Microlearning: une ontologie

Connaissances et raisonnement

LAUBÉ Nicolas, MILLON Romain et STUTZ Antoine

28/02/2022

CentraleSupélec

Academical year 2021-2022

Contents

1 Introduction & contexte

Ce projet s'inscrit dans le cadre de la filière entrepreneuriale et du cours de connaissances et raisonnement. Capsule est une plateforme de micro learning dont l'objectif est d'améliorer la rétention d'information et l'amusement pendant l'apprentissage.

Le micro learning est un concept qui consiste à diviser un cours en une multitude de petits modules de quelques minutes. Décomposer la connaissance en petits morceaux aide à augmenter l'attention et permet un taux de rétention plus élevé. Par exemple, la recherche montre que le micro-apprentissage peut entraîner une augmentation significative des taux de réussite aux examens (jusqu'à 18%) [1].

La plateforme d'apprentissage est ouverte au grand public et devra, à terme, couvrir des thématiques variées telles que la biologie, l'histoire ou l'astronomie etc. Chaque thématique peut contenir plusieurs cours. Par exemple, la thématique "Biologie" contient les cours "Le microbiote" et "La théorie de l'évolution". Ensuite, chaque cours est divisé en petits modules. Par exemple le cours sur "Le microbiote" contient les modules "Influence de l'environnement sur le microbiote" et "Saisonalité du microbiote". Un module contient plusieurs connaissances qui peuvent être présentées de diverses manières (petit texte à lire, questionnaire, audio, petite vidéo, etc.).

Ce projet entrepreneurial semble bien se prêter à l'application du cours de connaissance et raisonnement. Nous essaierons dans la suite de ce document de présenter notre démarche de construction d'une ontologie adaptée à notre objectif, flexible, à jour et évolutive. Pour cela, nous allons nous baser sur la méthode de construction d'ontologies en casacade.

2 Outil de programmation & code

Nous avons choisi d'implémenter notre base de connaissance avec python et le package owlready2, d'autres parties de notre projet entrepreneurial étant également écrites en python. OWL (Ontology Web Language) est de loin le langage le plus répandu pour la création d'ontologies.

De plus, l'ontologie formelle ne possédant pas d'équivalent pour les méthodes de classes, python pourra éventuellement nous permettre d'aller plus loin dans la conception de notre ontologie. Par ailleurs, il est possible de copier-coller des définitions, ce qui permet de créer rapidement des classes similaires.

Le code est disponible sur **Gitlab**.

3 Construction de la base de connaissance

Dans cette partie, nous résumons notre démarche de création de l'ontologie par la méthode en cascade. Nous avons également choisi d'implémenter cette méthode en plusieurs étapes incrémentales suivant la méthode agile. Cela nous permet d'avoir une ontologie fonctionnelle plus rapidement et de l'adapter progressivement.

Ainsi, nous construisons l'ontologie suivant les quatre grandes étapes suivantes:

- **Tout ce qui a trait aux enseignements:** thèmes, modules, parties, cours, connaissances, fragments. Pas de notion d'utilisateur, de gamification, d'affichage, de test

- **Ce qui concerne les utilisateurs et créateurs:** qui suit quoi, qui a accès à quoi, qui doit réviser quoi et quand...
- **Notions d'implémentation:** affichage possible, tests
- **Ce qui concerne la gamification:** points, badges...

Nous présenterons dans la suite uniquement les deux étapes qui ont été implémentées.

3.1 Étape 1: Tout ce qui a trait aux enseignements

3.1.1 Domaine et portée

La première étape de la création d'une ontologie par cascade consiste à définir le domaine et la portée. En l'occurrence, nous essayons de définir le domaine et la portée spécifiques aux enseignements. Pour cela, nous listons ici l'ensemble des questions auxquelles l'ontologie doit répondre.

- Quelles sont les thématiques ?
- À quelle thématique est associée le cours C ?
- Quels sont les modules du cours C ?
- Quelles sont les connaissances associées au module M ?
- Quels sont les prérequis du module M ?
- Quels sont les apprentissages d'un module ?
- Quels sont les apprentissages d'un cours ?
- Quels sont les modules débloqués par un module ?

3.1.2 Réutilisation d'une ontologie

Dans cette étape, nous ne réutiliserons pas d'ontologie préexistante.

3.1.3 Énumération des termes

- **Thématique** : Une thématique est, comme son nom l'indique, un ensemble cohérent selon un même thème.
- **Cours** : Un cours s'apparente à la notion commune de cours.
- **Module** : Un module est une partie d'un cours.
- **Connaissance** : Une connaissance représente une idée, un fait, un point précis qui ne peut être divisé en plusieurs connaissances.
- **Prérequis** : Des modules/connaissances peuvent être requises pour comprendre d'autres modules/connaissances.

- **Premier module/connaissance et ordre** : Par où commence un cours et plus généralement quel est l'enchaînement des modules.
- **Apprentissage** : Ce qu'a appris un utilisateur
- **Cours débloqués** : Les cours ou modules qui sont débloqués lorsqu'un utilisateur termine une cours/-module.
- **Contenu** : Ce que contient un cours/un module
- **Fragment** : Une extension liée à une connaissance
- **Type de fragment** : Un fragment peut être un complément, une exemple ou une anecdote
- **Association** : La façon dont les élément sont associés entre eux.

3.1.4 Défintion des classes

- **Thematic**, représentant un thème autour duquel s'articulent plusieurs cours
- **Course**, représentant la connaissance associée à un sujet particulier, associée à une Thématique
- **Module**, rassemblant un petit ensemble de connaissances autour d'un sujet très précis, associé à un Cours
- **Knowledge**, unité indivisible de connaissances, associée à un Module
- **Fragment**, donne des précisions sur une connaissance Associé à une Connaissance Partitionné en : complément, exemple, anecdote
- **Example**, qui illustre une connaissance, sous-classe de Fragment
- **Complement**, donne des précisions sur une connaissance
- **Anecdote**, une anecdote sur une connaissance, sous-classe de Fragment

3.1.5 Définition des propriétés objet

A partir des questions définies dans la première partie, nous pouvons identifier un certain nombre de propriétés que doivent vérifier les classes. Par simplicité, nous ne définissons pas les propriétés inverses lorsqu'elles existent et indiquerons simplement que la propriété inverse existe.

Dans cette partie, nous définissons les propriétés objet qui relient les différentes classes. Nous essayons notamment d'en créer le moins possible tout en gardant toute la représentation des cas de figure. Par exemple, il n'est pas nécessaire de définir la propriété d'ordre des modules $is\_preceeded\_by$ car celle-ci peut être induite par la propriété $requires\_module$ de prérequis sur les modules. Les propriétés s'appliquent entre deux classes définies précédement et peuvent être symétriques, réflexives, transitives, fonctionnelle (unicité) etc.

- `is_in_thematic` : Cette propriété indique si un cours fait partie d'une certaine thématique.
  - Range : Thematic
  - Domain : Cours
  - Propriété inverse : `has_as_course`
- `is_in_course` : Cette propriété indique si un module fait partie d'un cours
  - Range : Cours
  - Domain : Partie
  - Propriété inverse : `has_as_course`
- `is_in_module` : Cette propriété indique si une connaissance fait partie d'un module
  - Range : Module
  - Domain : Connaissance
  - Propriété inverse : `contains_knowledge`
- `requires_modules` : Cette propriété indique les modules requis pour pouvoir en commencer un autre
  - Range : Module
  - Domain : Module
  - Propriété inverse : `is_required_by_modules`
- `follows_knowledge` : Cette propriété indique quelle connaissance suit une autre connaissance
  - Range : Connaissance
  - Domain : Connaissance
  - Propriété inverse : `is_followed_by_knowledge`

Figure 1: Object properties

Nous avons choisi d'ajouter la dépendance de restriction sur les modules plutot que sur les connaissance. En effet, même si elle semblerait plus logique sur les connaissances, il est plus facile de comprendre de quels modules nous avons besoin d'étudier avant d'en étudier un nouveau, comme les modules présentent de toute façon des connaissances autour d'un sujet assez précis. De plus, comme l'enchaînement des connaissances est linéaire au sein d'un module, il vaut mieux laisser le créateur choisir leur ordre.

Par simplicité, nous supposons dans cette première étape que tous les modules prérequis pour un cours sont inclus dans celui-ci. Dans une itération suivante, nous pourrons intégrer ces dépendances.

3.1.6 Définition des propriétés donnée

Nous avons aussi pu déterminer quelques propriétés donnée, selon les classes que nous avons pu déterminer et les contraintes que nous avons exprimer.

- Thematic
  - Titre (string)
  - Description (string)
- Course
  - Titre (string)
  - Description (string)
  - Author
- Module
  - Titre
- Knowledge
  - Titre (string)
  - Description (string)
- Fragment
  - Contenu (string)
- Complement
- Example
- Anecdote

Figure 2: Data properties

3.1.7 Définition des contraintes

Cependant, nous n'avons pas défini ces propriétés partout comme c'est un exercice long qui ne présente pas grand intérêt. Par ailleurs, nous avons ajouté quelques label et commentaires dans quatre langues différentes.

- Un Fragment peut être défini par extension, c'est à dire en listant ces instances plutôt qu'en définissant ces propriétés. Un fragment peut être un complément, une anecdote ou un exemple.
- Un cours contient au moins 2 modules.
- Une partie contient au moins un module.
- Un module contient au moins une connaissance.
- Une thématique contient au moins un cours.
- L'ontologie repose sur l'hypothèse du monde ouvert : c'est à dire que tout ce qui n'est pas expressément interdit est considéré comme possible. Nous devons donc également définir des propriétés d'unicité. Par exemple, il faudra définir précisément qu'un cours a comme unique auteur l'auteur du cours.

3.1.8 Création des instances

Au cours de notre travail, nous avons créé un certain nombre d'instances pour les différentes classes définies précédemment. Bien entendu, il ne nous a pas été possible de définir toutes les connaissances de tous les cours qui ont été créés.

Toutefois, nous avons créé partiellement 16 cours dans cinq thématiques différentes. Le cours sur la musique pop est le plus complet et possède également des instances pour les connaissances.

3.1.9 Test de l'ontologie

Pour tester notre ontologie, nous utilisons le raisonneur Pellet et nous effectuons des requêtes sur notre base de données. Le raisonneur permet principalement de vérifier que l'ontologie n'est pas inconsistante et d'inférer de nouveaux triplets rdf à partir des contraintes et règles qui ont été définies.

Lorsque l'ontologie s'avère inconsistante, nous privilégions le logiciel protégé pour expliquer les potentielles erreurs.

3.2 Étape 2 : Utilisateurs et créateurs

Il y a plusieurs types de personnes, les créateurs, les apprenants et les suiveurs (follower). Une personne peut d'ailleurs être dans toutes les catégories à la fois.

Dans cette section, nous utilisons la même démarche que dans la section précédente. Cependant, nous ne détaillerons pas ici toutes les étapes.

3.2.1 Domaine et portée

- Qui a terminé le cours C ?
- Qui a créé le cours C ?
- Quelles connaissances à acquis X ?
- À quel cours à accès X ?
- Quels cours X doit-il réviser ?
- Quels cours X devrait-il suivre ?
- Où en est X dans le cours C ?
- Quels modules du cours C X a-t-il débloqué ?
- Combien de personnes ont commencé le cours C ?
- Combien de personnes ont terminé un cours C ?
- Quels fragments X a vu ?

3.2.2 Définissions des classes

- **Person**, une personne quelconque qui utilise (ou pas) notre outil.
- **Learner**, une personne qui suit un cours.
- **Creator**, une personne qui a créé au moins un cours.
- **Follower**, une personne qui suit une autre personne.
- **Reviewer**, une personne qui a modifié un module existant.

3.2.3 Définition des propriétés

- *created_course*: Est défini entre une personne et un cours. Correspond au fait q'une personne a créé le cours.
- *created_module*: Est defini entre une personne et un module. Correspond au fait qu'une personne a créé un module.
- *reviewed*: Est défini entre une personne et un module. Correspond au fait qu'une personne à apporter des modifications à un module existant.
- *finished_module*: Est défini entre une personne et un module. Correspond au fait qu'une personne a terminé un module.
- *started_course* : Est défini entre une personne et un cours. Correspond au fait qu'une personne a commencé un cours.
- *finished_cours*: Est défini entre une personne et un cours. Correspond au fait qu'une personne a terminé un cours.
- *started_module* : Est défini entre une personne et un module. Correspond au fait qu'une personne a commencé un module.
- *failed* : Est défini entre une personne et un module. Correspond au fait qu'une personne a manquer un module.
- *follows* : Est défini entre deux personnes. Correspond au fait qu'une personne suit une autre personne (son activité, les cours créés etc.).

3.2.4 Création des instances

Les instances créés sont essentiellement utiles pour tester notre ontologie mais ne correspondent pas à de vrais utilisateurs.

3.2.5 Définition des contraintes

Nous avons défini un certain nombre de contraintes en utilisant SWRL.

- Une personne doit avoir réussi/vu tous fragments (questions, exemples, vidéos etc.) pour qu'un module soit considérer comme terminé.
- Un cours est terminé lorsque tous les modules de ce cours ont été terminés.
- Une personne qui a revu un module est un *Reviewer*.
- Une personne qui a fini un cours a fini tous les modules qui sont dans ce cours.
- Une personne qui a commencé ou fini un module est un apprenant.
- Une personne qui a fini un cours a également commencé ce cours.
- Une personne qui a créé un module ou un cours est un créateur.
- une personne qui suit une autre personne est un suiveur.

4 Logique Prolog

Des règles Prolog ont été définies pour raisonner sur la base de connaissance.

Avant de présenter ces règles, nous devons définir deux concepts.

- **Le niveau de graphe d'un module.** C'est le rang n du groupe de modules correspondant à l'ensemble des modules indépendamment réalisables après avoir fini les n-1 premiers groupes de modules (cf. la figure 3 en-dessous).
- **Les dépendances minimales d'un module.** C'est le plus petit ensemble de dépendances de ce module M tel que tout module M' duquel M est dépendant est dans ce sous-ensemble ou dans l'une des dépendances lointaines d'un module de ce sous-ensemble.

  Par exemple, sur la figure 3, si les modules A, D et E dont définis comme les dépendances de G, les dépendances minimales de G sont les modules D et E comme A fait partie des dépendances de D (on remarquera ici que le graphe est construit avec les dépendances minimales).

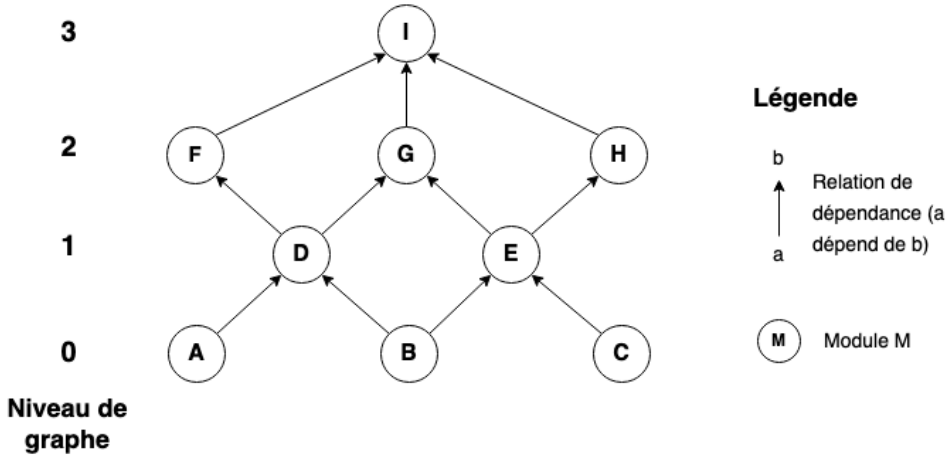

Figure 3: Schéma des niveaux de graphe

Les règles définies en Prolog permettent de d'effectuer les opérations suivantes :

- Récupérer toutes les dépendances d'un module dans un cours spécifique
- Récupérer tous les modules d'un cours
- Récupérer le niveau de graphe d'un module
- Récupérer les niveaux de graphe de tous les modules d'un cours
- Récupérer les dépendances minimales d'un module
- Récupérer les dépendances minimales de tous les modules d'un cours

Ces règles nous permettent de détecter les cycles de dépendance dans un cours. Ils nous permettent aussi de définir les graphes de module.

5 Démonstrateur

Nous avons créé un démonstrateur en utilisant le package **Streamlit**.

Il se compose de deux pages:

- Une page pour lancer des requêtes SparQL
- Une page pour visualiser les graphes des cours

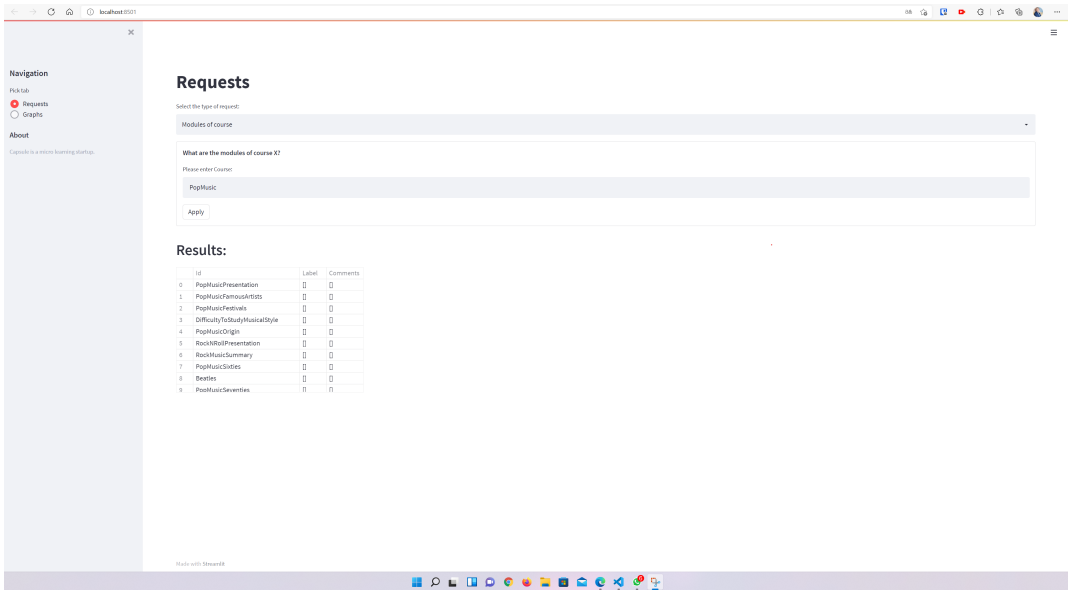

Figure 4: Écran de requêtes du démonstrateur. Il est possible de sélectionner le type de requête. En fonction du type sélectionné, plusieurs champs sont à remplir. Enfin, les résultats sont affichés sous forme de tableau.

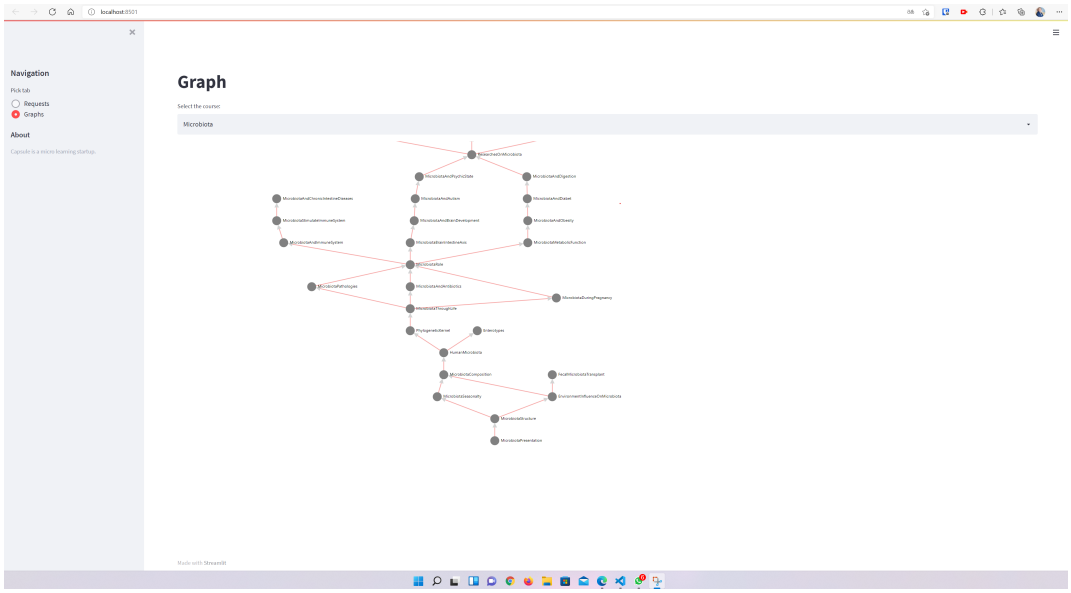

Figure 5: Écran du graphe d'un cours. Il est possible de sélectionner le cours de son choix. En fonction des contraintes de pré-requis qui ont été spécifiées dans l'ontologie, la plate-forme renvoie le graphe de modules correspondant, c'est à dire le graphe présentant l'ordre dans lequel les modules doivent être suivis.

6 Conclusion

Nous avons créé une ontologie spécifique à notre projet entrepreneurial en python sans réutiliser des ontologies existantes. Ce projet nous a permis d'utiliser de nombreux outils/langages vus en classe (Prolog, SWRL, OWL et SPARQL). Nous avons fait le choix d'utiliser python ce qui nous a permis de facilement créer un démonstrateur et de pouvoir réutiliser notre code pour notre projet. Il semble néanmoins que les outils comme Protégé sont plus adaptés pour identifier les erreurs d'inconsistances et pour créer rapidement des ontologies.

Concernant les améliorations qui pourrait être apportées à notre ontologie, outre la finalisation des étapes qui ont été définies dans 3, il serait intéressant de réussir à connecter notre outils à dbpedia et d'améliorer la représentation des connaissances. Par ailleurs, d'autres subsomption de classes pourraient être ajoutées (Créateur expert, début, intermédiaire etc.).

References

[1] The Effectiveness of Microlearning to Improve Students' Learning Ability, *Sirwan Mohammed, Gona; Wakil, Karzan; M. Nawroly, Sarkhell Sirwan*, **International Journal of Educational Research Review (3): 35. doi:10.24331/ijere.415824.**. 2018. Retrieved March 28, 2021.

Annexes

Portée de l'ontologie (questions)

En réfléchissant sur le microlearning et ses applications, nous avons formulé les questions suivantes :

- Quelles sont les thématiques ?
- Quels sont tous les cours ?
- À quelle thématique est associé le cours C ?
- Quels sont les modules du cours C ?
- Quelles sont les connaissances associées au cours C ?
- Quelles sont les connaissances associées au module M ?
- Quels sont les prérequis du module M ?
- Quels sont les prérequis du cours C ?
- Quel est le premier module du cours C ?
- Quel est le module après M ?
- Quels sont les prérequis du module M ?
- Quels sont les apprentissages d'un module ?
- Combien de points sont rapportés par l'utilisateur grâce à ce module ?
- Quels sont les cours débloqués par un module ?
- Dans quel ordre faut-il afficher les connaissances liées au module M ?
- Quand faut-il que je lance un test de connaissances ?
- Quel est résumé d'une étape ? Les points/achievements associés ?
- Quel est le contenu de la connaissance C ?
- Quelles sont les langues disponibles pour la connaissance C ?
- Combien de fois une connaissance doit-elle être revue ?
- Quels sont les fragments associés à la connaissance C ?
- Quel est le type de ce fragment ? (exemple, fait amusant)
- De quelle forme ce fragment peut être représenté ?
- Ce fragment peut-il constituer un test de connaissance ?
- Qui a terminé le cours C ?

- Qui a créé le cours C ?
- Quelles connaissances à acquis X ?
- À quel cours à accès X ?
- Quels cours X doit-il réviser ?
- Quels cours X devrait-il suivre ?
- Où en est X dans le cours C ?
- Quels modules du cours C X a-t-il débloqué ?
- Combien de personnes ont commencé le cours C ?
- Combien de personnes ont terminé un cours C ?

Seule une partie a pour le moment été utilisée.