Université de Montpellier — Master Informatique

Traitement sémantique des données (HAI824I)

TP : Système OBDA avec Obi-wan

Dans ce TP, nous allons utiliser **Obi-wan** un système d'accès aux données basé sur une ontologie (OBDA), qui a été développé dans le cadre de la thèse de Maxime Buron. Obi-wan permet l'interrogation de **sources de données hétérogènes** (actuellement, PostgreSQL, SQLite, MongoDB, Jena TDB) via un **niveau ontologique en RDFS**. Les mappings permettent d'associer une requête sur une base de données source à un ensemble de triplets RDF utilisant le vocabulaire de l'ontologie.

Obi-wan intègre un certain nombre d'outils existants, comme le médiateur Tatooine (`https://team.inria.fr/cedar/tatooine/`) capable de traiter des plans de requêtes portant sur plusieurs bases de données hétérogènes, et un algorithme de reformulation de requête avec des règles provenant du moteur de règles Graal (`https://graphik-team.github.io/graal/`).

L'interrogation peut reposer soit sur la matérialisation totale (création d'une base de faits avec les mappings et saturation de cette base), soit sur la réécriture totale (reformulation de la requête avec l'ontologie et les règles d'implication de RDFS puis réécriture de la requête obtenue avec les mappings), soit sur des techniques mixtes. Ce TP est effectué par défaut avec l'approche de **reformulation totale** (appelée REW-CA dans Obi-wan), mais vous pouvez tester d'autres approches.

Le but de ce TP est de construire un système OBDA simplifié sur le thème de Star wars, en utilisant deux bases de données :

- l'une consacrée aux personnages de star wars et aux objets (véhicules, armes) que ces personnages utilisent ;
- l'autre consacrée aux films - cette deuxième base de données a été extraite de IMDb (`https://www.imdb.com/interfaces/`), avec des simplifications dans les tables choisies et une restriction aux films de Star Wars pour obtenir une petite base plus facile à explorer.

Les deux bases de données sont en SQLite, un système léger de gestion de BD relationnelles qui a l'avantage d'être intégré à Linux (et MacOS) et qui ne nécessite pas de service fonctionnant en arrière plan.

Pour en savoir plus sur Obi-wan et les techniques qu'il implémente :

- Site wiki d'Obi-wan : `https://gitlab.inria.fr/cedar/obi-wan/-/wikis/Home`
- Démo à la conférence internationale VLDB (Very Large Data Bases) : `https://hal.inria.fr/hal-02921434/file/main.pdf`
- Thèse de Maxime Buron : `https://hal.inria.fr/tel-03107689/`

Pour commencer, téléchargez ces 2 fichiers :

- le fichier `tp-obiwan.zip` depuis moodle. Une fois décompressé, vous obtenez un répertoire `tp-obiwan` qui contient un projet Obi-wan appelé `sw-ris` (RIS pour "RDF Integration System").
- le fichier JAR exécutable de Obi-Wan, disponible ici : `https://seafile.lirmm.fr/f/8da61b3243534b45a269/`. Mettez le fichier `obiwan.jar` dans le répertoire `tp-obiwan`.

1 Structure d'un projet Obi-wan

Ouvrez le répertoire `tp-obiwan/sw-ris` pour examiner la structure d'un projet Obi-wan.

De façon générale, un projet Obi-wan contient les 4 fichiers suivants (en italiques, des noms indicatifs que vous pouvez changer, seule `l'extension` du fichier est importante) :

- *obi-wan*`.properties` : le fichier qui pilote l'application
- *ontology*`.nt` : l'ontologie RDFS au format N-triples
- *ris*`.json` : les mappings au format JSON
- un fichier `querysession.properties` qui est nécessaire pour des raisons techniques quand on utilise l'approche de matérialisation mais dont vous n'aurez pas à vous préoccuper. Ne pas modifier ce fichier.

A noter : Le fichier `obi-wan.properties` spécifie entre autres le nom du fichier qui contient les mappings (ici `ris.json`) et la stratégie de réponse aux requêtes (diverses stratégies qui vont de la matérialisation totale à la réécriture totale) :

`obiwan.ris.file = ris.json`
`obiwan.qastrategy.type = REW_CA` (par exemple)

Quand vous lancerez Obi-wan, un sous-répertoire de nom `sessions` sera créé automatiquement et contiendra des informations sur les interrogations que vous aurez effectuées.

Pour ce TP, on a ajouté également dans `tp-obiwan/sw-ris` les deux bases de données utilisées :

- `sw_bank.db` sur les personnages de Star Wars
- `sw_imdb.db` sur les films.

2 Ontologie RDFS

L'ontologie RDFS introduit des classes et des propriétés, leurs liens de sécialisation (sous-classe ou sous-propriété) et des informations de signature des propriétés (ici, `range` uniquement).

Ouvrez le fichier `ontology.nt` avec un éditeur de texte. Vous y trouvez le squelette d'ontologie ci-dessous, où nous avons enlevé les préfixes des IRIs pour que ce soit plus lisible :

```
<LightSaber> <subClassOf> <Object> .
<StarShip> <subClassOf> <Vehicle> .
<Vehicle> <subClassOf> <Object> .

<pilotOf> <subPropertyOf> <uses> .
<usesWeapon> <subPropertyOf> <uses> .

<pilotOf> <range> <Vehicle> .
<uses> <range> <Object> .
```

Remarque : Pour que la visualisation graphique de l'ontologie (voir plus loin) fonctionne correctement, il faut sur chaque ligne du fichier .nt un **espace avant le point final**.

3 Mappings

De façon abstraite, les mappings sont des objets de la forme *name* : *body* → *head*, où :

- *name* est le nom du mapping ;
- *body* décrit une requête sur une source de données et un modèle (*template*) de construction des ressources RDF (IRIs ou valeurs littérales) à partir des valeurs des variables réponses ; à chaque ressource RDF, on peut associer directement une variable réponse (on aura une valeur littérale) ou bien construire un IRI.
- *head* décrit la structure des triplets RDF obtenus, où les variables (dénotées ici par un $) seront remplacées par les ressources RDF correspondantes selon le template. Si une variable n'a pas de correspondance dans le template, elle sera remplacé par un nouveau noeud blanc (blank node).

De façon concrète, l'ensemble des mappings est décrit avec une syntaxe JSON. Le fichier de mappings `ris.json` décrit 4 mappings (de nom `pilot`, `jedi`, `person` et `title`) et des informations sur l'environnement (connexion à des sources de données, nom de l'ontologie, etc.). Vous pouvez visualiser ce fichier avec un éditeur de texte, mais pour avoir une coloration syntaxique et un contrôle de la syntaxe (ce sera très utile quand vous écrirez vos propres mappings !), nous vous conseillons d'utiliser un éditeur de code (par exemple Visual Code).

Ci-dessous une partie de ce fichier (1 mapping et les informations d'environnement). Le mapping a pour nom `pilot` et sa requête SQL interroge la table `vehicle` de la base de données `SW_BANK` en faisant une projection sur l'attribut `character` (variable réponse) ; la partie `templates` décrit comment la variable `char_iri` est construite à partir de la variable réponse `character` ; à partir de chaque réponse à la requête SQL, ce mapping produit deux triplets, où `$char_iri` est remplacé par l'IRI construite à partir de la réponse, et `$z` qui n'a pas de variable réponse associée est remplacé par un noeud blanc .

```
{
  "mappings": [
    {
      "name": "pilot",
      "head":
        [ "$char_iri", "<https://www.starwars.com/databank/pilotOf>", "$z" ],
        [ "$z", "<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>",
                                "<https://www.starwars.com/databank/StarShip>" ]
      ],
      "body": {
        "templates": {
          "char_iri": "<https://www.starwars.com/databank/{character}>"
        },
        "query": "select character from vehicle where vehicleType='starship' ",
        "datasource": "SW_BANK"
      }
    },
      ..... Autres mappings .....
  ],
  "datasources": [
    {
      "name": "SW_BANK",
      "type": "SQLITE",
      "parameters": {
        "databasePath":"./sw_bank.db"
      }
  },
  ..... Autres sources de données .....
```

```
  ],
  "name": "Star Wars Tutorial",
  "description": "Small integration of two sources about Star Wars movies and characters",
  "RDFSRuleSet": "RDFS",
  "ontology": "ontology.nt"
}
```

A noter : Les requêtes SQL utilisées dans les mappings doivent avoir la forme `SELECT ... FROM ... WHERE ...` où `WHERE` ne contient pas de requête imbriquée.

4 Utilisation du serveur Obi-wan

Pour **démarrer le serveur Obi-wan**, ouvrez un terminal, placez-vous dans le répertoire `sw-ris` et exécutez la commande suivante

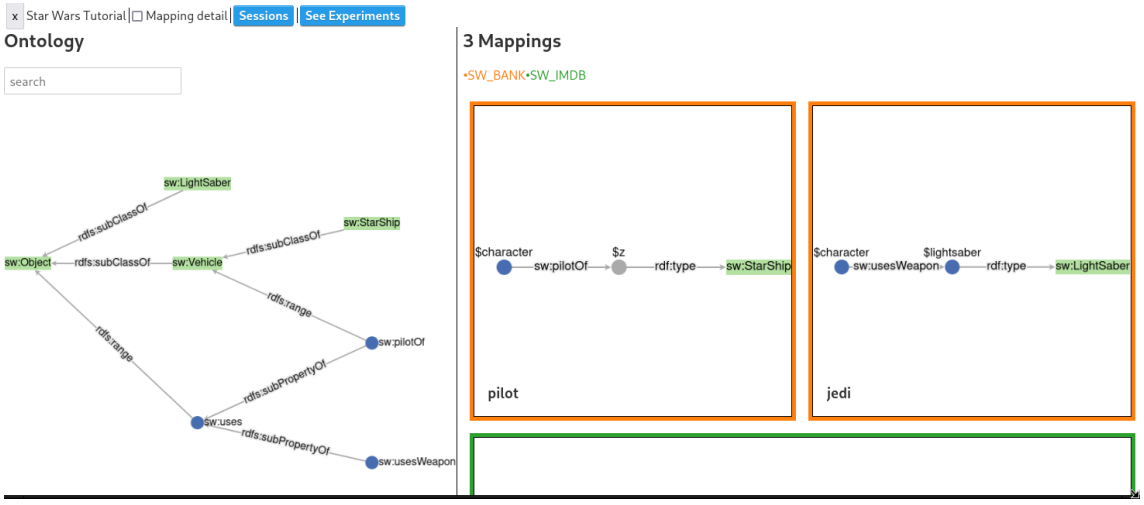
```
java -jar ../obiwan.jar server obi-wan.properties
```

Il vous faut indiquer le chemin d'accès à `obiwan.jar` (on suppose ici que le fichier est placé dans `tp-obiwan`).

Une fois le serveur lancé, vous avez accès à une visualisation graphique du système d'intégration (RIS) et une interface d'interrogation en SPARQL (SPARQL end point) qui sont présentées dans les deux sections suivantes.

4.1 Visualisation du système d'intégration

Cette visualisation est accessible ici : `http://localhost:8080/ris/sw-ris/index.html`. L'ontologie est dessinée à gauche et les mappings à droite comme dans l'exemple ci-dessous.



4.2 SPARQL endpoint

L'interface de requêtage SPARQL est accessible ici : `http://localhost:8080/client/index.html`. L'interface est séparée en deux ; en haut, un éditeur de requêtes SPARQL et en bas, un tableau des réponses.

Pour faciliter l'écriture des requêtes SPARQL, vous pouvez indiquer les préfixes suivants au-dessus de la requête (les deux premiers vous sont déjà fournis, faites un copier-coller des deux autres qui sont propres à votre projet).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sw: <https://www.starwars.com/databank/>
PREFIX imdb: <http://www.imdb.com/>
```

Ceci va permettre d'utiliser des IRI abrégés, par exemple `rdf:type` au lieu de `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`.

Exécutez d'abord la requête qui permet de visualiser tous les triplets RDFS du système d'intégration, c'est-à-dire tous ceux obtenus par saturation de l'ontologie et de la base de faits issue des mappings (c'est la requête qui devrait vous être proposée par défaut). Examinez ces différents triplets.

```
SELECT ?sub ?pred ?obj WHERE {
?sub ?pred ?obj .
}
```

Essayez ensuite des requêtes plus sélectives et examinez leur résultat. Par exemple celle-ci :

```
SELECT ?x ?y WHERE {
    ?x sw:uses ?z .
    ?z rdf:type ?y .
    ?y rdfs:subClassOf sw:Object .
}
```

Remarque. Ce SPARQL end point ne sait gérer que des requêtes de type "Basic Graph Pattern" (requêtes conjonctives).

Le système d'intégration (`sw-ris`) contient un répertoire de nom *sessions* qui stocke des informations sur toutes les sessions de requpetage (une session correspond à un lancement du serveur). Pour chaque session et chaque évaluation de requête effectuée durant cette session on trouve : la requête, les réponses obtenues, les reformulations avec l'ontologie (si applicable), les réécritures avec les mappings (si applicable), le plan d'exécution logique de la requête transmis à Tatooine, sa version optimisée (où les opérateurs relationnels ont été dans la mesure du possible "poussés" dans les requêtes SQL sur la source SQLite) et sa version physique (avec des opérateurs plus précis que la version logique).

Vous pouvez visualiser les sessions via le bouton "sessions" de l'interface graphique du RIS.

5 Les bases de données du TP

La base de données `sw_bank.db` contient 2 tables :

- `vehicle`
  - `character` : the name of the driving character
  - `vehicleType` : the type of vehicule driven by the character
- `lightsaber`
  - `character` : the name of the character using the weapon
  - `saber` : the name of the lightsaber

La base de données `sw_imdb.db` contient 3 tables :

- `title`
  - `tconst` : alphanumeric unique identifier of the title

  - `primaryTitle` : the more popular title
  - `startYear` : represents the release year of a title
  - `runtimeMinutes` : primary runtime of the title, in minutes
- `person`
  - `nconst` : alphanumeric unique identifier of the name/person
  - `primaryName` : name by which the person is most often credited
  - `birthYear` : in YYYY format
  - `deathYear` : in YYYY format if applicable, else NULL
- `casting`
  - `tconst` : alphanumeric unique identifier of the title
  - `nconst` : alphanumeric unique identifier of the name/person
  - `category` : the category of job that person was in ('actor', 'actress', 'director' or 'composer')
  - `character` : the name of the character played if applicable, else NULL

Vous pouvez explorer ces bases de données en utilisant SQLite, qui en général est déjà installé sous Linux et MacOS : taper la commande sqlite3 dans un terminal pour le vérifier.

Tutoriel sur SQL Lite : `https://www.tutorialspoint.com/sqlite/index.htm`

En particulier, essayez ceci (en vous plaçant dans le répertoire qui contient la base de données) :

`sqlite3` (ceci ouvre l'invite de commande de SQLite : vous pouvez ensuite taper des commandes commençant par un point, ou des commandes SQL qui doivent impérativement se terminer par un point virgule)
`.open xxx.db` (ceci ouvre la base de données xxx ou la crée si elle n'existe pas)
`.tables` (ceci affiche les noms des tables de la base)
`.schema` (ceci affiche les requêtes SQL de création du schéma de la base et permet de voir quelles sont les tables et leurs attributs)
`.header on` (ceci met les noms des attributs quand les réponses sont affichées)
`SELECT * FROM table_name;` (en mettant le nom de la table voulue : ceci affiche le contenu de la table ; ne pas oublier le point virgule à la fin)
`.quit` (ceci quitte SQLite)

6 A vous de jouer !

Vous allez maintenant étendre le système d'intégration.

Attention : A chaque fois que vous modifiez les mappings (fichier `ris.json`) ou l'ontologie (fichier `ontology.nt`), vous devez tuer le serveur Obi-wan (Crtl C par exemple) et relancer le serveur pour que les changements soient pris en compte.

```
java -jar ../obiwan.jar server obi-wan.properties
```

6.1 Complétion de l'ontologie

Ajoutez la classe PodRacer ("module de course") qui est une sous-classe de véhicule.

Précisez également que la propriété *uses* a pour domaine Character (personnage). Est-il nécessaire d'indiquer que les propriétés *pilotOf* et *usesWeapon* ont pour domaine Character (et pourquoi ?) ?

6.2 Ajout d'un mapping

Le premier mapping du fichier `ris.json` permet de récupérer les pilotes d'objets (anonymes) de type vaisseau spatial (StarShip). Et c'est le seul type de véhicule qu'on récupère.

Testez-ce mapping avec une requête SPARQL qui demande tous les x et z tels que x pilote quelque chose de type z.

Remarquez au passage que vous ne pouvez pas récupérer des noeuds "blancs" : si vous demandez tous les x, y et z tels que x pilote y de type z, vous n'obtenez plus aucune réponse.

Ajoutez un mapping de nom "pilot2" similaire à "pilot" qui permet de récupérer également les objets de la classe PodRacer.

Conseil : Commencez par copier le mapping "pilot" puis modifiez la copie.

Obtenez-vous de nouvelles réponses à votre requête ? [N'oubliez pas de relancer le serveur Obi-wan pour que vos modifications soient prises en compte.]

6.3 Modification d'un mapping

Nous nous intéressons maintenant aux mappings depuis la BD `sw_imdb.db`.

Editez le dernier mapping (de nom "title") et ajoutez de quoi récupérer aussi l'année de sortie du film via une propriété dont l'IRI est `<http://www.imdb.com/releaseYear>`.

Il vous faut ajouter un triplet dans la tête, une variable réponse (attribut) dans la requête SQL et préciser la partie template.

Relancez le serveur. Vous devez par exemple obtenir une réponse à la requête suivante :

```
SELECT * WHERE {
  ?movie imdb:title ?title .
  ?movie imdb:releaseYear 1977 .
}
```

6.4 Lier les acteurs aux films

Créez un mapping de nom “actress” qui, à partir de la table `casting` de `sw_imdb.db`, extrait les actrices et les films dans lesquels elles jouent. Les triplets produits utilisent une propriété dont l'IRI est `<http://www.imdb.com/actressIn>`.

Répondez à la requête suivante (“quelle actrice joue dans quel film ? ”) :

```
SELECT ?name ?title WHERE {
  ?actress imdb:fullName ?name .
  ?actress imdb:actressIn ?movie .
  ?movie imdb:title ?title .
}
```

Faites un mapping similaire pour les acteurs masculins (en utilisant une propriété d'IRI `<http://www.imdb.com/maleActorIn>`), de façon à pouvoir répondre à la requête suivante :

```
SELECT ?name ?title WHERE {
  ?actor imdb:fullName ?name .
  ?actor imdb:maleActorIn ?movie .
  ?movie imdb:title ?title .
}
```

6.5 Généralisation de propriétés

Nous voudrions savoir qui joue dans quel film en utilisant une seule requête SPARQL au lieu d'avoir une requête pour les actrices et une autre pour les acteurs masculins. Pour cela ajouter dans l'ontologie une propriété `imdb:actorIn` qui généralise les deux propriétés `imdb:actressIn` and `imdb:maleActorIn` (attention, il vous faut indiquer le préfixe en entier dans l'ontologie).

Vérifiez que vous obtenez le comportement attendu avec la requête :

```
SELECT ?name ?title WHERE {
  ?actor imdb:fullName ?name .
  ?actor imdb:actorIn ?movie .
  ?movie imdb:title ?title .
}
```

6.6 Lier les acteurs aux personnages

Ajoutez un mapping à partir de la table `casting` de `sw_imdb.db` qui permette de lier (un identifiant d') acteur - homme ou femme - au personnage qu'il ou elle joue via la propriété `imdb:plays`.

Indication : Si besoin, vous pouvez utiliser dans la requête SQL la syntaxe suivante : `category in ('actor', 'actress')`. Ceci évite d'avoir à créer deux mappings.

Vous devez par exemple pouvoir répondre à une requête de cette forme :

```
SELECT ?actorName ?character WHERE
{ ?actor imdb:fullName ?actorName .
?actor imdb:plays ?character .
?character sw:uses ?object .
?object rdf:type sw:StarShip .
}
```

Ecrivez les deux requêtes SPARQL suivantes :

- Quels sont tous les personnages ?
- Quels sont les acteurs qui jouent Dark Wador (Darth Wader en anglais).

6.7 Lier les personnages aux films

On veut retrouver quels personnages apparaissent dans quels films. Pourquoi la requête suivante n'est-elle pas correcte (au sens où elle ne retourne pas ce qu'il faut) :

```
SELECT  ?character ?movieTitle
WHERE {
  ?actor imdb:plays ?character .
  ?actor imdb:actorIn ?movie .
  ?movie imdb:title ?movieTitle .
}
```

Proposez une façon de faire.

Précision de syntaxe des mappings. Si votre solution implique un mapping dans lequel il y a une jointure SQL à faire (donc plusieurs tables), il faut préfixer le nom de chaque attribut par le nom de sa table (T.A pour un attribut A d'une table T) dans la requête SQL et dans les templates qui utilisent ces attributs.

6.8 Finalisation de l'ontologie

Insérez les classes `imdb: Movie` et `imdb: Person` dans votre ontologie, de façon à ce qu'elles contiennent respectivement les identifiants de films et de personnes. Il s'agit de faire apparaitre ces classes dans l'ontologie, vous n'avez pas à ajouter de mappings.

Vérifiez par une ou des requête(s) SPARQL que vous avez effectivement les instances de Movie et de Film attendues.

6.9 Sous le capot

Considérons cette requête SPARQL très simple :

```
SELECT ?name WHERE
{ ?actor imdb:fullName ?name .
?actor imdb:actorIn ?movie .
?movie imdb:title "Star Wars: Episode IV - A New Hope" .
}
```

Nous avons choisi la stratégie d'interrogation qui reformule la requête avec l'ontologie, puis réécrit les reformulations avec les mappings (option "REW-CA" dans le fichier obiwan. properties).

Quelles sont les reformulations de cette requête avec l'ontologie, puis les réécritures avec les mappings ?

Essayez de répondre à cette question en considérant votre ontologie et vos mappings, puis vérifiez votre réponse en vous aidant du répertoire "sessions" : explorez directement ce répertoire ou utilisez la visualisation du système d'intégration (bouton "sessions"). Remarquez au passage qu'Obi-wan associe un plan d'exécution à la réécriture obtenue, puis optimise ce plan, et finalement l'implémente avec des opérations précises (par exemple "join" est remplacé par un algorithme de "join" particulier).