Une approche fondée sur le raisonnement à partir de cas pour la mise à jour interactive d'objets du Web sémantique

Alice Hermann, Mireille Ducassé, Sébastien Ferré, Jean Lieber

Une approche fondée sur le raisonnement à partir de cas pour la mise à jour interactive d'objets du Web sémantique

Alice Hermann[1,2,3], Mireille Ducassé[4], Sébastien Ferré[5] et Jean Lieber[1,2,3]
[1]Université de Lorraine, LORIA, UMR 7503 — 54506 Vandœuvre-lès-Nancy, France,
Prénom.Nom@loria.fr
[2]CNRS — 54506 Vandœuvre-lès-Nancy, France
[3]Inria — 54602 Villers-lès-Nancy, France
[4]INSA Rennes, IRISA, UMR 6074 – 35042 Rennes cedex, FRANCE,
ducasse@irisa.fr
[5]Université de Rennes 1, IRISA, UMR 6074 –35042 Rennes cedex, FRANCE,
ferre@irisa.fr

Résumé

Ce papier présente UTILIS, un outil du Web sémantique pour aider les utilisateurs à mettre à jour des objets RDFS. Les fondements d'UTILIS ont été réinterprétés selon les paradigmes du raisonnement à partir de cas. La recherche de cas utilise un ensemble de règles de relâchement pour trouver les cas sources similaires à l'objet en cours de modification. Ces cas sont utilisés comme suggestions pour étendre la description de l'objet. UTILIS fournit des suggestions finement appropriées à l'objet en cours de modification. Une expérience utilisateur a prouvé qu'UTILIS était déjà utile, en particulier la base résultant de l'utilisation d'UTILIS était plus consistante et contenait moins d'erreurs que lors de l'utilisation d'un outil de référence. La ré-interprétation en raisonnement à partir de cas, d'une part, montre trois pistes d'améliorations qui peuvent immédiatement bénéficier à UTILIS. D'autre part, l'algorithme efficace qui produit des suggestions, développé pour cet outil, pourrait bénéficier aux systèmes de raisonnement à partir de cas.

Mots clés : mise à jour d'objets, Web sémantique, RDFS, formulaire de saisie, recherche de cas par relâchement

1 Introduction

Le raisonnement à partir de cas (RÀPC) [24] a pour but de résoudre des problèmes à l'aide de solutions de problèmes déjà résolus. Cet article reformule, selon les paradigmes du RÀPC, les bases d'UTILIS (*Updating Through Interaction in Logical Information Systems*) [15], un outil du Web sémantique pour aider les utilisateurs à mettre à jour des objets RDFS, où RDFS est un formalisme bien connu du Web sémantique [13].

UTILIS recherche les objets similaires à l'objet mis à jour, c'est-à-dire les objets ayant des propriétés et des valeurs en commun avec lui. Les objets sont directement utilisés comme suggestions pour étendre la description de l'objet mis à jour. UTILIS fournit des suggestions finement appropriées à l'objet mis à jour ou en cours de création.

La contribution principale de cet article est un enrichissement mutuel entre le RÀPC et le Web sémantique. La ré-interprétation en RÀPC, d'une part, montre trois pistes d'amélioration qui peuvent immédiatement bénéficier à UTILIS. Une des perspectives concerne l'étape de réutilisation avec une adaptation des cas sources au cas cible. Une autre perspective concerne l'ordonnancement des cas sources à une même distance. D'autre part, l'algorithme efficace qui produit des suggestions, développé pour cet outil, pourrait bénéficier aux systèmes de RÀPC.

Dans la suite, les exemples et l'expérience sont liés à la mise à jour d'une base d'annotation de cases de bandes dessinées. Un extrait de cette base est montré en figure 1. La case A est une case de la collection *Ma vie à deux*, avec Missbean et son chat comme personnages ainsi qu'une bulle de dialogue dite par Missbean à son chat.

Case A > collection : Ma vie à deux > personnage : Missbean, Chat de Missbean > bulle : (a bulle de dialogue, dit par Missbean, s'adresse à Chat de Missbean)	Case B > collection : Calvin et Hobbes > personnage : Calvin, Hobbes > bulle : (a bulle de dialogue, dit par Calvin, s'adresse à Hobbes)
Case C > collection : Peanuts > personnage : Snoopy, Sally Brown > bulle : (a bulle de dialogue, dit par Sally Brown, s'adresse à Snoopy) > bulle : (a bulle de pensée, pensé par Snoopy)	Case D > collection : Ma vie à deux > personnage : MissBean, Babybean > bulle : (a bulle de dialogue, dit par MissBean, s'adresse à Babybean) > bulle : (a bulle de dialogue, dit par Babybean, s'adresse à Missbean)

FIGURE 1 – Extrait de la base d'annotation de cases de bandes dessinées. `a` est une abréviation pour `rdf:type`.

L'approche la plus courante en RÀPC, selon le modèle des $4R$ de [1], se compose de quatre étapes : (1) *retrieve*, (2) *reuse*, (3) *revise* et (4) *retain*. La première étape consiste à retrouver, dans la base de cas, les cas sources les plus similaires au cas cible. Dans UTILIS, la recherche de cas utilise un ensemble de règles de relâchement, inspiré par les travaux d'Hurtado et al. [16], et un algorithme efficace pour le calcul des suggestions. Les propriétés déjà connues d'un nouvel objet sont utilisées pour en suggérer d'autres. Supposons qu'un utilisateur ajoute une case et indique la collection de bande-dessinée à laquelle elle appartient, cette dernière peut aider à suggérer des personnages, il est probable que cette case et celles de la même collection aient des personnages en commun. Les cas sources sont directement utilisés par UTILIS. Ici, les suggestions faites à l'utilisateur sont les personnages des cases de la même collection que la nouvelle case. Cela implique que l'étape de réutilisation est une simple copie pour cette version d'UTILIS. Les cas sources pourraient être adaptés au cas cible pour fournir une meilleure solution. La solution proposée à l'utilisateur peut être testée. Si la solution est réussie, la cible complétée est ajoutée à la base de cas. C'est l'étape d'apprentissage. Sinon, l'étape de révision est commencée : le système essaye de déterminer les raisons de l'échec. Dans UTILIS, quand un utilisateur décide que la description d'un objet est complète, ce cas est ajouté à la base de cas et sera utilisé pour les prochaines suggestions. Actuellement, seules la première et la quatrième étapes sont implémentées dans UTILIS. Les perspectives pour améliorer les suggestions sont discutées à la fin de cet article. Elles concernent les étapes de recherche, de réutilisation et de révision. Pour l'étape de recherche, l'ajout d'une seconde étape à l'algorithme permettrait d'ordonner les cas sources à une même distance. La seconde étape utilisera une description étendue des objets. Actuellement, aucune adaptation n'est faite. Une adaptation permettra d'utiliser des cas sources moins similaires au cas cible. Une étape de révision peut être réalisée en utilisant les précédents choix des utilisateurs.

La section 2 donne des définitions relatives aux langages du Web sémantique. Elle introduit les systèmes d'information logiques utilisés pour l'interaction avec l'utilisateur et la problématique du guidage pour la mise à jour. La section 3 présente la création de la description d'une case. La section 4 rappelle les caractéristiques d'UTILIS qui sont essentielles pour cet article (voir [15] pour plus de détails). La section 5 compare notre approche aux travaux relatifs au RÀPC. La section 6 présente les perspectives. La section 7 présente une conclusion.

2 Préliminaires

2.1 RDF et RDFS

RDF et RDFS sont des langages du Web sémantique. Les éléments de base de ces langages sont les ressources et les triplets. Une ressource peut être soit un URI (nom absolu d'une ressource), un littéral ou une ressource anonyme. Un triplet est composé de 3 ressources. Un triplet *(s, p, o)* peut être lu comme une phrase où *s* est le *sujet*, *p* est le verbe, appelé *prédicat*, et *o* est l'*objet*.

Français	Quelles sont les cases avec au moins une bulle dite par MissBean ?
Requête SPARQL	SELECT ?x WHERE { ?x a :case . ?x :bulle ?y . ?y :ditPar <MissBean>)}

FIGURE 2 – Une question et sa traduction en requête SPARQL.

RDF [18] permet de représenter des données. Par exemple, dans la base d'annotation le triplet (*<CaseK>, :personnage, <Missbean>*) peut être lu comme "La case K a pour personnage Missbean". RDF possède un vocabulaire prédéfini pour représenter l'appartenance à une classe de ressource (`rdf:type`). Par exemple, le triplet (*:Bulle1,* `rdf:type`*, :BulleDeDialogue)* dit que "La bulle 1 est de type BulleDeDialogue" ou plus simplement "La bulle 1 est une bulle de dialogue". La ressource *:BulleDeDialogue* est une classe.

De plus, RDFS [13] est un langage pour structurer les ressources RDF. RDFS possède un vocabulaire prédéfini avec une sémantique fixée, en particulier les propriétés `rdfs:subClassOf` et `rdfs:subPropertyOf`. Par exemple, le triplet (*:BulleDeDialogue,* `rdfs:subClassOf`*, :Bulle)* dit que "La classe BulleDeDialogue est une sous-classe de Bulle", ou plus simplement "Chaque bulle de dialogue est une bulle". Les deux derniers triplets peuvent être utilisés pour déduire le triplet (*<Bulle1>,* `rdf:type`*, :Bulle).*

Dans la suite, pour des raisons de lisibilité, les descriptions des objets sont écrites en notation Turtle [6]. Par exemple, les triplets suivants ((`<CaseK>, rdf:type, Case`) (`<CaseK>, :personnage, <Missbean>`)) s'écrivent en Turtle (`<CaseK> a case; :personnage <Missbean>`).

2.2 SPARQL

SPARQL est un langage de requête pour RDF fondé sur le *graph pattern matching* [23]. Une question et sa traduction en SPARQL sont montrées dans la figure 2. La clause SELECT définit les variables que l'utilisateur souhaite obtenir. La clause WHERE définit les triplets RDF qui spécifient les valeurs des variables. Une variable commence par un point d'interrogation (`?`). Ici, la variable `?x` correspond aux cases. Le point (`.`) correspond à une conjonction et est équivalente au `AND` de SQL. Des contraintes peuvent également être ajoutées à la clause FILTER qui ajoute une ou plusieurs conditions qui doivent être vraies pour que le résultat soit conservé.

2.3 Systèmes d'information logiques (LIS)

Les LIS [12] forment un paradigme de recherche d'information et d'exploration, combinant l'interrogation et la navigation. Ils se rapprochent du paradigme de la recherche par facettes [25]. Leur langage de requête a une expressivité proche de celle de SPARQL et une syntaxe similaire à Turtle [11]. Un prototype *open source*, Sewelis [1], a été implémenté. L'utilisateur navigue de requête en requête. Les liens de navigation sont calculés automatiquement à partir des données, et suggérés aux utilisateurs, de manière à assurer une navigation guidée sûre (pas de requête sans résultats), et complète (toutes les requêtes avec des résultats sont atteignables). UTILIS est implémenté dans Sewelis.

2.4 Guidage pour la mise à jour

Les données du Web sémantique sont rarement mises à jour par les utilisateurs. Pour les aider dans cette mise à jour, des suggestions peuvent lui être faites. Ces suggestions fournissent aux utilisateurs des exemples de format et du type de valeur à ajouter. Toutefois, il est parfois difficile de fournir à l'utilisateur une liste de suggestions appropriées. La connaissance utilisée pour assister les utilisateurs pendant la mise à jour peut être de différents types et change selon l'éditeur.

Les éditeurs pour les données du Web sémantique peuvent être des outils dédiés (par exemple, Protégé [22], OKM [10], Gino [8], QuiKey [14]) ou intégré dans des environnements wiki (par exemple, Semantic Media Wiki [29], KiWI [26], ACEWiki [19]). Leurs interfaces peuvent être fondées sur des formulaires (par exemple, Protégé, OKM), sur des langages naturels (par exemple, Gino, ACEWiki), ou ne permettent seulement la

1. `http://www.irisa.fr/LIS/softwares/sewelis`

FIGURE 3 – Case de bande dessinée extraite de *Ma vie à deux : Pour le meilleur et pour le pire !*, écrite et dessinée par *Missbean, City Editions*.

création d'un seul triplet (ou lien sémantique) à la fois (par exemple, Quikey). Nous discutons dans la suite des informations sur lesquelles les suggestions de ces éditeurs sont fondées.

Les suggestions sont dérivées des axiomes de l'ontologie. Tous les éditeurs mentionnés ci-dessus, à l'exception OKM et Quikey, n'utilisent que les axiomes de l'ontologie, c'est-à-dire le schéma, pour calculer les suggestions, généralement à partir de la classe du nouvel objet. Par exemple, Protégé requiert la définition des domaines et des co-domaines pour faire des suggestions, en effet les formulaires sont dérivées de ces informations. Les suggestions sont uniquement fondées sur la classe des objets.

Les suggestions dépendent de la classe de l'objet. Les formulaires dynamiques d'OKM dépendent seulement, et requièrent seulement la classe du nouvel objet. Quikey suggère simplement (par auto-complétion) toutes les propriétés et valeurs existantes. Les autres informations sur l'objet ne sont pas utilisées pour calculer les suggestions.

Les suggestions ne sont pas finement appropriées au nouvel objet. Les éditeurs listent toutes les instances de certaines classes, comme dans Protégé, ou demandent à l'utilisateur d'entrer quelques caractères afin d'obtenir des suggestions de ressources, comme dans Gino.

3 Exemple

Pour illustrer UTILIS, cette section détaille des étapes de la création de la description d'une nouvelle case dans la base d'annotation. La base contient les cases montrées dans la figure 1 et six autres cases. Imaginons qu'un utilisateur souhaite ajouter la case de la figure 3, nommée CaseK. Elle fait partie de la collection *Ma vie à deux*. Elle a deux personnages, Missbean et Fatbean et une bulle de dialogue dit par Missbean à Fatbean.

La figure 4 présente les étapes 2 à 7 de la création. À chaque étape, la première boîte contient la description courante de l'objet. La partie que l'utilisateur veut étendre est soulignée. Elle est appelée le focus. La seconde boîte contient les suggestions avec, devant chacune d'elles, le nombre d'élargissement pour obtenir la suggestion. En effet, les suggestions sont ordonnées. Par défaut, un sous-ensemble est proposé et l'utilisateur peut l'élargir. Pour des raisons de place, nous ne montrons à chaque étape qu'un nombre limité de suggestions. L'élément en gras correspond au choix de l'utilisateur. L'utilisateur peut choisir une ou plusieurs suggestions.

À l'étape 1, UTILIS n'a aucune information sur le nouvel objet, les suggestions sont toutes les classes et propriétés de la base. L'utilisateur sélectionne une suggestion, ici *a case* pour compléter la description. À l'étape 2, la description du nouvel objet est $\underline{<CaseK>}$ `a` *:case*. Les suggestions s'adaptent à cette description : ce sont les propriétés d'au moins une case. L'utilisateur choisit `:collection []`. La partie supérieure de la figure 5 montre l'interface utilisateur pour l'étape 3, des annotations ont été faites à la main, la description est dans la partie gauche. Dans la partie droite, les ressources suggérées pour le focus sont listées, ici les collections. Au-dessus de cette partie, le nombre de suggestions est indiqué et un bouton *More* permet de les élargir aux distances

Étape 2

<CaseK> a :case
0 :bulle [] 0 :personnage [] 0 **:collection []** 1 :dit par [] 1 :s'adresse à []

Étape 3

<CaseK> a :case; :collection []
0 <Boule et Bill> 0 <Calvin et Hobbes> 0 <**Ma vie à deux**> 0 <Peanuts>

Étape 4

<CaseK> a :case; :collection <M>
0 :bulle [] 0 **:personnage []** 1 :dit par [] 1 :s'adresse à []

Étape 5

<CaseK> a :case; :collection <M>; :personnage []
0 <**Missbean**> 0 <Babybean> 0 <**Fatbean**> 0 <Chat de Missbean> 1 <Calvin>

Étape 6

<CaseK> a :case; :collection <M>; :personnage <MB>, <FB>
0 **:bulle []** 1 :dit par []

Étape 7

<CaseK> a :case; :collection <M>; :personnage <MB>, <FB>; :bulle []
0 **a :bulle de dialogue** 0 :dit par [] 0 :s'adresse à [] 1 a :bulle de pensée []

FIGURE 4 – **Étapes de création de la nouvelle case CaseK** : Elle fait partie de la collection *Ma vie à deux*. Elle a pour personnage Missbean et Fatbean et une bulle de dialogue dit par Missbean à Fatbean.

Étape 3

Étape 7

FIGURE 5 – Captures d'écran d'UTILIS des étapes 3 et 7 de la création de la case de la figure 3.

supérieures contenant des résultats. À l'étape 3, l'utilisateur choisit <*Ma vie à deux*> parmi toutes les collections. Retournons à l'étape 4 de la figure 4, la description est <CaseK> a :case; :collection <Ma vie à deux>. Les suggestions sont les propriétés d'au moins une case de *Ma vie à deux*. La propriété sélectionnée par l'utilisateur est :personnage [] pour spécifier les personnages de la nouvelle case. À l'étape 5, les suggestions sont seulement un sous-ensemble des personnages, tous les personnages des cases de *Ma vie à deux* déjà annotées. L'utilisateur choisit <Missbean> et <Fatbean>. À l'étape 6, aucun objet de la base possède tous les éléments de la description du nouvel objet. En effet, ces deux personnages apparaissent ensemble dans aucune case de la base. Par relâchement de la description, UTILIS continue à faire des suggestions. La partie inférieure de la figure 5 montre l'interface utilisateur pour l'étape 7, les classes et propriétés suggérées sont dans la partie du milieu. À partir de là, l'utilisateur peut continuer la description de la case et quand il décide qu'elle est complète, il l'ajoute à la base avec le bouton *Assert*.

`Cible`	<CaseK> a :case ; :collection <Ma vie à deux> ; :personnage [ ]
Étape 1	?x a :case ; :collection <Ma vie à deux> ; :personnage [ ]
Étape 2	?x a :case . ?x :collection ?y . ?x :personnage ?z . FILTER (?y = <Ma vie à deux>)
Requête cible	SELECT ?z WHERE { ?x a :case . ?x :collection ?y . ?x :personnage ?z . FILTER (?y = <Ma vie à deux>)}

FIGURE 6 – Un cas cible et sa transformation en requête cible.

4 Recherche des cas

Cette section décrit la recherche des cas d'UTILIS. Les cas sources sont les descriptions des objets de la base. Le cas cible est la description d'un objet que l'utilisateur souhaite compléter. La section 4.1 décrit la représentation des cas. Le principe de recherche est montré dans la section 4.2. UTILIS recherche les cas sources ayant le plus de propriétés et valeurs en commun avec le cas cible. La cible complétée est le cas cible et toutes les propriétés et valeurs des cas source qui ne sont pas dans le cas cible. Cela implique que l'étape de réutilisation est une simple copie pour cette version d'UTILIS. Pour rechercher les cas sources, le cas cible est transformé en une requête cible. Cette requête est utilisée comme point de départ pour obtenir des requêtes généralisées en utilisant des règles de relâchement (Section 4.3). Un algorithme efficace pour sélectionner les cas sources à partir de la base de cas et du cas cible a été implémenté avec des techniques de programmation dynamique (Section 4.4). Une évaluation de la recherche a été faite et est brièvement rapportée dans la section 4.5.

4.1 Représentation des cas

Cas source. Un cas source (`Source`) est la description d'un objet de la base. La description d'un objet est représentée par une conjonction de triplets RDFS. Elle contient tous les éléments directement en relation avec l'objet. Les nœuds anonymes de la description sont remplacés par leurs descriptions. En effet, un nœud anonyme est une ressource pour laquelle aucun nom a été donné. Par exemple, la case A de la figure 1 est une case de la collection *Ma vie à deux*. Elle a deux personnages, Missbean et son chat, et une bulle de dialogue dite par Missbean à son chat.

`Source` =<CaseA> a :case ;
:collection <Ma vie à deux> ;
:personnage <Missbean>, <Chat de Missbean> ;
:bulle [a :bulleDeDialogue ; :ditPar <Missbean> ; :s'AdresseA <Chat de Missbean>].

Requête cible. Dans UTILIS, une requête cible est obtenue à partir de la description courante du cas cible (`Cible`) et à partir d'un focus qui est un élément de la description que l'utilisateur souhaite compléter. Le focus est représenté par la partie soulignée. Un exemple de `Cible` est la case K, une case de la collection *Ma vie à deux*, elle a des personnages qui ne sont pas encore spécifiés.

`Cible` = <CaseK> a :case ; :collection <Ma vie à deux> ; :personnage [ ]

La figure 6 montre un exemple de `Cible` et la requête cible obtenue après transformation. `Cible` est transformé en une requête en trois étapes. Premièrement, l'identité du nouvel objet est remplacée par une variable. Par exemple, `<CaseK>` est remplacé par `?x`. Deuxièmement, la description est transformé en un *graph pattern* SPARQL pour avoir seulement un élément modifiable par triplet. Chaque triplet est composé d'un seul individu. Les autres éléments du triplet sont des variables ou la propriété prédéfinie `rdf:type`. Par exemple, (`:collection <Ma vie à deux>`) est transformé en (`?x :collection ?y . FILTER (?y = <Ma vie à deux>)`). Enfin, `Cible` est transformé en une requête en mettant la variable du focus comme variable à rechercher, et le corps de la requête cible correspond au *graph pattern* de l'étape 2.

Règle	Triplet initial	Triplet relâché	Condition
SuperProperty	$?x\ p_1\ ?y$	$?x\ p_2\ ?y$	p_1 subp p_2
SuperClass	$?x$ a c_1	$?x$ a c_2	c_1 subc c_2
Resource	$?x = r_2$	nil	
Property	$?x\ p_1\ ?y$	nil	$\nexists p \neq p_1.(p_1$ subp $p)$
Class	$?x$ a c_1	nil	$\nexists c \neq c_1.(c_1$ subc $c)$

FIGURE 7 – **Règles de relâchement**. Les variables sont précédées d'un point d'interrogation, r_i sont des ressources, p_j des propriétés, et c_k des classes. a, subc, subp sont des abréviations pour rdf:type, rdfs:subClassOf et rdfs:subPropertyOf.

4.2 Principe de recherche

Soit Γ(Cible) une généralisation de Cible. Quand Cible a été modifié par d étapes de généralisations, cela est noté Γ_d(Cible), et Γ_0 est l'identité. Si Γ_0(Cible) n'a aucun résultat (c'est-à-dire, aucun cas source correspond exactement à Γ_0(Cible)), alors elle est généralisée avec des règles de relâchement et un coût est calculé. Une des règles de relâchement permet par exemple de remplacer une ressource par une variable. L'ensemble de ces règles est présenté à la section suivante. Plus Γ(Cible) est généralisée, plus le coût devient élevé. Les sources recherchés sont celles qui correspondent exactement à la généralisation minimale de Cible : ce sont les cas sources qui correspondent à une Γ(Cible) tel que pour chaque Γ' tel que coût(Γ') $<$ coût(Γ), aucun cas source correspond à Γ'(Cible).

Les cas sources sont utilisés comme suggestions pour compléter Cible. Les suggestions sont des ressources, des classes ou des propriétés. Le focus peut être soit sur une ressource soit sur une variable. Si le focus est sur une ressource, les suggestions sont des classes et des propriétés. Si le focus est sur une variable, les suggestions sont des ressources. Une fonctionnalité importante d'UTILIS est que les utilisateurs peuvent ajouter des éléments à Cible à tout moment et à chaque "endroit" de Cible. Par exemple, un utilisateur peut décider d'ajouter une information à une ressource déjà annotée présente dans Cible, attachant cette information à un objet existant. Le focus permet d'identifier l'endroit de la description que l'utilisateur souhaite étendre. Pour éviter que les utilisateurs spécifient explicitement le focus à chaque étape, une stratégie par défaut est proposée. Quand une propriété est ajoutée à Cible, le focus est déplacé sur la variable représentant la valeur de la propriété, car on suppose que l'utilisateur souhaitera probablement sélectionner ou entrer une valeur après cela. Quand la valeur de la propriété est renseignée, le focus est déplacé sur la description de l'objet. Quand une classe est ajoutée à la description de l'objet, le focus reste sur la description de l'objet.

Après raffinement, CibleComplétée sera le nouveau cas cible et sera utilisé pour définir une nouvelle requête cible et ainsi de suite. De cette manière, tous les éléments de Cible sont pris en compte. Le raffinement s'arrête quand l'utilisateur décide que la description est complète. CibleComplétée est ajouté à la base de cas et sera utilisé pour les prochaines suggestions.

4.3 Relâchement des cas

Règles de relâchement. Pour généraliser Cible, nous avons défini des règles de relâchement, inspirées des règles d'approximation de requêtes d'Hurtado et al. [16]. La figure 7 montre l'ensemble des règles de relâchement, applicables aux triplets. La première colonne montre le nom de la règle, la deuxième le triplet avant relâchement, la troisième le triplet relâché et la quatrième les possibles conditions pour l'application de la règle. À l'exception de règles SuperProperty et SuperClass, les règles ne dépendent pas d'une ontologie de domaine. La règle SuperProperty s'applique sur un triplet ayant une variable comme sujet et objet et une ressource p_1 comme prédicat. La condition d'application de cette règle est que p_1 soit une sous-propriété d'une autre propriété p_2 dans l'ontologie de domaine. Après l'application de cette règle, le sujet et l'objet restent identiques mais la propriété p_1 est remplacée par la propriété p_2. Par exemple, le triplet (?x, :s'AdresseA, ?y) peut être relâché en (?x, :converseAvec, ?y) par l'application de cette règle si le triplet (:s'AdresseA, rdfs:subPropertyOf, :converseAvec) appartient à l'ontolo-

gie de domaine. Le triplet relâché `nil` correspond à la suppression du triplet initial.

Distance entre les cas sources et le cas cible. Le coût de Γ est le nombre de règles appliquées pour aller de `Cible` à Γ(`Cible`). Chaque règle de relaxation a un coût de 1 dans la version actuelle du système.

À coût 0, ce sont les résultats de Γ_0, ceux obtenus directement à partir de `Cible` sans généralisation. Par exemple, supposons que Γ_0 est (`SELECT ?x WHERE {?x a :case . ?x :collection ?y . ?x :personnage ?z . ?x :personnage ?a . FILTER (?y = <Ma vie à deux> && ?z = <MissBean> && ?a = <Fatbean>)}`). Cette requête recherche les cases de la collection *Ma vie à deux* avec les personnages Missbean et Fatbean. Une requête généralisée peut être obtenue en relâchant la contrainte `(?z = <Missbean>)` en appliquant la règle `Resource` sur cette contrainte. La requête généralisée a un coût de 1. Pour le même coût, la règle `Resource` peut aussi être utilisée sur une autre contrainte, par exemple sur la contrainte `(?a = <Fatbean>)`. Les règles peuvent être combinées. L'ordre d'application des règles n'a pas d'impact sur les requêtes généralisées.

La distance entre `Source` et `Cible` est le coût minimal de la requête ayant `Source` comme résultat : $d(\texttt{Source}, \texttt{Cible}) = \texttt{coût}(\Gamma^*)$.

4.4 Algorithme de recherche

Un algorithme naïf pour le calcul des cas sources à une distance d de `Cible` consiste à générer toutes les requêtes à distance d, Γ_d(`Cible`), et à calculer l'union de leurs résultats. Γ_d(`Cible`) sont obtenues en appliquant d relaxations sur les n triplets de `Cible`, ce qui fait au moins C_n^d requêtes généralisées. Le nombre de requêtes de 0 à d est égal à $\sum_d \binom{n}{d}$, c'est-à-dire 2^n. En raison de la nature des requêtes dont les *graph patterns* sont des arbres car ils sont obtenus à partie d'expression Turtle, les résultats de chaque Γ_d(`Cible`) peuvent être calculés en $O(n)$ opérations ensemblistes (intersection et traversée de relations). Le coût de l'algorithme naïf, $O(n2^n)$, est trop important, surtout que les requêtes généralisées sont uniquement des intermédiaires dans notre approche.

Dans UTILIS, nous utilisons un algorithme plus efficace fondé sur la technique de la programmation dynamique. Les cas source sont directement calculés à distance d. Γ_0(`Cible`) est représenté par une requête `SELECT ?`x `WHERE {?`x D`}`, où D est une description Turtle sans sujet (liste de couples propriété-valeur). Pour obtenir cette requête, il suffit de définir D comme la réécriture de la description de l'objet en fonction du focus. Par exemple, si `Cible` = <CaseK> a :case ; :collection [], alors D = :collection of [a :case]. La fonction tabulée $E(d, D)$ calcule les cas sources à distance d de D. La fonction E est définie récursivement en s'appelant elle-même avec de plus petites distances et/ou des sous-requêtes. Les cas de base sont quand $d = 0$ et quand D est une description atomique (c'est-à-dire une ressource, une classe, ou `p r`). L'algorithme est défini par un ensemble d'équations, couvrant toutes les combinaisons d'une distance et d'une description. Par exemple, l'équation qui définit E pour les conjonctions de description est :

$$E(d, D_1; D_2) = \bigcup_{i=0}^{d} (E(i, D_1) \cap E(d - i, D_2)).$$

Cette équation dit qu'un résultat d'une description de la forme $D_1; D_2$ est un objet qui est à la fois un résultat à une distance d_1 de D_1 et un résultat à une distance d_2 de D_2, tel que $d_1 + d_2 = d$. La distance d est distribuée entre les deux sous-descriptions ($d_1 = i$ et $d_2 = d - i$) de toutes les manières possibles (pour $i = 0..d$). Les équations pour les descriptions de la forme $p\ \ D_1$ et `is` p `of` D_1 sont similaires, en utilisant la traversée d'une relation au lieu d'une intersection. Les résultats à distance d pour les classes (resp. propriétés) sont fondés sur les super-classes (resp. super-propriétés) à distance d dans la hiérarchie des classes (resp. propriétés) de l'ontologie de domaine.

Supposons que nous souhaitons calculer les résultats de la généralisation de la description D : `a :case; :collection <Ma vie à deux>`. Pour calculer ces résultats, plusieurs calculs intermédiaires sont nécessaires. Le tableau 1 montre $E(d, D)$ ainsi que les calculs intermédiaires qui ont permis son calcul. Ces calculs ont été effectués sur l'extrait de base montré en introduction dans la figure 1. Pour calculer $E(d, D)$, il faut d'abord calculer $E(d,$ `a :case`$)$ et $E(d,$ `:collection <Ma vie à deux>`$)$. Le

	a :case; :collection <Ma vie à deux>	a :case	:collection <Ma vie à deux>	<Ma vie à deux>
0	<CaseA> <CaseD>	<CaseA> <CaseB> <CaseC> <CaseD>	<CaseA> <CaseD>	<Ma vie à deux>
1	<CaseA> <CaseB> <CaseC> <CaseD>	all	<CaseA> <CaseB> <CaseC> <CaseD>	all
2	<CaseA> <CaseB> <CaseC> <CaseD>		all	
3	all			

TABLE 1 – Calcul des généralisations de a :case; :collection <Ma vie à deux>.

calcul de $E(d,$:collection <Ma vie à deux>$)$ entraîne le calcul de $E(d,$ <Ma vie à deux>$)$. Après avoir calculés tous ces résultats intermédiaires, le calcul de $E(d, D)$ est effectué en utilisant l'équation présentée ci-dessus avec D_1 = a :case et D_2 = :collection <Ma vie à deux>. À distance 1,

$$E(1, D) = (E(0, \text{a :case}) \cap E(1, \text{:collection <Ma vie à deux>}) \cup (E(1, \text{a :case}) \cap E(0, \text{:collection <Ma vie à deux>})).$$

Le calcul de $E(d, q)$ est effectué pour toutes les distances jusqu'à ce que $E(d, q)$ soit égal à l'ensemble des objets de la base.

Les calculs intermédiaires de $E(d, D)$ sont stockés dans une table avec une ligne pour chaque distance de 0 à d ($O(d)$ lignes), et une colonne pour D et chacune des sous-descriptions de D ($O(n)$ colonnes). La complexité de calcul d'une cellule de ce tableau est de $O(d)$ opérations ensemblistes. Par conséquent, la complexité de notre algorithme est de $O(nd^2)$ opérations ensemblistes, c'est-à-dire polynomial au lieu de combinatoire pour l'algorithme naïf. De plus, la table est conservée jusqu'à ce que l'utilisateur ajoute CibleComplétée à la base de cas.

4.5 Évaluation de la recherche d'UTILIS

L'expérience rapportée dans [15] évalue l'étape de recherche d'UTILIS lors de la création de nouveaux objets. Cette section donne un bref résumé de celle-ci. 18 étudiants en master d'informatique ont testé et évalué l'utilisabilité d'UTILIS et de Protégé [22] [2]. L'interface et le guidage de Protégé sont représentatifs des éditeurs actuels. En effet, deux tiers des utilisateurs du Web sémantique l'utilisent comme éditeur [9].

Les sujets avaient une connaissance préalable en bases de données relationnelles, mais ne connaissaient ni Sewelis, ni Protégé, ni le Web sémantique. Pour chaque éditeur, ils devaient effectuer les mêmes tâches. Les sujets devaient mettre à jour une base déjà existante, décrivant des cases de bandes dessinées. La base était identique à chaque début de session de test. Les cases existaient déjà dans la base, mais n'avaient aucune classe ni propriété. Les sujets avaient pour instruction d'entrer un maximum d'informations sur chaque case, prenant la description des cases existantes comme modèle. Ils avaient pour consigne de réutiliser autant que possible l'information existante. Ils pouvaient néanmoins créer des nouveaux éléments s'ils en avaient besoin.

Les deux principaux résultats étaient que les suggestions étaient pertinentes et que la consistance de la base était mieux conservée sous UTILIS. En effet, l'ajout des personnages dans une case a été fait dans 90% des

2. http://protege.stanford.edu

cas par la sélection d'une suggestion. Les éléments relatif à la collection, aux personnages et aux interlocuteurs des bulles, ont été, s'ils existaient déjà, sélectionnés dans plus de 80% des cas dans les 3 premiers ensembles de suggestions. Pour les lieux et les autres éléments, les suggestions ont été moins utiles. Ces éléments de la description sont plus subjectifs que les précédents et ont été beaucoup créés par les utilisateurs. Par exemple, pour les autres éléments, plus de la moitié (58%) ont été créés. Les questionnaires remplis après les expériences donnent une image cohérente : 16 sujets ont trouvé les suggestions pertinentes pour les personnages, 15 pour la collection, 12 pour les interlocuteurs, mais seulement 6 pour les lieux et 3 pour les autres éléments.

Le résultat le plus important est que 27 doublons ont été introduits dans la base lors de l'utilisation de Protégé, alors qu'aucun n'a été créé sous UTILIS. Un doublon est un nouvel individu, créé alors qu'il existait déjà un individu pertinent. De plus, sous Protégé les utilisateurs ont introduit 50% de plus de valeurs erronées par rapport à UTILIS. Il convient également de noter que les doublons ont été créés par plus de la moitié des utilisateurs (11 sujets), sous Protégé. De plus, 9 des 18 utilisateurs n'ont introduit ni doublons ni valeurs erronées dans toute la base sous UTILIS contre 2 seulement sous Protégé. La cohérence a été ainsi mieux conservée avec UTILIS.

Néanmoins, les utilisateurs ont eu certaines difficultés à cause de la flexibilité de Sewelis. Trois types d'erreurs ont seulement été faites sous Sewelis : erreur de format, par exemple la création d'un littéral au lieu d'un objet ; propriété erronée, l'utilisation d'une super-propriété ou l'utilisation d'une propriété dans la mauvaise direction, et un focus mal placé, un élément attaché au mauvais endroit dans la description. Ces erreurs ne peuvent pas être faites par les utilisateurs sous Protégé, les formulaires à remplir sont statiques. Toutefois, il faut noter que deux utilisateurs n'ont fait aucune erreur sous Sewelis alors qu'aucun utilisateur ne l'a réussi sous Protégé.

5 Comparaison avec l'état de l'art

Plusieurs approches ont été explorées pour la recherche de cas [20, 7, 28]. Certaines d'entre elles utilisent l'approche par relâchement de requête [21] ou une approche par recherche incrémentale [17]. Dans UTILIS, l'utilisateur souhaite créer une description complète d'un objet. Baccigalupo et Plaza [4] utilise le RÀPC pour recommander une liste de chansons à partir d'une chanson et d'une taille pour la liste. À partir de la chanson de l'utilisateur, le système recherche les listes de chansons comprenant cette chanson et avec de la variété, en effet les auteurs font l'hypothèse que l'utilisateur ne souhaite pas une liste de chansons avec uniquement des chansons du même artiste. Les listes de chansons retrouvées sont des listes de chansons comprenant la chanson de l'utilisateur. À partir ces listes, le système crée un nouvelle liste. Il cherche pour chaque chanson le meilleur successeur ou antécédent en commençant par la chanson demandée par l'utilisateur et s'arrête quand la taille de la liste de lecture est atteinte. Le processus de création est fait sans interaction avec l'utilisateur, contrairement à UTILIS. Les interactions avec l'utilisateur permettent d'obtenir une description plus détaillée du cas cible.

Dans le RÀPC *conversationnel* [2], la création du problème cible est incrémental. En effet, les utilisateurs n'entrent pas une description complète du problème. Le système propose des questions à l'utilisateur pour raffiner le problème. Durant cette conversation, le système ordonne et affiche progressivement les cas retrouvées. Aktas et al. [3] utilisent le RÀPC conversationnel pour aider les utilisateurs à sélectionner des ressources. Comme dans UTILIS, les cas sont représentés par une conjonction de triplets RDF et le cas cible est construit par interactions avec l'utilisateur. Le système recherche les cas ayant le plus d'éléments en commun avec sa requête. Par contre, dans UTILIS, le processus de recherche utilise le relâchement de requête. Aktas et al. ne font que de la suppression de triplets. Le relâchement de requête permet d'ajouter des étapes intermédiaires à la suppression de triplet.

Le relâchement de requête a été étudié dans les systèmes de RÀPC. Taaable [5] [3] est un système de recommandation de recettes de cuisine. L'utilisateur recherche une recette à l'aide d'une requête. Si des recettes correspondent à la requête de l'utilisateur, Taaable retourne ces recettes. Si aucune recette ne satisfait la requête de l'utilisateur, Taaable recherche des recettes similaires et les adapte. Les recettes sont représentées dans un fragment de logique propositionnelle. Une requête est une conjonction de contraintes. Si aucune recette ne correspond à la requête, elle est généralisée. Les résultats des requêtes généralisées sont adaptés. L'adaptation

3. `http://taaable.fr`

d'une recette est la suppression ou la substitution d'un ingrédient par un autre pour un ou plusieurs ingrédients de la recette. UTILIS utilise le relâchement de requête, mais ses cas sont représentés par des triplets RDFS. Les différences dans les langages de représentations entraînent l'utilisation de différents types de relâchement.

6 Apport du RÀPC à UTILIS

La réinterprétation d'UTILIS en RÀPC montre trois pistes d'améliorations futures qui peuvent immédiatement bénéficier à UTILIS. Les étapes de recherche, de réutilisation et de révision sont concernées. Une perspective concerne l'étape de réutilisation avec une adaptation des cas sources au cas cible (section 6.1). Pour ordonner les cas sources à une même distance, un algorithme de recherche en deux étapes est étudié (section 6.2) aussi bien qu'une étape de révision (section 6.3).

6.1 Vers une adaptation des cas pour UTILIS

Les cas sources sont directement utilisés par UTILIS. Cela implique que l'étape de réutilisation est une simple copie dans cette version d'UTILIS. Une perspective concerne l'étape de réutilisation, appelé ici *adaptation*.

Les suggestions proposées à l'utilisateur sont les cas sources qui sont les plus similaires au cas cible. Avec des connaissances d'adaptation, des cas sources moins similaires seront utilisés par UTILIS. Ces connaissances peuvent spécifier des équivalences entre certaines ressources en fonction du contexte. Les informations sur l'annotation des ressources peuvent être utilisées pour définir des équivalences. Par exemple, un personnage peut avoir certains caractéristiques, par exemple principal ou secondaire, humain ou animal. Le personnage Calvin a les mêmes caractéristiques que Charlie Brown. Une situation similaire existe entre Hobbes et Snoopy. Par conséquent, dans la base d'annotation de cas, Calvin est équivalent à Charlie Brown et Hobbes à Snoopy. Supposons que `Cible` est égale à `<CaseL> a :case; collection <Calvin et Hobbes>; :personnage <Calvin>, []`. L'utilisateur souhaite ajouter une autre personnage à `Cible`. Les cases de la collection *Peanuts* peuvent être utilisées en adaptant les valeurs de ces cases à `Cible`. Ainsi, la case B peut être utilisée pour étendre `Cible`. Au lieu de suggérer le personnage Snoopy, son équivalent pour Calvin et Hobbes sera suggéré, c'est-à-dire Hobbes.

6.2 Vers un algorithme de recherche à deux étapes

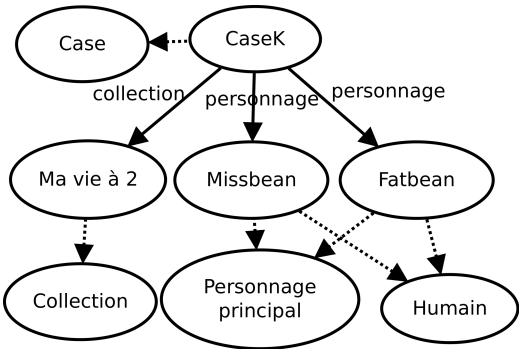
L'étape de recherche utilise des règles de relâchement. L'union de tous les résultats des requêtes produites par d généralisations sont les résultats proposés à distance d. Pour une même distance, les cas sources ne sont pas ordonnés. Une second étape de recherche peut être faite pour déterminer un ordre.

Pour la première étape de recherche, un case cible est représenté par les éléments directement en relation avec le nouvel objet. À la seconde étape, nous pouvons étendre cette description aux éléments en relation avec ceux de la description. La description étendue peut être comparée à la description étendue des cas sources d'une même distance. Un exemple de `Cible` est `<CaseK> a :case; :collection <Ma vie à deux>; :personnage <Missbean>, <Fatbean>`. À distance 1, les cas sources sont les cases A et D. La figure 8 montre la description étendue de la case K et une partie des descriptions étendues des cases A et D. La case D est plus proche de la case K que la case A, car Fatbean partage plus de propriétés avec Babybean, celle d'être un humain, qu'avec le chat de Missbean. Ainsi les suggestions obtenues avec la case D seront mieux classées que celles obtenues avec la case A.

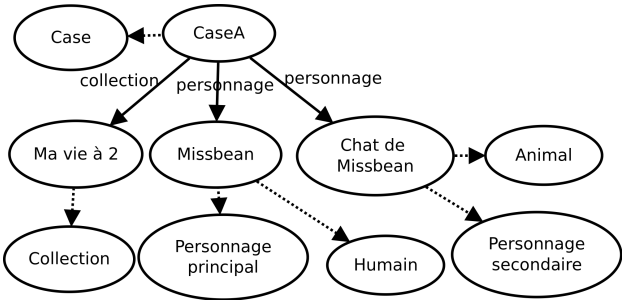
6.3 Utiliser les retours des utilisateurs pour améliorer les résultats

Actuellement, UTILIS n'a pas d'étape de révision. L'étape de révision essaye de déterminer les raisons de l'échec de certaines solutions. Dans UTILIS, cette étape sera appliquée pour comprendre les différences entre les cas sources sélectionnés par les utilisateurs et les autres cas sources.

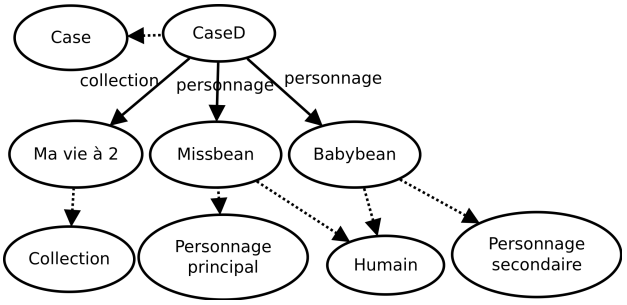
Les retours des utilisateurs correspondent aux précédents choix des utilisateurs. Il est possible que les solutions sélectionnées soient majoritairement les résultats d'un même relâchement. Cette information peut être utilisée pour ordonner les cas sources à une distance donnée. Stahl [27] propose un système de RÀPC



(a) Description de la case K



(b) Partie de la description de la case A



(c) Partie de la description de la case D

FIGURE 8 – Description étendue de différentes cases. La propriété `rdf:type` est représentée par une flèche en pointillé.

avec une rétro-action faite par un *enseignant de similarité*. Le rôle de cet enseignant est d'évaluer les résultats retournés. Cet enseignant peut être l'utilisateur. L'ordre des résultats retournés est modifié par les précédents choix des utilisateurs. Le résultat le plus sélectionné est placé en premier et ainsi de suite.

Pour UTILIS, les suggestions peuvent être ordonnées en fonction des choix précédents des utilisateurs. Par exemple, supposons que pour ajouter les personnages, les solutions sélectionnées soient plus souvent obtenues par relâchement du lieu que par relâchement de la collection, alors les solutions sélectionnées obtenues par relâchement du lieu seront mises en avant. Les retours des utilisateurs peuvent être utilisés pour la seconde étape de l'algorithme de recherche, avec un calcul de poids. Quand une solution obtenue par un certain relâchement est sélectionnée, le poids de ce relâchement est modifié. Avec ces poids, un ordre entre les cas sources de différents relâchements peut être établi.

Un exemple de `Cible` est égal à "`a case; collection : <Ma vie à deux>; lieu : <Rue>`". À distance 1, les cas sources peuvent être le résultat des différentes requêtes, avec le relâchement de la collection ou du lieu. Actuellement, il n'existe pas de différences entre les résultats de ces deux requêtes relâchées. Toutefois, la collection et le lieu n'ont pas la même importance dans `Cible`. Supposons que sur 100 sélections, 80 sont créés par le relâchement du lieu et 20 par le relâchement de la collection. Dans ce cas, les suggestions obtenues par le relâchement du lieu seront favorisées comparé à celles obtenues par le relâchement de la collection.

7 Conclusion

Ce papier propose une ré-interprétation en RÀPC d'UTILIS. UTILIS est une méthode pour guider les utilisateurs lors de la création d'objets RDFS. Le guidage s'appuie sur les objets existants, et sur la description courante du nouvel objet. Les suggestions sont bien appropriées à chaque objet en cours de création. Comparé à d'autres éditeurs RDFS, les suggestions d'UTILIS sont fondées sur les objets existants, plutôt qu'uniquement sur le schéma RDFS. Les cases retrouvés sont les résultats de requêtes qui sont généralisées à partir du cas cible. Ces requêtes sont obtenues par application de règles de relâchement. L'algorithme efficace pour le calcul

des suggestions pourrait bénéficier aux systèmes de RÀPC.

Cette ré-interprétation apporte des perspectives pour améliorer les suggestions d'UTILIS. Actuellement, seulement les étapes de recherche et d'apprentissage sont faites. L'étape d'apprentissage consiste simplement à l'ajout des triplets édités à la base de cas. Une seconde étape dans l'algorithme de recherche ordonnera les cas sources à une même distance. Une étape d'adaptation utilisera les cas sources moins similaires au cas cible. Une étape de révision utilisera les retours des utilisateurs pour ordonner les cas sources.

Remerciements

Nous tenons à remercier Marie Levesque (alias Missbean) pour l'utilisation de la case extraite de *Ma vie à deux : Pour le meilleur et pour le pire !* et son éditeur *City Éditions*.

Références

[1] Agnar Aamodt et Enric Plaza. Case-based reasoning ; foundational issues, methodological variations, and system approaches. *AI COMMUNICATIONS*, 7(1) :39–59, 1994.

[2] David W. Aha, Leonard A. Breslow, et Héctor Muñoz Avila. Conversational case-based reasoning. *Applied Intelligence*, 14 :9–32, 2001.

[3] Mehmet S. Aktas, Marlon Pierce, Geoffrey C. Fox, et David Leake. A web based conversational case-based recommender system for ontology aided metadata discovery. In *IEEE/ACM international workshop on grid computing*. IEEE Computer Society, 2004.

[4] Claudio Baccigalupo et Enric Plaza. Case-based sequential ordering of songs for playlist recommendation. In *European Conference on Case-Based Reasoning (ECCBR)*, volume 4106 of *LNCS*, pages 286–300. Springer, 2006.

[5] Fadi Badra, Rokia Bendaoud, Rim Bentebibel, Pierre-Antoine Champin, Julien Cojan, Amélie Cordier, Sylvie Després, Stéphanie Jean-Daubias, Jean Lieber, Thomas Meilender, Alain Mille, Emmanuel Nauer, Amedeo Napoli, et Yannick Toussaint. TAAABLE : Text Mining, Ontology Engineering, and Hierarchical Classification for Textual Case-Based Cooking. In *European Conference on Case-Based Reasoning (ECCBR), Workshop Proceedings*, pages 219–228, 2008.

[6] David Beckett, Tim Berners-Lee, et Eric Prud'hommeaux. Turtle - Terse RDF Triple Language. W3C Recommendation, January 2010.

[7] Ralph Bergmann et Yolanda Gil. Retrieval of semantic workflows with knowledge intensive similarity measures. In *International Conference on Case-Based Reasoning (ICCBR)*, volume 6880 of *LNCS*, pages 17–31. Springer, 2011.

[8] Abraham Bernstein et Esther Kaufmann. Gino - a guided input natural language ontology editor. In *International Semantic Web Conference (ISWC)*. Springer, 2006.

[9] Jorge Cardoso. The semantic web vision : Where are we ? *IEEE Intelligent Systems*, 2007.

[10] Stephen Davies, Chris Donaher, et Jesse Hatfield. Making the Semantic Web usable : interface principles to empower the layperson. *Journal of Digital Information*, 12(1), 2010.

[11] Sébastien Ferré et Alice Hermann. Semantic search : Reconciling expressive querying and exploratory search. In *International Semantic Web Conference (ISWC)*. Springer, 2011.

[12] Sébastien Ferré et Olivier Ridoux. An introduction to logical information systems. *Information Processing & Management*, 40(3) :383–419, 2004.

[13] Ramanathan V. Guha et Dan Brickley. RDF Vocabulary Description Language 1.0 : RDF Schema. World Wide Web Consortium (W3C) Recommendation, 2004.

[14] Heiko Haller. QuiKey – an efficient semantic command line. In *Knowledge Engineering and Management by the Masses (EKAW)*, pages 473–482. Springer, 2010.

[15] Alice Hermann, Sébastien Ferré, et Mireille Ducassé. An interactive guidance process supporting consistent updates of RDFS graphs. In *Int. Conf. Knowledge Engineering and Knowledge Management (EKAW)*, LNAI 7603, pages 185–199. Springer, 2012.

[16] Carlos A. Hurtado, Alexandra Poulovassilis, et Peter T. Wood. Query relaxation in RDF. *J. Data Semantics*, 10 :31–61, 2008.

[17] Igor Jurisica, Janice Glasgow, et John Mylopoulos. Incremental iterative retrieval and browsing for efficient conversational cbr systems. *Applied Intelligence*, 2000.

[18] Graham Klyne et Jeremy J. Carroll. Resource Description Framework (RDF) : Concepts and Abstract Syntax. World Wide Web Consortium (W3C) Recommendation, 2004.

[19] Tobias Kuhn. How controlled english can improve semantic wikis. In *Semantic Wiki Workshop (SemWiki)*. CEUR-WS.org, 2009.

[20] T. Warren Liao, Zhiming Zhang, et Claude Mount. Similarity measures for retrieval in case-based reasoning systems. *Applied Artificial Intelligence*, 12(4) :267–288, 1998.

[21] Nader Mirzadeh, Francesco Ricci, et Mukesh Bansal. Supporting user query relaxation in a recommender system. In *International Conference on Electronic Commerce and Web Technologies (EC-Web)*, volume 3182 of *LNCS*, pages 31–40. Springer, 2004.

[22] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubezy, Ray W. Fergerson, et Mark A. Musen. Creating semantic web contents with Protégé-2000. *Intelligent Systems, IEEE*, 16(2) :60–71, 2001.

[23] Eric Prud'hommeaux et Andy Seaborne. SPARQL query language for RDF. W3C Recommendation, 2008.

[24] Christopher K. Riesbeck et Roger C. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, 1989.

[25] Giovanni Sacco et Yannis Tzitzikas, editors. *Dynamic Taxonomies and Faceted Search : Theory, Practice, and Experience*, volume 25 of *The Information Retrieval Series*. Springer, Berlin, 2009.

[26] Sebastian Schaffert, Julia Eder, Szaby Grünwald, Thomas Kurz, Mihai Radulescu, Rolf Sint, et Stephanie Stroka. Kiwi - a platform for semantic social software. In *Semantic Wiki Workshop (SemWiki)*. CEUR-WS.org, 2009.

[27] Armin Stahl. Learning feature weights from case order feedback. In *International Conference on Case-Based Reasoning (ICCBR)*, volume 2080 of *LNCS*. Springer, 2001.

[28] Armin Stahl et Sascha Schmitt. Optimizing retrieval in CBR by introducing solution similarity. In *International Conference on Artificial Intelligence (ICAI)*. CSREA Press, 2002.

[29] Max Völkel, Markus Krötzsch, Denny Vrandecic, Heiko Haller, et Rudi Studer. Semantic Wikipedia. In *International conference on World Wide Web (WWW)*. ACM Press, 2006.