_annonce}

On pourrait écrire tout un document sur la stratégie à adopter pour le choix des URLs, et ce sera un élément à bien prendre à considération pour une mise en œuvre concrète.

9 Versioning des données

Pour gérer différentes versions d'une même ressource qui évolue dans le temps (une personne, une adresse, etc.), il existe plusieurs possibilités en RDF, dont notamment :

- Inclure des métadonnées de version
  Ceci consiste à ajouter dans les données des informations sur la version. Pour le faire, différentes ontologies proposent des classes et propriétés, comme Dublin Core ou PROV-O (Provenance Ontology)[49].
  A noter la solution mise en place dans LINDAs pour la « liste historisée des communes » basée sur l'ontologie version.link: Identity Versioning[50]. En ouvrant par exemple la page de la municipalité Lausanne[51], une information de « version » est disponible.
- Utiliser des graphes nommés
  A l'intérieur d'une même base de données d'un Triple Store, il est possible de gérer des sous-ensembles de données que l'on appelle graphes nommés (named graph). Suivant le cas, il est envisageable de gérer différentes versions d'une ressource dans différents graphes nommés
- Versioning natif du triple store
  Certains triples stores gèrent une historisation des versions des données, comme GraphDB[52].

Dans nos exemples nous démontrons la mise en œuvre de métadonnées pour une relation d'annonce qui est un événement avec une date début et date de fin. Cette modélisation est possible avec le Core Criterion and Core Evidence Vocabulary que nous avons choisi.

[49] https://www.w3.org/TR/prov-o/
[50] https://version.link/
[51] https://ld.admin.ch/municipality/5586
[52] https://graphdb.ontotext.com/documentation/10.7/data-history-and-versioning.html?highlight=history%20plugin

10 Conversion des données

10.1 Les RDFizers

Pour convertir les données à partir d'un certain format (CSV, JSON, etc.) vers RDF, il existe une quantité d'outils existants nommés communément RDFizers.

Certains outils sont propres à un format spécifique en entrée, d'autres gèrent différents formats. Il est intéressant de noter l'existence de la norme RDF Mapping Language (RML)[53] qui sert de base à de nombreux outils, et qui fournit un langage générique pour configurer la conversion en RDF de données en différents formats.

Pour trouver les outils, une recherche Web peut faire l'affaire. Il existe aussi des listes, que ce soit la page du W3C[54] qui est maintenue depuis de nombreuses années, ou différents sites qui présentent certains outils comme celui du FAIR Cookbook[55].

10.2 Formats de sérialisation RDF

RDF peut être représenté dans plusieurs formats pour faciliter sa lecture et son échange. Les principaux formats sont :

- **RDF/XML** : Un format basé sur XML, adapté pour les machines mais plus difficile à lire pour les humains.
- **Turtle** : Un format plus lisible pour les humains, avec une syntaxe simplifiée.
- **JSON-LD** : Un format basé sur JSON, populaire pour les applications web car il est facile à intégrer avec JavaScript.
- **N-Triples** : Un format très simple, où chaque triplet est représenté sur une ligne distincte.

L'outil de conversion génère en général un fichier texte qui contiendra les données RDF dans l'un des formats ci-dessus. Ce format de sortie peut d'ailleurs être un paramètre de l'outil.

10.3 CSV vers RDF

Pour le projet nous fournissons des exemples de conversion CSV vers RDF pour les jeux de données UPI et EWR[56].

Nous avons utilisé l'outil Tarql[57] (SPARQL for Tables), qui ne convertit que des données CSV. Mais il a l'avantage d'être simple de configuration et performant. Il est intéressant de noter que la configuration se fait par la syntaxe SPARQL, ce qui est beaucoup plus simple et concis que le RML notamment.

On trouve dans les répertoires d'exécution[58] [59]:

[53] https://rml.io/specs/rml/
[54] https://www.w3.org/wiki/ConverterToRdf (lors de l'écriture de ce rapport, dernière mise à jour en 2020)
[55] https://egonw.github.io/cookbook-dev/content/recipes/interoperability/rdf-conversion.html
[56] https://github.com/swiss/ld-prototype-data/tree/main?tab=readme-ov-file#upi-and-ewr-data---csv-to-rdf-transformation
[57] https://tarql.github.io/
[58] https://github.com/swiss/ld-prototype-data/tree/main/UC-Serafe/UPI-Dataset-fiction/tarql-1.2/bin
[59] https://github.com/swiss/ld-prototype-data/tree/main/UC-Serafe/EWR-Dataset-fiction/tarql-1.2/bin

- mapping.sparql : le fichier de configuration de la conversion
- le .csv : le fichier de données sources
- le .ttl : le fichier de résultat, au format Turtle

La conversion peut être exécutée sous Linux ou Windows avec les fichiers respectifs (run.sh et run.bat). Ces fichiers contiennent d'ailleurs quelques options de conversion.

10.4 XML vers RDF

Comme beaucoup d'échange de données ont lieu avec des fichiers XML et selon les standards eCH, nous avons aussi testé quelques outils pour la transformation XML vers RDF[60].

Bien que la fonctionnalité de base de transformation XML à RDF est proposée par de nombreux outils, c'est avec surprise que nous avons rencontré de petits problèmes/bugs pour des cas précis.

Sur notre GitHub, nous avons mis en œuvre les deux outils présentés ici : SPARQL-Generate et RocketRML.

10.4.1 SPARQL-Generate

Notre premier exemple[61] démontre l'utilisation de SPARQL-Generate[62], un outil simple d'utilisation et similaire à Tarql par sa configuration basée sur la syntaxe SPARQL, mais qui permet de gérer différents types de fichiers en entrée (JSON, CSV, XML, etc.).

Le bug découvert dans SPARQL-Generate est qu'il n'est pas capable de gérer les tags XML en camelCase (avec un mélange de majuscules et minuscules). Nous en avons discuté ici[63] avec les auteurs. La correction semble aisée et nous verrons s'ils fournissent un prochain livrable.

Pour notre exemple, nous avons simplement exécuté une commande juste avant la transformation pour convertir tous les tags XML en minuscules (voir le run.sh[64]).

10.4.2 RocketRML

Notre deuxième exemple[65] démontre l'utilisation de RocketRML[66], un outil basé sur la norme RML que nous avons présentée ci-dessus.

Nous avons ici rencontré un premier problème commun à plusieurs RDFizers, soit la difficulté de leur parseur XPath à gérer les namespaces (xmlns) qui peuvent être présents dans le tag racine. Cet outil fournit une très élégante solution qui permet de configurer une option « removeNameSpace » (voir dans le fichier index.js[67]).

[60] https://github.com/swiss/ld-prototype-data/tree/main?tab=readme-ov-file#upi-data---xml-to-rdf-transformation
[61] https://github.com/swiss/ld-prototype-data/tree/main/XML2RDF/sparql-generate
[62] https://github.com/sparql-generate/sparql-generate
[63] https://github.com/sparql-generate/sparql-generate/issues/109#issuecomment-2358331037
[64] https://github.com/swiss/ld-prototype-data/blob/main/XML2RDF/sparql-generate/run.sh
[65] https://github.com/swiss/ld-prototype-data/tree/main/XML2RDF/rocketRML
[66] https://github.com/semantifyit/RocketRML
[67] https://github.com/swiss/ld-prototype-data/blob/main/XML2RDF/rocketRML/index.js

Pour l'utilisation d'outils basés sur RML, il peut être nécessaire d'écrire des fonctions personnalisées pour gérer des situations particulières. Dans notre cas, il faut transformer la valeur numérique du sexe de la personne dans les données source (1 ou 2) en une URL prédéfinie. Ceci, très simplement géré avec des outils comme Tarql ou SPARQL-Generate, demande cependant d'ajouter du code dans un outil basé sur RML. Bien que cet ajout soit simple à mettre en œuvre pour RocketRML (fonction getSexUrl() définies dans index.js et appelée dans mapping.ttl), le résultat contient un petit bug car l'URL générée est convertie en chaine de caractère. N'ayant pas reçu de réponse à ce jour de la part des auteurs[68], nous avons à nouveau apporté une solution simple pour corriger le fichier généré directement dans le run.sh[69].

10.4.3 Autres Outils

En complément d'information, voici des résultats de tests effectués avec d'autres outils :

- SDM-RDFizer[70]
  Outil également basé sur RML.
  Lors de nos premiers tests, il rencontrait aussi un problème pour gérer les namespaces (xmlns), mais les auteurs ont rapidement fourni un correctif à notre demande[71]. Nous ne sommes pas allé plus loin, mais il faudrait encore tester l'ajout de fonctions personnalisées, certainement par la mise en œuvre de Dragoman[72].
- RMLMapper[73]
  Également basé sur RML, c'est outil fut relativement simple à mettre en œuvre pour la conversion de base. Cela demanderait un peu de travail pour voir comment implémenter les fonctions personnalisées[74], mais nous ne sommes pas allés plus loin étant donné que l'outil semble peu performant pour gérer des fichiers XML[75].

Mentionnons finalement l'outil Barnard59[76] grandement utilisé pour les pipelines de transformation de données de la plateforme LINDAS. Selon les auteurs que nous avons contactés, il ne fournit pas de solution prête à l'emploi pour le XML, mais cette mise en œuvre ne demanderait certainement pas trop de travail.

10.4.4 Conclusion

En conclusion, il semblerait que le XML n'est pas grandement utilisé pour les conversions vers RDF vu les petits problèmes que nous avons identifiés dans plusieurs outils connus. D'un autre côté il n'y aurait aucun souci majeur pour une mise en œuvre, soit en apportant de petites corrections aux outils existants, soit en contournant les problèmes rencontrés comme nous l'avons implémenté. Nous recommanderions un test plus complet avec l'outil Barnard59 qui fonctionne peut-être parfaitement bien pour le XML, et qui est déjà utilisé pour des conversions CSV ou JSON dans LINDAS.

[68] https://github.com/semantifyit/RocketRML/issues/48
[69] https://github.com/swiss/ld-prototype-data/blob/main/XML2RDF/rocketRML/run.sh
[70] https://github.com/SDM-TIB/SDM-RDFizer
[71] https://github.com/SDM-TIB/SDM-RDFizer/issues/117
[72] https://github.com/SDM-TIB/Dragoman
[73] https://github.com/RMLio/rmlmapper-java
[74] https://github.com/RMLio/rmlmapper-java?tab=readme-ov-file#including-functions
[75] https://github.com/RMLio/rmlmapper-java?tab=readme-ov-file#xml-file-parsing-performance
[76] https://github.com/zazuko/barnard59/

10.5 Graphe de connaissances virtuel

Il existe des outils qui permettent de l'interrogation SPARQL sur des bases relationnelles. Au lieu de convertir les données d'origine en RDF pour les stocker dans un triple store, il est alors possible de conserver les données dans une base de données relationnelle et de les présenter comme un « knowledge graph » virtuel.

Lors d'une requête SPARQL, la requête est alors traduite en SQL et les données de la réponse sont converties en RDF à la volée.

Ceci est possible avec différents outils comme Ontop[77], ou certains triples stores qui gèrent des graphes virtuels comme Stardog[78].

[77] https://ontop-vkg.org/
[78] https://docs.stardog.com/virtual-graphs/

11 Validation de l'intégrité des données (SHACL)

Pour valider les données RDF, le W3C a normalisé le Shapes Constraint Language (SHACL)[79].

Après analyse, nous avons découvert que les Core Vocabularies utilisés pour nos jeux de données, fournissent déjà les fichiers SHACL pour la validation des données, directement sur les pages officielles[80].

La mise en œuvre de la validation des données est démontrée sur le dépôt GitHub du projet [81] : un outil SHACL (Apache Jena[82] dans notre exemple) est exécuté avec en entrée d'une part le fichier RDF à vérifier (par exemple nos données UPI ou EWR), d'autre part le fichier SHACL.

Cette validation nous a permis de corriger nos transformations, car le SHACL nécessite que les noms et prénoms des personnes soient des chaines de caractères avec un tag de langue, ce qui n'était pas le cas dans nos premières versions.

Nous avons trouvé une petite inconsistance dans le fichier SHACL du Core Criterion and Core Evidence Vocabulary, et l'avons mentionné sur leur GitHub[83]. Il s'agit du fait que l'URL de l'ontologie Time est référencée généralement avec le protocole « http », sauf pour <https://www.w3.org/2006/time#Instant>. Ils corrigeront cela dans une prochaine version et nous avions déjà adapté le fichier utilisé (voir la note au sujet de cccev-ap-SHACL.ttl corrigé dans cccev-ap-SHACL_corrected.ttl [84]).

[79] https://www.w3.org/TR/shacl/
[80] https://semiceu.github.io/Core-Person-Vocabulary/releases/2.1.1/
[81] https://github.com/swiss/ld-prototype-data/tree/main?tab=readme-ov-file#data-validation-with-shacl
[82] https://jena.apache.org/download/index.cgi
[83] https://github.com/SEMICeu/CCCEV/issues/58
[84] https://github.com/swiss/ld-prototype-data?tab=readme-ov-file#data-validation-with-shacl

12 Prototype : hébergement et interrogation des données

Ce prototype[85] démontre plusieurs choses :

- Instancier à la volée deux triples stores, un pour les données UPI l'autre pour les données EWR que nous avons générées en RDF
- Exposer les données hébergées dans les triples stores sur deux SPARQL end-points respectifs
- Montrer comment du code client peut poser des requêtes SPARQL qui combinent les données de ces deux end-points, ceci de différentes manières (avec ou sans SPARQL fédéré notamment).

Le code, réalisé en Python, est basé sur rdflib[86], une librairie qui offre des fonctionnalités pour RDF/Linked Data. Cette librairie fournit également l'outil rdflib-endpoint utilisé pour charger dans un triple store, en mémoire, les fichiers Turtle que générés pour les jeux de données UPI et EWR, tout en exposant ces données sur un SPARQL endpoint.

Le POC se compose ainsi de 3 composants :

- Les deux triples stores et leurs SPARQL endpoints UPI et EWR qu'il faut exécuter (startEWR.sh et startUPI.sh)
- Le code client serafe_sparql_query.py qui lance plusieurs requêtes SPARQL d'interrogation des données

Le code client démontre comment exécuter une requête sur les données EWR pour obtenir toutes les résidences principales et les identifiants des personnes concernées, conjuguée avec une requête sur les données UPI pour obtenir les informations sur ces personnes (nom et prénom en l'occurrence).

Trois possibilités sont démontrées pour la manière de réaliser ces requêtes sur deux endpoints différents :

- Une seule requête SPARQL : Federated Query
- Une requête EWR puis une seule requête UPI
- Une requête EWR puis une requête UPI par personne

[85] https://github.com/swiss/ld-prototype-data?tab=readme-ov-file#data-stored-in-two-triple-stores-and-client-interrogation-on-the-sparql-endpoints
[86] https://github.com/RDFLib/rdflib

12.1 Requêtes SPARQL

Les requêtes se retrouvent à disposition dans l'Annexe.

12.1.1 Une seule requête SPARQL: Federated Query

Il est possible depuis SPARQL 1.1 d'exécuter des requêtes fédérées[87], c'est-à-dire de questionner plusieurs SPARQL endpoints à partir d'une seule requête.

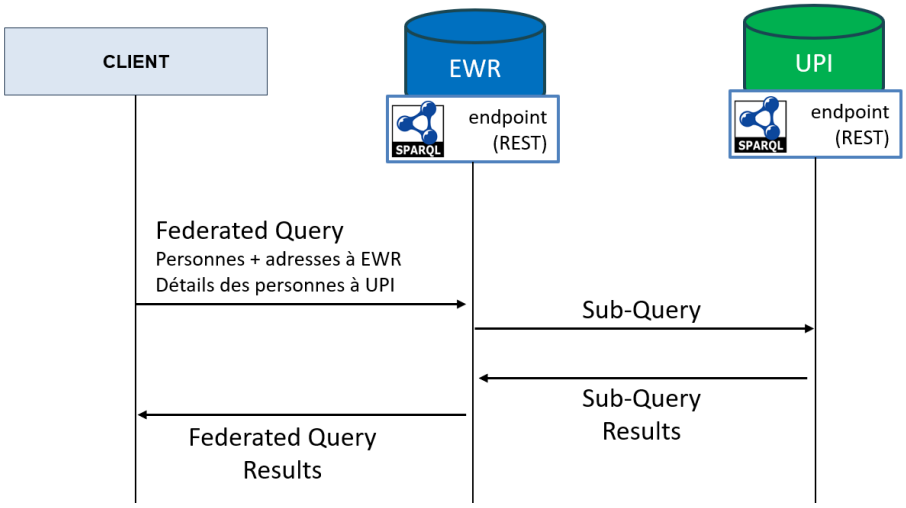

Figure 8 - SPARQL Federated Query - Séquence

En effet, le mot clé SERVICE permet de diriger une partie d'une requête vers un endpoint particulier. Les résultats sont renvoyés au processeur de requêtes fédérées et sont combinés avec les résultats du reste de la requête.

```
SELECT ?person ?city ?name WHERE {
?evidence dct:conformsTo ex:evidenceTypePrincipalResidence ;
  dct:subject ?person ;
  cv:registeredAddress/locn:postName ?city .

SERVICE <http://upi-endpoint.ch> {
   ?person foaf:familyName ?name .
   }
}
```

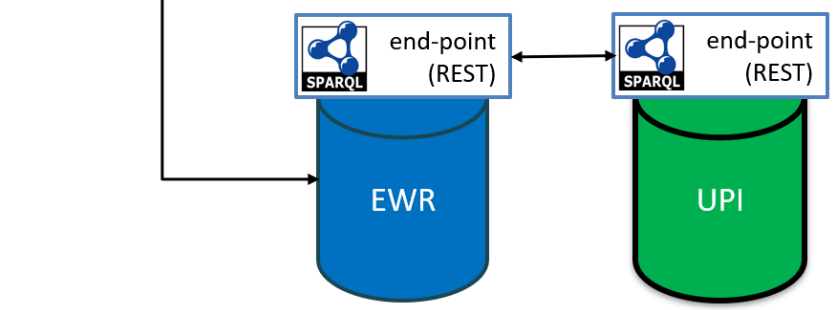

Figure 9 - SPARQL Federated Query - Exemple

Ici la jointure des résultats est donc réalisée de manière transparente.

[87] https://www.w3.org/TR/sparql11-federated-query/

12.1.2 Une requête EWR puis une seule requête UPI

Une autre possibilité de SPARQL 1.1 est d'utiliser le mot-clé VALUES[88] pour limiter les valeurs que peuvent prendre une variable.

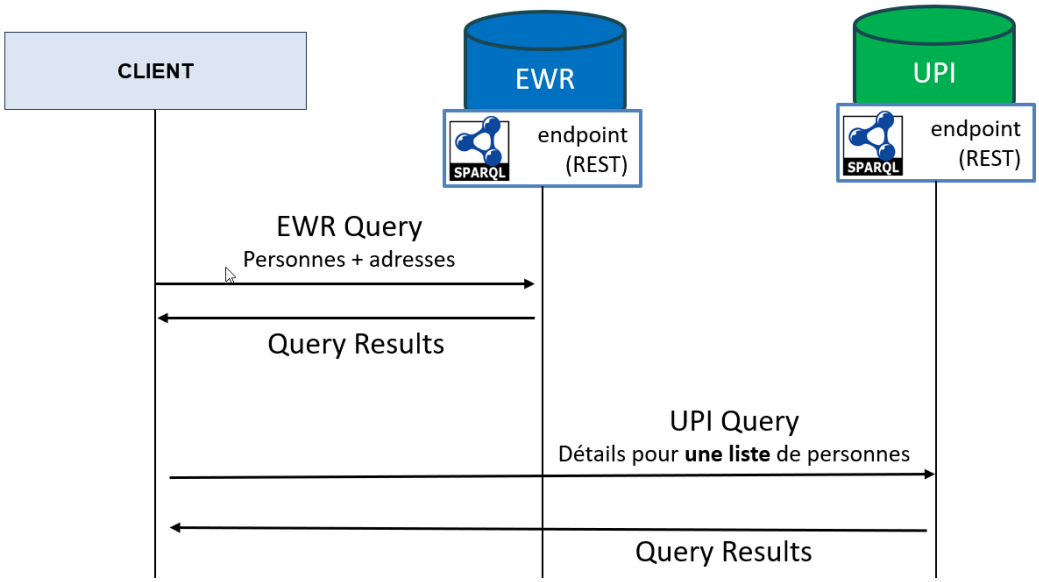

Figure 10 - SPARQL VALUES – Séquence

Dans cet exemple, la liste des identifiants de personnes est donc récupérée de la première requête EWR, puis utilisées dans la requête UPI pour n'interroger qu'une liste de personnes précise.

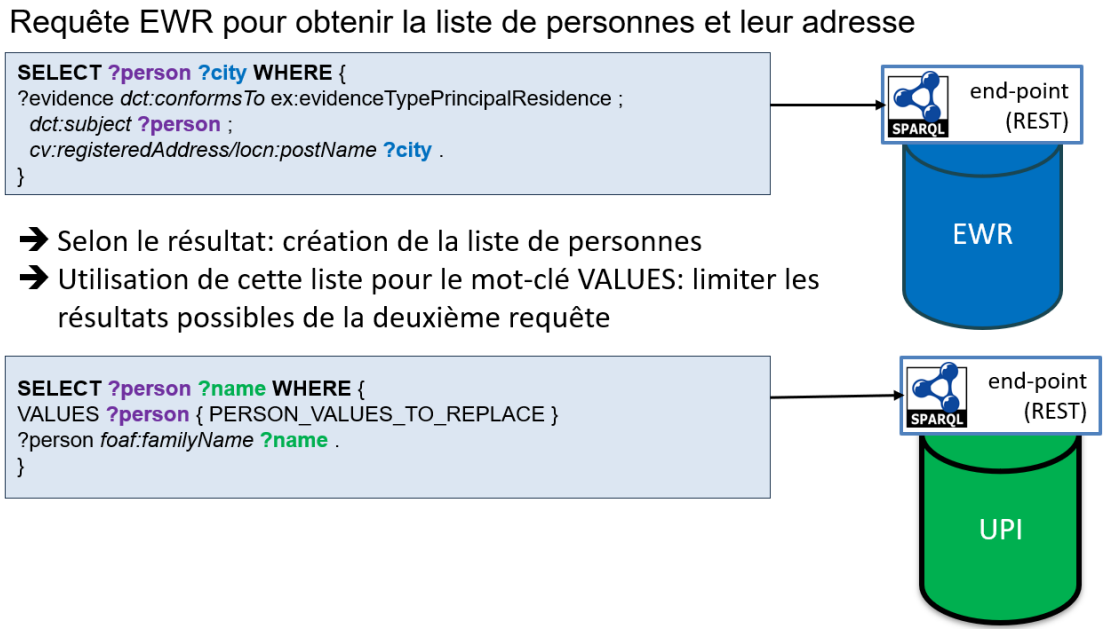

Figure 11 - SPARQL VALUES – Exemple

Ici la jointure des résultats est réalisée par le code client qui récupère une liste de la première requête et l'utilise pour adapter dynamiquement la seconde requête.

[88] https://www.w3.org/TR/sparql11-query/#inline-data

12.1.3 Une requête EWR puis une requête UPI par personne

De manière beaucoup plus classique, cet exemple exécute simplement une boucle sur les résultats de la première requête EWR pour ensuite interroger, pour chaque personne, le endpoint UPI.

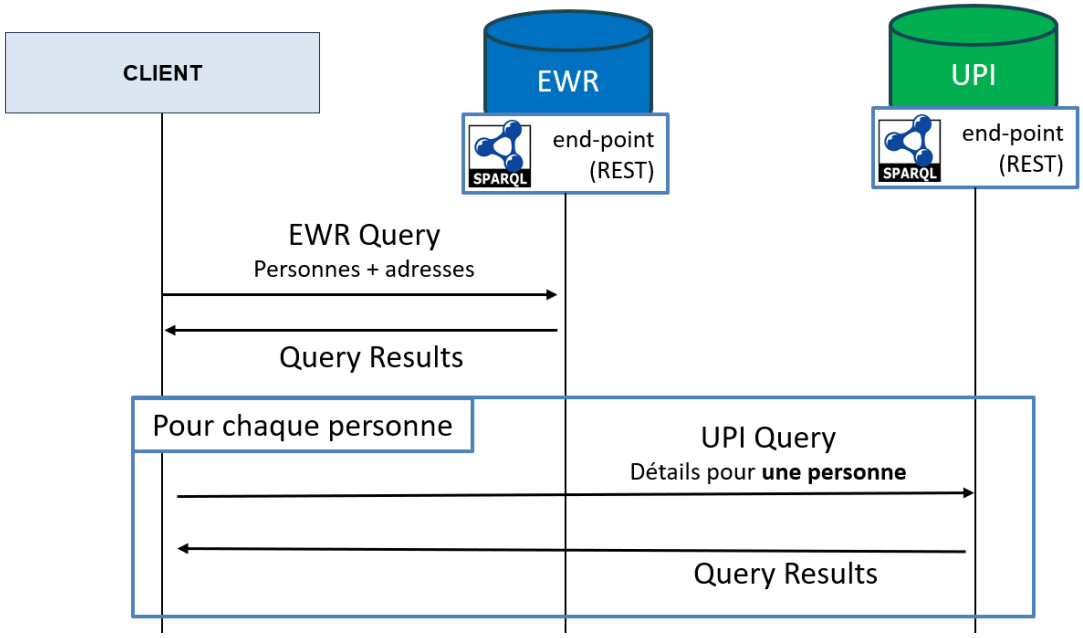

Figure 12 - Interrogation classique – Séquence

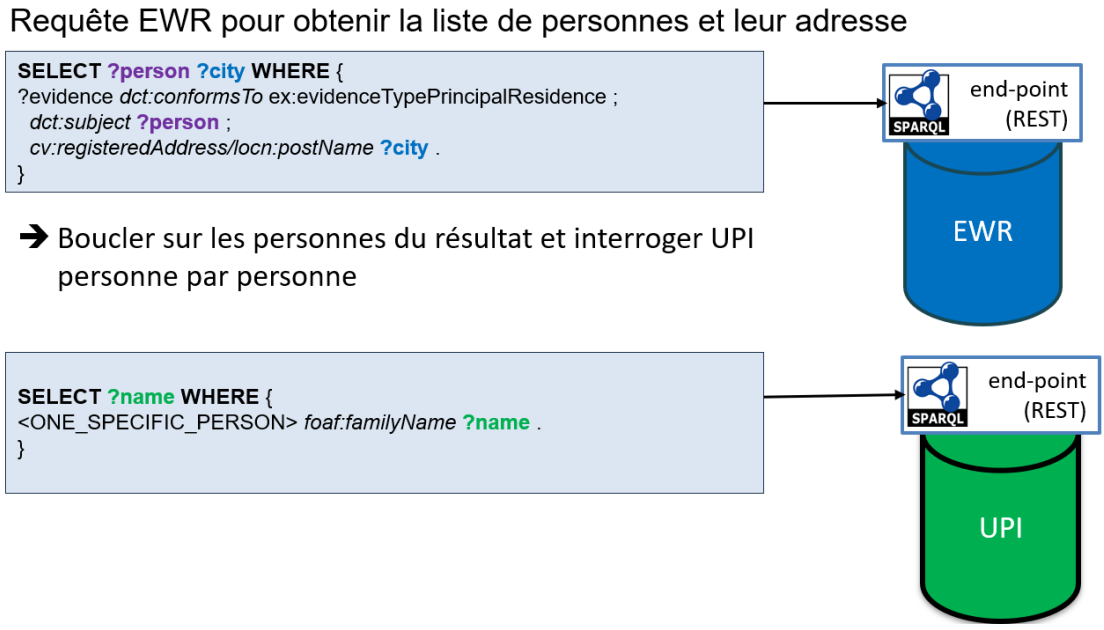

Figure 13 - Interrogation classique - Exemple

12.1.4 Performances

La performance de ces trois manières de faire devrait être analysée selon les outils utilisés, les triples stores utilisés, les SPARQL endpoints utilisés.

Lors de nos tests avec le prototype, la requête la plus performante est celle avec le mot-clé VALUES, puis le Federated Query, et finalement, logiquement, celle avec une requête par personne.

12.2 Déréférencement d'URL

Le code client démontre encore une autre chose : l'exécution d'une requête UPI pour obtenir les personnes et leur URL de Wikidata, puis un HTTP GET pour chaque URL Wikidata.

Cet exemple illustre un mécanisme important : l'utilisation des liens existants entre différents jeux de données dans le monde Linked Data, en exploitant ici la relation « rdfs:seeAlso » des données de base UPI.

Ceci est rendu possible par le choix de caractères fictifs jouant le rôle des personnes du registre UPI, ces caractères étant des caractères Disney qui ont des entrées dans Wikidata. Ceci permet aussi de montrer comment tirer des liens entre jeux de données lorsque des ressources n'ont pas forcément la même URL : en effet, entre le jeu de données EWR et UPI il est aisé de reconstituer l'identifiant d'une personne à partir de son numéro AVS et connaissant la manière de recréer l'URL d'une personne (`http://example.com/person/{numéro AVS}`). Le fait d'expliciter qu'une même ressource peut avoir des identifiants différents dans des jeux de données gérés de manière indépendante est une pratique commune en Linked Data, surtout que la réconciliation automatique de données est une tâche hasardeuse. On voit par exemple sur la page Wikidata de Mickey Mouse[89] les correspondances vers d'autres jeux de données (voir Figure 14 - Liens Wikidata vers d'autres jeux de données).

Figure 14 - Liens Wikidata vers d'autres jeux de données

[89] https://www.wikidata.org/wiki/Q11934

Une fois l'identifiant Wikidata obtenu, il serait tout à fait possible d'exécuter des requêtes SPARQL sur le endpoint Wikidata. Nous démontrons cependant une autre approche intéressante comme possibilité en Linked Data : le déréférencement d'URL, soit la possibilité d'accéder aux données d'une ressource en « ouvrant » simplement cette URL, comme nous avons l'habitude de le faire avec des URLs de pages Web classiques. Il s'agit d'un appel classique HTTP Get sur une URL.

L'avantage du déréférencement est que l'on sait où se trouve une ressource, l'identifiant de la ressource (URL avec L pour Locator) donnant également son emplacement. Le désavantage est que le requêtage d'un ensemble de ressources, une à une, est bien plus lent que d'exécuter une seule requête (voir ce que nous avons démontré ci-dessus dans le prototype). Finalement, ce qui peut être un avantage comme un inconvénient, et que le déréférencement va alors retourner l'ensemble des données concernant cette ressource : on ne peut pas demander seulement le nom et prénom, il faudra ensuite parcourir l'ensemble des résultats pour en extraire ce qui nous intéresse.

Dernière remarque : le fait de publier ou non les données des ressources sur leur URL respective est une recommandation pour le Linked Open Data, mais il s'agit d'un choix d'implémentation qui demande une certaine mise en œuvre technique (ce n'est pas une fonctionnalité native d'un triple store par exemple).

13 Intégration avec le travail de la BFH

Le travail de la BFH ajoute une couche supplémentaire au-dessus de ce qui est présenté dans ce rapport-ci. L'intégration fut donc simple à ce niveau, comme l'illustre le bloque Proxy de la « Figure 2 - Implémentation Linked Data pour le projet ». Merci de consulter le rapport de la BFH ainsi que leur dépôt GitHub du projet[90] pour plus d'information.

[90] https://github.com/swiss/ld-prototype-proxy

14 Au sujet du Swiss Personalized Health Network (SPHN)

Nous tenons à mentionner une belle mise en œuvre du Linked Data par le projet SPHN[91], dans le domaine de la recherche pour les soins de santé dans les hôpitaux universitaires. Le but étant d'uniformiser des données, les anonymiser, et les mettre à disposition des chercheurs.

Il s'agit d'un projet qui couvre toute la durée de vie de la donnée jusqu'à son hébergement dans un Triple Store :

- Définition d'ontologies par l'outil « schemaForge », qui permet à des utilisateurs ne connaissant pas le RDF de décrire des classes et des attributs dans un fichier Excel puis d'en générer une ontologie
- Transformation et intégration des données avec le « SPHN Connector »[92]
- Validation des données avec SHACL, incluant un l'outil « SHACLer » qui génère les fichiers SHACL à la volée à partir de la sémantique décrite dans les ontologies

Les différents outils sont accessibles dans leur Git[93].

Voici le résultat de notre analyse et discussions avec les auteurs :

- L'outil est basé sur l'ontologie SPHN RDF Schema[94] et les données SPHN Dataset[95], qui forment l'épine dorsale de la solution. Selon les auteurs, une adaptation de l'outil pour gérer d'autres types de données (ontologie plus générique, etc.), serait une tâche relativement complexe.
- L'outil gère spécifiquement des données des hôpitaux et des patients
  Le fournisseur de donnée est typiquement un hôpital, et les données d'un patient sont stockées dans le Triple Store dans un Named Graph qui lui est propre. L'organisation des données, dans les Named Graph propres aux patients, serait une contrainte dans une solution plus générique.
- Le SPHN Connector est basé sur le SPHN RDF Schema et en tire des contraintes sur les données qu'il traite. Ceci garantit que la transformation en RDF est cohérente et que les données RDF résultantes sont conformes à la sémantique SPHN. Un changement d'ontologie demanderait donc des modifications non triviales de ce RDFizer.
- Les auteurs ont relevé que le SPHN Connector est conçu pour traiter des données du patient, et que ces données ne proviennent que d'un seul fournisseur. Ceci peut être une contrainte sur l'organisation des données, comme la mention ci-dessus de gérer les données d'un patient dans un Named Graph spécifique.
- Le SPHN Connector peut actuellement traiter des données JSON et d'une base de données relationnelle. Il ne gère donc actuellement pas le XML qui pourrait être nécessaire dans notre cas.

[91] https://sphn.ch/network/data-coordination-center/the-sphn-semantic-interoperability-framework/
[92] https://sphn.ch/fr/2023/10/20/sphn-connector-mise-a-disposition-de-notre-outil-open-source/
[93] https://git.dcc.sib.swiss/sphn-semantic-framework
[94] https://www.biomedit.ch/rdf/sphn-schema/sphn
[95] https://sphn.ch/document/sphn-dataset/

- La génération d'URL se fait selon un schéma spécifique aux fournisseurs de données (Hôpitaux universitaires), ceci devrait être aussi adapté, de plus pour gérer différents types d'URLs dans notre cas.
- L'outil SHACler permet une génération automatique des fichiers SHACL de validation des données. Il se base sur les informations sémantiques de RDFS (v1.1) et OWL 2, plus spécifiquement le profil OWL 2 Expression Logic (EL). Pour tirer parti de cet outil, il est donc nécessaire de se baser sur des ontologies qui contiennent une certaine sémantique, ce qui est souvent le cas dans le domaine médical, beaucoup moins dans le Linked Data.

Nous avons donc analysé ces outils et discuté avec les auteurs afin de clarifier s'ils seraient réutilisables « sans trop de travail », dans le cadre de notre projet. La conclusion étant qu'il serait possible d'adapter les outils qui sont spécifiques au domaine de la santé (les ontologies, les données du patient, etc.), mais il n'est pas certain que cela serait un gain de temps par rapport à la réalisation d'une nouvelle solution, basée bien entendu elle aussi sur des outils existants comme nous l'avons démontré.

15 Conclusion

Dans notre rôle de Data Architect, nous présentons dans ce rapport différentes étapes de la durée de vie des données, à partir des données originales dans un format standard jusqu'à l'hébergement et au requêtage de ces données traduites en RDF/Linked Data.

Basé sur le cas d'utilisation Serafe, les données sources des personnes (registre UPI) et leurs adresses principales (registre EWR) ont été analysées par la lecture des documents en ligne, des normes eCH ainsi que des schémas (.xsd) disponibles. Ceci a permis de rechercher des schémas de données (ontologie) pour décrire les deux jeux de données spécifiques en Linked Data, et valider que les Core Vocabularies de l'Union européenne conviennent à cet effet. Nous avons démontré la transformation des données sources en Linked Data en utilisant des outils existants (RDFizers), configurés pour des données de différents formats (CSV et XML). Les données Linked Data sont alors validées avec les outils SHACL, puis hébergées dans deux Triple Store : un pour les données UPI, l'autre pour les données EWR. Ces deux Triple Store exposent des SPARQL endpoint qui permettent le requêtage sur les données, et nous proposons différentes manières d'interroger ces deux jeux de données de : requête SPARQL fédérée, plusieurs requêtes SPARQL en gérant la jointure des données coté client, déréférencement d'URLs.

Dans le cadre de ce projet, tout ce processus est indépendant de la mise à disposition finale des données, qu'il s'agisse de données ouvertes ou à accès restreint et contrôlé. C'est au niveau de la configuration du Triple Store et de son SPARQL end-point que l'accès aux données est géré, et dans ce projet le travail de la BFH ajoute une couche supplémentaire de gestion des accès par un proxy qui s'installe devant le SPARQL end-point, et qui n'influence donc pas le travail présenté dans ce rapport.

Mentionnons finalement que nous avons apporté une modeste contribution à l'écosystème des outils Linked Data, en reportant quelques bugs aux auteurs de différents outils (des RDFizers mais aussi le un fichier SHACL de l'Union européenne), bugs qui ont été corrigés dans la plupart des cas.

16 Annexe

16.1 Données source

16.1.1 Données UPI – CSV

Contenu du fichier UPI_Personnes_fiction.csv[96] :

```
WikidataURL;Universe;familyName;givenName;sex;birthDate;AVSNr
Q111082709;Disney;Mouse;Margie;female;1912-10-22;7565626146561
Q112223829;Disney;Goat;Gideon;male;1937-03-18;7563797564085
Q2334015;Disney;Mouse;Markus;male;1939-12-06;7566939230183
Q27304877;Disney;Robin;Christopher;male;1945-11-28;7566422590732
Q3437147;Disney;Hood;Robin;male;1947-08-29;7569468062614
Q11934;Disney;Mouse;Mickey;male;1949-12-14;7565691946349
Q11936;Disney;Mouse;Minerva;female;1951-05-07;7564934397498
Q11936;Disney;Mouse;Minnie;female;1972-05-10;7563038909224
Q11937;Disney;McDuck;Scrooge;male;1976-04-22;7561230860077
Q6550;Disney;Duck;Donald;male;1977-01-20;7562378067595
Q104869376;fiction;Armstrong;Tricia;female;1991-04-12;7562186711192
Q100927311;fiction;Tataroglu;Inci;female;1991-07-30;7564098604807
Q101052856;fiction;Caldwell;Alan;male;1996-03-12;7568849389654
Q101052928;fiction;McMullen;Jessie;male;1998-01-21;7565949876756
Q101053041;fiction;Pierce;Edward;male;1999-07-30;7565554356834
Q101062929;fiction;Richmond;Anthony;male;2006-05-23;7563100078216
Q101062929;fiction;Richmond;Tony;male;2009-04-23;7563429487423
Q101063012;fiction;Rutland;Mark;male;2016-09-29;7564386875681
Q101068819;fiction;Armstrong;Paul;male;2016-10-14;7561256952909
Q101072692;fiction;Bradley;Paul;male;2019-07-02;7562907209281
Q101078234;fiction;Meredith;Douglas;male;1922-06-06;7560605260238
Q101096123;fiction;Miller;Reno;male;1978-07-03;7562629102143
Q101112186;fiction;Campbell;Robert;male;1996-12-31;7562966486647
Q101112187;fiction;Hale;Patrick;male;2010-03-31;7569629683115
```

16.1.2 Données UPI – XML

Contenu du fichier persons-eCH0044.xml[97] :

```
<?xml version="1.0" encoding="utf-8"?>
<personIdentificationRoot xmlns="http://www.ech.ch/xmlns/eCH-0044/4"
xsi:schemaLocation="http://www.ech.ch/xmlns/eCH-0044/4 schema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <personIdentification>
    <vn>9791234567890</vn>
    <localPersonId>
        <personIdCategory>LOC</personIdCategory>
        <personId>EMP001234</personId>
    </localPersonId>
    <officialName>Parr</officialName>
    <firstName>Violet</firstName>
```

[96] https://github.com/swiss/ld-prototype-data/blob/main/UC-Serafe/UPI-Dataset-fiction/UPI_Personnes_fiction.csv
[97] https://github.com/swiss/ld-prototype-data/blob/main/XML2RDF/sparql-generate/persons-eCH0044.xml

```
  <originalName>Doe</originalName>
  <sex>2</sex>
  <dateOfBirth>
      <yearMonthDay>1990-05-15</yearMonthDay>
  </dateOfBirth>
</personIdentification>
<personIdentification>
  <vn>9799876543210</vn>
  <localPersonId>
      <personIdCategory>LOC</personIdCategory>
      <personId>P98765432</personId>
  </localPersonId>
  <officialName>Sullivan</officialName>
  <firstName>James</firstName>
  <sex>1</sex>
  <dateOfBirth>
      <yearMonthDay>1985-10-12</yearMonthDay>
  </dateOfBirth>
</personIdentification>
<personIdentification>
  <vn>9795555555555</vn>
  <localPersonId>
      <personIdCategory>LOC</personIdCategory>
      <personId>NID345678</personId>
  </localPersonId>
  <officialName>Corona</officialName>
  <firstName>Rapunzel</firstName>
  <sex>2</sex>
  <dateOfBirth>
      <yearMonthDay>1993-07-23</yearMonthDay>
  </dateOfBirth>
</personIdentification>
 <personIdentification>
  <vn>9792222222222</vn>
  <localPersonId>
      <personIdCategory>LOC</personIdCategory>
      <personId>D78901234</personId>
  </localPersonId>
  <officialName>Parr</officialName>
  <firstName>Robert</firstName>
  <sex>1</sex>
  <dateOfBirth>
      <yearMonthDay>1975-02-03</yearMonthDay>
  </dateOfBirth>
</personIdentification>
<personIdentification>
  <vn>9793333333333</vn>
  <localPersonId>
      <personIdCategory>LOC</personIdCategory>
      <personId>979.3333.4444.55</personId>
  </localPersonId>
  <officialName>Porter</officialName>
  <firstName>Jane</firstName>
  <sex>2</sex>
  <dateOfBirth>
```

```
        <yearMonthDay>1992-03-21</yearMonthDay>
    </dateOfBirth>
  </personIdentification>
</personIdentificationRoot>
```

16.1.3 Données EWR – CSV

Contenu du fichier EWR_ResidencesPrincipales.csv[98]

```
id;personAvsNr;street;streetNr;postalCode;city;startDate;endDate
1234567890;7565626146561;Rue de la Montagne;14;3000;Berne;1945-08-15;1953-10-20
2345678901;7563797564085;Avenue de la Liberté;32;3010;Berne;1998-04-22;2009-06-18
3456789012;7566939230183;Boulevard des Alpes;7;3020;Berne;1950-11-07;1959-12-03
4567890123;7566422590732;Place du Marché;5;3030;Berne;2005-09-18;2008-11-29
5678901234;7569468062614;Rue de la Paix;21;3040;Berne;1994-12-30;2006-05-12
6789012345;7565691946349;Avenue des Fleurs;10;3050;Berne;2008-07-03;2010-02-15
7890123456;7564934397498;Rue des Champs;3;3060;Berne;1999-10-11;2004-09-07
8901234567;7563038909224;Boulevard de l'Europe;88;3070;Berne;2007-02-28;2010-11-24
9012345678;7561230860077;Place de la Gare;15;3080;Berne;2001-06-14;2009-08-30
123456789;7562378067595;Rue du Lac;42;3090;Berne;1997-03-25;2005-04-19
9876543210;7562186711192;Avenue du Soleil;77;3100;Berne;2009-12-05;2010-10-10
8765432109;7564098604807;Chemin des Érables;25;3110;Berne;2004-08-19;2010-03-27
7654321098;7568849389654;Rue des Roses;6;3120;Berne;2000-01-10;2007-07-02
6543210987;7565949876756;Boulevard des Arts;19;3130;Berne;2010-07-28;2015-11-14
5432109876;7565554356834;Place de la Cathédrale;3;3140;Berne;2003-10-02;2010-09-08
4321098765;7563100078216;Rue des Vignes;8;3150;Berne;2015-05-14;2020-01-25
3210987654;7563429487423;Avenue de la République;11;3160;Berne;2013-04-17;2019-04-30
2109876543;7564386875681;Chemin du Moulin;2;3170;Berne;2016-09-29;2018-09-29
1098765432;7561256952909;Rue de la Croix;30;3180;Berne;2016-10-14;2019-10-14
987654321;7562907209281;Boulevard de la Plage;14;3190;Berne;2019-07-02;2019-10-02
9876543211;7560605260238;Place du Palais;9;3200;Berne;1932-06-06;1942-06-06
8765432108;7562629102143;Rue du Paradis;20;3210;Berne;2004-03-08;2010-08-14
7654321095;7562966486647;Avenue des Cèdres;6;3220;Berne;1996-12-31;2002-03-06
6543210984;7569629683115;Chemin de la Montagne;18;3230;Berne;2010-03-31;2013-03-31
```

16.2 Requêtes SPARQL du prototype

16.2.1 Une seule requête SPARQL: Federated Query

```
SELECT ?person ?city ?name WHERE {
?evidence dct:conformsTo ex:evidenceTypePrincipalResidence ;
 dct:subject ?person ;
 cv:registeredAddress/locn:postName ?city .
SERVICE <http://upi-endpoint.ch> {
  ?person foaf:familyName ?name .
  }
}
```

[98] https://github.com/swiss/ld-prototype-data/blob/main/UC-Serafe/EWR-Dataset-fiction/EWR_ResidencesPrincipales.csv

16.2.2 Une requête EWR puis une seule requête UPI

Requête EWR pour obtenir la liste de personnes et leur adresse

```
SELECT ?person ?city WHERE {
?evidence dct:conformsTo ex:evidenceTypePrincipalResidence ;
  dct:subject ?person ;
  cv:registeredAddress/locn:postName ?city .
}
```

Utilisation de cette liste pour le mot-clé VALUES: limiter les résultats possibles de la deuxième requête

```
SELECT ?person ?name WHERE {
VALUES ?person { PERSON_VALUES_TO_REPLACE }
?person foaf:familyName ?name .
}
```

16.2.3 Une requête EWR puis une requête UPI par personne

Requête EWR pour obtenir la liste de personnes et leur adresse

```
SELECT ?person ?city WHERE {
?evidence dct:conformsTo ex:evidenceTypePrincipalResidence ;
  dct:subject ?person ;
  cv:registeredAddress/locn:postName ?city .
}
```

Boucler sur les personnes du résultat et interroger UPI personne par personne

```
SELECT ?name WHERE {
<ONE_SPECIFIC_PERSON> foaf:familyName ?name .
}
```