Suivi thérapeutique intelligent par recommandation à base d'ontologie et de règles

Xavier Goblet, Christophe Rey

Suivi thérapeutique intelligent par recommandation à base d'ontologie et de règles

X. Goblet[1], C. Rey[2]

[1] Jeolis Solutions – Chamalières, France
[2] LIMOS – Université Clermont Auvergne, France

xavier.goblet@lojelis.com, christophe.rey@uca.fr

Résumé

Le domaine de l'Education Thérapeutique du Patient vise à habituer un patient à adopter les bons comportements qui lui permettront au quotidien de renforcer l'action d'un traitement médical. Dans ce contexte, nous présentons une application proposant automatiquement des défis ludiques à un patient souffrant d'obésité. Le processus de recommandation est basé sur l'usage d'une ontologie des patients, des défis et de règles pédagogiques pour le choix des meilleurs défis. Nous utilisons les standards du web sémantique que sont OWL2, SWRL, et le raisonneur Pellet, orchestrés par un programme en Python grâce au module Owlready2. Nous discutons des résultats obtenus ainsi que de l'intérêt d'utiliser des langages déclaratifs tels que OWL2 et SWRL du point de vue du génie logiciel.

Mots-clés

Education thérapeutique du patient, recommandation, OWL2, SWRL, Owlready2, langage déclaratif

Abstract

The field of patient education aims at accustoming a patient to adopt the right behaviors which will allow him daily to reinforce the action of a medical treatment. In this context, we present an application automatically offering fun challenges to an obese patient. The recommendation process is based on the use of an ontology that models the patients, the challenges and on educational rules for choosing the best challenges. We use the semantic web standards OWL2, SWRL, and the Pellet reasoner, orchestrated by a program in Python thanks to the Owlready2 module. We discuss the obtained practical results as well as the interest to use declarative languages such as OWL2 and SWRL in software engineering.

Keywords

patient education, recommendation, OWL2, SWRL, Owlready2, declarative language

1 Introduction

De plus en plus de pathologies chroniques sont prises en charge par des entretiens de suivi avec les professionnels de santé (PS). Ce suivi est efficace si les entretiens sont le moins espacés dans le temps, le plus souvent en présentiel (face-à-face) et si le patient est pleinement acteur de sa thérapie. L'objectif d'un entretien est, après un diagnostic, la mise en place de bonnes pratiques et leur maintien. C'est en cela que l'on peut parler d'éducation thérapeutique du patient (ETP) [6]. Les principaux freins à une ETP efficace sont : le manque de temps des praticiens et patients, les contraintes économiques du monde de la santé, les contraintes écologiques (limiter les déplacements physiques par exemple), une accessibilité grandissante d'informations sur internet (pas toujours fiables), un contenu peu adapté à l'individu, une faible adhésion si la motivation du patient n'est pas présente [6, 18]. Ces freins sont par exemple vérifiés dans le contexte d'une expérience d'ETP sur le bassin clermontois pour prévenir l'obésité infantile dans une famille à risques (au moins un enfant en surpoids) [17]. Pendant six mois, une famille incluse dans ce protocole est suivie sur les axes Nutrition, Activités Physiques Adaptées (APA), et Parentalité/Sociabilité par trois spécialistes qui se déplacent chacun dans la famille pour cinq ateliers de suivi. Les six mois suivant cette première période, la famille n'est plus suivie ; elle rentre alors en phase d'autonomie. Au bout d'un an, le protocole se finit par des entretiens de bilan avec chaque PS et le(s) médecin(s). Il a ainsi été constaté, en l'absence d'un suivi régulier, une baisse de motivation des participants. De plus, la phase d'autonomie est souvent préjudiciable aux progrès observés les six premiers mois. Dans ce contexte, dans le même esprit que [7], un projet de digitalisation de cette ETP est en cours d'élaboration. L'objectif à partir d'une application mobile est d'augmenter singulièrement le suivi des patients, de l'individualiser en utilisant des outils basés sur de l'IA et de maintenir la motivation via une ludification du protocole au moyen de défis personnels en complément des entretiens de suivi. Ce concept de défis (cf. tableau 1) est le pendant des tâches à accomplir par le patient dans le cadre des thérapies cognitives et comportementales (TCC). « Chaque patient progresse à son propre rythme » est aussi le principe fondateur des psychologues cliniciens des TCC. De plus, mis à part la toute première fois, c'est toujours le patient qui choisit son prochain défi. Nous avons conçu « ORALOOS » un Outil de Recommandation d'Activités Ludiques basé sur une Ontologie Opération-

Nutrition (nb=499)	- Buvez seulement de l'eau à vos repas. - La télévision est éteinte lors de vos repas.
Activités Physiques Adaptées (nb=207)	- Descendez un arrêt de bus/métro/tram avant destination et faites-le reste à pieds. - Marchez d'un bon pas sur les trajets courts.
Parentalité /Sociabilité (nb=144)	- Essaie de parler sans pouce ni doudou dans la bouche. - Prenez un moment pour faire une activité avec votre conjoint(e).

TABLE 1 – Des exemples de défis, définis par une psychologue de Jeolis avec les praticiens de Proxob [17].

nelle de Suivi, pour permettre un suivi adaptatif du patient. Nous l'avons défini en utilisant les langages standard du Web sémantique OWL2 et SWRL [9, 10], et le raisonneur Pellet, le tout orchestré par du code Python (via le framework Owlready2 [11]). L'ontologie est dite opérationnelle puisque son contenu va déterminer les traitements effectués par l'application. Les deux contributions de cet article sont : (i) la description de la conception d'ORALOOS, de son ontologie et de son usage dans la gestion et la recommandation des défis personnels pour une meilleure ETP, et (ii) les conséquences, en termes de génie logiciel, de la nature déclarative d'OWL2 et de SWRL, dans le contexte d'une application en langage impératif Python.

2 Travaux antérieurs

Nous nous sommes inspirés des travaux autour de l'utilisation de la logique du premier ordre dans les AEHS (Adaptive Educational Hypermedia Systems) [14, 8]. Chaque système adaptatif, en tant que système hypermédia éducatif, fait des hypothèses sur les documents et leurs relations dans un espace documentaire. Il propose un modèle d'utilisateur pour spécifier diverses caractéristiques d'utilisateurs individuels ou de groupes d'utilisateurs. Pendant l'exécution, il collecte des observations sur les interactions de l'utilisateur. Sur la base de l'organisation de l'espace documentaire sous-jacent, ainsi que des informations du modèle utilisateur et de l'observation du système, une fonctionnalité adaptative est fournie. Un AEHS se conceptualise sous la forme de quatre composants ou espaces : DOCS pour DOCument Space, UM pour User Model, OBS pour OBServations et AC pour Adaptation Component. Cette architecture correspond à nos besoins dans le domaine de l'ETP si l'on remplace les utilisateurs par les patients et si l'on considère que la fonctionnalité adaptative consiste à recommander les meilleurs défis aux patients. Pour la modélisation du domaine, nous nous sommes tournés vers les langages de la logique du premier ordre. En effet, des sous-langages de cette logique permettent la représentation de connaissances dynamiques sous forme de règles et la représentation de connaissances statiques sous forme d'énoncés et de faits. Ces deux types de connaissances sont cruciaux pour un système hypermédia éducatif adaptatif qui doit permettre l'expression et la réutilisation des règles d'adaptation (connaissances dynamiques) dans différents contextes en prenant en compte des métadonnées pour l'adaptation (connaissances statiques) [2]. Dans le domaine du web sémantique, OWL2 et SWRL figurent parmi les langages du premier ordre les plus utilisés et permettent le découplage entre représentation de connaissances dynamiques (en SWRL) et statiques (en OWL2). Le profil EL d'OWL2 [12] assure de plus de bonnes performances dans les raisonnements associés (subsomption, satisfaisabilité, traitement de requêtes), ces derniers étant traitables dans la plupart des cas. SWRL est quant à lui un langage de règles basé sur les clauses définies (sous-ensemble de la logique du premier ordre) étendu par des prédicats qui sont des concepts OWL2. En dehors des langages du web sémantique, nous avons aussi regardé le langage IDP (premier ordre augmenté par la récursivité) [4] qui apparaît très intéressant sur le plan théorique étant donné la possibilité d'exécuter plusieurs types d'inférences différentes. Cependant sur le plan pratique, il semble moins intéressant que les langages du web sémantique étant donné une plus grande difficulté d'apprentissage a priori, l'absence de recommandation W3C associée, ainsi que d'outil de modélisation du type Protégé [15]. De plus on s'interroge sur la possibilité de passage à l'échelle d'un tel langage. C'est pourquoi nous avons choisi OWL2 avec un profil EL et SWRL. Plus précisément, nous utilisons le profil EL d'OWL2 augmenté avec des propriétés d'objet fonctionnelles ainsi qu'avec le constructeur d'union de classes et la possibilité de définir des types intervalles de valeurs ayant une valeur minimale. Nous gagnons donc en expressivité au détriment de la complexité des raisonnements qui perd son caractère traitable mais n'en devient pas prohibitive pour autant en pratique (voir la section 5). Dans la suite, nous supposons le lecteur familier avec OWL2 et SWRL qu'on ne peut redéfinir ici par soucis de synthèse.

3 Approche

En section 3.1, nous expliquons la modélisation déclarative sous la forme des quatre espaces/modèles d'un AEHS en utilisant OWL2 et SWRL. En section 3.2, nous présentons le module python permettant d'accéder à l'ontologie et de raisonner avec elle.

3.1 Modélisation déclarative de l'ontologie

L'ontologie OWL2 et les règles SWRL ont été définies en utilisant l'éditeur Protégé. Comme évoqué précédemment, OWL2 est utilisé pour la modélisation statique du domaine de l'ETP, c'est-à-dire pour décrire les notions qui structurent le domaine (comme par exemple les catégories de patients ou les caractéristiques d'un défi), et SWRL est utilisé pour décrire les aspects dynamiques du domaine, c'est-à-dire les règles métier (notamment celles qui définissent le processus de recommandation de défis aux patients). Dans ce qui suit, nous présentons certaines parties de l'ontologie OWL2 sous forme de diagrammes de classes UML. En effet, la connaissance pouvant être décrite avec le profil EL de OWL2 utilisé correspond aux notions de classe et de relation en UML. Plus précisément, une classe OWL2 est représentée par une classe UML, une propriété OWL2

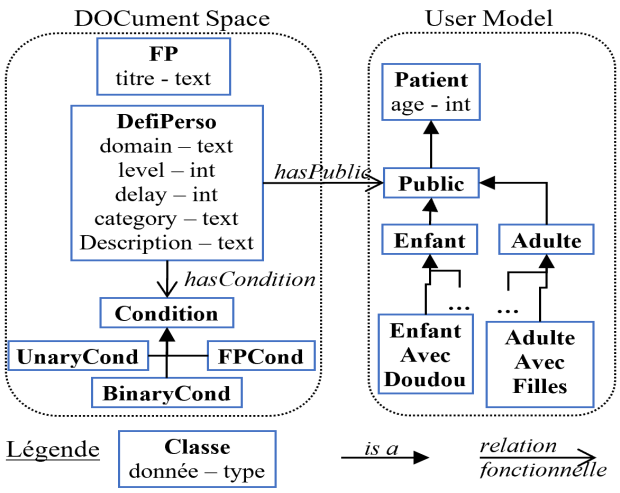

FIGURE 1 – Les espaces Document et Utilisateur

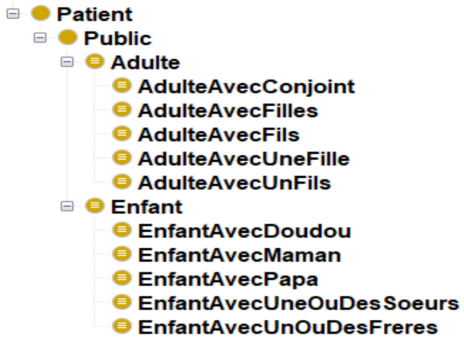

FIGURE 2 – Hiérarchie des différents types de patients

par une relation (ou association) UML [1] , et une relation de subsomption entre classes par une relation d'héritage. Cela nous permet de regrouper en une seule vue graphique, les classes et les propriétés OWL2. Nous voyons maintenant la modélisation des quatre parties du cadre des AEHS.

DOCument Space (cf. Fig.1) : c'est l'espace documentaire de l'ontologie qui contient les défis personnels, ainsi que des fiches pédagogiques (FP). Un défi personnel est défini obligatoirement par un domaine d'application (Nutrition, APA, ou Parentalité), le public concerné par la relation fonctionnelle [2] hasPublic, un niveau de difficulté (niveau croissant avec un entier), et une échéance (entier croissant : 1 → Jour, 2 → WE, 3 → Semaine, 4 → Mois). Les autres données ou relations sont optionnelles : une condition d'exécution du défi par la relation hasCondition (avoir lu une fiche pédagogique ou une caractéristique spécifique du public ciblé), une catégorie du défi dans le domaine et une description textuelle du défi. Dans un domaine, les défis sont définis indépendamment et il n'y pas non plus de dépendances entre les domaines.

User Model (cf. Fig.1) : c'est l'espace des utilisateurs qui porte sur la classe Patient défini par un âge qui permet de classifier le public auquel il appartient. Les classes Adulte et Enfant peuvent être spécialisées chacune en sous-

1. Dans la suite, on parlera de manière interchangeable de propriété (OWL2) ou de relation (UML).

2. Une relation est fonctionnelle lorsqu'il n'y a qu'une seule valeur possible entre les deux classes (1 → 1).

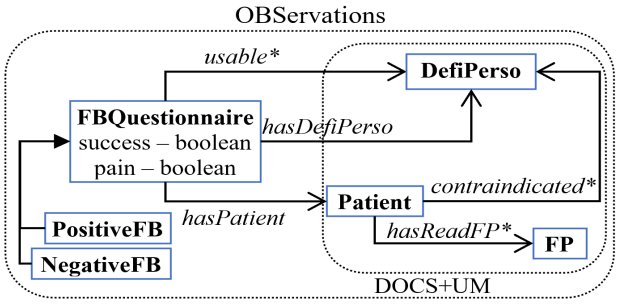

FIGURE 3 – Espace des observations

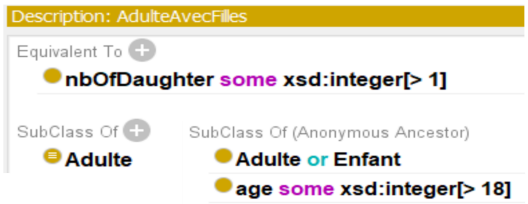

FIGURE 4 – La classe AdulteAvecFilles sous Protégé

classes qui portent une caractéristique optionnelle du patient (exemples : nombre de frères ou de sœurs, avoir un conjoint. . .). La figure 2 montre la hiérarchie des différents types de patients et la figure 4 montre la modélisation de la classe AdulteAvecFilles sous Protégé. Un individu peut appartenir à plusieurs classes : un enfant de 12 ans, vivant uniquement avec sa mère et possédant un doudou est : [Enfant, EnfantAvecDoudou, EnfantAvecMaman]. A contrario un adulte sans conjoint, ni fils, ni fille appartient uniquement à la classe Adulte.

OBServations (cf. Fig.3) : c'est l'espace des observations qui contient toutes les interactions des utilisateurs avec le système. Le premier type d'interaction est le questionnaire d'auto-évaluation (classe FBQuestionnaire, FB pour "feedback"). A échéance d'un défi, un questionnaire d'auto-évaluation est envoyé au patient sur son application mobile. En répondant à une série de questions, il évalue son succès ou non dans la réalisation du défi (booléen success), ainsi que sa difficulté ou non d'exécution (booléen pain). Chaque feed-back est contextualisé par le défi courant (relation fonctionnelle hasDefiPerso) et le patient ayant réalisé le défi (relation fonctionnelle hasPatient). La relation non fonctionnelle [3] usable filtre en amont tous les défis applicables de même domaine et même public que le défi courant à l'instant du feed-back. Ainsi tous les défis qui ne sont plus applicables après un feed-back ne sont pas dans usable. Un défi est applicable dans deux cas. Premièrement, il ne doit pas être contre-indiqué pour le patient. Cela correspond à la relation non fonctionnelle contraindicated. Par exemple un défi réalisé avec succès n'est plus proposé, ou bien un professionnel de santé peut explicitement interdire certains défis pour certains patients. Dans les deux cas le défi est dans contraindicated. Deuxièmement, si ce défi porte une condition, elle doit être vérifiée à cet instant, sinon le défi n'est pas applicable. C'est le cas de la relation

3. Une relation est non fonctionnelle lorsqu'il y a plusieurs valeurs possibles entre les deux classes (1 → *). Ce type de relation exprime aussi une liste dynamique d'informations.

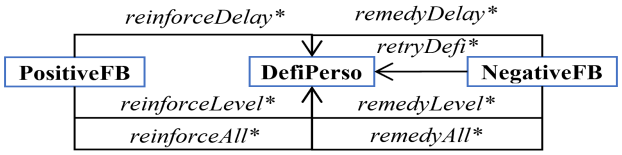

FIGURE 5 – Les relations selon la classe de feed-back

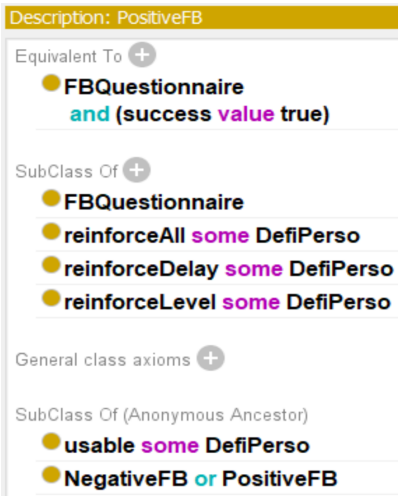

FIGURE 6 – Spécification Protégé de PositiveFB

non fonctionnelle hasReadFP qui concerne les fiches pédagogiques lues/vues par le patient et qui conditionne l'exécution de certains défis. En résumé, la relation usable est une liste dynamique des défis applicables. Comme évoqué à la section 5, mettre à jour cette liste nécessite du code Python. A contrario, une fois usable calculée, seules les règles SWRL (cf. Fig.7) sont nécessaires pour que le système établisse dynamiquement ses recommandations de défis à venir.

Adaptation Component (cf. Fig.5) : c'est l'espace des règles d'adaptation qui implémente les principes de progression au rythme de chaque patient et qui détermine les défis à recommander. Nous avons vu que l'espace des défis est important et que, sans guide, le patient ne progressera pas. Nous avons donc, en relation avec les experts psychologues et pédagogues, défini des règles métier en s'inspirant de la pédagogie behavioriste [5]. Le premier cas est celui où l'on renforce le succès en proposant des défis un peu plus difficiles. Cela se traduit dans la classe DefiPerso par une incrémentation du niveau, de l'échéance ou bien des deux. Cette incrémentation est exprimée par les règles R1, R2 et R3 de la figure 7. Le second cas est celui où l'on remédie à l'échec en proposant des défis un peu moins difficiles. Cela se traduit dans la classe DefiPerso par une décrémentation du niveau, de l'échéance ou bien des deux. Cette décrémentation est exprimée par les règles R5, R6 et R7 de la figure 7. On peut vérifier que les classes et les propriétés non fonctionnelles OWL2 présentes dans les figures 5 et 6 sont bien utilisées en prédicats dans les règles précédemment citées. Pour un même type de feed-back, chaque règle associée à une relation non fonctionnelle peut être activée en même temps (cependant, elles ne contiennent pas les mêmes défis). La règle R4 de la figure 7 implémente le fait qu'un défi évalué avec succès n'est plus proposé au patient en augmentant sa relation non fonctionnelle contraindicated. De même, les règles R8 et R9 de la figure 7 stipulent qu'un défi en échec peut soit être proposé de nouveau (retryDefi) au patient si une cause extérieure sans lien avec sa pathologie en a empêché la réussite (par ex. une météo défavorable), soit n'être plus proposé au patient s'il a échoué et éprouvé une difficulté.

```
R1 PositiveFB(?fb) ^ hasDefiPerso(?fb, ?df) ^
    hasLevel(?df, ?l) ^ hasDelay(?df, ?delay) ^
    swrlb:add(?nl, ?l, 1) ^ usable(?fb, ?ndf) ^
    hasLevel(?ndf, ?nl) ^ hasDelay(?ndf, ?delay)
    -> reinforceLevel(?fb, ?ndf)
R2 PositiveFB(?fb) ^ hasDefiPerso(?fb, ?df) ^
    hasDelay(?df, ?delay) ^ hasLevel(?df, ?l) ^
    swrlb:add(?ndelay, ?delay, 1) ^ usable(?fb, ?ndf) ^
    hasLevel(?ndf, ?l) ^ hasDelay(?ndf, ?ndelay)
    -> reinforceDelay(?fb, ?ndf)
R3 hasLevel(?ndf, ?nl) ^ usable(?fb, ?ndf) ^
    hasDefiPerso(?fb, ?df) ^ swrlb:add(?nl, ?l, 1) ^
    hasDelay(?df, ?delay) ^ hasDelay(?ndf, ?ndelay) ^
    swrlb:add(?ndelay, ?delay, 1) ^ PositiveFB(?fb) ^
    hasLevel(?df, ?l)
    -> reinforceAll(?fb, ?ndf)
R4 PositiveFB(?fb) ^ hasDefiPerso(?fb, ?df) ^
    hasPatient(?fb, ?pat)
    -> contraindicated(?pat, ?df)
R5 hasLevel(?ndf, ?nl) ^ usable(?fb, ?ndf) ^
    hasDefiPerso(?fb, ?df) ^ swrlb:greaterThan(?l, 1) ^
    hasDelay(?ndf, ?delay) ^ hasDelay(?df, ?delay) ^
    swrlb:subtract(?nl, ?l, 1) ^ NegativeFB(?fb) ^
    hasLevel(?df, ?l)
    -> remedyLevel(?fb, ?ndf)
R6 hasLevel(?ndf, ?l) ^ usable(?fb, ?ndf) ^
    hasDefiPerso(?fb, ?df) ^
    swrlb:greaterThan(?delay, 1) ^
    swrlb:subtract(?ndelay, ?delay, 1) ^
    hasDelay(?df, ?delay) ^
    hasDelay(?ndf, ?ndelay) ^ NegativeFB(?fb) ^
    hasLevel(?df, ?l)
    -> remedyDelay(?fb, ?ndf)
R7 hasLevel(?ndf, ?nl) ^ usable(?fb, ?ndf) ^
    hasDefiPerso(?fb, ?df) ^ swrlb:greaterThan(?l, 1) ^
    swrlb:greaterThan(?delay, 1) ^
    swrlb:subtract(?ndelay, ?delay, 1) ^
    hasDelay(?df, ?delay) ^ swrlb:subtract(?nl, ?l, 1) ^
    hasDelay(?ndf, ?ndelay) ^ NegativeFB(?fb) ^
    hasLevel(?df, ?l)
    -> remedyAll(?fb, ?ndf)
R8 NegativeFB(?fb) ^ pain(?fb, false) ^
    hasDefiPerso(?fb, ?df)
    -> retryDefi(?fb, ?df)
R9 NegativeFB(?fb) ^ pain(?fb, true) ^
    hasDefiPerso(?fb, ?df) ^ hasPatient(?fb, ?pat)
    -> contraindicated(?pat, ?df)
```

FIGURE 7 – Les règles SWRL de recommandation de défis. Les paramètres ?x sont des variables, les prédicats sont des classes ou des relations OWL2, et swrlb préfixe des fonctions prédéfinies SWRL

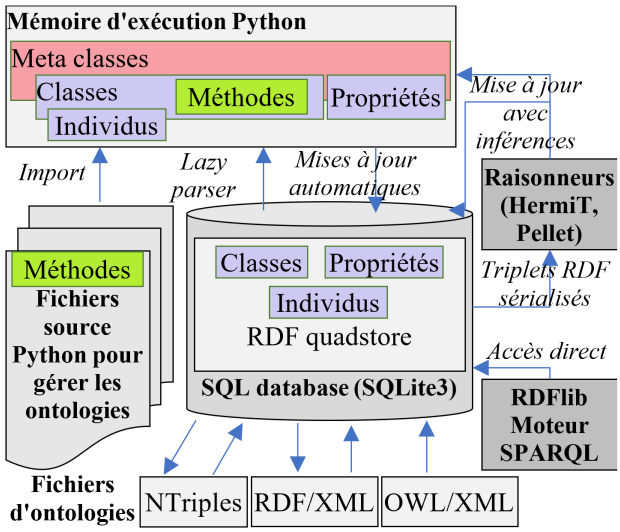

FIGURE 8 – Architecture d'Owlready2 extrait de [11]

```
with onto:
 class FBQuestionnaire(Thing):
  def preFilter(self):
   dom = self.hasDefiPerso.domain
   pub = self.hasDefiPerso.hasPublic
   contraindicates = self.hasPatient.contraindicated
   usables = onto.search(type=onto.DefiPerso,
                         domain=dom,
                         hasPublic=pub)
   # le defi courant doit etre enleve
   current = self.hasDefiPerso
   usables.remove(current)
   if len(contraindicates) != 0:
    for df in usables:
     if df in contraindicates:
      usables.remove(df)
   for df in usables:
    cond = df.hasCondition
    if (cond is None)
       or
       (cond.checkApplicability(self.hasPatient)):
     self.usable.append(df)
   return True
```

FIGURE 9 – Classe FBQuestionnaire

3.2 Programmation impérative d'ontologie

Nous avons programmé notre application en langage Python, avec l'aide d'Owlready2 un module Python (version 3 du langage) facilitant la programmation d'ontologies (accès aux données et connaissances, exécutions d'inférences, lecture et écriture avec les formats standards du Web sémantique). Ce module inclut nativement une base de triplets (appelée quadstore) et les raisonneurs HermiT et Pellet[4]. En plus des raisonnements fournis par ces deux raisonneurs, l'utilisateur d'Owlready2 peut en développer d'autres en Python. L'architecture d'Owlready2 est présentée à la figure 8. Avec Owlready2, toutes les classes OWL2 correspondent à des classes Python. Nous illustrons ce principe avec la classe OWL2 FBQuestionnaire contenant la méthode preFilter() qui met à jour la relation usable. La figure 9 montre le code python de cette classe.

Par ailleurs les classes FPCond et UnaryCond implémentent la vérification dynamique d'une condition d'applicabilité d'un défi (cf. Fig.10). On ne peut pas implémenter cette vérification grâce à des règles SWRL car il faudrait pouvoir exprimer des négations dans le corps des règles, voire même écrire des règles hors de la logique du premier ordre (cf. section 5).

4. Voir les sites web `http://www.hermit-reasoner.com/` et `https://github.com/stardog-union/pellet`.

```
class FPCond(Thing):
 def checkApplicability(self, patient):
  fp = self.hasFP
  readings = list(patient.hasReadFP)
  if fp in readings:
   return True
  return False

class UnaryCond(Thing):
 def checkApplicability(self, patient):
  classe = self.type
  subclasses = str(patient.is_a)
  if classe in subclasses:
   return True
  return False
```

FIGURE 10 – Les classes FPCond et UnaryCond

```
with onto:
 pat = onto.Patient(patient.ident,
                    age=patient.age,
                    **patient.kargs)
 try:
  sync_reasoner_pellet([onto],
          fer_property_values = True,
          infer_data_property_values = True)
 except OwlReadyInconsistentOntologyError:
  destroy_entity(pat)
  raise HTTPException(status_code=500,
           detail="Inconsistent ontology!"+
           "Check if patient age " +
           "is aligned with optional" +
           "characteristics...")
return patient
```

FIGURE 11 – Création et classement d'un patient

La figure 11 illustre le raisonnement permettant de classer le patient en adulte ou enfant ; si l'individu possède des caractéristiques optionnelles, le raisonnement affine la classification dans l'une des deux branches de la hiérarchie des patients (cf. Fig.2), tout en gérant les inconsistantes éventuelles (comme par exemple le fait qu'un enfant n'a pas de conjoint).

Le second raisonnement est la recherche des réponses à une requête et est implémenté en utilisant le raisonneur Pellet couplé à notre fonction ad-hoc preFilter() (cf. Fig.12) : la première étape consiste à créer un individu de classe FBQuestionnaire avec les différents paramètres d'entrées. La seconde étape consiste à appeler la méthode preFilter() qui permet de contextualiser dynamiquement le feed-back en calculant les défis applicables du domaine pour le patient (c'est-à-dire en mettant à jour la relation usable). La troisième étape consiste à lancer le raisonneur Pellet pour trouver le type de feed-back (de la classe FB) positif ou négatif et appliquer en conséquence les règles SWRL.

4 Intégration en application de suivi

Dans le cadre d'une architecture orientée services, ORALOOS est un composant micro-service qui interagit avec d'autres applications via une API Web REST comme illustré sur la figure 13. Il utilise pleinement la base de données nativement intégrée d'Owlready2 pour persister la base ontologique.

```
with onto:
 fdbk = onto.FBQuestionnaire(fb.fbID,
                             success=fb.success,
                             pain=fb.pain,
                             hasDefiPerso=defi,
                             hasPatient= pat)
 fdbk.preFilter()
 try:
  sync_reasoner_pellet([onto],
             infer_property_values = True,
             infer_data_property_values = True)
  except OwlReadyInconsistentOntologyError:
   raise HTTPException(status_code=500,
                       detail="Inconsistent ontology!")
  if isinstance(fdbk, onto.PositiveFB):
   response = PosFBAPI(ID = fb.fbID,
            patientID = fb.patientID,
                defiID = fb.defiID,
    reinforceAll = format_defis(fdbk.reinforceAll),
    reinforceLevel = format_defis(fdbk.reinforceLevel),
    reinforceDelay = format_defis(fdbk.reinforceDelay)
                    )
  else:
   response = NegFBAPI(ID = fb.fbID,
            patientID = fb.patientID,
                defiID = fb.defiID,
    remedyAll = format_defis(fdbk.remedyAll),
    remedyLevel = format_defis(fdbk.remedyLevel),
    remedyDelay = format_defis(fdbk.remedyDelay),
    retry = format_defis(fdbk.retryDefi)
                    )
return response
```

FIGURE 12 – Mise à jour des défis à recommander

4.1 API Web

Les principales interactions entre le back office et Oraloos permettent de peupler la base d'individus : createPatient, createChallenge, createFeedBack. Ces requêtes sont initiées par le serveur BO. Seules createPatient et createFeedBack déclenchent le raisonneur Pellet comme présenté en section 3.2. La réponse principale d'ORALOOS, contient les recommandations calculées par les règles dans les différentes listes, selon le type de feed-back. C'est le serveur BO qui les notifie alors au Patient.

4.2 Premiers résultats

Le tableau 2 montrent quelques métriques issues de Protégé. La figure (a) du tableau 2 concerne l'ontologie avant insertion des défis. Après peuplement de la base avec les 850 défis, et après inférence, nous constatons normalement une augmentation du nombre d'individus, mais aussi une augmentation importante du nombre d'axiomes (figure (b) du tableau 2). Evidemment, il y a une augmentation d'individus et d'axiomes lors des créations des patients et des feedbacks. Nos premiers tests montrent un temps de raisonnement moyen de l'ordre de 3-4 secondes lors de la création d'un patient et un temps de 6-7 secondes lors de la création d'un feed-back. Ces temps peuvent sembler importants mais nous rappelons que notre besoin de recommandations n'est pas en temps réel ; l'échéance minimale d'un défi est la journée. Ces résultats sont à confirmer avec une utilisation par des patients en conditions réelles lorsque l'application sera industrialisée.

5 Apports et limites de l'ontologie

Le principal avantage lié à l'usage d'une ontologie et d'un raisonneur dans notre application d'ETP est l'ajout d'une dimension déclarative forte au développement et à la maintenance de cette application. Cela permet ainsi d'éviter

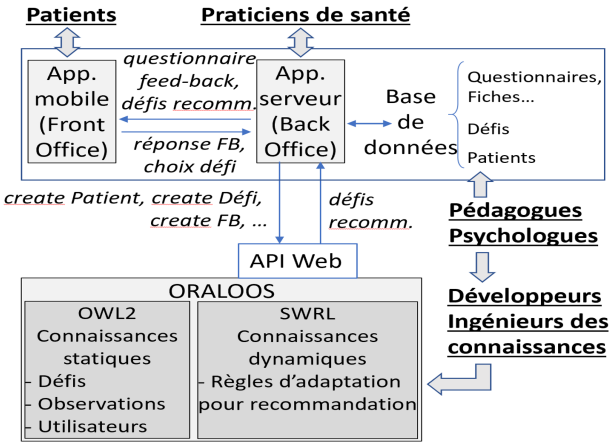

FIGURE 13 – Architecture de l'application de suivi à distance pour une ETP interactive, ludique et adaptative

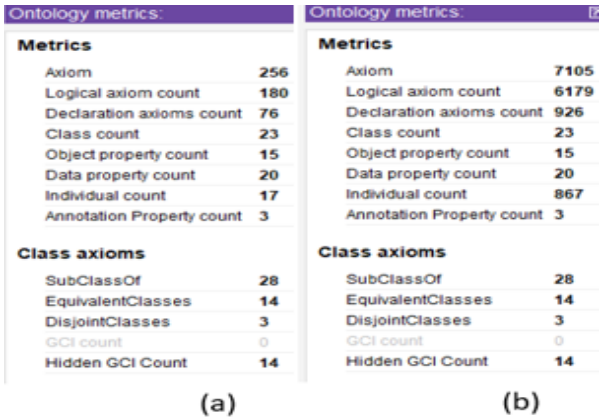

TABLE 2 – Quelques métriques Protégé

l'écriture d'un code Python complexe puisque toute la partie nécessitant la classification des connaissances et données ainsi que le requêtage avec les règles sont entièrement pris en charge par le raisonneur. Par ailleurs, la connaissance stockée dans l'ontologie est facilement extensible et maintenable puisqu'il suffit d'éditer le fichier avec un éditeur dédié comme Protégé.

Nous mettons ici en avant le fait que le niveau de déclarativité permis par le couple (ontologie avec règles, raisonneur) va bien au-delà de celui permis par un couple (fichiers texte de paramètres, bibliothèque de fonctions en python). En effet, grâce aux règles notamment, le développeur (cf. Fig.13) a la possibilité, par exemple, d'ajouter une nouvelle règle métier à l'application et ce déclarativement, c'est-à-dire en décrivant seulement les états initial et final de l'application de cette règle. Avec un langage impératif tel que Python, cela correspondrait à l'ajout d'une nouvelle fonction au sein d'une bibliothèque où, en plus des états initial et final associés à la fonction, nous devrions aussi décrire la gestion de la mémoire et les structures de contrôle nécessaires au calcul fait par la fonction. En résumé, nous pouvons étendre ou modifier très simplement les règles métier sans manipulations compliquées dans le code de l'application. Une autre conséquence intéressante de cette approche déclarative en OWL2 est la possibilité d'ajouter de nouveaux concepts du domaine uti-

lisables dans de nouvelles règles. C'est donc bien plus puissant qu'un fichier texte contenant des paramètres qu'on ne peut que modifier sans pouvoir en ajouter d'autres non prévus par le développeur d'origine. Ainsi une telle application nécessite-t-elle moins de développeurs, de temps de développement et de maintenance. Cependant, l'ontologie OWL2 avec règles SWRL présente quelques limites. Comme évoqué en section 3.2, certains traitements, notamment de réification , sortent du cadre de la logique du premier ordre. Par exemple, la méthode checkApplicability() de la classe FPCond de la figure 10 pourrait être traduite par la règle suivante :
UnaryCond(?cond) ^ type(?cond, ?classe) ^?classe(?pat) -> applicable(?cond, ?pat)
où la variable ?classe est tantôt utilisée en objet du triplet type(?cond, ?classe) de propriété type, tantôt utilisée en propriété dans le triplet ?classe(?pat). Une autre limite importante citée précédemment est celle de l'impossibilité d'exprimer la négation dans le corps d'une règle SWRL. Par exemple, à la figure 9, le calcul d'une différence ensembliste avec l'instruction :

```
if df in contraindicates:
        usables.remove(df)
```

pourrait être exprimée avec la règle non SWRL (avec ?x0 un FBQuestionnaire et ?x un DefiPerso) :
NOT contraindicates(?x0, ?x) -> usables(?x0, ?x)
Enfin, l'impossibilité bien connue d'exprimer certains cas de jointures en OWL2 impose de décrire certaines connaissances statiques sous forme de règles SWRL. Ceci peut être une gêne dans la mesure où cela brise le principe de modélisation réservant les règles SWRL à la modélisation des règles métier, c'est-à-dire à la partie dynamique des connaissances (cf. section 3.1).

6 Perspectives

Un axe de nos futurs travaux est de conceptualiser les notions de profil et de parcours Patient. Nous allons évidemment nous intéresser au choix du défi par le patient suite aux recommandations faites par ORALOOS. Comme le feed-back mémorise de quelle relation non fonctionnelle est issue le défi choisi par le patient, nous allons définir avec les psychologues et pédagogues des profils pour le patient, par exemple : performant, persévérant, prudent, etc. Ensuite, en analysant plus finement le parcours de chaque patient et son profil, à partir de modèles machine learning ou de métarègles, nous pourrons moduler les règles de recommandation behavioristes.
Par ailleurs, nous voulons améliorer la modélisation statique OWL2 de l'ontologie : (i) introduire une hiérarchie dans les défis et l'espace documentaire, (ii) étudier le concept d'échéance (delay/deadline) d'un défi qui peut être interprété différemment selon les domaines et/ou professionnels de santé, (iii) retravailler la modélisation des conditions et (iv) conceptualiser la notion d'espace pour améliorer l'organisation de l'ontologie. Nous discutons maintenant des moyens d'atteindre un plus grand niveau de déclarativité en proposant des solutions aux problèmes évoqués en section 5.
Le problème du besoin de réification pourrait être résolu en utilisant la sémantique basée sur RDF de OWL2 qui permet des cas de réification. Cette sémantique n'est cependant pas la même que celle utilisée par les règles SWRL ou par les raisonneurs HermiT et Pellet inclus dans Owlready2 (qui est la sémantique directe d'OWL2 basée sur la théorie des modèles). Utiliser la sémantique basée sur RDF apparaît donc très compliqué car cela implique de changer une bonne partie du code de l'application et de gérer la cohabitation de plusieurs sémantiques différentes. Ainsi, soit nous changeons, dans la mesure du possible, la modélisation de l'ontologie pour éviter les cas de réification, soit nous gardons le code Python permettant ce genre de traitement.
La question de l'ajout de la négation dans les règles SWRL est plus ouverte. La première solution serait d'utiliser dans les règles des noms de prédicats définis dans la partie OWL2 de l'ontologie par des descriptions de classes utilisant la négation (c'est possible avec OWL2). De fait, nous augmentons alors la complexité de raisonnement relative à la partie OWL2 (en s'éloignant un peu plus de la traitabilité du profil EL). De plus, les raisonneurs HermiT et Pellet ne peuvent pas raisonner avec des négations dans des descriptions de propriétés. Ainsi cette solution ne couvrirait pas les cas de négation de prédicats binaires dans les règles.
La deuxième solution pour ajouter la négation dans les règles est d'adopter une stratégie de raisonnement différente de celle des raisonneurs HermiT et Pellet. En effet, il est possible de traduire les connaissances d'une ontologie OWL2 en profil EL (sans les extensions évoquées en section 2). en règles existentielles (ou datalog$^{\pm}$), qui sont des règles SWRL auxquelles nous ajoutons des variables existentielles en tête [13]. Il existe depuis quelques années des travaux permettant d'étendre à la négation les algorithmes des deux grandes familles permettant de raisonner avec ces règles existentielles (les algorithmes de saturation et les algorithmes de réécriture de requêtes) [1]. Cependant, ces travaux restent pour le moment essentiellement théoriques. En dehors de ces algorithmes, une autre possibilité est de skolémiser les variables existentielles des règles, c'est-à-dire de les remplacer par des termes fonctionnels uniques pour chaque variable. Nous obtenons alors des programmes datalog avec des fonctions, passant ainsi dans le monde de la programmation logique où de nombreux travaux décrivent l'ajout de la négation aux règles. Citons par exemple : (i) la SLDNF-resolution avec stratification et complétion du programme [16], (ii) la sémantique bien fondée [16], et même l'answer set programming (ASP) avec la sémantique des modèles stables [3]. Le choix de l'approche dépend alors de la sémantique de la négation souhaitée. Globalement cette solution impose donc de changer de raisonneur (et donc sans doute de framework et de langage de programmation) et de corriger la partie OWL2 de l'ontologie pour rester dans le profil EL strict.
Le dernier problème évoqué en section 5 est celui de la perte de séparation nette entre la modélisation statique en OWL2 et la modélisation dynamique en SWRL, dans le cas où des connaissances statiques ne peuvent être exprimées que par des règles. Une solution peut être d'abandon-

ner cette distinction pour s'orienter vers une modélisation initiale de toute la connaissance sous forme de règles existentielles. Mais alors en plus de perdre le principe de séparation statique-dynamique permettant une modélisation claire de la connaissance, nous abandonnons aussi tous les standards et outils du web sémantiques dont la diffusion est de plus en plus large, notamment en lien avec les bases de données de graphes et plus généralement avec le domaine des linked data.

7 Conclusion

Nous montrons comment créer une application d'ETP basée sur une ontologie avec recommandation automatique et adaptative de défis. Les temps d'exécution sont encourageants et permettent une mise en production de l'application à court terme. D'un point de vue génie logiciel, la présence d'une ontologie et d'un raisonneur amènent une dimension déclarative au développement, se traduisant par un code impératif plus petit, des besoins en développeurs moins grands et des temps de développement plus courts. Aller vers une plus grande déclarativité imposerait d'étendre l'expressivité des langages OWL2 et SWRL (en ajoutant par exemple la négation aux règles). Cela pourrait avoir comme conséquence une remise en question de l'application en profondeur (avec un changement possible de langage de programmation). De même, une mise à jour de l'ontologie vers le profil EL sans extension pourrait être nécessaire, ce qui, paradoxalement, constitue une réduction de l'expressivité. Ainsi la question importante devient-elle celle du meilleur compromis entre expressivité et déclarativité. Dans cette optique, le langage des règles existentielles est intéressant puisqu'il peut généraliser SWRL et le profil EL d'OWL2. Cependant l'ajout de la négation reste encore un problème ouvert et principalement théorique.

Références

[1] M. Alviano, M. Morak, and A. Pieris. Stable model semantics for tuple-generating dependencies revisited. In *36th ACM Symposium on Principles of Database Systems (PODS'17)*, page 377–388, 2017.

[2] S. Angeletou, M. Rigou, and S. Sirmakessis. A logic-based approach to learner assessment. In *the 1st Int. Conf. on Educational Technologies, Tenerife, Spain*, pages 200–205, December 2005.

[3] J.-F. Baget, L. Garcia, F. Garreau, C. Lefevre, S. Rocher, and I. Stéphan. Bringing existential variables in answer set programming and bringing non-monotony in existential rules : two sides of the same coin. *Annals of mathematics and artificial intelligence*, 82(1-3) :3–41, 2018.

[4] B. De Cat, B. Bogaerts, M. Bruynooghe, and M. Denecker. Predicate Logic as a Modelling Language : The IDP System. *CoRR*, 2014.

[5] A. Giordan. Education thérapeutique du patient : les grands modèles pédagogiques qui les sous-tendent. *Médecin des maladies métaboliques*, 4(3), 2010.

[6] S. Hamy-Shoshany. Freins et dynamiques à la mise en place de programmes d'éducation thérapeutique du patient en soins primaires. *Thèse de Docteur en Médecine, Université Claude Bernard - Lyon 1*, 2015.

[7] B. Hansel, P. Giral, L. Gambotti, A. Lafourcade, G. Peres, C. Filipecki, D. Kadouch, A. Hartemann, J.-M. Oppert, E. Bruckert, M. Marre, A. Bruneel, E. Duchene, and R. Roussel. A fully automated web-based program improves lifestyle habits and hba1c in patients with type 2 diabetes and abdominal obesity : Randomized trial of patient e-coaching nutritional support (the anode study). *J Med Internet Res*, 19(11), 2017.

[8] N. Henze and W. Nejdl. Logically characterizing adaptive educational hypermedia systems. In *In International Workshop on Adaptive Hypermedia and Adaptive Web-based Systems (AH 2003)*.

[9] P. Hitzler, M. Krötzsch, B. Parsia, P. Patel-Schneider, and S. Rudolph. Owl 2 web ontology language primer (second edition), 2012. https ://www.w3.org/TR/owl2-overview/.

[10] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosofand, and M. Dean. SWRL : A semantic web rule language combining OWL and RuleML, 2004. http ://www.w3.org/Submission/SWRL/.

[11] J.-B. Lamy. Owlready : Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine*, 80 :11 – 28, July 2017.

[12] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. Owl 2 web ontology language : Profiles (second edition), 2012.

[13] M.-L. Mugnier and M. Thomazo. An introduction to ontology-based query answering with existential rules. In *Reasoning Web Summer School*, 2014.

[14] C. Mulwa, S. Lawless, M. Sharp, I. Arnedillo-Sanchez, and V. Wade. Adaptive educational hypermedia systems in technology enhanced learning : A literature review. SIGITE '10, page 73–84, 2010.

[15] M. A. Musen. The protégé project : A look back and a look forward. *AI Matters*, 1(4) :4–12, June 2015.

[16] U. Nilsson and J. Maluszynski. *Logic, programming and PROLOG (2ed)*. 2001.

[17] R. Rigondet, A. Rigal, C. Desblès, Q. Lesaichot, J. Masurier, C. Cardenoux, D. Thivel, B. Pereira, C. Lambert, Y. Boirie, and M. Miolanne. Accompagnement familial à domicile et de proximité de l'obésité infanto-juvénile proxob : étude pilote de faisabilité en recherche interventionnelle en santé. *Nutrition Clinique et Métabolisme*, 33(1) :83 – 84, 2019.

[18] K. Sandid. Usage des nouvelles technologies en éducation thérapeutique du patient., 2019. https ://www.slideshare.net/KarimSandid/.