Université de Genève

Technologies du web sémantiques
D400009

TP 2: Ontological Skiing

Author: Sajaendra Thevamanoharan

E-mail: sajaendra.thevamanoharan@etu.unige.ch

Author: Deniz Sungurtekin

E-mail: deniz.sungurtekin@etu.unige.ch

Décembre 2020

UNIVERSITÉ
DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Introduction

Le but de ce TP est de modéliser une grammaire représentant une station de Ski. En informatique et en science de l'information, une ontologie est l'ensemble structuré des termes et concepts représentant le sens d'un champ d'information, que ce soit par les métadonnées d'un espace de noms, ou les éléments d'un domaine de connaissances. Nous allons comprendre le fonctionnement de l'ontologie à travers cet exemple de station de Ski.

Ce TP se divise en trois grandes parties. Premièrement nous allons expliquer la modélisation de cette ontologie avec l'editeur nommé Protégé. Deuxièmement, nous allons créer une base de connaissance que nous allons manipuler avec SPARQL. Finallement, nous allons combiner les requêtes SPARQL avec les inferrences OWL.

Modélisation

On possède plusieurs classes pour la connaissance d'une station de Ski : SkiRun, BlackRun, RedRun, BlueRun, Skier, GoodSkier, AverageSkier, BeginnerSkier, SkiLift, Place, Restaurant.

Les propriétés à définir sont :

- canSkiOn (lien entre Skier et Skirun)
- startsAt (La place où débute un Skirun ou Skilift)
- isStartAt (l'inverse de startsAt)
- endsAt (la place où finit un Skirun ou Skilift)
- isEndof (inverse de endsAt)
- locatedAt (location d'un restaurant)
- isLocationOf (inverse de locatedAt)

La modélisation et la validation se font en deux étapes. Premièrement, nous allons créer plusieurs axiomes pour définir la sémantique de ces classes et de ces propriétés. Une fois l'ontologie définie, on vérifie la consistance à l'aide de OWL reasoner.

En analysant les classes, on peut facilement y retrouver une hiérarchie. On présente cette hiérarchie comme telle (version Protégé App) :

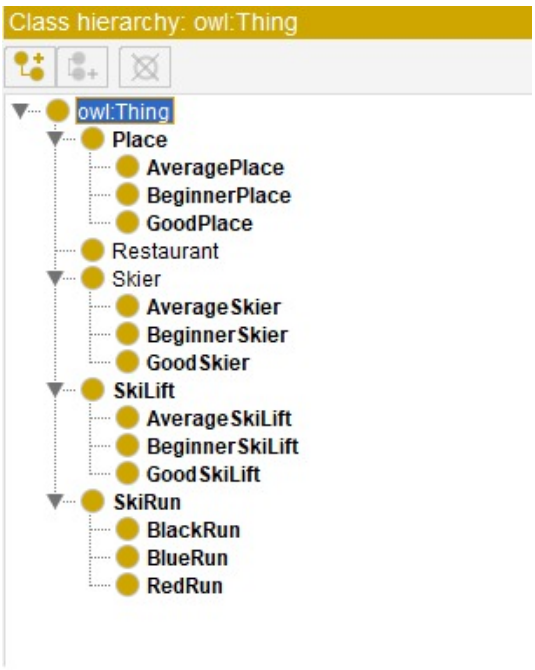

Figure 1: Hiérarchie des classes (version Protégé App)

Les propriétés

On va mainteant montrer comment on définit les propriétés dans l'application Protégé.

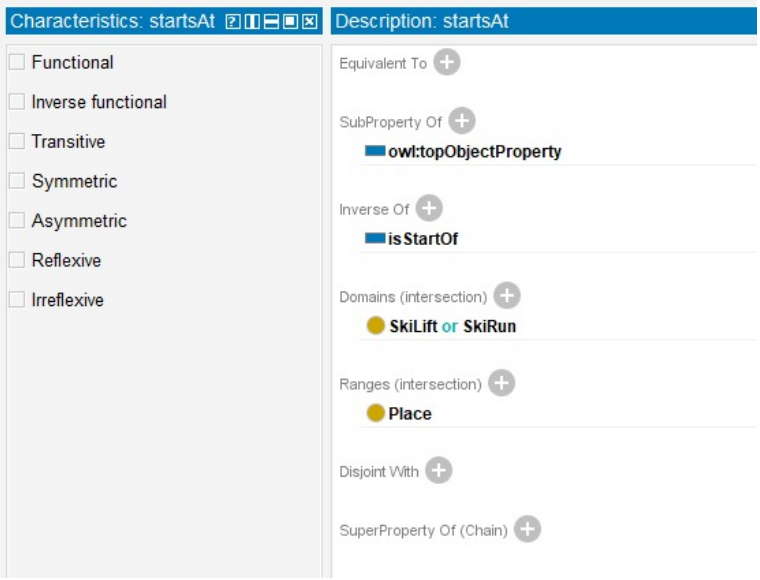

Figure 2: Définition d'une propriété sur Protégé

Les deux paramètres intéressants sont Domaines et Ranges. Il peuvent se définir comme l'ensemble de départ et d'arrivé d'une fonction. L'image ci-dessus est un exemple de la propriété startsAt. La propriété startsAt est en effet unSkiRun ou un Skilift qui débute sur une place. Donc le domaine est soit un Skirun ou un Skilift et le range, la classe Place.

Axiomes

On va présenter maintenant les axiomes que nous avons mis en place pour construire la logique de cette base de connaissane.

- Un Skirun ou un Skilift débute et finit exactement à une place.
- Une place qui est le début d'un Skirun est nécessairement la fin d'un Skirun ou d'un Skilift.
- S'il y a un restaurant à une place, alors un Skilift fini à cette place.
- Un BeginnerSkier peut skier uniquement sur les BlueRuns.
- Un AverageSkier peut skier uniquement sur les RedRuns ou BlueRuns.
- Un GreatSkier peut skier sur toute les pistes.
- Un Bluerun n'est pas un RedRun ou un BlackRun.
- Un Redrun n'est pas un BlueRun ou un BlackRun.
- Un Blackrun n'est pas un RedRun ou un BlueRun.
- Un Skier ne peut être qu'un BeginnerSkier ou un AverageSkier ou un GoodSkier.
- Un BeginnerPlace est une place où au moins un BlueRun commence.
- Un AvegragePlace est une place où au moins un RedRun commence.
- Un GreatPlace est une place où au moins un BlackRun commence.
- Un Skilift pour BeginnerSkier se dirige vers un BeginnerPlace.
- Un Skilift pour AverageSkier se dirige vers un AveragePlace.
- Un Skilift pour GreatSkier se dirige vers un GreatPlace.

Définiton des axiomes dans Protégé

On va maintenant montrer comment on définit les axiomes dans l'application Protégé.

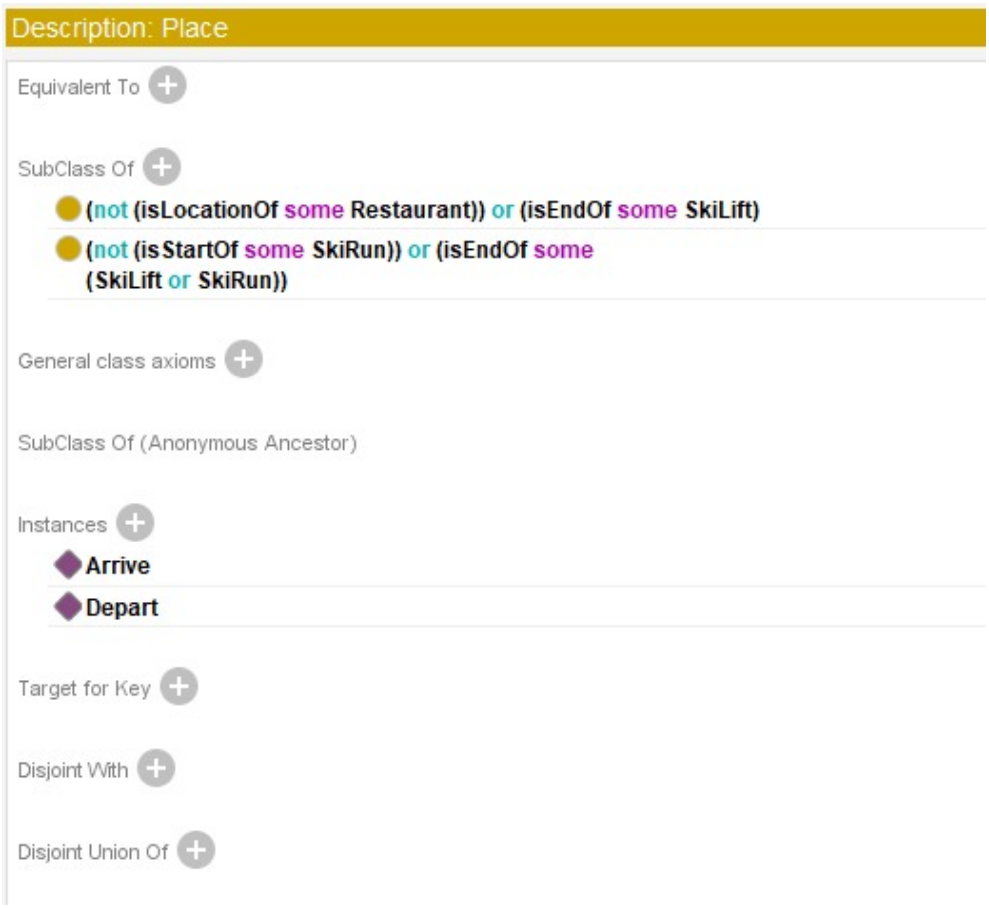

Figure 3: éfinition d'une axiome sur Protégé

Les axiomes se définissent principalement dans la partie *Subclass Of*. Ici on définit les axiomes concernant la classe Place :

- S'il y a un restaurant à une place, alors un Skilift fini à cette place.
- Une place qui est le début d'un Skirun est nécessairement la fin d'un Skirun ou d'un Skilift.

Vérification de la consistance avec OWL reasoner

OWL reasoner est un logiciel capable de déduire des conséquences logiques à partir d'un ensemble de faits ou d'axiomes affirmés. Lorsqu'il rencontre une ou plusieurs classes non-satisfaisable, il les affiche en rouge. S'il constate qu'il y a un problème majeure (Inconsistence), alors il affiche un message d'erreur en précisant les faits ou les axiomes mal définies.

Dans notre cas, on constate qu'il n'y a aucun de ces deux cas lorsque le reasoner OWL tourne. On peut donc conclure que notre base de connaissances est au moins consistante et satisfaisable.

Création d'une base de connaissance

Dans cette partie, nous allons vérifier la consistance et la performance de cette base de connaissance que nous avons modélisée dans la première partie. Pour ce faire, nous allons exporter l'ontologie sous forme de fichier Turtle et l'importer dans GraphDB. Nous allons égalements y inclure des instances de Classe, afin d'observer les inférences produites pas OWL reasoner.

Voici un exemple d'inférence pour une instance de Classe Place:

Figure 4: Caption

Effectivement, il nous ai pas nécessaire de déclarer explicitement les propriétés de cette place contenant une piste rouge, puisque si l'on déclare "piste noir" endsAt "debut piste rouge", OWL reasoner infére la règle "début piste rouge" isEndOf "piste noir", car il sait que la propriété "isEndOf" est l'inverse de "endsAt".

En effectuant la bonne commande SPARQL, on observe bien ces inférences:

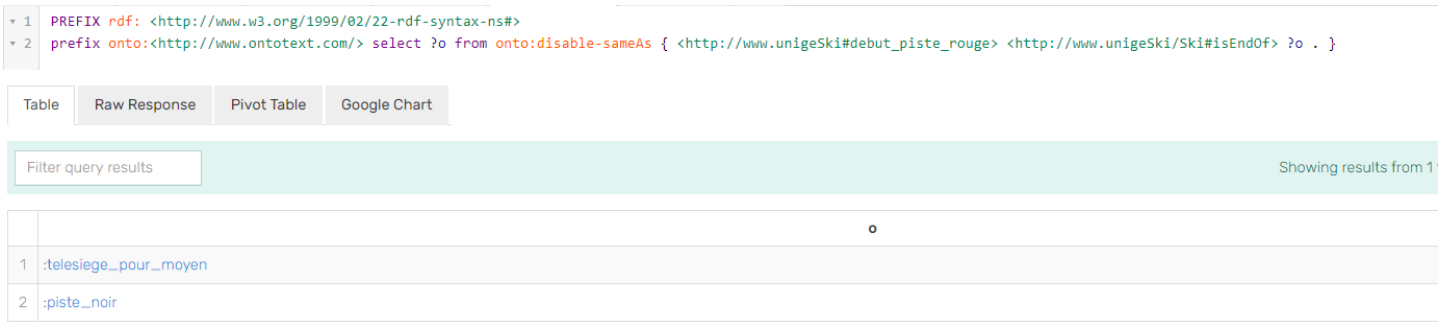

Figure 5: Caption

Evidemment, ces éléments ne représentent pas la totalité de la base de connaissance constituée. Voici ci-dessous le modèle de notre base de connaissance qui nous permet de vérifier l'ensemble de nos axiomes, des classes que nous avons définit et que nous définirons dans la prochaine Section:

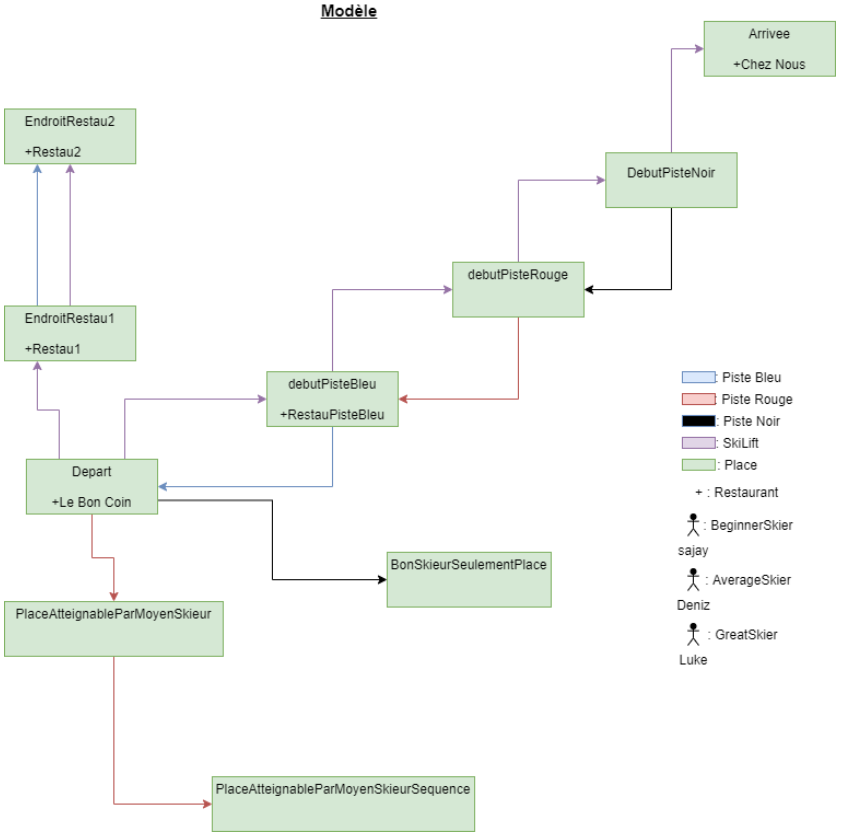

Figure 6: Caption

SPARQL avec OWL inférences

Pour les expressions en SPARQL, il nous a été nécessaire de définir des classes dont les membres sont inférés par OWL reasoner. Pour ce faire, nous avons déclaré six nouvelle classes:

- La classe Start, une sous-classe de Place contenant une/des instance(s) d'un lieu à partir duquel nous souhaitons commencer la recherche.
- La classe PlacesReachedByStart définit par la relation : "**Place and (isEndOf some ((SkiLift and (startsAt some Start)) or (SkiRun and (startsAt some Start))))**", nous renvoyant les places qui sont atteignable en prenant un SkiLift ou un SkiRun à partir des instances de Start.
- La classe PlacesReachedByStartSequence définit par la relation : "**PlacesReachedByStart or (isEndOf some ((SkiLift and (startsAt some PlacesReachedByStartSequence)) or (SkiRun and (startsAt some PlacesReachedByStartSequence))))**", nous renvoyant toutes les places atteignables depuis les éléments de Start en prenant un ou plusieurs SkiLift ou SkiRun.
- La classe AllPlacesReachedByAverage définit par la relation : "**PlacesReachedByStartSequence and ((isEndOf some SkiLift) or (isEndOf some (BlueRun or RedRun)))**", renvoyant l'ensemble des places atteignables par un AverageSkier depuis les éléments se trouvant dans la Classe Start.
- La classe ResaurantFromStart définit par la relation: "**PlacesReachedByStart and (isLocationOf some Restaurant)**", donnant les places possèdants un restaurant atteignables se trouvant au(x) voisinage(s) de Start.

- La classe RestaurantFromStartSequence définit par la relation: "**RestaurantFromStart or ((isEndOf some ((SkiLift and (startsAt some RestaurantFromStartSequence)) or (SkiRun and (startsAt some RestaurantFromStartSequence)))) and (isLocationOf some Restaurant))**", qui retourne l'ensemble des Places possèdants un restaurant atteignable uniquement en passant par d'autre places possèdant des restaurants depuis Start.

Après avoir importé ces nouvelles classes dans GrapheDB, il nous ai donc possible d'exécuter de manière simple des commandes SPARQL trés complexe de la façon suivante:

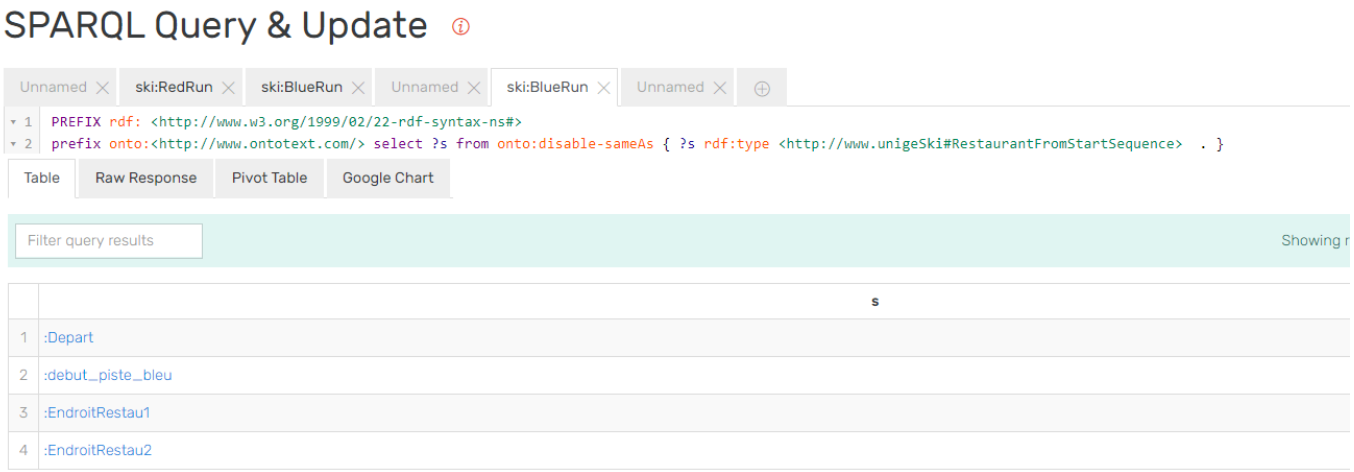

Figure 7: Caption

En observant notre modèle, on remarque effectivement que la classe RestaurantFromStartSequence retourne les bons éléments en partant de la place "Départ".

Voici donc notre hiérarchie de classe finale:

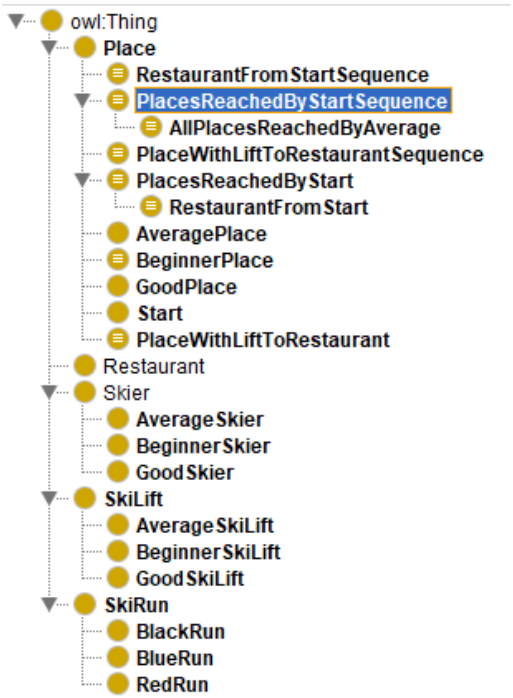

Figure 8: Caption