Métamodélisation pour le Web sémantique

Olivier Gerbé*, Guy Mineau**

*HEC Montréal, 3000, chemin de la Côte-Ste-Catherine, Montréal, Québec, Canada H3T 2A7
olivier.gerbe@hec.ca
**Université Laval, Québec, Québec, Canada G1K 7P4
mineau@ift.ulaval.ca

Résumé. Le Web sémantique nécessite la standardisation des mécanismes de représentation qui permettent à la connaissance contenue dans les pages Web d'être identifiée et traitée au niveau sémantique. La popularité de RDF est de plus en plus grande, cependant il existe de nombreux documents disponibles sur le Web qui utilisent d'autres modes de représentation. Nous proposons dans cet article d'utiliser les graphes conceptuels comme langage pivot du Web. Les graphes conceptuels sont souvent identifiés comme un langage clé dans la représentation de la connaissance de par leur simplicité, leur expressivité et leur similitude à d'autres langages de modélisation graphiques tels UML ou Entité-Relation. Nous proposons donc d'utiliser les techniques de métamodélisation pour mettre en œuvre ce langage pivot, et ainsi tirer profit, entre autres, de la proximité inhérente entre les graphes conceptuels et la langue naturelle pour faciliter l'implantation d'outils automatiques de traduction entre diverses représentations.

1 Introduction

L'avénement du Web sémantique [3] implique la standardisation des mécanismes de représentation afin que la connaissance incluse dans un document puisse être extraite et traitée à un niveau sémantique. RDF (Resource Description Framework) [23] semble être le standard émergent. RDF et RDF-S (RDF Schema) [21] définissent une façon de décrire les ressources du Web à l'aide d'un vocabulaire (propriétés et valeurs) qui est traitable par un ordinateur. RDF-S spécifie comment décrire le vocabulaire RDF. La popularité de RDF est de plus en plus grande, cependant il existe de nombreux documents disponibles sur le Web qui utilisent d'autres modes de représentation. Citons HTML (HyperText Markup Language) et sa balise <keyword>, XMI (XML Metadata Interchange) qui permet de coder des diagrammes UML (Unified Modeling Language) [17], E-R (Entity-Relationship) [7] utilisé dans le développement de schéma de bases de données, sans oublier OWL [22] qui vient étendre RDF-S en augmentant son pouvoir d'expression et qui est utilisé pour la description d'ontologies . Certains de ces documents resteront encodés avec leur format d'origine et ne pourront pas être traduits en RDF pour plusieurs raisons : a) leur utilité est mieux rendue par leur format, b) l'expressivité de RDF est insuffisante, c) les coûts de traduction seraient prohibitifs, d) les applications

qui utilisent ces documents devraient être modifiées et enfin e) cela nécessiterait la formation d'experts humains pour valider les traductions. Il apparaîtra donc, selon les besoins, de nombreuses correspondances un à un entre ces différents formalismes, comme cela a été fait dans [9]. Ceci va créer une situation où un jeune standard comme RDF va engendrer des problèmes inévitables de mise à niveau à chacune de ses évolutions et dans ce contexte l'effet domino sera important. C'est pourquoi nous préconisons l'utilisation d'un langage commun pour tous ces formalismes. Une correspondance entre ce langage commun et chacun des formalismes sera alors suffisant pour permettre des traductions (parfois partielles) d'un vers n'importe quel autre. Toute traduction d'un formalisme à un autre passera alors par le langage commun. Nous pensons qu'à long terme, un tel langage améliorera l'interopérabilité entre les systèmes.

Bien qu'il y ait plusieurs formalismes de représentation de la connaissance qui conviendraient, nous préconisons l'utilisation du formalisme des graphes conceptuels. Tim Berners-Lee a comparé RDF et les graphes conceptuels et a conclu que les graphes conceptuels pourraient être très facilement intégrés dans le Web sémantique [2]. Martin et Eklund ont utilisé les graphes conceptuels pour décrire et indexer des documents Web [14]. Delteil et al. [10] ont proposé une extension de RDF-S basé sur les graphes conceptuels dédiée à la représentation de connaissances contextuelles. Carloni et al. [4] s'intéressent à l'utilisation des graphes conceptuels en relation avec les Topic Maps. Un autre exemple de l'intérêt porté aux graphes conceptuels est le moteur de recherche Corese [8] pour le Web sémantique qui est basé sur les graphes conceptuels pour leur puissance d'inférence. Tous ces travaux militent pour l'adoption des graphes conceptuels comme langage pivot pour les échanges sur le Web, d'où notre intérêt à explorer cette avenue.

Notre démarche s'inscrit dans une démarche plus générale, appelée ingénierie dirigée par les modèles, qui consiste à travailler aux niveaux des modèles et métamodèles pour la transformation de données.

Dans cet article, nous présentons d'abord l'architecture des différents métamodèles : métamodèle UML, métamodèle des graphes conceptuels et le métamodèle de RDF-S (section 2). Ensuite nous introduisons le formalisme des graphes conceptuels (section 3). Puis dans la section 4 nous détaillons les métamodèles de RDF (4.1) et de RDF-S (4.2) , nous illustrons (4.3) l'utilisation de ces métamodèles et donnons quelques exemples des métamodèles de OWL (4.4) et UML (4.5). Puis nous montrons (section 5) comment définir des méta-règles permettant de passer d'un formalisme à un autre, que nous illustrons par des exemples. Enfin nous terminons par une revue des travaux futurs.

2 L'architecture

La figure 1 présente l'organisation des couches de modélisation avec trois métamodèles au niveau M2 : le métamodèle de UML avec les définitions de UML Class, UML Object et les autres éléments de UML, le métamodèle des graphes conceptuels avec les définitions de Concept Type, Concept et les autres éléments du formalisme, et le métamodèle de RDF avec les définitions de RDF-S Class, RDF Resource et les autres éléments de RDF et RDF-S.

Les modèles sont définis au niveau M1. Dans notre exemple, nous avons représenté à l'aide des graphes conceptuels l'assertion "Marie est une personne" en utilisant les termes définis au niveau M2 selon les trois métamodèles : métamodèle UML, métamodèle des graphes conceptuels et le métamodèle de RDF.

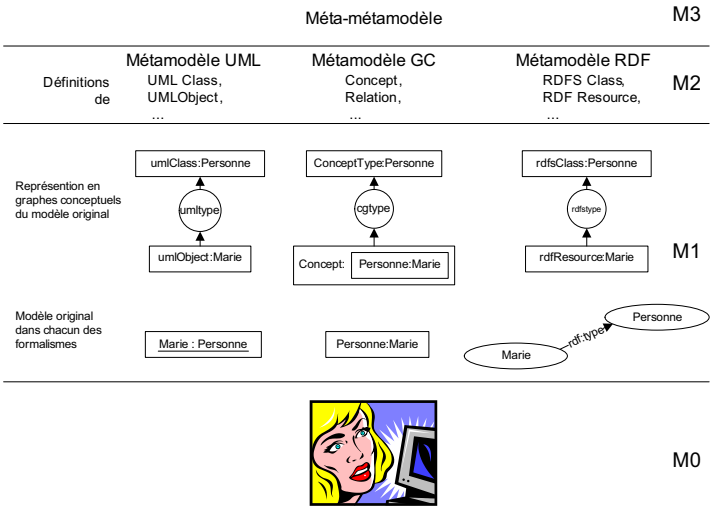

FIG. 1 – *Les trois couches de modélisation et le monde modélisé.*

Dans la partie gauche, sous le métamodèle de UML, l'assertion qui est représentée en UML par Marie : Personne est représentée par Marie qui est un umlObject qui a une relation umlType avec la umlClass Personne. Au centre se trouve la représentation en graphes conceptuels. À droite, l'assertion est représentée par Marie qui est une rdfResource dont le rdfType est la rdfClass Personne.

3 Les graphes conceptuels

Étant au centre de notre présentation car utilisé comme langage pivot, cette section présente l'essentiel de la théorie des graphes conceptuels.

Sowa a introduit dans [19] la première version du formalisme des graphes conceptuels. Depuis, de nombreux travaux ont porté sur ce formalisme ainsi que sur ses extensions [5, 15, 16, 24]. Mugnier et Chein [5, 16, 1, 6] ont généralisé le modèle de base de Sowa [19] en levant certaines restrictions (connexité des graphes, treillis de types).

3.1 Introduction

Le lecteur peu familier avec les graphes conceptuels trouvera ici une brève introduction ; seulement le minimum requis pour la compréhension de la suite de cet article est présenté. Plus d'information peut être trouvée dans [19, 5, 20].

Les graphes conceptuels sont un formalisme par lequel l'univers du discours est représenté par des concepts et des relations conceptuelles. Un concept représente un objet d'intérêt. Les relations conceptuelles permettent d'associer les concepts. La figure 2 représente la phrase "John est un chauffeur de taxi qui conduit le taxi numéro 23". La même phrase peut être représentée sous une forme linéaire ou sous une forme graphique.

```
[ChauffeurTaxi :John]->(conduit)->[Taxi :#23].
```

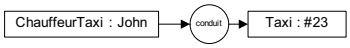

FIG. 2 – *Graphes conceptuels : concepts et relations.*

Les concepts peuvent être catégorisés par les relations qu'ils entretiennent avec les autres concepts. Les types de concepts appelés Concept Type définissent ces catégories. Un type de concept est défini par un graphe de définition. La figure 3 présente le graphe de définition du type ChauffeurTaxi qui spécifie qu'un chauffeur de taxi est un conducteur qui conduit un taxi. De la même façon, les relations peuvent être catégorisées par les concepts qu'elles lient. Les types de relation sont appelés Relation Type. Un type de relation est également définie par un graphe. La figure 3 présente également le graphe de définition du type mange.

```
Type ChauffeurTaxi(*x) is
    [Chauffeur :?x]->(conduit)->[Taxi]
Type mange(*x,*y) is
    [EtreAnime :?x]->(mange)->[Nourriture :?y]
```

FIG. 3 – *Graphes conceptuels : Type de concept et type de relation.*

Dans la figure 3, $?x$ et $?y$ font référence respectivement à $*x$ et $*y$. Ceci signifie que la variable $?x$ a la même valeur que $*x$ et que la variable $?y$ a la même valeur que $*y$. Il s'agit de la notion de coréférence des graphes conceptuels.

3.2 Vue d'ensemble du formalisme

La figure 4 présente la hiérarchie des types du formalisme des graphes conceptuels[1].

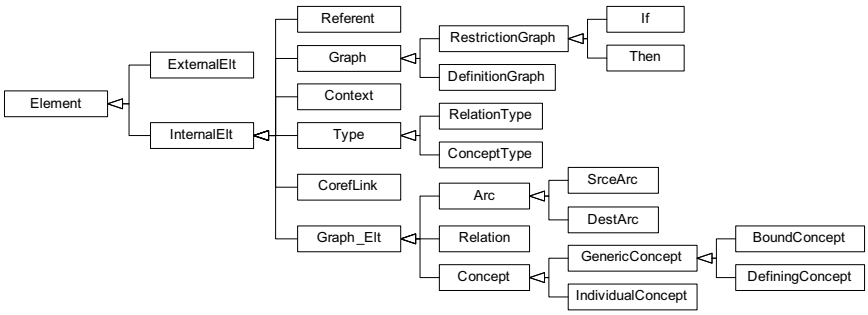

FIG. 4 – *Hiérarchie des types de concepts du langage.*

Au plus haut niveau de la hiérarchie nous distinguons deux catégories, les éléments externes (`ExternalElt`), et les éléments internes (`InternalElt`) qui sont les constituants du formalisme. Les éléments externes représentent les objets de l'univers du discours qui sont à l'extérieur du système et qui peuvent être référencés par des éléments internes. Les éléments

[1]Les libellés des type de concept sont en anglais pour des raisons d'intégration avec le standard en cours de définition.

internes sont regroupés en six types : référent, graphe, contexte, type, lien de co-référence et élément de graphe. Les référents (`Referent`) sont les représentants des objets de l'univers du discours ; les graphes (`Graph`) sont les phrases du langage, les contextes (`Context`) regroupent les graphes représentant la connaissance, les types (`Type`) permettent de regrouper les référents en catégories, les liens de co-référence (`CoRefLink`) lient les concepts représentant les mêmes éléments et les éléments de graphes (`Graph_Elt`) sont les éléments : arc (`Arc`), relation (`Relation`) ou concept (`Concept`), que l'on retrouve dans un graphe.

Parmi les concepts nous distinguons deux types de concepts : les concepts individuels (`IndividualConcept`) qui représentent des entités identifiées de l'univers du discours et les concepts génériques (`GenericConcept`) qui représentent des entités non identifiées (qui sont généralement représentés par des variables).

Parmi les graphes nous distinguons deux sous-types : définition et restriction. Les graphes de définition `DefinitionGraph` sont des graphes utilisés pour la définition des types de concept et des types de relation. Les graphes de restriction `RestrictionGraph` sont des graphes qui doivent être toujours faux et qui contraignent les définitions de types de concept. Les graphes `If` et `Then` sont des cas particuliers de graphes de restriction.

Les figures 5 et 6 montrent les liens qui existent entre ces types en utilisant le formalisme UML. La figure 5 présente les éléments qui constituent un graphe. Un graphe est constitué de concepts, de relations et d'arcs source et destination qui lient les concepts et les relations. Un concept est composé de deux parties : un type de concept, lié à d'autres types de concepts par la relation de sous-typage, et un référent qui représente un et un seul élément. Une relation est associée à un type de relation qui définit la sémantique de la relation conceptuelle. Enfin un contexte permet de segmenter l'information en regroupant des graphes par une relation d'extension et en spécifiant par une relation d'intension les graphes pour lesquels les graphes de l'extension sont vrais.

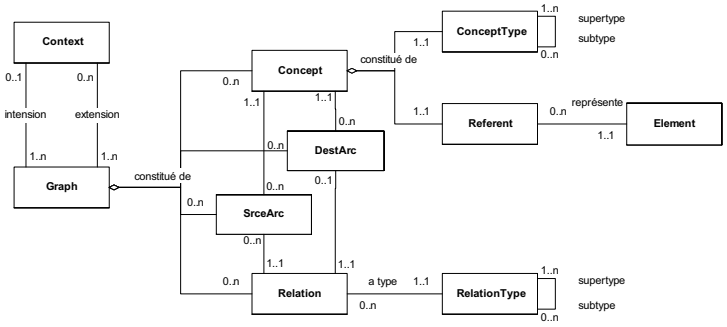

FIG. 5 – *Le modèle du langage - partie 1 (formalisme UML).*

La figure 6 présente les éléments définissant les types, types de concepts et types de relations. Un type est spécifié par : un graphe de définition, des graphes de restriction et des graphes de règles.

La figure 7 présente la hiérarchie des types de relations définis dans le langage. La relation de définition `def` et la relation de restriction `rstrct` associent les types à leurs graphes de spécification.

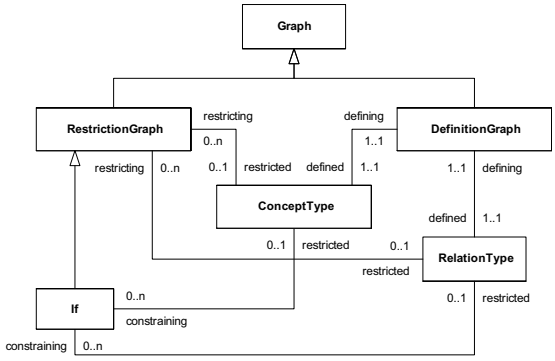

FIG. 6 – *Le modèle du langage - partie 2 (formalisme UML).*

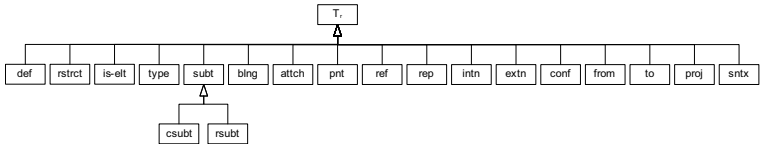

FIG. 7 – *Hiérarchie des types de relations du langage.*

La relation d'appartenance `is-elt` associe les éléments d'un graphe au graphe. La relation `type` lie un concept (resp. relation) à son type de concept (resp. relation). La relation de sous-typage `subt`, avec ses deux spécialisations `csubt` pour les types de concepts et `rsbut` pour les types de relations, organise les types en hiérarchie. La relation `blng` associe les arcs aux relations et la relation `attch` associe les arcs aux concepts. La relation `pnt` est une relation d'orientation pour les arcs. Les relations `ref` et `rep` associent respectivement un concept à son référent et un référent à l'objet qu'il représente. Les relations `intn` et `extn` associent un contexte aux graphes qui le définissent. La relation de conformité `conf` associe un référent à un type. Les relations `from` et `to` associent un lien de co-référence à respectivement la source du lien et la destination du lien. La relation de projection `proj` associe un graphe à un graphe plus spécialisé. Enfin la relation de syntaxe `sntx` associe un type au graphe qui spécifie la syntaxe.

Le lecteur intéressé trouvera dans [12] une présentation complète du métamodèle des graphes conceptuels.

3.3 Opérateur de correspondance

Dans [18] Sowa décrit un 'mapping' entre le niveau méta et le niveau données. Les graphes conceptuels permettent d'exprimer des assertions aussi bien au niveau méta qu'au niveau données comme dans l'exemple ci-dessous où il est exprimé que le type Entité a pour sous-type Plante.

`[Type :Plante]->(subt)->[Type :Entité] [Plante :∀⋆x] [Entité :?x]`

Pour traduire une expression du niveau méta (expression de gauche) au niveau données (expression de droite) Sowa définit deux opérateurs notés τ et ρ permettant de passer d'un

niveau au niveau inférieur. L'opérateur τ traduit le nom du référent en nom de type de concepts comme dans l'exemple ci-dessous qui traduit un énoncé à un niveau en un énoncé au niveau inférieur. Si t_1 est un sous-type de t_2 alors tout référent conforme à t_1 est conforme à t_2.

`IF : [Type :*t1]->(subt)->[Type :*t2]`
`THEN : [τt1 : ∀*x] [τt2 :?x]`

L'opérateur ρ joue le même rôle que τ mais pour les types de relations. L'opérateur transforme le nom d'une relation en un type de relation. L'exemple ci-dessous montre la règle de traduction du niveau méta au niveau données.

`IF : [Type :*t1]->(arg1)->[Relation :*r]<-(arg2)<-[Type :*t2]`
`THEN : [τt1]->(ρr)->[τt2]`

Plus généralement, nous avons besoin d'une fonction qui permet d'accéder à l'objet représenté par un concept, soit l'objet référé, afin de l'utiliser. Pour ce, il faut naturellement que l'objet représenté soit un élément interne au système. Par exemple, nous aimerions accéder à l'objet photoMarie représenté par le concept `[Photo :`  `]`, ou encore être capable de manipuler le graphe représenté par le concept `[Graph : [Cat]->(on)->[Mat] ]`.

Nous définissons ω une telle fonction.

Définition 1 *La fonction ω est définie de $\mathcal{C} \rightarrow \mathcal{E}$ où $\mathcal{C}$ est l'ensemble des concepts représentant des entités et $\mathcal{E}$ est l'ensemble de tous les éléments référencés.*

Appliquée à un concept, elle retourne l'objet dont le concept est une représentation conceptuelle.

ω`([Photo :`  `]) =` 
ω`( [Graph : [Cat]->(on)->[Mat] ]) = [Cat]->(on)->[Mat]`

L'exemple avec l'opérateur τ devient :

`IF : [Type :*t1]->(subt)->[Type :*t2]`
`THEN : [ω([Type :?t1]) :∀*x] [ω([Type :?t2]) :?x]`

Pour simplifier l'écriture ω`([Type :*t1])` pourra être noté tout simplement ωt_1 puisque c'est l'objet référencé qui est accédé et que tout référent identifie de manière unique l'objet qu'il référence. La règle s'écrit alors :

`IF : [Type :*t1]->(subt)->[Type :*t2]`
`THEN : [ωt1 :∀*x] [ωt2 :?x]`

De la même façon, la règle de traduction pour les relations peut s'écrire :

`IF : [Type :*t1]->(arg1)->[Relation :*r]<-(arg2)<-[Type :*t2]`
`THEN : [ωt1]->(ωr)->[ωt2]`

La fonction ω est plus générale que les opérateurs τ et ρ et peut être utilisée en lieu et place de ces opérateurs.

Exemples d'utilisation de ω

À titre d'exemple d'utilisation de la fonction ω pour la métamodélisation, nous donnons ci-dessous la définition, sous forme de graphes conceptuels, de la transitivité et de la réflexivité d'une relation. Nous verrons d'autres exemples appliqués à la traduction de modèles à la section 5.

Transitivité. La figure 8 montre le graphe définissant une relation transitive. Une relation $\mathcal{R}$ est transitive si et seulement si :

$$x\mathcal{R}y \wedge y\mathcal{R}z \Rightarrow x\mathcal{R}z \tag{1}$$

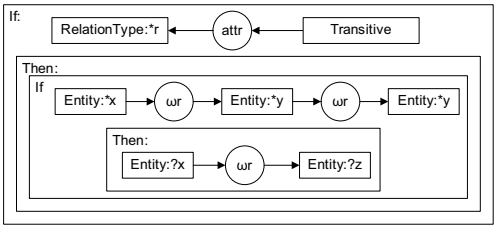

FIG. 8 – *Règle de définition d'une relation transitive.*

Réflexivité. La figure 9 montre le graphe définissant une relation réflexive. Une relation $\mathcal{R}$ définie sur T1 est réflexive si et seulement si :

$$\forall x \in \text{T1}, x\mathcal{R}x \tag{2}$$

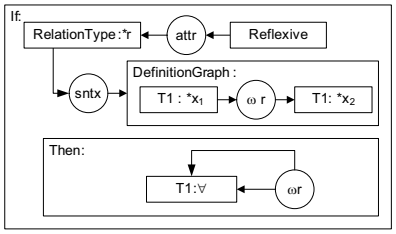

FIG. 9 – *Règle de définition d'une relation réflexive.*

Si une relation r, définie par un graphe de syntaxe qui montre que la relation r est définie sur l'ensemble T1, est réflexive alors tout concept T1 de l'ensemble de définition est en relation r avec lui-même.

4 Les métamodèles

Nous avons vu qu'il était possible de représenter les différents niveaux de modélisation avec les graphes conceptuels. Dans cette section, nous illustrons la démarche en présentant, exprimés en graphes conceptuels, les métamodèles de RDF et RDF-S, ainsi que des éléments des métamodèles de OWL et UML.

4.1 RDF

Nous présentons les trois principaux éléments de RDF : Resource, Property, et Statement. Ce sont les trois éléments fondamentaux de RDF et le lecteur pourra en déduire comment les autres éléments pourraient être représentés en graphes conceptuels.

RDF Resource

Le principal élément de RDF est la notion de ressource que nous noterons rdfsResource. rdfsResource est au sommet de la hiérarchie de classe et est sous-classe d'elle-même. La figure 10 présente sa spécification en graphes conceptuels. Une rdfsResource a une relation rdfstype avec rdfsClass, elle a également une relation rdfslabel et une relation rdfsComment avec un rdfsLitteral.

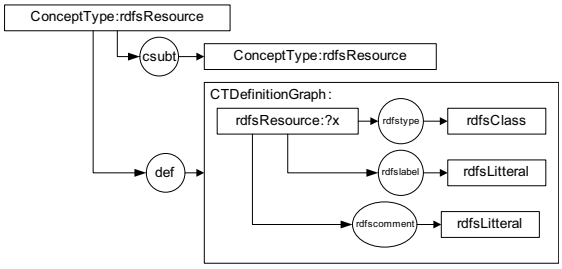

FIG. 10 – *La spécification de* rdfsResource.

RDF Property

Dans RDF les notions d'attributs et de relations sont réalisées par la notion unique de propriété (Property). Une Property relie deux classes. L'une est la classe à laquelle s'applique la propriété (attribut de la classe ou source de la relation), l'autre est la classe dans laquelle les valeurs sont prises (valeur de l'attributs ou cible de la relation). Associées à Property, RDF-S définit deux relations (propriétés) : domain et range qui seront détaillées ultérieurement. La figure 11 montre la représentation en graphes conceptuels de la spécification de Property.

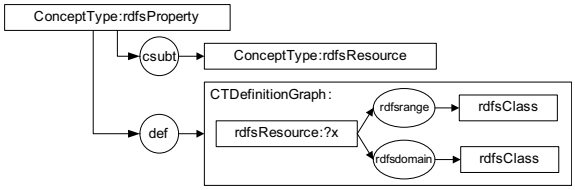

FIG. 11 – *La spécification de* Property.

RDF Statement

Dans RDF, la connaissance est représentée sous forme de déclarations (statements). Une déclaration est une association entre une ressource, une propriété et une valeur. La figure 12 illustre la représentation de connaissance en RDF et sa transcription en graphes conceptuels : la propriété et la valeur sont liées à la ressource qu'elles qualifient. Les arcs subject, object et predicate identifient les rôles joués par chacun des concepts [2].

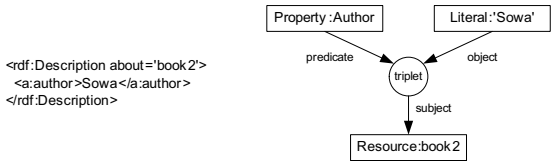

FIG. 12 – *Une déclaration RDF.*

RDF High Order Statement

Il est parfois nécessaire d'exprimer des connaissances sur des déclarations RDF. En RDF les déclarations peuvent être réifiés ce qui permet la représentation de déclarations à propos de déclarations. Ces déclarations sont appelés déclarations d'ordre supérieur (High Order Statement).

La figure 13 montre une déclaration d'ordre supérieur exprimée en RDF et sa représentation en graphes conceptuels. Une déclaration d'ordre supérieur est représentée par une ressource ayant quatre propriétés. La propriété subject identifie la ressource décrite. La propriété predicate identifie la propriété de la déclaration, la propriété object identifie la valeur de la propriété ou la ressource liée par cette propriété et la propriété type qui statue que la ressource est une déclaration. Avec la notion de déclaration d'ordre supérieur, il est possible de déclarer "Paul dit que le livre est écrit par Sowa". l'assertion est représentée par une déclaration d'ordre supérieur attribuée à Paul

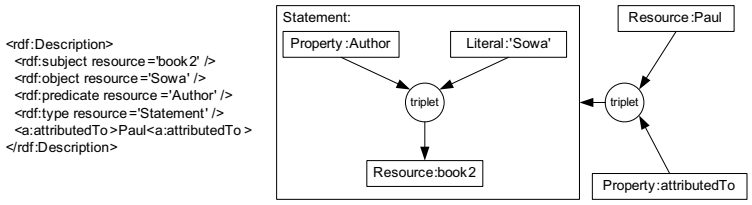

FIG. 13 – *Une déclaration d'ordre supérieur.*

[2] Les libellés des arcs ont été ajoutés pour une meilleure compréhension. Dans le formalisme des graphes conceptuels, les arcs sont numérotés afin de les différencier.

4.2 RDF Schema

Nous présentons ici les principaux éléments du métamodèle de RDF Schema. Nous ne détaillerons pas tous les éléments du métamodèle, nous nous concentrerons sur les éléments et propriétés fondamentaux. La figure 14 présente la hiérarchie des éléments détaillés ici.

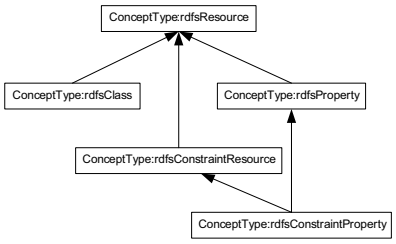

FIG. 14 – *La hiérarchie de type de RDF Schema.*

Au sommet se trouve le type rdfsResource. Tous les éléments de RDF et RDF-S sont considérées comme des ressources et héritent des propriétés. Juste en-dessous se trouvent les deux éléments fondamentaux rdfsClass et rdfsProperty qui correspondent respectivement à Concept Type et Relation Type dans le formalisme des graphes conceptuels. rdfsConstraintResource est un élément ad-hoc utilisé pour spécifier des contraintes. En particulier rdfsConstraintProperty est utilisé pour implémenter les propriétés de contraintes domain et range.

RDF Schema Class

Class correspond à la notion abstraite de type. Comme il est précisé dans le document de spécification de RDF Schema [21], cette notion est similaire à la notion de classe dans les langages de programmation orienté objet. Ceci signifie que la spécification des membres de la classe est exprimée au niveau des types comme dans UML [17] et non aux niveaux des instances comme dans le formalisme des graphes conceptuels [11]. La figure 15 présente la spécification de rdfsClass en graphes conceptuels. Au niveau métamodèle, une rdfsClass est une sorte de rdfsResource, il y a donc une relation csubt entre les deux concepts `[ConceptType :rdfsClass]` et `[ConceptType :rdfsResource]`. Au niveau modèle, une rdfsClass est une rdfssubClass d'une autre rdfsClass et est identifiée par un ID.

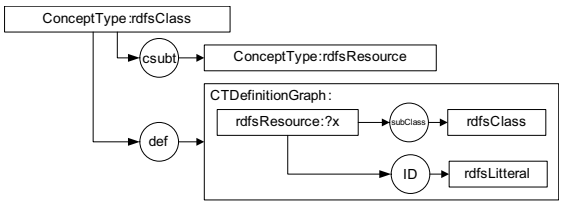

FIG. 15 – *La spécification de* Class.

RDF Schema type

La relation type est utilisée pour indiquer qu'une ressource RDF est membre d'une classe. La relation lie une ressource à sa classe. Dans RDF-S une ressource peut être associée à plus d'une classe. La figure 16 présente la spécification de la relation type.

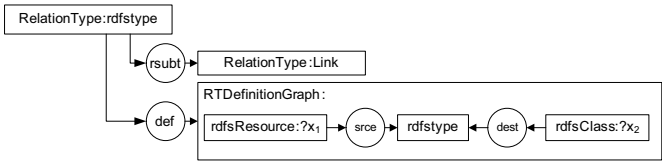

FIG. 16 – *La spécification de la relation* type.

RDF Schema subClassOf

La relation subClassOf lie une classe à sa super classe. La figure 17 présente la spécification de la relation. Il est à noter que dans RDF-S une classe peut être associée à plus d'une super-classe.

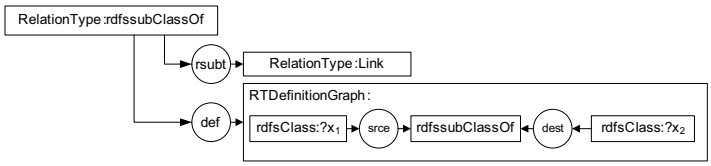

FIG. 17 – *La spécification de la relation* subClassOf.

RDF Schema domain

La relation domain lie une propriété aux classes dont les membres peuvent avoir cette propriété. La figure 18 présente la spécification de la relation.

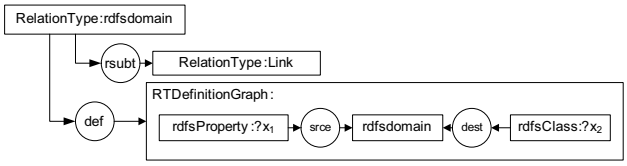

FIG. 18 – *La spécification de la relation* domain.

RDF Schema range **Relationship**

La relation range lie une propriété à la classe dans laquelle cette propriété prend sa valeur. La figure 19 présente la spécification de la relation.

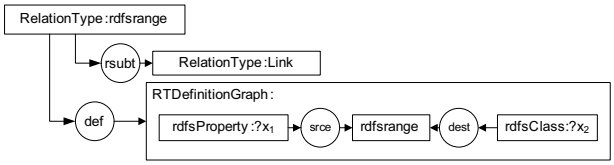

FIG. 19 – *La spécification de la relation* range.

Nous venons de montrer comment les graphes conceptuels pouvaient être utilisés pour représenter le métamodèle RDF-S. Nous allons voir maintenant comment exprimer les déclarations RDF en graphes conceptuels.

4.3 Exemple de représentation RDF Schema et RDF

Avec les éléments de RDF et RDF-S définis précédemment, nous pouvons représenter les faits RDF et RDF-S dans le même formalisme. Les figures 20 et 21 illustrent l'intégration de ces deux niveaux de modélisation.

La figure 20 présente une partie du métamodèle de RDF-S avec les éléments principaux de RDF-S, les types rdfsClass, rdfsProperty, rdfsRessource, et les relations rdfsdomain, rdfsrange, rdfstype et rdfstriplet et intègre les éléments spécifiques à l'application. La partie inférieure gauche exprime que la propriété Author prend ses valeurs dans la classe Litteral que cette propriété peut s'appliquer à la classe Book.

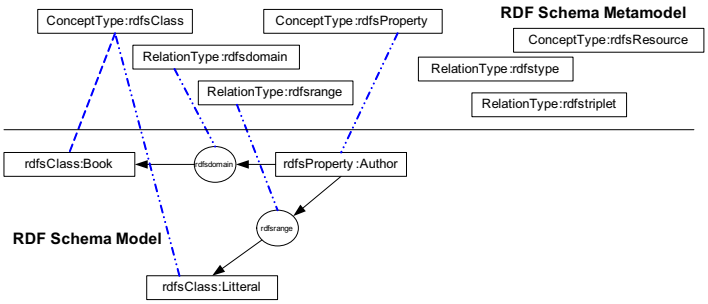

FIG. 20 – *Le modèle de* Livre.

La figure 21 intègre la déclaration "Sowa est l'auteur du Livre2". La partie inférieure droite exprime la déclaration elle-même : la ressource Book2 est le sujet de la déclaration, la ressource Sowa dont le type est la classe Litteral est l'objet de la déclaration et la propriété Author est le prédicat de la déclaration. Nous avons ajouté des lignes pointillées entre les concepts et leurs types pour bien montrer les relations entre les deux niveaux bien que ces relations soient implicitement représentées par les noms des types des concepts.

4.4 OWL

Nous ne développerons pas dans cette section le métamodèle de OWL. Nous donnerons juste un élément du métamodèle de OWL. La figure 22 présente le graphe de spécification de

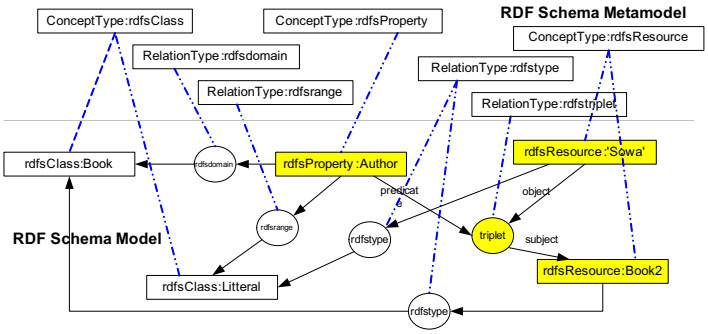

FIG. 21 – *La représentation de "Sowa est l'auteur du Livre2".*

owlRestriction qui permet d'exprimer des contraintes sur les classes.

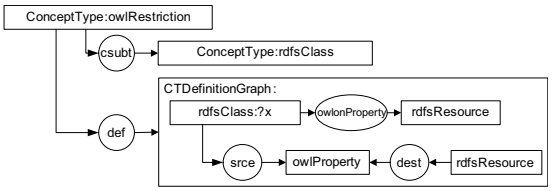

FIG. 22 – *La spécification de* Restriction.

owlRestriction est une sous-classe de owlClass, elle définit une contrainte sur (owlonProperty) une ressource (rdfsRessource). Cette contrainte est exprimé par un critère owlProperty et une valeur (rdfsRessource). Un exemple de restriction est minCardinality qui exprime la cardinalité minimale pour une relation. Le lecteur trouvera plus loin (figure 32) une règle de transformation mettant en œuvre cette contrainte.

4.5 UML

De même pour UML, nous ne présenterons que deux exemples illustrant le métamodèle de UML exprimé en graphes conceptuels. La figure 23 présente la spécification de la notion de umlSlot qui correspond à la valeur d'un attribut ou propriété d'une instance (umlInstanceSpecification).

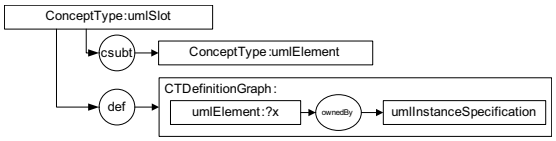

FIG. 23 – *La spécification de* Slot.

La figure 24 présente la spécification de la notion de umlProperty qui généralise la notion d'attribut. Une umlProperty est une sous-classe de StructuralFeature et est associée au classificateur (umlclassifier) qu'elle caractérise.

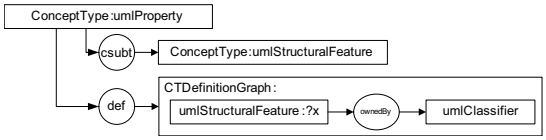

FIG. 24 – *La spécification de* Property.

5 Règles de transformation

Pour différents formalismes, nous pouvons avoir une représentation de leur métamodèle en graphes conceptuels et nous pouvons alors définir des règles de transformation d'un modèle à un autre[3] comme illustré ci-dessous.

En utilisant l'opérateur ω (section 3.3) nous pouvons définir des méta-règles pour transformer des modèles d'un formalisme à un autre[4]. Pour illustrer ces transformations, nous présentons ci-dessous six exemples de règles.

La première règle présentée ici permet de transformer une ressource RDF dans son équivalent en graphes conceptuels. Une ressource RDF correspond à la notion de concept dans les graphes conceptuels. La règle établit donc que s'il existe une ressource RDF x instance d'une classe RDF-S y alors nous avons un concept dont le type est y et le référent x. La figure 25 présente cette règle.

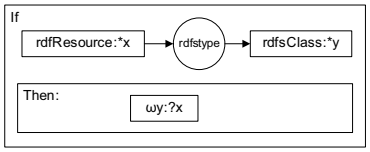

FIG. 25 – *Une ressource RDF est transformée en un concept..*

Les deuxième et troisième règles permettent de transformer un modèle RDF avec contraintes en son correspondant en graphes conceptuels. Les contraintes ne sont pas toujours définies au même niveau dans RDF et dans les graphes conceptuels. Dans RDF les contraintes sont définies entre les classes et les propriétés et s'appliquent au niveau des ressources. Dans les graphes conceptuels les contraintes sont exprimées au niveau des concepts dans les graphes de définition.

Les contraintes sur les classes, comme par exemple, rdfssubClassOf, sont transformées en contraintes sur les types de concepts. La propriété rdfssubClassOf entre classes est transformée en relation csubt entre deux types de concepts (voir figure 26).

Par contre, les contraintes comme rdfsrange et rdfsdomain sont transformées en contraintes sur les concepts. La règle présentée dans la figure 27 transforme une relation entre deux classes en son équivalent en graphes conceptuels. Une propriété RDF-S x applicable à la classe y et prenant ses valeurs dans la classe t sera transformée en un type de relation dont le graphe de définition montre que cette relation x lie deux concepts de types z et t.

[3] Si l'expressivité des deux formalismes est équivalente. Si ce n'est pas le cas, il y a seulement un sous-ensemble des construits du formalisme le plus expressif qui pourra être transformé dans le formalisme le moins expressif.

[4] À condition que les méta-règles n'utilisent pas de symboles fonctionnels ni ne soient récursives, nous pouvons faire une transformation directe entre construits équivalents.

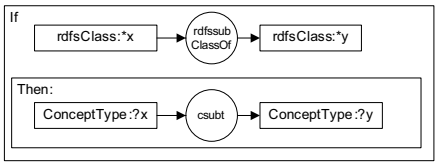

FIG. 26 – *La relation* subClassOf *de RDF-S est transformée en la relation* csubt..

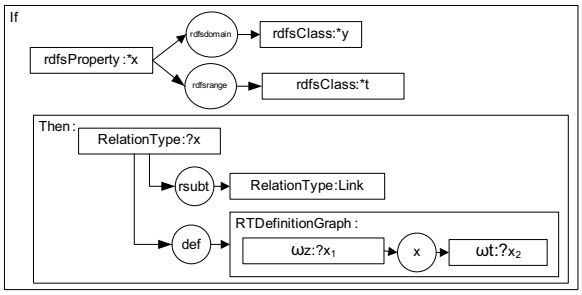

FIG. 27 – *Une contrainte RDF-S est transformée en type de relation.*

La quatrième règle illustre la transformation dans l'autre sens d'un modèle en graphes conceptuels vers un modèle RDF-S. Une définition de type de concept est transformée en un ensemble de contraintes RDF-S. La figure 28 présente la méta-règle et la figure 29 présente un exemple d'application.

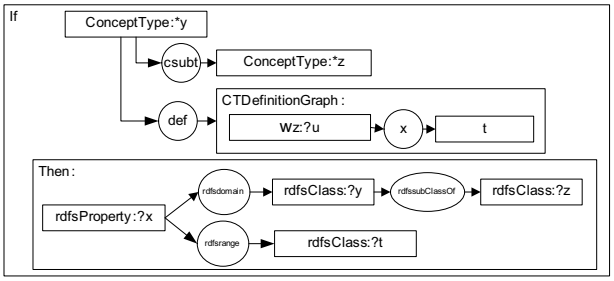

FIG. 28 – *Une définition de type de concept est transformée en contraintes RDF-S.*

En appliquant les règles vues précédemment nous pouvons transformer une définition de type de concept en contraintes RDF-S et réciproquement comme illustré dans la figure 29.

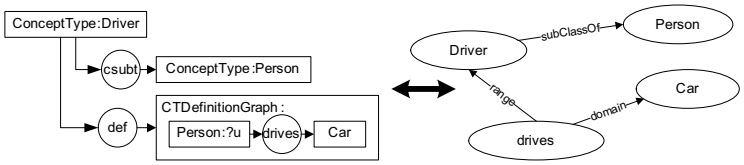

FIG. 29 – *Exemple de transformation.*

La cinquième règle permet de transformer des objets UML en ressources RDF. Comme il y a une correspondance entre UML Class et RDF-S Class et entre UML Object et RDF-S Ressource, la règle de transformation est simple. La figure 30 présente cette règle.

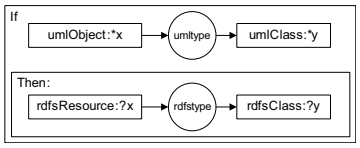

FIG. 30 – *Un objet UML transformé en ressource RDF.*

En appliquant cette règle, nous pouvons transformer un modèle UML en sa représentation en RDF. La figure 31 illustre un tel exemple. L'objet UML représentant la personne Mary est transformé via le formalisme des graphes conceptuels en une représentation RDF.

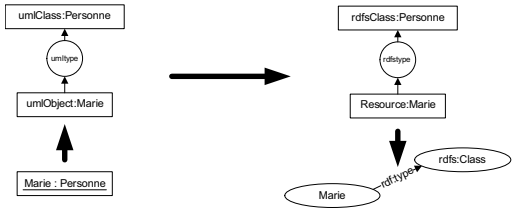

FIG. 31 – *Application de la règle de la figure 30.*

Enfin nous présentons une règle (figure 32) qui permet de transformer une contrainte exprimée en OWL en une contrainte exprimée en graphes conceptuels.

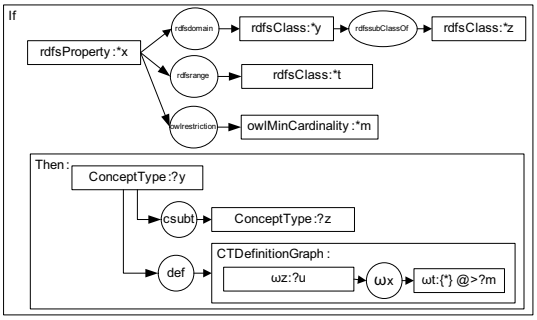

FIG. 32 – *Une contrainte OWL est transformée en une contrainte CG.*

Une classe y, sous-classe de z, étant liée à une classe t par la propriété x, si cette propriété a une cardinalité minimale m, alors le graphe de définition de y montre que toute instance de y est liée a au moins m t. Le symbole @ exprime la cardinalité de l'ensemble {*} dans le formalisme des graphes conceptuels.

En appliquant cette règle, nous pouvons exprimer une expression OWL en graphes conceptuels. La figure 32 en montre un exemple.

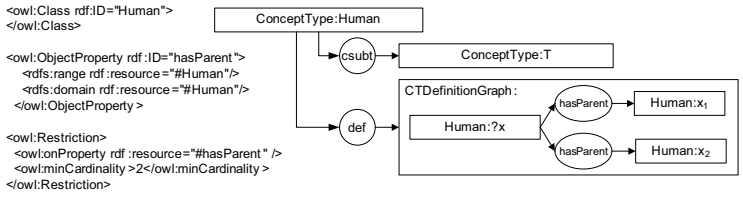

FIG. 33 – *Application de la règle de la figure 32.*

La partie gauche de la figure présente l'expression OWL. Un être humain a au moins deux parents. Ceci est traduit par le règle de la figure 32 en un graphe conceptuel qui montre qu'un humain est lié à au moins deux humains comme parents.

6 Conclusion

Dans cet article, nous avons présenté comment un langage pivot pour le Web sémantique pourrait aider à la communication. Plutôt que de développer des traducteurs pour chaque paire de langages utilisés sur le Web, nous préconisons l'usage d'un langage pivot vers et à partir duquel les traductions se feraient. Nous avons également montré que le formalisme des graphes conceptuels est un très bon candidat, d'autant plus que ses capacités d'inférence permettrait alors le raisonnement sur les connaissances codées dans les documents. Nous avons illustré nos propos par quelques exemples qui montrent le potentiel de cette approche, mais il reste encore beaucoup de travail à faire. Il faut développer d'une manière exhaustive les représentations des différents métamodèles des langages utilisés sur le Web. Il faut également définir les règles de transformation qui ne sont pas toujours aussi simples que celles présentées ici. En effet, lorsque la sémantique des éléments considérés diffère, il faut être capable de traduire ces nuances et d'avertir les utilisateurs. Nous y travaillons actuellement.

Enfin l'utilisation des graphes comme langage pivot permettrait d'exploiter pleinement des travaux comme ceux de Martin et Eklund [14], Carloni et al. [4], et le moteur de recherche Corese [8].

Références

[1] J.-F. Baget and M.-L. Mugnier. Extensions of Simple Conceptual Graphs : The Complexity of Rules and Constraints. *Journal of Artificial Intelligence Research*, 16 :425–465, 2002.

[2] T. Berners-Lee. Conceptual Graphs and the Semantic Web. february 2001. available at http ://www.w3.org/DesignIssues/CG.html.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2002.

[4] O. Carloni, M. Leclère, and M.-L. Mugnier. Introducing Graph-based Reasoning into a Knowledge Management Tool : an Industrial Case. In *Proceedings of IEA / AIE'06 (19th International Conference on Industrial, Engineering and Other Applications of Applied*

Intelligent Systems), number 4031 in Lecture Notes in Computer Science, pages 590–599. Springer, 2006.

[5] M. Chein and M-L. Mugnier. Conceptual Graphs : Fundamental Notions. *Revue d'intelligence artificielle*, 6(4) :365–406, 1992.

[6] M. Chein and M.-L. Mugnier. Types and Coreference in Simple Conceptual Graphs. In *Proceedings of the International Conference on Conceptual Structures, ICCS2004*, number 3127 in LNAI, pages 303–318, Hunstville, USA, 2004. Springer.

[7] P. Chen. The Entity-Relationship Model – Towards a Unified View of Data. *ACM Transactions on Database systems*, 1(1) :9–36, 1976.

[8] O. Corby, R. Dieng-Kuntz, C. Faron-Zucker, and F. Gandon. Searching the Semantic Web : Approximate Query Processing based on Ontologies. *IEEE Intelligent Systems Journal*, 22(1) :20–27, 2006.

[9] A. Delteil, R. Dieng, and C. Faron-Zucker. Extension of RDFS Based on the CGs Formalism. In H. Delugach and G. Stumme, editors, *Proceedings of the 9th International Conference on Conceptual Structures (ICCS 2001)*, number 2120 in LNAI, pages 275–289, Stanford, CA, USA, August 2001. Springer-Verlag.

[10] A. Delteil, C. Faron, and R. Dieng. Le modèle des graphes conceptuels pour le web sémantique : Extensions de RDF et RDFS basées sur le modèle des graphes conceptuels. *L'Objet*, 9(3) :95–122, 2003.

[11] O. Gerbé. Conceptual Graphs for Corporate Knowledge Repositories. In H. Delugach, M. Keeler, L. Searle, and J. Sowa, editors, *Proceedings of the 5th International Conference on Conceptual Structures (ICCS'1997)*, number 1257 in LNAI, pages 474–488, Seattle, Washington, USA, August 1997. Springer-Verlag.

[12] O. Gerbé, G. Mineau, and R. Keller. Un métamodèle des graphes conceptuels. *Revue d'intelligence artificielle*, 21(2/2007) :255–284, 2007.

[13] ISO/JTC1/SC 32. *Information technology - Common Logic (CL) - A Framework for a Family of Logic-Based Languages*, December 2005. Final Committee Draft ISO/IEC.

[14] P. Martin and P. Eklund. Embedding Knowledge in Web Documents : CGs versus XML-based Metadata Languages. In W. Cyre and W. Tepfenhart, editors, *Proceedings of the 7th International Conference on Conceptual Structures (ICCS'1999)*, LNAI, pages 230–246, Blacksburg, VA, USA, July 1999. Springer-Verlag.

[15] M.-L. Mugnier. On Generalization/Specialization for Conceptual Graphs. *Journal of Experimental and Theorical Artificial Intelligence*, 7(3) :325–344, 1995.

[16] M.-L. Mugnier and M. Chein. Représenter des connaissances et raisonner avec des graphes. *Revue d'intelligence Artificielle*, 10(1) :7–56, 1996.

[17] Object Management Group. *OMG Unified Modeling Language Specification*, 2004.

[18] J. Sowa. Relating Diagrams to Logic. In G. Mineau, B. Moulin, and J. Sowa, editors, *Proceedings of the First International Conference on Conceptual Structures (ICCS'93)*, number 699 in LNAI, pages 1–35, Québec, Canada, August 1993. Springer-Verlag.

[19] J.F. Sowa. *Conceptual Structures - Information processing in mind and machine*. Addison wesley 14472, 1984.

[20] J.F. Sowa. *Knowledge Representation : Logical, Philosophical and Computational Foundations*. BooksCole, 2000.

[21] W3C. *Resource Description Framework (RDF) Schema Specification 1.0*, March 2000.

[22] W3C. *OWL Web Ontology Language - Overview* , 2004. Document disponible à : http ://www.w3.org/TR/2004/REC-owl-features-20040210/.

[23] W3C. *RDF/XML Syntax Specification* , 2004.

[24] M. Wermelinger. Conceptual Graphs and First-Order Logic. In G. Ellis, R. Levinson, W. Rich, and J. Sowa, editors, *Proceedings of the Third International Conference on Conceptual Structures, ICCS1995*, number 954 in LNAI, pages 323–337. Springer-Verlag, August 1995.

Summary

The semantic Web entails the standardization of representation mechanisms so that the knowledge contained in a Web document can be retrieved and processed on a semantic level. RDF seems to be the emerging encoding scheme for that purpose. However, there are many different sorts of documents on the Web that do not use RDF as their primary coding scheme. It is expected that many one-to-one mappings between pairs of document representation formalisms will eventually arise. Rather, we advocate the use of a common language for all these encoding formalisms. Though there may be many knowledge representation formalisms suited for that task, we advocate the use of the conceptual graph formalism.