Réponses coopératives dans l'interrogation de documents RDF

Adrian Tanasescu, Mohand-Saïd Hacid

LIRIS - Université Claude Bernard Lyon 1
43 Bld. du 11 Novembre 1918
69422 VILLEURBANNE Cedex – FRANCE
{atanases, mshacid}@bat710.univ-lyon1.fr

Résumé. Le développement du Web Sémantique a conduit à l'élaboration de standards pour la représentation des connaissances sur le Web. RDF, comme un de ces standards, est devenu une recommandation du W3C. Même s'il a été conçu pour être interprétable par l'homme et la machine (encodage XML, triplets, graphes étiquetés), RDF n'a pas été fourni avec des services d'interrogation et de raisonnement. La plupart des travaux concernant l'interrogation de documents RDF se sont concentrés sur l'usage de techniques issues de la programmation logique et sur des extensions de SQL. Nous portons un nouveau regard sur les techniques d'interrogation et de raisonnement sur les documents RDF et nous montrons que la sémantique des termes OSF (Order Sorted Features) est compatible avec la représentation isomorphique (triplets) des propositions RDF. Cette transformation permet l'ordonnancement des ressources en ontologies et, à travers ceci, des meilleurs mécanismes de réponses (par approximation et recouvrement) aux interrogations de documents RDF.

Mots clés : RDF, approximation, interrogation.

1 Introduction

Les requêtes posées par des utilisateurs sur une base de documents RDF peuvent ne pas retourner de réponses satisfaisantes (variables de la requête pas toutes instanciées). Dans de nombreuses situations, plusieurs documents RDF peuvent contribuer à délivrer une réponse exacte ou approximative à une requête.

Considérons l'exemple suivant : supposons que notre base de documents RDF est constituée de deux documents définis par les graphes étiquetés de la figure 1. Considérons aussi la requête suivante (exprimée sous forme de ψ-terme) posée sur cette base :

```
Q : ISMIS05 (type -> X ;
             editor -> Y (affiliation -> Z))
```

Un mécanisme "classique" répond à cette requête en cherchant un homomorphisme dans chacun des deux documents RDF. Comme on peut le constater facilement, aucun des deux documents ne peut fournir une réponse complète à la requête posée. Néanmoins, la combinaison de certaines parties des deux documents (signalées sur la figure 1 en pointillés) peuvent, ensemble, délivrer une réponse complète. Dans ce pa-

pier nous étudions une approche capable de fournir ce genre de réponse combinant différentes parties de documents RDF.

Nous proposons, dans la suite du papier, un cadre logique pour fournir des réponses coopératives à une requête posée sur des documents RDF. Dans un premier temps, chaque document RDF est transformé en un ψ-terme. Les ψ-termes peuvent être vus comme une conjonction de contraintes. Ainsi, nous proposons un algorithme basé sur la propagation de contraintes pour évaluer une requête sur une base de documents RDF. L'algorithme permet d'assembler des fragments de réponses pouvant provenir de documents différents.

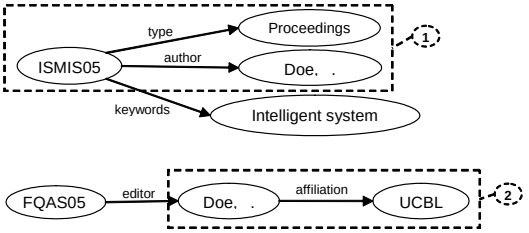

Fig. 1 – Exemple de base de documents RDF

2 Interrogation de documents RDF : État des lieux

Les triplets du modèle RDF peuvent être vus comme l'équivalent des faits dans un langage logique. De ce fait, les approches basées sur la logique, concernant la gestion et l'interrogation de l'information, peuvent s'appliquer facilement au modèle de RDF. De ce point de vue, le sujet de l'interrogation de documents RDF à travers un langage logique a déjà été traité. Ainsi, l'utilisation de F-logique (Kifer et al., 1995) a été proposée dans (Decker et al., 1998). Les propositions RDF sont traduites en termes dans la F-logique et des interrogations peuvent être formulées sur ces termes. Cette première approche ne prend pas en compte tous les constructeurs de RDF (propositions réifiées, ressources anonymes). Ces limitations ont été étudiées par Yang et Keifer (Yang et Kifer, 2003) qui proposent une extension de la F-logique qui permet la représentation des ressources anonymes et des propositions réifiées.

Dans (Sintek et Decker, 2001) TRIPLE est présenté comme un langage de requêtes, d'inférence et de transformation pour RDF. Son but principal est l'interrogation des ressources Web d'une manière déclarative. TRIPLE est un langage de règles basé sur des extensions syntaxiques de la logique de Horn pour la représentation des ressources et de propositions RDF. D'autre part, des modules d'extensions sémantiques de RDF comme RDF Schema , OIL et DAML+OIL peuvent être implémentés soit directement dans TRIPLE soit via une interaction avec des raisonneurs externes. En traduisant les propositions RDF en règles logiques, TRIPLE permet l'interrogation de données RDF. De plus, les règles issues de TRIPLE peuvent être représentées dans RDF permettant ainsi leur redistribution sur le Web.

Karvounarakis et al. (Karvounarakis et al., 2003) proposent un langage d'interrogation pour RDF nommé RQL. Il s'agit d'un langage fonctionnel typé (à la OQL) et

se base sur un modèle formel pour les graphes étiquetés permettant l'interprétation des descriptions de ressources surimposées à travers un ou plusieurs schémas RDF. RQL adapte la fonctionnalité des langages d'interrogation semistructurés/XML pour les particularités de RDF mais, surtout, permet l'interrogation uniforme à la fois des descriptions de ressources et des schémas.

Broekstra et al. (Broekstra et al., 2002) ont développé une architecture pour le stockage et l'interrogation de documents RDF et RDFS, nommée SESAME. Cette dernière permet le stockage persistant de données et schémas RDF et fournit des méthodes d'accès à ces informations à travers des modules d'exportation et d'interrogation.

3 Préliminaires

Avant de développer les détails techniques de notre approche il est important de rappeler brièvement les bases de RDF et des traits typés ordonnés (termes OSF).

3.1 Le modèle de données RDF

RDF (RDF, 1999) est un langage assertionnel conçu pour exprimer des propositions à l'aide de vocabulaires formels précis, plus particulièrement RDFS, en facilitant ainsi l'accès et l'utilisation sur le Web. Il est censé fournir une base pour des langages assertionnels plus complexes ayant un but similaire.

Le modèle de données RDF peut être représenté comme triplets, comme graphes étiquetés, ou comme documents XML (RDF, 1999). Ces représentations sont considérées comme équivalentes. Un document RDF est un ensemble finit de propositions de la forme {prédicat, sujet, objet }, où prédicat est une propriété, sujet est une ressource, et objet est une ressource ou un littéral. Une ressource décrit une entité conceptuelle ou réelle (ex : Ora Lassila). Par exemple, l'URI, http ://www.w3.org/TR/REC-rdf-syntax représente le concept abstrait de RDF même. Les ressources sont représentées par des URI (Berners-Lee et al., 1998), mais elle peuvent aussi être anonymes.

Un manière de représenter la sémantique consiste à donner une traduction de RDF dans un langage logique formel, ayant une théorie du modèle déjà attachée. Cette approche de "sémantique axiomatique" a déjà été utilisée avec de nombreuses versions alternatives du langage logique cible (Marchiori et Saarela, 1998, McGuinness et al., 2002). L'axiomatique sémantique présente des avantages dans le calcul machine et peut être plus lisible, mais dans le cas où elle échoue de se conformer à la sémantique du modèle théorique, ce dernier doit être considéré comme normatif.

3.2 Les traits typés ordonnés (termes OSF)

Dans (Ait-Kaci et Nasr, 1986), les ψ-termes ont été proposés comme structures d'enregistrements flexibles pour la programmation logique. Néanmoins, les ψ-termes ont une utilisation plus étendue (voir (Ait-Kaci et Podelski, 1991), (Ait-Kaci et Sasaki, 2001), (Holsheimer et al., 1994), (Ait-Kaci et Podelski, 1992)). Comme les ψ-termes représentent une généralisation des termes de premier ordre, et comme ces derniers sont des structures de données dominantes utilisées par les langages de programmation

symboliques, les ψ-termes, plus flexibles, offrent une alternative intéressante.
La manière la plus simple pour décrire un ψ-terme est à travers un exemple. Voila comment un ψ-terme peut être utilisé pour décrire un objet représentant une personne générique :

```
P :person(name⇒id(first⇒string,
                  last⇒S :string),
          age⇒30,
          spouse⇒person(name⇒id(last⇒S),
                        spouse⇒P)).
```

En simple mots : une personne, âgée de 30 ans, a un nom dont les parties first (prénom) et last (nom) sont des littéraux et son époux(se) est une personne avec le même nom, l'époux(se) de cette dernière personne étant la première personne en question.

Cette expression ressemble à la structure d'un enregistrement. Comme un enregistrement, elle a des noms de champs (les symboles à gauche de ⇒), appelés des traits ou attributs. Contrairement aux enregistrement conventionnels, les ψ-termes peuvent engendrer plus d'informations. Les champs sont attachés aux types (ex : person, id, string, 30, etc.). Ces types peuvent représenter indifféremment des valeurs individuelles (ex : 30) ou des ensembles de valeurs (ex : person, string). Les types sont partiellement ordonnés pour refléter l'inclusion des ensembles (ex : employee $<$ person veut dire que tout les employés sont des personnes). Enfin, le partage de la structure peut être exprimé avec des variables (ex : P et S) et ce partage peut être circulaire (ex : P).

Les ψ-termes représente la base d'une logique de structures d'enregistrements appelée logique OSF. Cette dernière est, en fait, la forme la plus primitive d'un langage de contraintes appelé LIFE (Logic, Inheritance, Functions and Equations). L'idée fondamentale de la logique OSF est la possibilité d'établir une relation de subsomption entre les termes OSF, fait qui relève l'importance de cette logique dans notre approche.

Les ψ-termes peuvent aussi être vu comme une conjonction de contraintes. Ainsi le ψ-terme présenté précédemment peut être directement réécrit en l'ensemble de contraintes suivant :

```
{(P, id) :name ; (id, string) :first ; (id, S) :last ; (P, 30) :age ;
(P, person) :spouse ; (person, id) :name ; (id, S) :last ; (person, P) :spouse }
with P < person, S < string.
```

Néanmoins, la traduction directe des ψ-termes en contraintes peut amener à une perte d'informations ou même à de fausses informations. Ceci est du à l'utilisation des types qui dénotent des ensembles de valeurs. Par exemple, dans l'ensemble de contraintes créé ci-dessus, on peut faussement conclure que toutes les personnes sont des époux(ses) de P. Pour éviter ce gendre d'information supplémentaire fausse, tout les types représentant des ensemble de valeurs seront individualisé en introduisant des variables. Notre ψ-terme devien ainsi :

P :person(name⇒Id1 :id(first⇒string,
last⇒S :string),
age⇒30,
spouse⇒P2 :person(name⇒Id2 :id(last⇒S),
spouse⇒P)).

et il est traduit en l'ensemble de contraintes suivant :

```
{(P, Id1) :name ; (Id1, string) :first ; (Id1, S) :last ; (P, 30) :age ;
(P, P2) :spouse ; (P2, Id2) :name ; (Id2, S) :last ; (P2, P) :spouse }
with P < person, S < string, Id1<id, Id2<id.
```

La possibilité de traduire les ψ-termes en contraintes nous permettra de construire un algorithme capable de fournir des réponses coopératives pour les requêtes formulées sur les documents RDF (voir section 5).

4 Expression des propositions RDF avec des ψ-termes

Dans cette section nous montrons, à travers quelques exemples, comment les propositions RDF peuvent être traduites sous forme de ψ-termes. La structure des ψ-termes (Ait-Kaci et Podelski, 1991) fournit une représentation adéquate pour les objets complexes. Ils généralisent les termes conventionnels utilisés en programmation logique. Ce sont des structures typées et attribuées, ordonnées grace à un ordre de sous-types.

Nous nous intéressons aussi aux cas des propositions RDF utilisant des ressources anonymes, des listes et les propositions réifiées. Les exemples que nous traitons sont inspirés de (RDF, 1999).

4.1 Propositions RDF simples avec des ressources nommées

La proposition RDF suivante :
Ora Lassila is the creator of the resource http ://www.w3c.org/Lassila/
qui est représentée comme le triplet
(http ://www.w3c.org/Home/Lassila, creator, Ora Lassila)
peut être traduite dans le ψ-terme :

```
'http ://www.w3c.org/Home/Lassila/'(creator =>'Ora Lassila').
```

4.2 Propositions RDF contenant des ressources anonymes

Dans le cas de RDF les ressources anonymes sont considérées comme uniques. Donc, nous supposons qu'en aucun cas deux ressources anonymes, se trouvant dans des ψ-termes différents, ayant les mêmes attributs (et valeurs) ne pourront être considérées comme étant la même ressource. Néanmoins, une même ressource anonyme doit pouvoir apparaître plusieurs fois à l'intérieur d'un même ψ-terme.

Dans la traduction des propositions RDF en ψ-termes nous noterons les ressources anonymes par $\#\_$. Pour différencier les ressources anonymes distinctes à l'intérieur d'un

ψ-terme nous utiliserons des chiffres à la suite du symbole #_ (ex : #_1, #_2, etc). Voici un exemple :

The individual whose name is Ora Lassila, email lassila@w3c.org is the creator of http ://.../Lassila/ sera traduite vers le ψ-terme

```
'http ://...Lassila/'(creator => #_ (name =>'Ora Lassila' ;
                                     email =>'lassila@w3c.org').
```

4.3 Les containers et collections dans les propositions RDF

Les containers et les collections sont utilisés dans RDF pour indiquer un ensemble d'objets (dans le sens de la logique des triplets [prédicat, sujet, objet]). Les containers sont des ensembles ouverts de ressources, tandis que les collections sont des ensembles fermés (elles contiennent tous les membres d'un ensemble). De plus, les containers se déclinent en trois catégories : (1) Bag - ensemble simple ; (2) Seq - ensemble dans lequel l'ordre d'apparition a une importance ; (3) Alt - ensemble de valeurs alternatives (équivalentes). Bien que RDF propose cette déclinaison des containers, aucune sémantique ne peut être attachée à ces types de containers dans RDF (RDF, 1999).

Nous présentons ces ensembles en créant une extension de la logique des OSF et nous montrons comment ces extensions respectent les principales règles concernant la subsomption et la généralisation dans les OSF.

Pour ces quatre constructeurs nous avons créé des types globaux que l'on a introduit comme une extension à la logique des ψ-termes. Ceci nous permet d'exprimer les containers et les collections en tant que types. Ainsi, nous prédéfinissons les quatre types suivants : `Bag, Alt, Seq` et `List`. Ces termes deviennent alors réservés et ne peuvent plus être utilisés en tant que noms de types dans la construction des ψ-termes.

4.3.1 Expression des containers et collections en tant que types

```
X : Book ( title => 'Modern Information Retrieval' ;
           Authors=> Bag1 ;
           editor =>'ACM Press')
 Bag1 := {'R. Baeza-Yates' ; 'B. Ribeiro-Neto'}
 Bag1 < Bag
```

Dans cet exemple, l'ensemble des auteurs d'un livre est regroupé dans un container `Bag1`, dont le type est défini par son appartenance au type prédéfini `Bag`.

L'expression syntaxique des trois autres ensembles (`Alt, Seq, List`) se fait de manière identique. L'interprétation sémantique de ces quatre types sera faite par l'application qui sera chargée des calculs dans les ψ-termes. Pour cela nous allons définir les règles de subsomption qui s'appliqueront aux quatre types.

4.3.2 Subsomption entre containers et collections

1. Subsomption entre les containers de type `Bag`

Soitent `Bag1 < Bag et Bag2 < Bag`.

`Bag1 < Bag2 ssi` : $\forall t_1 \in$ `Bag1`, $\exists t_2 \in$ `Bag2` $\mid t_1 \leq t_2$

Ex : {Mary, John} < {Mary, John, Tom}

Sachant que les collections sont aussi des ensembles non ordonnés, nous appliqueront la même règle de subsomption pour le type List.

2. Subsomption entre les containers de type Alt
Soit Alt1 < Alt et Alt2 < Alt.
Alt1 < Alt2 ssi : $\exists t_1 \in$ Alt1, $\exists t_2 \in$ Alt2 | $t_1 \leq t_2$
Ex : {Mary, John} < {John, Tom}

3. Subsomption entre les containers de type Seq
Soit Seq1 < Seq et Seq1 < Seq.
Seq1 < Seq2 ssi :
$\forall t_i \in$ Seq1,$\exists t_j^{'} \in$ Seq2| $(t_i \leq t_j^{'})$
et (si $(\exists t_k \in Seq1, \exists t_m \in Seq2, k < i, t_k < t_m)$ alors $m < j$)
Ex : {Mary, Fred, Tom} < {Mary, John, Fred, Tom}
{Mary, Tom, Fred} $\not<$ {Mary, John, Fred, Tom}

5 Interrogation de documents RDF et réponses coopératives

RDF a été conçu pour être un formalisme de représentation de données pour les applications sur le Web. Des travaux et des outils explorant la relation entre RDF et le monde de la représentation de connaissance et de la programmation logique existent (Peer, 2003), (Fikes et McGuinness, 2001), (Marchiori et Saarela, 1998), (Alferes et al., 2003). Notre objectif est de continuer cet effort, d'investiguer des services de raisonnement qui peuvent être employés sur les propositions RDF.

Comme nous l'avons montré, les documents RDF peuvent être représentés sous forme de ψ-termes. Notre intention est de développer une méthodologie permettant de fournir des réponses coopératives dans l'interrogation de documents RDF. Jusqu'à présent, l'évaluation d'une requête posée sur une base de documents RDF considère les documents individuellement et cherche un homomorphisme de la requête dans chaque document de la base. Nous proposons un mécanisme de recherche permettant de trouver une réponse exacte ou approximative en combinant des propositions RDF qui se trouvent dans différents documents.

5.1 Algorithme pour réponses coopératives

Afin de procéder à ce type de recherche, nous considérons que tous les ψ-termes ainsi que la requête sont décomposés sous forme de contraintes. La recherche consiste en deux étapes : (1) la propagation, étape dans laquelle on cherche à satisfaire les contraintes de la requête et (2) l'évaluation, étape dans laquelle on vérifie que les réponses trouvées existent réellement dans les documents RDF.

Soit F l'ensemble de faits (résultant de la décomposition de l'ensemble de ψ-termes) exprimés sous la forme $(s_i, s_j) : label_n$.
Soit G l'ensemble de contraintes de la requête exprimées dans une des formes : (s_i, X) :

$label_n$, $(X, s_j) : label_n$ ou $(X, Y) : label_n$ où X et Y sont les variables dans les contraintes. On définit un ensemble de règles permettant la mise en place d'un algorithme de recherche.

1. Étape de propagation

R1 :

$$\left.\begin{array}{l} si\ (s_i, X) : label_n \in G \\ et\ \exists\ (s_i, s_j) : label_n \in F \end{array}\right\} \Rightarrow alors\ G = G\ \mid\ X = s_j$$

R2 :

$$\left.\begin{array}{l} si\ (s_i, X) : label_n \in G \\ et\ \exists\ (s_k, s_j) : label_n \in F \\ et\ LUB(s_k, s_i) = s_i \end{array}\right\} \Rightarrow alors\ G = G\ \mid\ X = s_j$$

R3 :

$$\left.\begin{array}{l} si\ (s_i, X) : label_n \in G \\ et\ \exists\ (s_i, s_j) : label_m \in F \\ et\ LUB(label_m, label_n) = label_n \end{array}\right\} \Rightarrow alors\ G = G\ \mid\ X = s_j$$

Les contraintes à une seule variable sont résolues en priorité. Les règles sont utilisées dans l'ordre spécifié ci-dessus. Si une règle satisfait une contrainte, les règles suivantes sont ignorés et la prochaine contrainte est traitée. R2 et R3 utilisent un ordre prédéfini sur les types et les traits pour permettre de fournir des réponses approximatives aux requêtes posées.

2. Étape d'évaluation

$$\left.\begin{array}{l} si\ (s_i, s_j) : label_n \in G \\ et\ (s_i, s_j) : label_n \in F \\ avec\ s_k \leq s_i,\ s_l \leq s_j,\ label_m \leq label_n \end{array}\right\} \Rightarrow\ G = G \setminus \{(s_i, s_j) : label_n\}$$

Si à la fin de l'étape d'évaluation $G = \emptyset$, nous avons une réponse exacte à la requête. Dans le cas contraire il s'agira d'une réponse partielle et l'ensemble G contiendra les parties de la requête pour lesquelles aucune réponse n'a été trouvée.

5.2 Exemple

Nous considérons la base de ψ-termes suivante, obtenue par la traduction de documents RDF :

```
ψ1 : ISMIS (type -> proceedings ;
            author -> John ;
            keywords -> 'Intelligent systems')
ψ2 : EGC (type -> proceedings ;
          author -> Paul M.)
ψ3 : FQAS (editor -> John (affiliation -> UCBL))
ψ4 : PKDD (type -> proceedings ;
           author -> Matt S.(affiliation -> Lyon2))
```

Tous les ψ-termes sont ensuite traduits en un ensemble de contraintes (faits) nommé F. Nous formulons la requête suivante :

```
Q : ISMIS (type -> X ;
           editor -> Y (affiliation -> Z))
```

qui est traduite en l'ensemble de contraintes (objectifs) G :

```
G = {(ISMIS, X) : type, (ISMIS, Y) : editor, (Y, Z) : affiliation}
```

Nous disposons aussi d'un ordre prédéfini sur les attributs (traits) : `author` $<$ `editor`. Voici une simulation de l'exécution de l'algorithme, commençant par la phase de propagation.

- 1^{ere} contrainte de la requête : `(ISMIS, X) : type`
  → R1 trouve un réponse : `(ISMIS,proceedings) :type` $\Rightarrow X =$ *proceedings*
  $\Rightarrow G =$ {*(ISMIS, proceedings) : type, (ISMIS, Y) : editor, (Y, Z) : affiliation*}

- 2^{eme} contrainte de la requête : `(ISMIS, Y) : editor`
  → R1 ne trouve pas de réponse
  → R2 ne trouve pas de réponse
  → R3 trouve un réponse : `(ISMIS,John T.) :author` $\Rightarrow Y =$ *John*
  $\Rightarrow G =$ {*(ISMIS, proceedings) : type, (ISMIS, John) : editor, (John, Z) : affiliation*}

- 3^{eme} contrainte de la requête : `(John, Z) : affiliation`
  → R1 trouve un réponse : `(John, UCBL) : affiliation` $\Rightarrow Z =$ *UCBL*
  $\Rightarrow G =$ {*(ISMIS, proceedings) : type, (ISMIS, John) : editor, (John, UCBL) : affiliation*}

Dans cet exemple la phase de validation trouve toutes les contraintes de G dans la base de faits F. Cela nous indique que la réponse à notre requête est complète. Cette réponse est construite dans la phase de validation en considérant les contraintes qui sont successivement enlevées de G.

6 Conclusion

Dans ce papier nous nous intéressons au problème de l'interrogation complexe de documents RDF. La nouveauté de notre approche consiste dans la possibilité de fournir des réponses coopératives aux requêtes posées sur un ensemble de documents RDF. Pour ce faire, nous avons traduit ces documents en traits typés ordonnés (OSF) et nous nous sommes servis des propriétés de ces derniers pour permettre l'approximation des réponses aux requêtes. Ce papier ne présente que très brièvement notre travail en cours qui vise a fournir une sémantique complète pour RDF à l'aide d'un langage logique. Cela permettrait des interrogations sémantiques sur les différents types de données complexes pouvant être décrites avec RDF. Dans nos travaux futurs, nous traiterons de plus près les propriétés de l'algorithme proposé et nous nous intéresserons à son optimisation.

Références

Hassan Ait-Kaci et Roger Nasr (1986), LOGIN : A logic programming language with built-in inheritance. The Journal of Logic Programming, 3(3) : pp 185-215, 1986.

H. Ait-Kaci et A. Podelski (1991), Towards the meaning of LIFE. In J. Maluszynski et M. Wirsing, editors, Proceedings on 3rd International Symposium on Programming Language Implementation and Logic Programming, Berlin, pp 255-274, 1991. [Ait-Kaci et Podelski, 1992]

Hassan Ait-Kaci et Andreas Podelski (1992), Logic Programming with Functions over Order-Sorted Feature Terms. In Proceedings of the Third Workshop on Extensions of Logic Programming, ELP'92, Bologna, Italy, February 26-28, 1992, pp 100-119.

Hassan Ait-Kaci et Yutaka Sasaki (2001), An Axiomatic Approach to Feature Term Generalization. In In Proceedings of the 12th European Conference on Ma- chine Learning, Freiburg, Germany, pages 1-12. Springer-Verlag Lecture Notes in Computer Science (LNCS) no. 2167, September 5-7 2001.

Jose Julio Alferes, Carlos Viegas Damasio, et Luis Moniz Pereira (2003), Se- mantic Web Logic Programming Tools. In In Proceedings of the International Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR'03), Mumbai, India, pages 16-32. Springer-Verlag Lecture Notes in Computer Science (LNCS) no. 2901, December 8 2003.

T. Berners-Lee, R. Fielding, et L. Masinter (1998), Uniform Resource Identifiers (URI) : Generic Syntax. http ://www.isi.edu/in-notes/rfc2396.txt, August 1998.

Jeen Broekstra, Arjohn Kampman, et Frank van Harmelen (2002), Sesame : An Architecture for Storing and Querying RDF and RDF Schema. In In Proceedings of the First International Semantic Web Conference (ISWC 2002), Sardinia, Italy, pp 54-68. Springer-Verlag Lecture Notes in Computer Science (LNCS) no. 2342, June 9-12 2002.

S. Decker, D. Brickley, J. Saarela, et J. Angele (1998), A query and inference service for RDF. In QL'98 - The Query Languages Workshop, December 1998.

R. Fikes et D. L. McGuinness (2001), An Axiomatic Semantics for RDF, RDF Schema, and DAML+OIL. KSL Technical REport KSL-01-01, 2001.

Marcel Holsheimer, Rolf A. de By, et Hassan Ait-Kaci (1994), A Database Interface for Complex Objects. In In Proceedings of the Eleventh International Conference on Logic Programming, Santa Margherita Ligure, Italy, pp 437-455, June 13-18 1994.

G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, et K. Tolle (2003), Querying the Semantic Web with RQL. Computer Networks and ISDN Systems Journal, 42(5) : pp 617-640, 2003.

Michael Kifer, Georg Lausen, et James Wu (1995), Logical foundations of object-oriented and frame-based languages. Journal of the ACM, 42(4) : pp 741-843, july 1995.

M. Marchiori et J. Saarela (1998), Query + meta- data + logic = metalog. QL'98 : The Query Language Workshop, http ://www.w3.org/TandS/QL/QL98/pp/metalog.html, 1998.

D. L. McGuinness, R. Fikes, J. Hendler, et L. A. Stein (2002), DAML+OIL :An Ontology Language for the Semantic Web. IEEE Intelligent Systems, 17(5), 2002.

Joachim Peer (2003), Knowledge Transformation for the Semantic Web, volume 95 of Frontiers in Artificial Intelligence and Applications. IOS Press, 2003.

RDF Primer (1999), http ://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 1999.

M. Sintek et S. Decker (2001), TRIPLE - An RDF Query, Inference, and Transformation Language. In Deductive Databases and Knowledge Management (DDLP'2001), October 2001.

Guizhen Yang et Michael Kifer (2003), Reasoning about anonymous resources and meta statements on the semantic web. In Journal of Data Semantics, 2003.

Summary

The interest of representing data for the Semantic Web has generated standards for expressing knowledge on the Web. RDF, as one of those standards, has become a recommendation of the W3C. Even if it was designed to be human and machine readable (XML encoding, triples, labeled graphs), RDF was not provided with querying and reasoning services. Most work on querying RDF has concentrated on the use of logic programming evaluation techniques and SQL extensions. We take a new look at the problem of querying and reasoning on RDF statements and find that order-sorted feature (OSF) terms apply to this problem because OSF have been tailored for efficiency and their semantics is compatible with the isomorphic representation (triples) of RDF statements. This transformation allows to compute an ordering on resources and thus provide better answering mechanisms when querying RDF.