SPARE-LNC : un langage naturel contrôlé pour l'interrogation de traces d'interactions stockées dans une base RDF

Bryan Kong Win Chang, Marie Lefevre, Nathalie Guin, Pierre-Antoine Champin

SPARE-LNC : un langage naturel contrôlé pour l'interrogation de traces d'interactions stockées dans une base RDF

Bryan Kong Win Chang, Marie Lefevre, Nathalie Guin, Pierre-Antoine Champin

UNIVERSITÉ DE LYON, CNRS
Université Lyon 1, LIRIS, UMR 5205, F-69622, France

Résumé : Les traces issues de l'interaction d'un utilisateur avec un environnement informatique sont une source d'informations non négligeable concernant l'utilisation de celui-ci, si l'on est en mesure de les étudier correctement. Dans cet article, nous proposons un langage naturel contrôlé permettant à des analystes non-informaticiens d'interroger des traces stockées dans une base RDF. Nous expliquons les apports de ce langage, sa mise en œuvre ainsi que les évolutions envisagées suite aux premières mises à l'essai.

Mots-clés : Langage Naturel Contrôlé, Génération de Requêtes SPARQL, RDF, Interrogation de Traces

1 Introduction

Les traces sont communément définies comme le résultat d'une activité. Si l'on considère par exemple un logiciel informatique, une première définition des traces serait tout simplement ce que le logiciel produit, ce que le logiciel permet de réaliser comme action, ce qui résulte de cette action, comme par exemple un affichage visuel, mais aussi d'autres informations qui ne sont pas directement transmises à l'utilisateur. On retrouve ainsi dans ces traces au sens large les *logs*, ou encore les paramétrages d'un utilisateur concernant le logiciel.

Ces traces qui sont des restes, enregistrements ou témoins de l'activité qui a eu lieu, sont une source d'information primordiale pour différents utilisateurs de ces logiciels. Elles peuvent être utilisées en tant que source brute d'information, ou au contraire être traitées par un système tiers (ou même le système les générant) afin de réaliser d'autres actions.

Un exemple simple d'utilisation concerne par exemple les *logs*, qui peuvent contenir des éléments divers sur la manière dont s'est déroulé le programme et ainsi aider à le corriger lorsqu'il ne s'est pas exécuté correctement. Ici, l'expert accède directement au contenu des traces et agit en conséquence. D'autres systèmes utilisent également des mécanismes automatisés afin de fournir des données personnalisées aux différents acteurs du logiciel. Par exemple dans le domaine du jeu vidéo, les traces du joueur peuvent être utilisées pour adapter le comportement des personnages non-joueurs (Rubin & Ram, 2012). Sur le web, les traces peuvent être analysées pour fournir diverses recommandations (Zarka, 2013). Dans le domaine des *Environnements Informatiques pour l'Apprentissage Humain* (*EIAH*), l'analyse des traces représente un enjeu majeur pour la compréhension du comportement de l'apprenant et l'évaluation de ses connaissances afin de pouvoir fournir à ce dernier un accompagnement personnalisé lors de son apprentissage.

Cependant, malgré l'omniprésence des traces dans les logiciels, il est rare que celles-ci soient récoltées et stockées selon le même modèle et dans le même format. Or pour certains éléments de traces, et pour les traces en général, étant donné un domaine d'utilisation, on retrouve des mêmes besoins. Pour les logiciels éducatifs, ces besoins ont d'ailleurs donné lieu à des études afin de formaliser ce que sont ces données (nommées *indicateurs* (Dimitrakopoulou *et al.*, 2006)) que l'on calcule ou récupère afin d'instancier une représentation de ce que l'on sait sur un apprenant donné, le *profil d'apprenant*. Cependant, même si ces éléments ont pu faire l'objet d'une formalisation et d'une classification, la récupération des informations contenues dans les traces reste à la discrétion de ceux qui conçoivent le système.

Un premier pas vers l'homogénéisation des traces et l'explicitation de leur modèle a été proposé avec la notion de Système à Base de Traces (*SBT*). Le nom de Système à Base de Traces provient d'un parallèle avec les Systèmes à Base de Connaissances (Settouti *et al.*, 2009), où les traces viennent remplacer les connaissances. Une première proposition de ce formalisme est présenté par (Settouti *et al.*, 2009). Le méta-modèle proposé accorde une place prépondérante à la notion du temps, les traces étant représentées comme un ensemble d'observables situés dans le temps. Ce formalisme s'accompagne d'une étude des technologies potentielles pour l'implémentation de ces traces, qui a abouti à la création du kTBS.

KTBS, *kernel for Trace-Based System*, est ainsi une implémentation d'un SBT permettant de stocker les traces dans un ensemble de triplets RDF, l'ensemble agissant comme un RDF-store pour l'accès aux données (Champin *et al.*, 2011). Les requêtes de modification, ajout et récupération des données se font alors *via* le langage SPARQL. Toutefois, bien que le RDF et son langage associé SPARQL (version 1.1) satisfaisaient la majeure partie des besoins liée à l'exploitation des traces (Settouti, 2011), l'utilisation de ces technologies pose un nouveau problème à la démocratisation de l'usage du kTBS : comment permettre à des utilisateurs non-informaticiens d'accéder et d'utiliser le contenu des traces pour récupérer les informations qui les intéressent ?

Afin de répondre à cette problématique, nous présentons dans cet article un langage naturel contrôlé permettant l'interrogation des traces respectant le modèle SBT dans des triplestores [1]. La deuxième section de cet article est ainsi consacrée à un état de l'art concernant les outils existants pour l'interrogation des traces et des triplestores. Ensuite, nous présentons le langage en lui-même et son implémentation, avant de conclure par une discussion sur les utilisations actuelles puis par des perspectives de recherche.

2 État de l'art

Dans le but d'interroger une architecture basée sur RDF, il est pertinent de se pencher sur les différentes techniques utilisées pour faciliter l'accès de ces données à ce qui est appelé classiquement "l'utilisateur lambda" du web. Abstraire des requêtes SPARQL est en effet une problématique qui a déjà vu de nombreuses propositions plus ou moins réussies, et fait l'objet de plusieurs concours récompensant les outils de recommandation et de recherche permettant à l'utilisateur d'obtenir le plus facilement l'information qu'il cherche (Lopez *et al.*, 2013).

1. Un triplestore est une base de données spécialement conçue pour le stockage et la récupération de données RDF.

Les approches proposées afin de résoudre ce problème sont diverses. Une d'entre elle est l'approche des moteurs de recherche, dont l'utilisation intuitive fonctionne par la saisie d'un certain nombre de mots liés à la recherche, que chaque moteur interprétera à sa manière pour en sortir une liste de suggestions que l'utilisateur final est libre de choisir.

Une autre approche consiste à considérer un ensemble de requêtes que l'utilisateur final sera potentiellement amené à vouloir réaliser sur une base de données (Pradel *et al.*, 2012). Les principes de l'approche sont indépendants du domaine d'application, mais l'ensemble des patterns à définir est un travail d'ingénierie supplémentaire lourd pour chaque nouveau domaine.

Une dernière approche consiste à proposer des Langages Naturels Contrôlés (*LNC*). Il s'agit ici de restreindre le langage naturel pour en faire un langage utilisable sans ambiguïté. Les approches de ce genre sont beaucoup étudiées en anglais, où sont développés depuis longtemps des analyseurs sémantiques du langage (Montague, 1973). Par exemple, SQUALL (Ferré, 2012) est un langage, qui s'inspire de la théorie de Montague "Universal Grammar", théorie appuyant la possibilité de considérer les langages naturels et machines de la même manière (Montague, 1973). Sa théorie explique cette possibilité *via* la mise en application de procédés algorithmiques simples pour passer d'un langage naturel à des langages formels simples, ayant mené aux parsers de Montague. SQUALL utilise ainsi cette base pour traduire des questions posées en Anglais sur des données de DBpedia.

L'article présentant SQUALL est suivi en 2013 d'un article (Ferré, 2013) où les difficultés rencontrées lors de la traduction du langage au SPARQL sont explicités et où il est montré que l'expressivité de son langage est similaire à celui du SPARQL .

Dans le cadre d'une utilisation associée au kTBS, le langage SQUALL propose un langage indépendant de tout domaine, en se calquant sur le côté triplet "sujet prédication objet" du SPARQL lors de la traduction en langage formel. Cependant, le temps ou la date des informations ne sont pas considérées de manière particulière. Par exemple, le "Quand" ("When") est ainsi absent du langage. Cependant il reste un des langages les plus proches de la formulation utilisateur.

En parallèle des approches du web sémantique, de nombreux travaux ont été proposés dans le domaine des *EIAH* concernant la définition des indicateurs à partir des traces. On peut citer la capitalisation des requêtes, comme dans les travaux concernant la génération des indicateurs (Choquet & Iksal, 2007; Diagne, 2009; Gendron, 2010; Djouad, 2011). Il y a également les modèles qui servent à guider l'utilisateur dans son raisonnement pour la construction d'indicateurs (Dimitrakopoulou *et al.*, 2006) utilisés dans ces mêmes outils. Toutes ces approches présentent le défaut d'être liées à leur application ou à des contraintes supplémentaires concernant le contenu des traces, là où le kTBS recherche une indépendance au domaine.

Dans la section suivante, nous présentons notre Langage Naturel Contrôlé, indépendant du domaine et permettant de répondre à la limite de la prise en compte du temps induite par l'utilisation de SPARQL.

3 SPARE-LNC

Afin de présenter le langage, nous commençons par aborder le modèle des données à interroger puis le type du langage et les choix effectués afin de représenter les règles contraignant le LNC. Une fois les règles introduites, nous présenterons par des exemples les bases du langage,

avant d'introduire progressivement les différentes spécificités de ce dernier, pour terminer par l'implémentation du langage que nous avons effectuée.

3.1 Les données interrogées

Les traces du kTBS présentent un certain nombre de spécificités qui sont héritées du méta-modèle (Settouti *et al.*, 2009) dont il est issu. Une trace dans le kTBS est ainsi principalement composée d'un ensemble d'éléments indépendants, que l'on nomme des *obsels* (par contraction de *"observable elements"*). Chacun de ces obsels dispose d'attributs. Certains attributs sont présents dans tous les obsels, comme par exemple une date de début et une date de fin, ou encore le type de l'obsel. Chaque obsel peut ensuite posséder un nombre non défini d'attributs constituant les éléments considérés comme dépendants du domaine. Dans une logique de représentation RDF, ces obsels forment alors un ensemble de triplets liés par un sujet RDF unique pour un obsel donné. Les traces ayant pour but de récolter l'activité d'un individu, et selon ce que l'expert cherche à analyser, le détail et la durée d'une expérimentation, les traces peuvent rapidement atteindre une masse importante en terme de nombre de triplets RDF.

Afin de définir les bases de notre langage SPARE, il a fallu principalement s'intéresser à deux choses. La première concerne les fonctionnalités que le langage doit proposer afin de satisfaire les besoin exprimés concernant les interrogations de traces. La seconde est la forme que doit revêtir le langage, afin d'assurer autant que possible une utilisabilité convenable pour des non-informaticiens.

Dans le kTBS, bien qu'un modèle de traces décrivant les obsels pouvant être rencontrés dans la trace puisse être fourni, il n'est pas obligatoire. Il est possible de générer ce dernier à partir des traces, mais la génération *a posteriori* implique le risque de voir apparaitre des obsels dont le contenu n'est pas décrit dans le modèle généré car absent lors de sa génération. Par contre, le modèle de ces traces étant contraint par le méta-modèle du kTBS, nous avons choisi de nous intéresser plus particulièrement à ce dernier afin d'exprimer les besoins du langage.

Pour la forme, l'approche retenue est de se baser sur un langage d'interrogation textuel, et plus précisément un *LNC*. Si des interfaces graphiques permettant l'interrogation de traces existent (Settouti *et al.*, 2009), ces dernières sont généralement d'une manière ou d'une autre liées à leur domaine d'utilisation voire à l'application elle-même. Dans la mesure où aucune interface générique n'est associée au kTBS pour l'interrogation des traces, utiliser un langage textuel parsé tel qu'un *LNC* présente un certain nombre d'avantages, le premier étant la liberté de l'interface. À partir d'un langage n'ayant besoin que de texte, il est possible de définir des interfaces allant d'un simple champ textuel à compléter, à une interface graphique plus complète où le langage remplace simplement le SPARQL dans son utilisation. De plus, pour un domaine particulier, il est possible de reprendre le langage et de bâtir une interface adaptée. Le texte offre également une facilité supplémentaire pour l'échange de requêtes, en fournissant un format directement capitalisable, et dont le contenu, se basant sur le langage de l'utilisateur, est facilement compréhensible par l'utilisateur.

En résumé, dans le cadre d'une utilisation typique de notre approche (*cf. Figure 1*), afin d'utiliser le langage, l'utilisateur, à travers une interface personnalisée répondant aux besoins de l'application, choisit de manière directe (en écrivant les requêtes en LNC) ou indirecte (en utilisant des requêtes générées ou pré-écrites) une requête ou un ensemble de requêtes. Ces requêtes sont ensuite analysées par l'interpréteur du langage (ou parser), qui traduit la requête

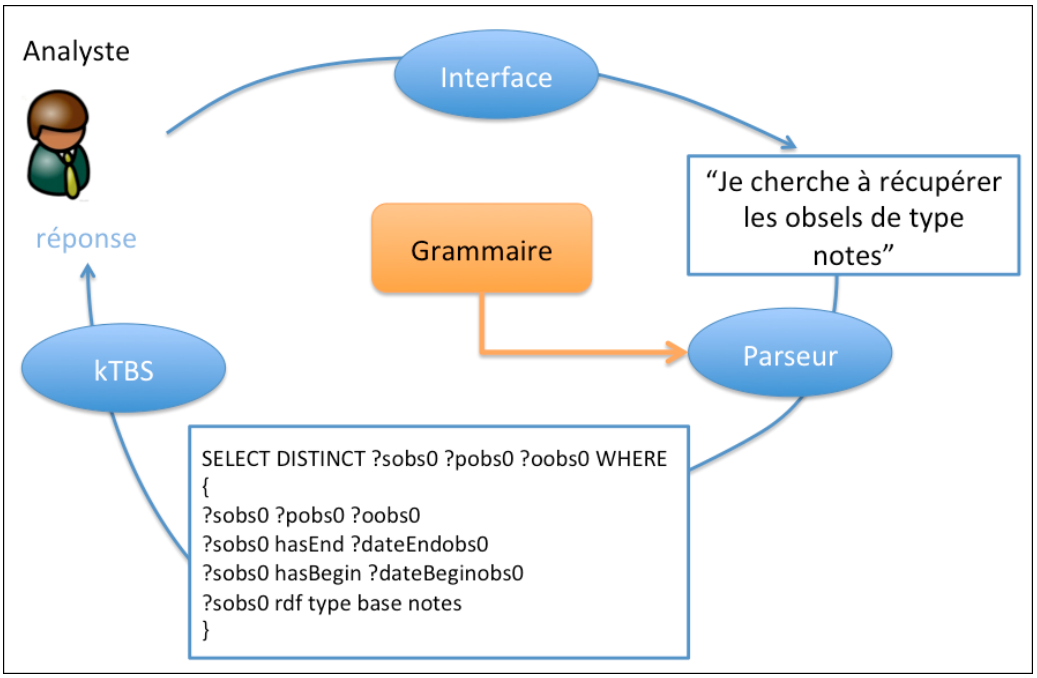

FIGURE 1 – Interrogation d'une trace RDF par l'utilisateur *via* SPARE-LNC

de l'utilisateur dans le langage compréhensible par le kTBS, le SPARQL. En réponse, le kTBS interprète la requête et retransmet la réponse à l'utilisateur.

3.2 Une grammaire pour SPARE-LNC

SPARE-LNC est l'abréviation pour SPArql REquest en Langage Naturel Contrôlé. Le principe du langage est ainsi de proposer une alternative au SPARQL pour interroger les traces stockées dans le système à base de traces kTBS. Pour formaliser le *LNC*, nous avons choisi d'opter pour un langage guidé par une grammaire. La formalisation du langage se base ainsi sur une grammaire algébrique. Le langage utilisé comme base est le français, mais l'adaptation de la grammaire pour de l'anglais, après première étude, ne semble pas poser de problèmes majeurs. L'entrée de l'utilisateur est analysée dans son intégralité *via* l'ensemble de règles composant la grammaire. L'ordre des règles n'est pas absolu, permettant certaines libertés. La *Figure 2* montre une version simplifiée de la grammaire de SPARE-LNC. La version complète de la grammaire est disponible en ligne (Kong Win Chang *et al.*, 2014). On peut différencier dans la définition de la grammaire deux groupes de règles. Le premier groupe (partie haute de la *Figure 2*) permet la création de phrases exprimant des conditions sur des éléments à récupérer dans les traces. Le deuxième permettant la définition de phrases gérant les éléments récupérés, par exemple en opérant des calculs ou des sélections sur ce qui est récupéré.

Le langage utilisé pour requêter la base est ainsi composé d'un ensemble de ces phrases qui satisfont la grammaire proposée. Chacune de ces phrases correspond ainsi à une sous-requête du langage, formant un texte décrivant les données à récupérer en énonçant un ensemble de contraintes et un ensemble d'actions à réaliser.

3.3 Contraintes sur les données

Les sous-requêtes exprimant les contraintes sur les données sont à la base du système d'interrogation. La phrase doit être simple et compréhensible par tout un chacun. La sous-requête la plus simple est ainsi composée de l'unique texte suivant : "Je cherche à récupérer les obsels.". Afin de faciliter l'utilisation *via* le texte, un certain nombre de phrases équivalentes existe dans le langage. Ainsi "Je cherche à récupérer les obsels" dispose d'un autre équivalent "Je veux récupérer les obsels".

Bien sûr, cet exemple de sous-requête ne permet pas vraiment de contraindre le contenu récupéré depuis la base. Toutefois il peut être intéressant de s'intéresser à une variante utilisable, "Je cherche à compter les obsels.", qui permet elle plus d'utilisations, comme par exemple, calculer une moyenne de notes ou compter le nombre d'évènements de la base.

```
liste_requête  = début* select?

début          = début_select1
               / début_nommé1
               / début_soit
               / début_parmi

début_select1  = "Je cherche à " action nommage point
début_nommé1   = "Je nomme" NOM_RES "les obsels" conditions_obsels* point
début_parmi    = "Parmi " NOM_RES début_select1

action         = " récupérer les obsels " conditions_obsels*
               / "récupérer les attributs" NOM_ATT? conditions_attributs*

conditions_obsels = " de type " NOM_TYPE
                  / " ayant un attribut " NOM_ATT?
                  / "suivi" within? "par un obsel " condobs_temp*

conditions_attributs = VALL_ATT
                     / "de valeur" VALL_ATT
                     / VALL_ATT "de " NOM

within           = " dans l'heure "
                 / " dans les " VALEUR " secondes qui suivent"
                 / " dans les " VALEUR " milisecondes qui suivent"

point            = " . "
point_virgule    = " ; "
-------------------------------------------------------------------------
select         = "Je sélectionne les requêtes : " (NUM, " : ")
                 + ( NOM, " : " ) "."

début_soit     = " Soit " NOM_RES action point
               / " Soit " NOM_RES " = " equation point_virgule
```

FIGURE 2 – Extrait court de la grammaire de SPARE-LNC

3.3.1 Les attributs

À ces sous-requêtes de base s'ajoutent les conditions les plus simples : les conditions sur les attributs, par exemple, ou sur le type de l'obsel. Pour ajouter une condition sur un attribut, l'utilisateur doit globalement pouvoir, d'une part, rechercher les obsels possédant un attribut particulier, et d'autre part, pouvoir poser une condition sur sa valeur. Le premier cas s'exprime par "Je veux récupérer un obsel ayant un attribut Notes", à qui on peut rajouter "de valeur

supérieure à 10." par exemple. Il est à noter que le nom de l'attribut est optionnel, et l'utilisateur pourra écrire : "Je veux récupérer les obsels ayant un attribut de valeur supérieure à 10.". Bien sur, cela implique que l'on récupérera potentiellement, si on ne précise rien d'autre, d'autres événements qui disposent d'un attribut numérique répondant à la condition.

Certains attributs sont communs à toutes les traces. Ces derniers étant des éléments importants pour l'interrogation des traces, le langage propose des formulations propres afin d'interroger ces derniers. Par exemple pour récupérer un ensemble d'obsel d'un type donné, il faut interroger l'attribut commun à toutes les traces nommé *"hasType"*. L'utilisateur pourra écrire "Je veux récupérer les obsels de type contrôle." au lieu de "Je veux récupérer les obsels ayant un attribut hasType de valeur contrôle.". Un autre ensemble d'attributs communs a eu droit à une considération toute particulière de par leur place centrale pour les traces du kTBS : l'ensemble des attributs temporels.

3.3.2 Les opérateurs temporels

Dans le but de pouvoir extraire des données des traces, les notions de position temporelle relative ou absolue d'un événement par rapport à un autre sont cruciales pour l'expression des conditions. L'expression des opérateurs permettant la vérification temporelle n'est pas impossible en SPARQL. Cependant, si l'on souhaite exprimer "un obsel A suivant un obsel B", l'utilisateur se retrouvera à écrire un ensemble de triplets conséquent, qui de plus ne sont pas forcément intuitifs.

Pour palier ce problème, nous avons décidé d'implémenter ces opérateurs temporels dans notre langage, en prenant comme base les opérateurs de Allen (Allen, 1983) qui expriment un ensemble de 13 relations possibles entre deux événements exprimés sous la forme d'intervalles. Ces treize opérateurs sont en fait sept opérateurs et leurs opposés, l'un d'entre eux étant son propre opposé (l'égalité). Ces opérateurs sont : *before, contains, overlaps, meets, starts, finishes, equals, after, during, overlapped by, met by, started by, finished by*. Chacun des opérateurs a son équivalent dans le langage SPARE-LNC. Un obsel de type A suivant un obsel de type B s'écrira tout simplement : "Je cherche à récupérer les obsels de type A suivi par un obsel de type B."

3.3.3 Liens et références

Si l'on utilise l'ensemble des contraintes exprimées ci-dessus dans une seule et même sous-requête, on risque de se retrouver avec des sous-requêtes extrêmement longues. C'est pourquoi nous proposons des possibilités de définir des sous-résultats durant la définition de la requête *via* un système de nommage. La sous-requête "Je veux récupérer les obsels de type contrôle ayant un attribut note de valeur supérieure à 10." peut se diviser et devenir : "Je nomme Controle les obsels de type contrôle. Parmi Controle, je cherche à récupérer les obsels ayant un attribut note de valeur supérieure à 10.". *Controle* devient l'identifiant de la sous-requête, et peut ainsi être repris comme entrée pour d'autres sous-requêtes.

Il existe ainsi dans le langage trois moyens de nommer un résultat. Le premier est de commencer la phrase par *"Je nomme"* avant de donner l'identifiant. La deuxième manière est de finir par *"que je nomme"*. Bien que la formulation ne soit pas forcément la plus lisible, cela permet de rajouter rapidement un identifiant à une phrase n'en disposant pas sans toutefois trop altérer la lisibilité. Enfin, la troisième façon est similaire à *"Je nomme"*, et utilise le mot clef *"Soit"*.

"Soit" a une autre utilité. Une phrase commençant par *"Soit"* a les mêmes possibilités de base qu'un *"Je nomme"*, mais permet également d'introduire des calculs mathématiques.

Par exemple, soit la requête suivante : "Je nomme Notes les obsels de type note. Parmi Notes, je cherche à récupérer les attributs 'resultat' que je nomme A. Soit Moyenne = AVG(A) ;" On récupère les obsels nous intéressant dans les deux premières phrases avant de calculer la moyenne dans la troisième. Les expressions mathématiques acceptées par *"Soit"* sont les mêmes que celles du SPARQL, et ont les mêmes contraintes de variables que le langage originel. Du fait que le point est utilisé en RDF pour l'expression de certaines valeurs numériques, une expression mathématique utilisant le mot clef *"Soit"* se termine par un point virgule au lieu d'un point.

Les contraintes sur les opérations possibles entre deux types de variables SPARQL se retrouvent aussi lors de la spécification de sous-requêtes dépendantes d'une précédente. Si on reprend le résultat d'une sous-requête à l'aide du mot clef *"Parmi"*, comme dans l'exemple précédent, "Notes" fait référence à l'ensemble des observables de type "note" et ne peut pas être l'objet d'une opération arithmétique. De même, "A" est un ensemble de variables numériques, et ne peut pas faire l'objet de filtres supplémentaires.

Définir un ensemble de sous-requêtes n'est cependant pas suffisant pour renvoyer un résultat en particulier. Le résultat voulu par l'utilisateur n'est pas forcément le résultat d'une des sous-requêtes, mais potentiellement plusieurs. Par défaut, le langage considère tous les résultats des sous-requêtes comme résultat potentiel pour l'utilisateur. Le résultat par défaut est ainsi l'union des résultats des sous-requêtes. Pour définir ce qu'il faut récupérer, il faut utiliser une requête spécifique indiquant quelles variables récupérer, qui s'écrit tout simplement *"Je garde uniquement ..."* suivi de la liste des éléments à récupérer. Ces éléments de la liste peuvent être exprimés de deux manières, soit par l'utilisation des identificateurs proposés par l'utilisateur lors de la définition de la sous-requête (*via* le mécanisme de nommage), soit par des numéros de phrase dans l'ordre de la déclaration.

3.4 Implémentation

Pour une première implémentation du langage, et voulant appliquer les technologies du web aux traces stockées sous format RDF sur un kTBS en ligne dans le cadre d'un MOOC (Massive Open Online Course), le choix s'est dirigé pour une implémentation javascript du parser de grammaire avec du html pour les interfaces. Afin d'utiliser notre langage, l'utilisateur manipule ainsi une interface html qui lui permet de définir une requête du langage, qui est ensuite traduite en requête compréhensible par le kTBS, qui interprète et renvoie à l'utilisateur sa réponse (*cf. Figure 1*) .

Pour implémenter le parseur, nous avons utilisé un outil de génération de parser pour javascript [2] utilisant en entrée un pseudo-code javascript contenant les règles du langage. Le parseur résultant est structurellement une fonction javascript facilement utilisable prenant en paramètre une chaine de caractères contenant la requête en langage naturel contrôlé et ayant pour sortie une requête SPARQL directement utilisable sur le kTBS.

Le parser procède par sous-requêtes, avant de construire les liens entre les différentes sous-requêtes grâce au système d'identifiants. Ainsi, le parser extrait l'ensemble des conditions expri-

2. http://pegjs.org/

mées par chaque phrase et construit une première table de liens qui contient les identifiants référençant des sous-requêtes trouvés lors de l'analyse et une deuxième table contenant l'expression SPARQL des conditions. Après une première analyse du texte pour récupérer l'ensemble de ces tables, une deuxième lecture est réalisée pour résoudre les liens entre les sous-requêtes. Cette deuxième passe opère par un parcours simple dans les tables de liens en ajoutant les tables référées à la table en cours de traitement. La requête finale SPARQL est ensuite extraite soit par la fusion des tables dans le cas où la sous-requête de sélection est absente, soit par la sélection exprimée par la sous-requête de sélection.

Nous avons également codé deux interfaces pour le langage [3]. Une première est une simple interface textuelle, qui se caractérise par l'usage de plusieurs champs textes, dont un pour l'écriture de la requête et l'autre pour l'affichage de la requête SPARQL correspondante. Cette première interface textuelle est pratique, mais l'utilisation nécessite de connaître le langage. La deuxième interface est une interface graphique pour construire des sous-requêtes du langage. Les différents mots du langages sont accessible *via* des boutons qui les ajoutent à une liste pour construire la requête. Chacune des sous-requêtes peut être sélectionnée par l'utilisateur et ce dernier peut rajouter autant de sous-requêtes qu'il veut par l'utilisation des boutons ajoutant les débuts de sous-requête. Lorsque une sous-requête est sélectionnée, l'interface désactive les boutons qui ne permettent pas de construire une sous-requête grammaticalement correcte. Une fois la requête terminée, l'utilisateur peut interroger un kTBS. Il est également possible pour l'utilisateur de consulter la requête en langage naturel contrôlé et la requête SPARQL. L'affichage de la requête à l'utilisateur lui permet d'avoir accès au contenu de la requête et de la copier ou de la modifier directement s'il le souhaite.

4 Utilisation du langage et limitations

Le développement du parser a suivi la logique du développement progressif des fonctionnalités du langage. Les premières fonctionnalités ont été la récupération des obsels sans conditions et la possibilité de compter le nombre d'obsels d'une trace. Chaque étape d'ajout d'une nouvelle fonctionnalité se suivait d'une étape de validation de la requête SPARQL générée. La validation consistant à vérifier que la requête générée est valide et génère le résultat souhaité exprimé par SPARE-LNC. La vérification des fonctionnalités a eu lieu en premier lieu sur des bases de traces jouets contenant l'ensemble des cas recherchés pour tester la fonctionnalité nouvellement implémentée, puis sur des bases de traces d'un MOOC [4] utilisant le kTBS. La mise à disposition de ces bases de traces a notamment permis de débuter les tests du langage dans le cadre de la création d'indicateurs d'EIAH (trois pour le moment), retranscrits *via* les deux interfaces.

Cependant l'utilisation du langage par des informaticiens ou des non-informaticiens a posé un certain nombre de problèmes et de questions concernant l'expressivité du langage. Un des premiers points concerne l'équivalence entre SPARE-LNC et le SPARQL. Si le langage naturel contrôlé SPARE permet de s'affranchir du SPARQL et de n'écrire qu'un ensemble de phrases en français, il existe des opérations que SPARE-LNC ne couvre pas. L'une de ces opérations

3. Les interfaces de manipulation du langage SPARE-LNC ainsi que des exemples de requêtes SPARE-LNC associées à leur équivalent SPARQL sont disponibles sur http://liris.cnrs.fr/~bkongwin/SPARE/

4. MOOC FOVEA, Session primtemps 2014, environ 4000 inscrits :
http://anatomie3d.univ-lyon1.fr/webapp/website/website.html?id=3346735&pageId=223206

manquantes est par exemple la possibilité d'ordonnancer la réponse de la requête par rapport à certains critères. Trier un ensemble de notes directement *via* une seule sous-requête n'est pas possible, bien qu'il soit possible d'utiliser un moyen dévié, comme récupérer une par une les notes de l'ensemble en séparant l'ensemble entre la meilleure note et le reste. Ces opérations sont réalisables de façon simple sur les données finales, et doivent être considérées comme un prochain ajout au langage, bien qu'il faille réaliser un certain travail en amont. Car s'il est facile d'ajouter une règle au langage par le simple ajout de règles supplémentaires à la grammaire, il convient de s'assurer qu'une phrase grammaticalement correcte au sens de SPARE n'ait pas plusieurs interprétations.

Ce point tout particulier est un point d'importance majeure dans SPARE, où l'on cherche à permettre plusieurs formulations similaires. Par exemple, si l'on veut exprimer qu'un élément est précédé par un autre et se déroule en même temps qu'un troisième, la phrase SPARE-LNC correcte serait : "Je cherche à récupérer les obsels de type A suivi par un obsel de type B, pendant un obsel de type C.". Si maintenant on veut rajouter que l'événement de type B de notre exemple se déroule en même temps qu'un événement de type D et est suivi par un événement de type E, on devrait écrire : "Je cherche à récupérer les obsels de type A suivi par un obsel de type B pendant un obsel de type D et suivi par un obsel de type E, pendant un obsel de type C.". Nous voyons ici qu'il est possible d'avoir plusieurs interprétations d'une même phrase. Afin d'éviter des confusions, le changement d'un référant dans la phrase se fait *via* l'utilisation du mot clef *"lui-même"* que l'on ajoute après la déclaration du nouvel obsel. Bien sûr, cette solution ne fait que déplacer le problème à une ou deux conditions temporelles plus tard, c'est pourquoi nous conseillons de séparer la grande sous-requête en plusieurs requêtes plus petites pour éviter toute ambiguïté.

Une autre contrainte forte a aussi dû être posée lorsque l'on doit introduire des conditions sur des éléments RDF. Si le parser reconnaît les chaînes de caractères sans espace et les nombres sans problèmes, tout élément inclus dans les conditions contenant un caractère spécial ou un espace doit être mis entre guillemets simples. Un bon exemple serait si l'on veut récupérer tous les obsels du 21 janvier d'une personne du nom de Francis Dac. La requête s'exprimerait "Je veux récupérer les obsels ayant un attribut nom de valeur 'Francis Dac' et ayant un attribut date de valeur '21-01-2015'". Sans la contrainte concernant les guillemets afin de délimiter les valeurs des attributs, la phrase pourrait alors avoir plusieurs interprétations différentes, l'une d'elle étant de rechercher tous les obsels ayant un attribut appelé "nom" et ayant une valeur "Francis Dac et ayant un attribut date de valeur 21-01-2015".

Or cette utilisation d'informations provenant directement de la base RDF dans le langage SPARE-LNC a été entre autres l'un des principaux points de discussion lors de rencontres avec des utilisateurs potentiels. En effet, les éléments de la base du MOOC contenaient un très grand nombre de liens http qui étaient difficiles à manipuler pour un utilisateur non-informaticien. Ceci a mené à l'éventualité d'affecter à un ingénieur expert de la trace la tâche de mettre en place un équivalent entre certains éléments contenus dans la trace et une formulation adaptée pour l'utilisateur. Cependant, les formulations de ces équivalents étant dépendants à la fois du vocabulaire de l'utilisateur et de la formulation choisie dans les traces, ces adaptations seraient à effectuer du côté des interfaces de l'application ou du groupe d'applications utilisant ces adaptations.

Concernant la compréhension des sous-requêtes, la présence obligatoire des termes du méta-modèle ('obsels', 'attributs'...) dans les requêtes a également soulevé des inquiétudes. Ces no-

tions nouvelles d'obsels et d'attributs d'un obsel peuvent dans l'absolu être supprimées sans changer le sens de la requête. Cependant, si la phrase en SPARE-LNC "Je veux les obsels de type note." n'est pas différente de la phrase en français "Je veux les notes", il existe l'ambiguïté de savoir si on parle des notes comme d'un type d'obsel ou comme d'un attribut ou encore comme la valeur d'un des attributs. Cette simplification n'est pas sans solution, la plus simple étant de récupérer dans ce cas les éléments de type notes, les éléments ayant un attribut notes et les éléments ayant un attribut de valeur notes, mais nous considérons que familiariser l'utilisateur avec les éléments de la logique des traces lui permettra de mieux comprendre à l'usage le contenu de ses traces et ainsi mieux les appréhender.

5 Conclusion et perspectives

Le langage SPARE-LNC et son implémentation en tant que parser permet la génération de requêtes SPARQL fonctionnelles. Les premières utilisations du langage directement par des utilisateurs non impliqués dans la genèse de ce dernier ont donné lieu à des discussions dont il faut tenir compte pour espérer une utilisation plus large de SPARE-LNC dans le cadre du kTBS. Ces discussions mettent en avant l'importance de proposer un certain nombre d'évolutions pour SPARE-LNC. L'inquiétude soulevée par l'utilisation du vocabulaire spécifique aux traces est par exemple un point qui peut être effacé par la réalisation d'une interface spécifique à l'application ciblée. La question se pose alors d'évaluer le coût de la réalisation d'une telle interface par rapport à une utilisation directe du langage d'origine que SPARE-LNC est censé remplacer. Pour un programme n'ayant besoin que d'une ou deux requêtes SPARQL, il sera probablement plus rapide de définir directement ces requêtes dans le programme plutôt que d'utiliser SPARE-LNC. SPARE-LNC est par contre plus utile lorsqu'écrire des requêtes pour un kTBS occupe une place importante dans les fonctionnalités du logiciel. C'est le cas par exemple de l'outil Samotrace-me (Cordier *et al.*, 2015) qui intègre SPARE-LNC pour calculer des valeurs à partir d'éléments récupérés dans les traces d'un kTBS.

Dans les deux cas, il reste des avantages à l'utilisation de SPARE-LNC. Du fait de son rapprochement avec le langage naturel, SPARE-LNC propose une description intelligible de ce que fait la requête, pour peu que la personne lisant la requête soit familière avec les notions de trace et d'obsel. Cette description propose aussi la particularité interessante d'être le format de stockage de base de la requête, et peut ainsi être échangé directement en format textuel, facilitant la réutilisation des requêtes déjà définies.

La réutilisation de requêtes amène également la question de la pertinence des requêtes d'une trace à une autre. En se basant sur le méta-modèle du kTBS, SPARE-LNC décrit des requêtes toujours valides pour un kTBS, mais il n'assure pas que le résultat soit cohérent d'une trace à l'autre. En effet, les modèles des observables contenus dans deux traces différentes ne sont pas forcément les mêmes. Il serait du coup intéressant de se pencher sur la possibilité de créer des patrons de requêtes pour SPARE-LNC, décrivant la requête en langage naturel contrôlé et les conditions que doivent satisfaire les différents obsels faisant l'objet de conditions dans le patron.

Le fait que l'utilisateur puisse potentiellement adapter les requêtes à ses propres besoins présuppose cependant une certaine compréhension de ce que contient la trace et de ce qu'il peut y récupérer. Dans les interfaces actuelles de SPARE, l'interface textuelle ne propose pas de représentation de la trace, et l'interface graphique propose des récupérations systématiques

de toutes les données, ce qui est peu envisageable sur des bases de traces très peuplées. La recherche et la proposition de nouvelles interfaces viables intégrant le contenu des traces pour le langage SPARE-LNC apparait alors comme une priorité pour son utilisation.

Références

ALLEN J. F. (1983). Maintaining knowledge about temporal intervals. *Commun. ACM*, **26**(11), 832–843.

CHAMPIN P.-A., PRIÉ Y., AUBERT O., CONIL F. & CRAM D. (2011). *kTBS : Kernel for Trace-Based Systems*. Software, LIRIS.

CHOQUET C. & IKSAL S. (2007). Modélisation et contruction de traces d'utilisation d'une activité d'apprentissage : une approche langage pour la réingénierie d'un eiah. *Revue STICEF*.

CORDIER A., DERBEL F. & MILLE A. (2015). *Observing a web based learning activity : a knowledge oriented approach*. Research report, hal-01128536, LIRIS.

DIAGNE F. (2009). *Instrumentation de la supervision de l'apprentissage par la réutilisation d'indicateurs : Modèles et Architecture*. Thèse de doctorat en informatique, Université Joseph Fourier - Grenoble.

DIMITRAKOPOULOU A., PETROU A., MARTINEZ A., MARCOS J. A., KOLLIAS V., JERMANN P., HARRER A., DIMITRIADIS Y. & BOLLEN L. (2006). *State of the art of interaction analysis for Metacognitive Support and Diagnosis*. Research report, Kaleidoscope.

DJOUAD T. (2011). *Ingénierie des indicateurs d'activités à partir de traces modélisées pour un Environnement Informatique d'Apprentissage Humain*. Thèse de doctorat en informatique, Université Claude Bernard Lyon 1.

FERRÉ S. (2012). Squall : A controlled natural language for querying and updating rdf graphs. In S. B. HEIDELBERG, Ed., *Controlled Natural Language*, p. 11–25.

FERRÉ S. (2013). squall2sparql : A translator from controlled english to full sparql 1.1. In *Proceeding of the Question Answering over Linked Data lab QALD-3 at CLEF lab*.

GENDRON E. (2010). *Cadre conceptuel pour l'élaboration d'indicateurs de collaboration à partir des traces d'activité*. Thèse de doctorat en informatique, Université Claude Bernard Lyon 1.

KONG WIN CHANG B., GUIN N., LEFEVRE M. & CHAMPIN P.-A. (2014). *Conception d'un langage d'interrogation des traces accessible à des non-informaticiens*. Research Report RR-LIRIS-2014-015, LIRIS.

LOPEZ V., UNGER C., CIMIANO P. & MOTTA E. (2013). Evaluating question answering over linked data. *Web Semantics Science Services And Agents On The World Wide Web*, **21**, 3–13.

MONTAGUE R. (1973). The proper treatment of quantification in ordinary english. In H. ET AL. (EDS.), Ed., *Approaches to Natural Language*, p. 221–242.

PRADEL C., HAEMMERLÉ O. & HERNADEZ N. (2012). A semantic web interface using patterns : The swip system. In M. CROITORU, S. RUDOLPH, N. WILSON, J. HOWSE & O. CORBY, Eds., *Graph Structures for Knowledge Representation and Reasoning*, volume 7205 of *Lecture Notes in Computer Science*, p. 172–187.

RUBIN J. & RAM A. (2012). Capturing and adapting traces for character control in computer role playing games. In *Proceedings of the ICCBR 2012 Workshop TRUE*, p. 193–201.

SETTOUTI L. S. (2011). *Systèmes à Base de traces modélisées - Modèles et langages pour l'exploitation des traces d'Interactions*. Thèse de doctorat en informatique, Université Claude Bernard Lyon 1.

SETTOUTI L. S., PRIÉ Y., CHAMPIN P.-A., MARTY J.-C. & MILLE A. (2009). *A Trace-Based Systems Framework : Models, Languages Semantics*. Research report, inria-00363260-v2, LIRIS.

ZARKA R. (2013). *Trace-Based Reasoning for User Assistance and Recommendations*. Thèse de doctorat en informatique, INSA de Lyon.