Une synthèse des modèles de représentation des connaissances à base de Graphes Conceptuels et OWL

Thomas Raimbault[*,**], Henri Briand[**], David Genest[*], Rémi Lehn[**], Stéphane Loiseau[*]

[*]LERIA, Université d'Angers, 2 boulevard Lavoisier 49045 Angers Cedex 01
{thomas.raimbault, david.genest, stephane.loiseau}@info.univ-angers.fr
[**]LINA, École Polytechnique de Nantes, rue C. Pauc BP 50609 44306 Nantes Cedex 3
{henri.briand, remi.lehn}@polytech.univ-nantes.fr

Résumé. Nous présentons et comparons deux approches de modélisation, formelles et concrètes, pour représenter et manipuler des connaissances d'un domaine. Le modèle des graphes conceptuels permet de modéliser des connaissances en terme de graphes, basés sur un support. Cette approche de modélisation est intensionnelle, est munie d'une sémantique en logique du premier ordre, et fait l'hypothèse d'un monde fermé pour ses raisonnements. Le langage OWL permet de décrire des ontologies et des faits sur le Web, suivant une approche de modélisation extensionnelle. Il possède une sémantique issue des logiques de descriptions, et fait l'hypothèse d'un monde ouvert pour ses raisonnements.

1 Introduction

La modélisation des connaissances consiste à représenter un ensemble de données ou connaissances sous une forme adaptée pour qu'un opérateur humain et/ou machine, puisse les interpréter et les manipuler. Une représentation est définie selon un *modèle* qui fournit les règles syntaxiques de modélisation, appelées la *syntaxe*. Le modèle peut être muni d'une *sémantique*, logique par exemple, pour définir clairement le sens de ce qui est modélisé. Le modèle est dit *formel* si les modélisations basées sur ce modèle peuvent être interprétées syntaxiquement et sémantiquement sans ambiguïté.

Le but de cet article est de présenter et comparer le *modèle des graphes conceptuels* et le *langage OWL*, qui sont deux approches de modélisation ayant les particularités (1) d'être formelles pour les versions prises en compte, (2) de représenter des connaissances, et (3) d'être concrètes dans le sens où il existe des outils efficaces pour manipuler les connaissances modélisées. Le modèle des graphes conceptuels permet de modéliser les connaissances d'un domaine en terme de graphes, basés sur un support. Cette approche de modélisation est intensionnelle, est munie d'une sémantique en logique du premier ordre, et fait l'hypothèse d'un monde fermé pour ses raisonnements. Le langage OWL permet de décrire des ontologies et des faits sur le Web, suivant une approche de modélisation extensionnelle. Il possède une sémantique issue des logiques de descriptions, et fait l'hypothèse d'un monde ouvert.

Le modèle des graphes conceptuels et le langage OWL modélisent les connaissances sur deux niveaux conceptuels, l'un *structurel* et l'autre *factuel*. Cette conceptualisation sur deux niveaux est nécessaire dès lors que la manipulation des connaissances doit s'appliquer de façon

générique et/ou de façon spécifique sur les connaissances modélisées. Par exemple, il peut être souhaitable de tenir compte des Personnes en général, ou plus spécifiquement de ne s'intéresser qu'à l'individu Pierre. Le niveau factuel fait référence à la spécificité des faits, il est composé des *individus* appelés aussi *instances* ou *assertions* de classes, et des *liens* entre les individus, appelés aussi *assertions* de relations. Ce niveau constitue les *connaissances factuelles* du domaine. Le niveau structurel confère à la modélisation une vision générique des connaissances, en décrivant et organisant des familles d'individus, appelées *classes* ou *concepts*, et des familles de *propriétés*, appelées aussi *relations* ou *rôles*. Ce niveau constitue les *connaissances structurelles* du domaine, et peut éventuellement constituer une ontologie du domaine. Notons que les connaissances factuelles d'un domaine n'ont de sens qu'avec les connaissances structurelles du domaine : sans ces dernières, les faits ne seraient que de simples données. Remarquons que cette vision à deux niveaux conceptuels en modélisation est intuitive chez l'humain, et que les systèmes informatiques l'ont adoptée. Par exemple, dans les systèmes de représentation des connaissances objet, les classes regroupent des caractéristiques communes qui sont partagées par les instances de ces classes. En bases de données, les schémas des tables structurent l'information tandis que les tuples sont les données. Dans le modèle des graphes conceptuels, le vocabulaire de base du domaine est défini dans un support et les faits par des graphes. Le langage OWL permet de décrire des ontologies et des faits basés sur ces ontologies.

Le terme *ontologie*, qui renvoie étymologiquement à la "théorie de l'existence", est un mot que l'informatique a emprunté à la philosophie au début des années 1990. En philosophie, l'Ontologie est l'étude de "l'être en tant qu'être". Il s'agit de la théorie qui tente d'expliquer les concepts qui existent dans le monde et comment ces concepts s'imbriquent et s'organisent. En Intelligence Artificielle, une définition communément admise d'ontologie est énoncée dans (Gruber, 1993) puis précisée dans (Studer et al., 1998) : "la spécification formelle et explicite d'une conceptualisation partagée" [traduction libre]. Nous adoptons la définition suivante : *une ontologie est une modélisation formelle d'une conceptualisation structurelle d'un domaine, décrite par des entités génériques appelées concepts qui sont organisées par des relations*.

Selon nous, toute approche de modélisation doit permettre les raisonnements suivants. Le premier est de pouvoir *interroger* les connaissances modélisées. Le second est la capacité de *déduire* de nouvelles connaissances à partir de celles déjà modélisées. Le troisième est de *vérifier* les connaissances, c'est-à-dire la capacité de savoir si la modélisation reste valide au cours de son cycle de vie. Remarquons que si ces raisonnements ne sont pas très différents d'un point de vue sémantique car il s'agit d'*inférences*, ils peuvent l'être d'un point de vue des mécanismes mis en place pour les réaliser. Nous verrons dans cet article les particularités et les limites du modèle des graphes conceptuels et du langage OWL pour raisonner. Le modèle des graphes conceptuels dispose de la projection, de règles et de contraintes comme mécanismes pour interroger, déduire et vérifier des connaissances, et des outils exploitant ces mécanismes. Le langage OWL dispose des raisonnements en logiques de descriptions et de langages et outils associés de requêtes et de règles.

Cet article est structuré de la façon suivante. La Section 2 fournit, en terme de représentation des connaissances, une comparaison entre le modèle des graphes conceptuels et le langage OWL. Ces deux approches de modélisation y sont préalablement introduites. La Section 3 présente une comparaison entre graphes conceptuels et OWL du point de vue des raisonnements. Les opérations du modèle des graphes conceptuels, les règles et les contraintes, ainsi que des outils logiciels y sont présentés ; tout comme les opérations de base en logiques de descriptions, et des langages et outils de règles et de requêtes pour OWL.

2 Modélisation des connaissances

2.1 Le modèle des Graphes Conceptuels

Le modèle des *graphes conceptuels* (GCs) est un modèle formel de représentation des connaissances, qui tient son origine des réseaux sémantiques (Lehmann, 1992). Un modèle simple des GCs est introduit dans (Sowa, 1984) et formalisé dans (Chein et Mugnier, 1992; Mugnier et Chein, 1996). Plusieurs extensions ont été proposées formant ainsi une famille de modèles de GCs selon les extensions utilisées, comme les GCs emboîtés typés (Chein et Mugnier, 1997), les liens de coréférence (Chein et Mugnier, 2004), les règles (Salvat, 1998), et les contraintes (Baget et Mugnier, 2002).

De façon générale, l'utilisation du modèle des GCs pour modéliser un domaine se fait en deux étapes : d'abord la définition d'un *support* qui spécifie les connaissances structurelles du domaine, ensuite la création de GCs qui modélisent les connaissances factuelles.

Support. *Un support est un quintuplet $S = (T_C, T_R, \sigma, I, \tau)$ (Mugnier et Chein, 1996). T_C est un ensemble partiellement ordonné de* types de concepts, *où $\top$ est le plus grand élément. T_R un ensemble de* types de relations *partitionné : $T_R = T_{R_{i_1}} \cup \cdots \cup T_{R_{i_p}}$, où $T_{R_{i_j}}$ est l'ensemble des types de relations d'arité i_j pour $i_j > 0$. Chaque $T_{R_{i_j}}$ est partiellement ordonné et possède un plus grand élément noté $\top_{i_j}$. La relation d'ordre pour les ensembles des types de concepts et des types de relations est notée $\leq$. σ est une application associant à chaque type de relation t_r une* signature, *qui spécifie l'arité de t_r et le plus grand type possible pour chaque argument. On note $\sigma_i(t_r)$ le $i^{ème}$ argument de $\sigma(t_r)$. I est l'ensemble des* marqueurs individuels. *À cet ensemble I s'ajoute un* marqueur générique, *noté $*$, qui permet de représenter un individu non spécifié. τ est une application, dite de* conformité, *de I dans T_C qui associe à chaque marqueur individuel son type.*

EXEMPLE. La Figure 1 constitue un support de GCs, avec un ensemble de types de concepts et un ensemble de types de relations binaires avec leurs signatures. La relation d'ordre $\leq$, représentée par une flèche, s'interprète dans le sens de lecture de la flèche avec la signification "sorte-de". Par exemple, le concept Auto est une sorte-de Véhicule. On dit que Auto *est subsumé par* Véhicule, ou que Véhicule *subsume* Auto ; que Véhicule est le *subsumant*, et Auto le *subsumé*. Pour chaque type de relation, la signature σ est donnée. Ainsi, conduire a pour premier argument une Personne (ou tout subsumé) et pour second argument un Véhicule (ou tout subsumé).

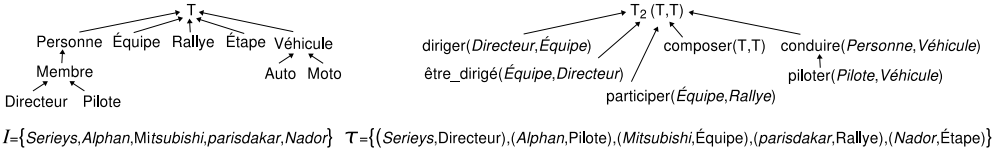

FIG. 1 – *Support*

Graphe simple[1]**.** *Un graphe conceptuel simple, défini sur un support S, est un multigraphe non orienté, biparti et étiqueté $G = (R, C, U, etiq)$ (Chein et Mugnier, 1992; Mugnier et Chein, 1996). R est l'ensemble de* sommets relations *et C l'ensemble de* sommets concepts.

[1]L'adjectif "simple" est employé ici par contraste avec des GCs plus complexes définis sur des supports plus riches, comme ceux utilisant les emboîtements, les types conjonctifs ou les types définis.

U est l'ensemble des arêtes. Les arêtes liées à un sommet relation sont étiquetées par une numérotation de 1 à l'arité de la relation. Le i^{ème} *voisin d'une relation r est noté $G_i(r)$. $etiq$ associe à chaque sommet une étiquette telle que $\forall r \in R, etiq(r) \in T_R$ et $\forall c \in C, etiq(c) = (type(c), ref(c))$. L'étiquette d'un sommet concept est formée d'un type $type(c) \in T_C$ et d'un référent $ref(c) \in I \cup \{*\}$. Un sommet concept tel que $ref(c) \in I$ est appelé* sommet concept individuel*, dans le cas contraire, il est appelé* sommet concept générique*. De plus, $etiq$ obéit aux contraintes fixées par σ et τ : $\forall r \in R, type(G_i(r)) \leq \sigma_i(type(r))$, et $\forall c \in C$ si $ref(c) \in I$ alors $\tau(ref(c)) \leq type(c)$.*

EXEMPLE. Le GC en Figure 2, basé sur le support en Figure 1, représente les connaissances factuelles suivantes. Le Directeur *Serieys* dirige l'Équipe *Mitsubishi*, qui participe au Rallye *Paris-Dakar*. Ce rallye est composé de l'Étape *Nador*. Le Pilote *Alphan* compose *Mitsubishi* et pilote une Auto. Remarquons que l'Auto n'est pas précisée, mais qu'elle existe (cf. marqueur générique).

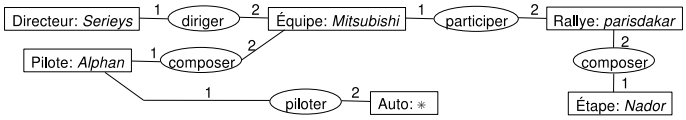

FIG. 2 – *Exemple de GC, défini sur le support en Fig. 1*

Sémantique logique. *À tout graphe conceptuel G et le support S sur lequel il est défini, est associée une formule logique du premier ordre $\Phi(S) \wedge \Phi(G)$ (Sowa, 1984; Mugnier et Chein, 1996), où $\Phi(S)$ est la formule associée à S et $\Phi(G)$ la formule associée à G, telles qu'elles sont définies en colonne "Sémantique" des Tableaux 1 et 2.*

	Syntaxe	**Sémantique**
type universel	$\top$	$\forall x\ \top(x)$
hiérarchie de concepts	$t_{c_1} \leq t_{c_2}$	$\forall x\ t_{c_1}(x) \rightarrow t_{c_2}(x)$
hiérarchie de relations	$t_{r_1} \leq t_{r_2}$	$\forall x_1, ..., x_p\ t_{r_1}(x_1, ..., x_p) \rightarrow t_{r_2}(x_1, ..., x_p)$
signature	$\sigma(t_r) = (t_{c_1}, ..., t_{c_n})$	$\forall x_1, ..., x_n\ t_r(x_1, ..., x_n) \rightarrow t_{c_1}(x_1) \wedge ... \wedge t_{c_n}(x_n)$
conformité	$\tau(m) = t$	$t(m)$

t_{c_i} sont des noms de types de concepts, t_{r_i} des noms de types de relations, et m un marqueur individuel.

TAB. 1 – *Syntaxe et sémantique logique du support*

	Syntaxe	**Sémantique**
SC individuel	$t_c : m$	$t_c(m)$, où m est la constante associée au SC individuel
SC générique	$t_c : *$	$\exists x\ t_c(x)$, où x est la variable associée au SC ; une variable distincte est associée à chaque SC générique.
SR unaire	t_r —1— C_1	$t_r(id_1)$, où id_1 est le terme associé au SC C_1
SR binaire	C_1 —1— t_r —2— C_2	$t_r(id_1, id_2)$, où id_1 et id_2 sont les termes associés aux SCs C_1 et C_2
SR *n*-aire	t_r —1— C_1 ; 2/ C_2 ... \n C_n	$t_r(id_1, ..., id_n)$, avec n l'arité de t_r et $id_1, ..., id_n$ les termes associés aux voisins du SR

(SC : sommet concept, SR : sommet relation)

TAB. 2 – *Syntaxe et sémantique logique d'un graphe*

EXEMPLE. Un extrait de la formule logique $\Phi(S)$ du support en Figure 1, et la formule logique $\Phi(G)$ du graphe en Figure 2 sont les suivantes :
$\Phi(S) =$ (∀x Pilote(x) → Membre(x)) ∧ (∀x Membre(x) → ⊤(x)) ∧ ... ∧ (∀x,y piloter(x,y) → conduire(x,y)) ∧ (∀x,y conduire(x,y) → $\top_2$(x,y)) ∧ ... ∧ (∀x,y piloter(x,y) → Pilote(x) ∧ Véhicule(y)) ∧ ...
$\Phi(G) =$ ∃x Directeur(Serieys) ∧ Équipe(Mitsubishi) ∧ Pilote(Alphan) ∧ Rallye(parisdakar) ∧ Étape(Nador) ∧ diriger(Serieys,Mitsubishi) ∧ participer(Mitsubishi,parisdakar) ∧ composer(Nador,parisdakar) ∧ composer(Alphan,Mitsubishi) ∧ piloter(Alphan,x) .

2.2 OWL, un langage ontologique pour le Web

Issu des travaux sur le Web Sémantique (Berners-Lee et al., 2001), OWL - pseudo-acronyme de *Web Ontology Language* - est un langage formel pour décrire des ontologies mais aussi des faits, et les diffuser sur le Web (Dean et al., 2004). OWL repose sur la syntaxe RDF (Klyne et Carroll, 2004), étend le vocabulaire de RDFS (Brickley et Guha, 2004), et possède une sémantique logique issue des logiques de descriptions. OWL étant une quasi-réécriture de certaines logiques de descriptions, nous commençons par en donner une courte introduction.

2.2.1 Courte introduction aux logiques de descriptions

Les logiques de descriptions (LDs) forment une famille de langages, structurés et formels, pour représenter des connaissances et raisonner avec (Baader et al., 2003). Ces logiques ont été conçues à partir des réseaux sémantiques (Lehmann, 1992) et des frames (Minsky, 1980). Une solide introduction aux logiques de descriptions peut être trouvée dans (Napoli, 1997).

Concrètement, les connaissances d'un domaine sont divisées en deux niveaux. D'un coté la *terminologie*, appelée T-Box, qui "décrit" des concepts et des rôles. Elle spécifie les connaissances structurelles. De l'autre, le niveau *assertionnel*, appelé A-Box, qui constitue les connaissances factuelles par un ensemble d'assertions. On appellera une *base de connaissances*, notée $\mathcal{K}$, un couple (T-Box, A-Box). Une sémantique est associée à $\mathcal{K}$ par une *interprétation*.

Terminologie. *Une terminologie, ou T-Box, est l'ensemble des formules qui décrivent les concepts et les rôles. L'introduction d'un concept se fait soit par une description (*$\sqsubseteq$*), on parle de* concept primitif, *soit par une définition (*$\equiv$*), on parle de* concept défini. *Un concept primitif est introduit en tant qu'une dérivation d'un concept ou d'une construction de plusieurs concepts, qui en est son subsumant. Il existe un concept de plus haut niveau, noté* $\top$*, et un de plus bas niveau, noté* $\bot$*. Une construction se fait au moyen de* constructeurs *qui agissent comme des opérateurs de combinaison de concepts, suivant la syntaxe donnée au Tableau 3. Un concept défini est introduit par une construction de concepts qui le définit. De même que pour les concepts, un rôle peut être introduit dans la terminologie par définition ou par description, avec des constructeurs propres aux rôles et* toprole *comme rôle de plus haut niveau.*

La signification d'une description se traduit par "sorte-de", et ordonne en hiérarchie les concepts et en hiérarchie les rôles. Remarquons d'une part qu'un rôle associé par construction à un concept primitif est une condition nécessaire pour qu'un individu soit une instance de ce concept primitif. D'autre part, l'ensemble des rôles associés à un concept défini forme les conditions nécessaires et suffisantes pour qu'un individu soit une instance de ce concept défini.

EXEMPLE. La T-Box en Figure 3 reprend d'une part la hiérarchie des concepts et la hiérarchie des relations (ici les rôles) en Figure 1, et d'autre part complète la spécification de ce niveau

structurel. Par exemple, le concept Moto est décrit comme étant disjoint de Auto ; le concept Directeur est défini comme une spécialisation de Membre qui dirige au moins une (cf. quantificateur existentiel) Équipe ; le rôle composer est transitif ; le rôle être_dirigé est l'inverse de diriger. Deux remarques peuvent être faites. La première est que d'après les descriptions et définitions des concepts, on peut déduire les domaines et co-domaines des rôles diriger, être_dirigé, piloter et participer. Par exemple en ligne (t13), diriger a pour domaine Directeur et pour co-domaine Équipe. Par contre, le domaine et co-domaine de composer n'ont qu'une portée locale, car (i) ce rôle est employé de deux manières différentes aux lignes (t12) pour lier Membre et Équipe et (t15) pour lier Étape et Rallye, et (ii) il n'existe ni de lien hiérarchique entre Membre et Équipe ni entre Étape et Rallye. Le domaine et le co-domaine de conduire ne peuvent être déduits. La description suivante aurait pu être ajoutée : Personne $\sqsubseteq \forall$conduire.Véhicule. La seconde remarque est qu'il n'y a pas qu'une façon de spécifier une T-Box. Par exemple, la ligne (t4) peut être remplacée par les deux descriptions : Moto $\sqsubseteq$ Véhicule puis Auto $\sqcap$ Moto $\sqsubseteq \bot$. La restriction de nombre qualifiée (≥ 2 composer$^-$.Membre) peut être plus ou moins remplacée par (≥ 2 composer$^-$)$\sqcap\exists$composer$^-$.Membre.

(t1) Personne $\sqsubseteq \top$	(t6) composer$^+ \sqsubseteq$ toprole
(t2) Véhicule $\sqsubseteq \top$	(t7) diriger $\sqsubseteq$ toprole
(t3) Auto $\sqsubseteq$ Véhicule	(t8) participer $\sqsubseteq$ toprole
(t4) Moto $\sqsubseteq$ Véhicule $\sqcap \neg$Auto	(t9) conduire $\sqsubseteq$ toprole
(t5) Étape $\sqsubseteq \top$	(t10) piloter $\sqsubseteq$ conduire
(t12) Membre $\equiv$ Personne $\sqcap \exists$composer.Équipe	
(t13) Directeur $\equiv$ Membre $\sqcap \exists$diriger.Équipe	(t11) être_dirigé $\equiv$ diriger$^-$
(t14) Pilote $\equiv$ Membre $\sqcap \exists$piloter.Véhicule	
(t15) Rallye $\equiv \exists$composer$^-$.Étape	
(t16) Équipe $\equiv \forall$participer.Rallye $\sqcap \exists$être_dirigé.Directeur $\sqcap$ (≥ 2 composer$^-$.Membre)	

FIG. 3 – *T-Box, reprenant les hiérarchies des types de concepts et des types de relations Fig.1*

A-Box. *Une A-Box est l'ensemble des formules qui décrivent les* assertions de concepts, *appelés individus, et les* assertions de rôles, *suivant la syntaxe donnée au Tableau 4.*

EXEMPLE. La A-Box en Figure 4 est définie sur la T-Box en Figure 3, et reprend les faits du GC en Figure 2. Remarquons l'instance particulière pajero du concept Voiture par rapport au GC.

(a1) Pilote(Alphan)	(a6) piloter(Alphan,pajero)
(a2) Auto(pajero)	(a7) composer(Alphan,Mitsubishi)
(a3) Équipe(Mitsubishi)	(a8) diriger(Serieys,Mitsubishi)
(a4) Rallye(parisdakar)	(a9) participer(Mitsubishi,parisdakar)
(a5) Étape(Nador)	(a10) composer(Nador,parisdakar)

FIG. 4 – *A-Box, définie suivant la T-Box en Fig. 3, et reprenant les faits en Fig. 2*

Interprétation. *Une interprétation $\mathcal{I} = (\Delta_\mathcal{I}, .^\mathcal{I})$ consiste en un ensemble non vide $\Delta_\mathcal{I}$, appelé* domaine de l'interprétation, *et une* fonction d'interprétation $.^\mathcal{I}$ *qui fait correspondre à un concept un sous-ensemble de $\Delta_\mathcal{I}$, à un rôle un sous-ensemble de $\Delta_\mathcal{I} \times \Delta_\mathcal{I}$, et à un individu un élément de $\Delta_\mathcal{I}$ et à une assertion de rôle un élément de $\Delta_\mathcal{I} \times \Delta_\mathcal{I}$, de telle sorte que les expressions en colonne "Sémantique" des Tableaux 3 et 4 soient satisfaites.*

Modèle. *Une interprétation $\mathcal{I}$ est dite un* modèle *pour une T-Box $\mathcal{T}$ si et seulement si $\mathcal{I}$ satisfait toutes les expressions de $\mathcal{T}$. Pour une A-Box $\mathcal{A}$, $\mathcal{I}$ est un* modèle *si et seulement si $\mathcal{I}$ satisfait toutes les assertions de $\mathcal{A}$. $\mathcal{I}$ est un* modèle *pour $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ si et seulement si $\mathcal{I}$ est un modèle pour $\mathcal{T}$ et $\mathcal{A}$ à la fois.*

Constructeur (lettre associée)		**Syntaxe**	**Sémantique**
		$\top$	$\top^{\mathcal{I}} = \Delta_{\mathcal{I}}$
		$\bot$	$\bot^{\mathcal{I}} = \emptyset$
	hiérarchie de concepts	$C_1 \sqsubseteq C_2$	$(C_1 \sqsubseteq C_2)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid x \in C_1^{\mathcal{I}} \rightarrow x \in C_2^{\mathcal{I}}\}$
	conjonction	$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid x \in C_1^{\mathcal{I}} \wedge x \in C_2^{\mathcal{I}}\}$
	quantificateur universel	$\forall r.C$	$(\forall r.C)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid \forall y \in \Delta_{\mathcal{I}}, (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
$\mathcal{E}$	quantificateur existentiel	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid \exists y \in \Delta_{\mathcal{I}}, (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
$\mathcal{C}$	négation	$\neg C$	$(\neg C)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid x \notin C^{\mathcal{I}}\}$
$\mathcal{U}$	disjonction	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid x \in C_1^{\mathcal{I}} \vee x \in C_2^{\mathcal{I}}\}$
$\mathcal{N}$	restriction de nombre[2]	$\geq n\ r$	$(\geq n\ r)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid \text{card}(\{y \in \Delta_{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\}) \geq n\}$
	(au-plus et au-moins)	$\leq n\ r$	$(\leq n\ r)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid \text{card}(\{y \in \Delta_{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\}) \leq n\}$
$\mathcal{O}$	type énuméré	$\{a_1, ..., a_n\}$	$\{a_1, ..., a_n\}^{\mathcal{I}} = \{a_1^{\mathcal{I}}, ..., a_n^{\mathcal{I}}\}$
$\mathcal{H}$	*hiérarchie de rôles*	$r_1 \sqsubseteq r_2$	$(r_1 \sqsubseteq r_2)^{\mathcal{I}} = \{(x,y) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}} \mid (x,y) \in r_1^{\mathcal{I}} \rightarrow (x,y) \in r_2^{\mathcal{I}}\}$
$\mathcal{R}$	conjonction de rôles	$r_1 \sqcap r_2$	$(r_1 \sqcap r_2)^{\mathcal{I}} = \{(x,y) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}} \mid (x,y) \in r_1^{\mathcal{I}} \wedge (x,y) \in r_2^{\mathcal{I}}\}$
$\mathcal{I}$	rôle inverse	r^-	$(r^-)^{\mathcal{I}} = \{(y,x) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} = (r^{\mathcal{I}})^-$
$\mathcal{R}^+$	transitivité des rôles	r^+	$(r^+)^{\mathcal{I}} = \{(x,z) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}} \mid \forall y \in \Delta_{\mathcal{I}}, (x,y) \in r^{\mathcal{I}} \wedge (y,z) \in r^{\mathcal{I}} \rightarrow (x,z) \in r^{\mathcal{I}}\}$

C, C_1 et C_2 sont des noms de concepts, r, r_1 et r_2 sont des noms de rôles, et a_1 à a_n sont des individus.

TAB. 3 – *Les constructeurs du langage $\mathcal{AL}$ et de ses extensions les plus courantes*

Assertion	**Syntaxe**	**Sémantique**
inclusion d'assertion de concept (d'individu)	$C(a)$	$C(a)^{\mathcal{I}} = a^{\mathcal{I}} \in C^{\mathcal{I}}$
inclusion d'assertion de rôle	$r(a,b)$	$r(a,b)^{\mathcal{I}} = (a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$
égalité d'individus	$a = b$	$(a{=}b)^{\mathcal{I}} = (a^{\mathcal{I}}{=}b^{\mathcal{I}})$
inégalité d'individus	$a \neq b$	$(a{\neq}b)^{\mathcal{I}} = (a^{\mathcal{I}}{\neq}b^{\mathcal{I}})$

C est un nom de concept, r est un nom de rôle, et a ainsi que b identifient des individus.

TAB. 4 – *Les assertions de concepts (individus) et les assertions de rôles en LDs.*

Les langages des LDs se distinguent les uns des autres par l'ensemble des constructeurs qu'ils offrent.

L'une des premières LD est le langage $\mathcal{FL}^-$ (Brachman et Levesque, 1984)[3], qui offre les constructeurs de conjonction ($\sqcap$), de quantificateur universel ($\forall$), de quantificateur existentiel non typé ($\exists r$, c-à-d $\exists r.\top$), et la restriction du co-domaine d'un rôle ($r|C$)[4]. Le langage $\mathcal{FL}$ est une extension de $\mathcal{FL}^-$ où le quantifieur existentiel est typé ($\exists r.C$). Le langage $\mathcal{AL} = \{\top, \bot, C_1 \sqcap C_2, \forall r.C, \exists r, \neg A\}$ (Schmidt-Schauss et Smolka, 1991) étend le langage $\mathcal{FL}^-$ en y ajoutant la négation, sur des concepts primitifs uniquement. Ce langage peut-être considéré comme la logique de base des autres LDs. En effet, les LDs sont des combinaisons des différents éléments du Tableau 3. Par exemple si on ajoute la négation complète (étendue aux concepts définis), repérée par la lettre $\mathcal{C}$, à la logique $\mathcal{AL}$ on obtient la logique $\mathcal{ALC}$. On peut remarquer que le langage $\mathcal{ALC}$ équivaut à $\mathcal{ALUE}$, puisque l'union ($\mathcal{U}$) et la quantification existentielle typée ($\mathcal{E}$) peuvent s'exprimer par la négation complète et inversement : $C_1 \sqcup C_2$ correspond à $\neg(\neg C_1 \sqcap \neg C_2)$ et $\exists r.C$ correspond à $\neg\forall r.\neg C$ (Baader et al., 2003).

[2]La restriction de nombre peut être qualifiée (repérée par la lettre $\mathcal{Q}$), avec $(\geq n\ r.C)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid \text{card}(\{y \in \Delta_{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}) \geq n\}$ et $(\leq n\ r.C)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid \text{card}(\{y \in \Delta_{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}) \leq n\}$.

[3]Le langage $\mathcal{FL}^-$ (Frame Logic) a été proposé comme un formalisme pour la sémantique des *frames* de Minsky, et pour lequel ont été établi des résultats théoriques de complexité sur le subsomption par Brachman et Levesque.

[4]Ce constructeur peut être rapproché du constructeur existentiel typé ($\exists r.C$), mais avec deux différences : il s'applique aux rôles et n'a aucune contrainte d'existence ; $(r|C)^{\mathcal{I}} = \{(x,y) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$.

2.2.2 Le langage OWL

Le langage OWL est en réalité divisé en trois sous-langages d'expressivité croissante : *OWL Lite*, *OWL DL* et *OWL Full*, de sorte que tout document OWL Lite soit aussi un document OWL DL, et que tout document OWL DL soit un document OWL Full. OWL Lite permet principalement de modéliser en hiérarchies les concepts et les rôles, et possède des mécanismes simples de construction de modélisation. Par exemple, seules les cardinalités 0 ou 1 sont gérées par OWL Lite. OWL DL comprend toutes les structures du langage OWL, avec certaines restrictions d'utilisation, de telle sorte qu'il y ait le meilleur compromis entre expressivité et efficacité. On entend par *efficacité*, d'une part la complétude du calcul, c'est-à-dire toutes les inférences sont garanties calculables, et d'autre part la décidabilité, c'est-à-dire tous les calculs s'achèveront dans un intervalle de temps fini. OWL DL tire son appellation DL du fait que ce sous-langage possède une sémantique issue des LDs (Description Logics). OWL Full offre une expressivité maximale et la liberté syntaxique de RDF. Par exemple, une classe peut être une instance d'une autre classe. En contrepartie, l'efficacité de calcul n'est pas garantie.

Rappelons que OWL est syntaxiquement basé sur RDF, qui lui-même est historiquement bâti sur XML (Bray et al., 2004) Cette remarque éclaire sur le fait que OWL permet de décrire, en plus des concepts et des rôles issus des LDs, des types de données ou *datatype* en anglais. Un datatype regroupe un ensemble de valeurs, au même titre qu'un concept regroupe un ensemble d'individus. Ainsi, un datatype appartient au niveau structurel d'une modélisation tandis qu'une valeur (d'un datatype) appartient au niveau factuel.

Sémantique logique. *À un document OWL est associée une formule logique dans le langage $\mathcal{SHIF}$(D)*[5] *pour OWL Lite et dans le langage $\mathcal{SHOIN}$(D) pour OWL DL (Horrocks et Patel-Schneider, 2003). Une* classe *en OWL correspond à un* concept *en LD, une* propriété concept *en OWL correspond à un* rôle *en DL. Les* datatypes *et* propriétés dadatypes *sont nouvellement introduits ; une propriété datatype lie une classe à un datatype. L'interprétation $\mathcal{I}_{OWL}$, qui associe une sémantique à un document OWL, étend l'interprétation $\mathcal{I}$ définie en Section 2.2.1, de façon à prendre en compte les datatypes. L'interprétation $\mathcal{I}_{OWL} = (\Delta_\mathcal{I}, \Delta_D, .^\mathcal{I})$ consiste en un ensemble non vide $\Delta_\mathcal{I}$, appelé cette fois* domaine de l'interprétation des instances, *en un ensemble Δ_D disjoint de $\Delta_\mathcal{I}$, appelé* domaine de l'interprétation des datatypes, *et une* fonction d'interprétation[6] *$.^\mathcal{I}$ qui fait correspondre : à une classe un sous-ensemble de $\Delta_\mathcal{I}$, à une propriété concept un sous-ensemble de $\Delta_\mathcal{I} \times \Delta_\mathcal{I}$, à un individu un élément de $\Delta_\mathcal{I}$, à un datatype un sous-ensemble Δ_D, à une propriété datatype un sous-ensemble de $\Delta_\mathcal{I} \times \Delta_D$, et à une valeur un élément de Δ_D, de telle sorte que les expressions en colonne "Sémantique" des Tableaux 3, 4 et 5 soient satisfaites.*

EXEMPLE. La Figure 5 donne les définitions du concept Directeur, du rôle être_dirigé, et les individus Serieys et Mitsubishi avec la syntaxe OWL. Le domaine et le co-domaine du rôle diriger sont ici explicitement donnés.

[5] $\mathcal{S}$ désigne le langage $\mathcal{ALCR}^+$, $\mathcal{F}$ la restriction de nombre fonctionnelle (0 ou 1), et (D) fait référence à *datatype*.

[6] Certains auteurs dissocient cette fonction d'interprétation en deux sous-fonctions : $.^\mathcal{I}$ pour interpréter les concepts, les propriétés concepts et les individus, et $.^D$ pour interpréter les datatypes, les propriétés datatypes et les valeurs.

Constructeur	Syntaxe	Sémantique
hiérarchie de dadatypes	$D_1 \sqsubseteq D_2$	$(D_1 \sqsubseteq D_2)^{\mathcal{I}} = \{d \in \Delta_D \mid d \in D_1^{\mathcal{I}} \rightarrow d \in D_2^{\mathcal{I}}\}$
quantificateur universel	$\forall u.D$	$(\forall u.D)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid \forall d \in \Delta_D, (x,d) \in u^{\mathcal{I}} \rightarrow d \in D^{\mathcal{I}}\}$
quantificateur existentiel	$\exists u.D$	$(\exists u.D)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid \exists d \in \Delta_D, (x,d) \in u^{\mathcal{I}} \wedge d \in D^{\mathcal{I}}\}$
restrictions de nombre	$\geq n\ u$	$(\geq n\ u)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid \text{card}(\{d \in \Delta_D \mid (x,d) \in u^{\mathcal{I}}\}) \geq n\}$
(au-plus et au-moins)	$\leq n\ u$	$(\leq n\ u)^{\mathcal{I}} = \{x \in \Delta_{\mathcal{I}} \mid \text{card}(\{d \in \Delta_D \mid (x,d) \in u^{\mathcal{I}}\}) \leq n\}$
type énuméré	$\{v_1, ..., v_n\}$	$\{v_1, ..., v_n\}^{\mathcal{I}} = \{v_1^{\mathcal{I}}, ..., v_n^{\mathcal{I}}\}$
Axiome	**Syntaxe**	**Sémantique**
définition de valeur	$D(v)$	$D(v)^{\mathcal{I}} = v^{\mathcal{I}} \in D^{\mathcal{I}}$
affectation de valeur	$u(a,v)$	$u(a,v)^{\mathcal{I}} = (a^{\mathcal{I}}, v^{\mathcal{I}}) \in u^{\mathcal{I}}$

D, D_1 et D_2 sont des datatypes, u, u_1 et u_2 des propriétés datatypes, a un individu, et v, v_i des valeurs.

TAB. 5 – *Les datatypes et leurs valeurs en OWL*

```
<owl:Class rdf:ID="Directeur">
 <rdfs:subClassOf rdf:ID="Membre">
  <owl:Restriction>
  <owl:onProperty rdf:resource="#diriger"/>
   <owl:mincardinality rdf:datatype
     ="&xsd;nonNegativeInteger"> 1
   </owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="être_dirigé">
 <owl:inverseOf rdf:resource="#diriger" />
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="diriger">
 <rdfs:domain rdf:resource="#Directeur"/>
 <rdfs:range rdf:resource="#Équipe" />
</owl:ObjectProperty>


<Directeur rdf:ID="Serieys">
 <diriger rdf:resource="#Mitsubishi" />
</Directeur>

<Équipe rdf:ID="Mitsubishi">
 <être_dirigé rdf:resource="#Serieys" />
 <participer rdf:resource="#parisdakar"/>
</Équipe>
```

FIG. 5 – *Syntaxe OWL - extraits de la T-Box en Fig. 3 et de la A-Box en Fig. 4*

2.3 GCs *versus* OWL, aspect représentation.

Modélisation intensionnelle VS extensionnelle. La modélisation avec le modèle des GCs se fait suivant une démarche *intensionnelle* de capitalisation des connaissances, tandis qu'elle est généralement *extensionnelle* avec OWL. Une démarche intensionnelle de modélisation consiste donc à définir des classes et des relations, puis à instancier des individus et les lier entre eux. Une démarche extensionnelle de modélisation consiste à s'intéresser d'abord aux individus, puis ensuite à les regrouper selon des caractéristiques communes et ainsi définir des classes. On entend aussi par cette démarche extensionnelle le fait de décrire de façon "éparpillée" des individus, des concepts et des rôles, sans une véritable vue globale et organisée de l'ensemble des descriptions ; l'idée étant d'avoir une grande souplesse de modélisation, où des *raisonneurs* réorganiseront le tout. Rappelons que la définition d'une classe est appelée l'*intension* de la classe, et que les individus d'une classe en constituent l'*extension*.

Monde fermé VS ouvert, et UNA. Le choix de faire l'hypothèse d'un *monde fermé* (CWA) ou d'un *monde ouvert* (OWA) pour les raisonnements influence sur la façon de modéliser les connaissances, de même que pour la supposition ou non du *nom unique* (UNA), notament dans la possibilité ou non de représenter des connaissances incomplètes. Les raisonnements avec le modèle des GCs se fait sous l'hypothèse d'un monde fermé et présuppose du nom unique, tandis qu'avec OWL ils sont sous l'hypothèse d'un monde ouvert sans supposition du nom unique. OWL a donc pour vocation de traiter des informations incomplètes. Sous l'hypothèse du monde fermé - Closed World Assumption -, toute connaissance non explicitement déduite est considérée comme fausse, alors que sous l'hypothèse du monde ouvert - Open World As-

sumption -, ce qui est omis reste inconnu. La supposition du nom unique - Unique Name Assumption - associe à une entité (concept, relation ou individu) un symbole unique, c'est-à-dire que deux noms différents représentent des entités différentes.
Bien que OWL travaille sous l'hypothèse du monde ouvert et ne présuppose pas du nom unique, il est souvent nécessaire et légitime de pouvoir (i) distinguer les individus, et (ii) "fermer" le monde ; ce qui est d'ailleurs fait implicitement par certains raisonneurs. Par exemple, lorsque l'on cherche des individus ayant certains caractéristiques, on souhaite du système qu'il considère ces individus exactement et les retourne. Pour cela, le langage OWL dispose d'une part des propriétés owl:differentFrom et owl:sameAs pour différencier ou non les individus. D'autre part, il est possible en OWL de fermer explicitement le monde en utilisant la propriété owl:oneOf d'une façon particulière. Cette propriété permet de définir des classes, dites énumérées, en fournissant la liste exhaustive de leurs individus, comme par exemple la classe JourDeLaSemaine. Pour fermer le monde, il suffit donc de déclarer la classe owl:Thing comme une énumération de tous les individus du domaine.
Connaissances structurelles. Le modèle des GCs et le langage OWL permettent tous deux de représenter les connaissances d'un domaine sur deux niveaux conceptuels : le niveau structurel et le niveau factuel. Notons que si les connaissances structurelles d'une modélisation en OWL peuvent constituer une ontologie à part entière, le support des GCs est moins expressif. En effet, seul le lien de subsomption organise les concepts et les relations du support. Le support constitue donc une *taxonomie* du domaine. Cependant, en Section 3.3 est indiqué comment décrire de manière plus riche les connaissances structurelles avec le modèle des GCs.
Atelier de modélisation. La facilité d'utilisation des modèles pour manipuler des connaissances fait partie intégrante d'une approche de modélisation. En effet pour un opérateur humain, cet aspect atelier convivial de modélisation est essentielle, et passe par une visualisation des connaissances sous une forme intuitive. En effet, si la visualisation des connaissances peut sembler peu importante pour la machine, elle est essentielle pour l'opérateur humain.
Le modèle des GCs offre une représentation des connaissances sous forme de graphes, basés sur un support. L'utilisation de *relations n-aires* ou d'*emboîtements* dans les graphes, ne rend certes pas le modèle plus expressif (ni moins efficace), mais offrent une vision des graphes qui en facilite grandement la compréhension. Les GCs emboîtés (Chein et Mugnier, 1997; Chein et al., 1998) constituent une extension du modèle de base des GCs. Concrètement, un GC peut être défini à l'intérieur d'un sommet concept d'un autre graphe. Un emboîtement est une description partielle et interne d'un sommet concept, où il est possible de "zoomer".
Le langage OWL, textuel par nature, dispose de nombreux outils conviviaux pour visualiser les connaissances. Citons l'éditeur d'ontologies OilEd (http://oiled.man.ac.uk/), l'outil de visualisation de graphes RDF Isaviz (http://www.w3.org/2001/11/IsaViz/), et l'incontournable éditeur Protégé avec le plugin OWL, le plugin Jambalaya (Storey et al., 2001) ou le plugin OntoRama (Eklund et al., 2000) pour visualiser le graphe RDF. Le *graphe RDF* correspond aux triplets RDF, et offre une représentation d'un document OWL sous forme d'un graphe. Remarquons que dans un graphe RDF les connaissances structurelles et factuelles se mélangent. Ainsi, OWL permet de manipuler conjointement l'ontologie et les faits, car les connaissances structurelles et factuelles ont la même forme syntaxique (triplets RDF). Cette manipulation conjointe est moins évidente dans le modèle de GCs car le support et les graphes se retrouvent sous deux formes bien distinctes. Cependant, cette distinction permet d'une part de bénéficier d'algorithmes spécifiques sur le support et d'autres sur les GCs, et d'autre part de rendre plus lisible la visualisation entre connaissances structurelles et factuelles.

3 Manipulation des connaissances

3.1 Manipuler des connaissances avec les GCs

3.1.1 Opération de base dans le modèle des GCs

Projection. *Une projection (Chein et Mugnier, 1992; Mugnier et Chein, 1996) d'un graphe conceptuel $H = (R_H, C_H, U_H, etiq_H)$ dans un graphe conceptuel $G = (R_G, C_G, U_G, etiq_G)$ est un couple d'applications $\Pi = (f, g)$, avec $f : R_H \rightarrow R_G$ et $g : C_H \rightarrow C_G$, tel que :*
(1) Π *peut restreindre les étiquettes des sommets (pour chaque sommet t de H, $etiq_H(t)$ est identique ou plus générique que $etiq_G(\Pi(t))$), et*
(2) Π *préserve les arêtes et leurs étiquettes (si il existe une arête entre deux sommets t_1 et t_2 dans H, alors il doit exister une arête dans G avec la même étiquette entre $\Pi(t_1)$ et $\Pi(t_2)$).*

La *projection* est l'opération de base du modèle des GCs. Il s'agit concrètement d'un homomorphisme de graphes (Chein et Mugnier, 1992). La projection est adéquate et complète vis à vis de la déduction en LPO (Sowa, 1984; Chein et Mugnier, 1992; Wermelinger, 1995)[7] : H se projette dans $G \iff \Phi(S), \Phi(G) \models \Phi(H)$.

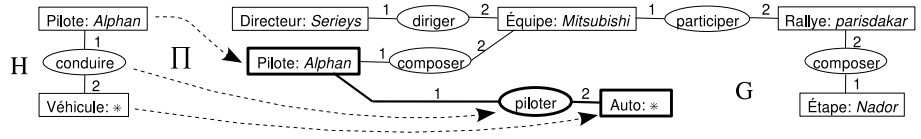

FIG. 6 – *Projection Π du graphe H dans le graphe G ; H et G définis sur le support en Fig. 1*

EXEMPLE. La Figure 6 illustre la projection Π du graphe H dans le graphe G, qui fait correspondre à H le sous-graphe de G mis en gras (H et G tous deux définis sur le support en Figure 1). On notera que la projection fait correspondre l'individu *Alphan* avec exactement l'individu *Alphan*, l'existence d'un individu de type Véhicule avec l'existence d'un individu du type subsumé Auto, et la relation conduire avec la relation subsumée piloter où les étiquettes des arêtes sont respectées.

3.1.2 Opérations et outils en GCs pour manipuler les connaissances

Interroger des connaissances. La définition et l'utilisation d'une *requête* pour interroger des connaissances modélisées dans un GC est une utilisation immédiate de la projection. Par exemple, le GC H en Figure 6 représente la requête "Le pilote Alphan conduit-il un véhicule ?" ; la projection Π constitue la réponse "Le pilote Alphan pilote une auto".

Requête. *Soit G un graphe conceptuel défini sur un support S. Une* requête *pour interroger G s'exprime par un graphe conceptuel Q défini sur ce même support S. Le résultat de Q sur G est l'ensemble (qui peut être vide) des projections de Q dans G.*

Déduire des connaissances. Une *règle* permet de déduire de nouvelles connaissances et de les ajouter à un GC. Elle est de la forme : "si G_1 alors G_2 ", où G_1 et G_2 sont des GCs. L'ap-

[7] En prenant pour modèle des GCs celui présenté en Section 2.1, et les graphes G et H doivent être définis sur un même support et mis sous une forme particulière appelée *forme normale* (Mugnier et Chein, 1996) .

plication de règles offre la possibilité d'exprimer explicitement des connaissances factuelles implicites présentes dans un GC.

Règle et fermeture. *Une règle (Salvat, 1998; Baget et Mugnier, 2002) est un GC bicolore, composé d'une* hypothèse *et d'une* conclusion*. L'hypothèse est formée de l'ensemble des sommets 0-colorés, et la conclusion des sommets 1-colorés. Une règle R est dite* applicable *sur un GC G si il existe une projection Π de l'hypothèse de R dans G. Dans ce cas, le résultat de l'application de R dans G selon Π est le GC $G_{R,\Pi}$ obtenu à partir de G auquel on a ajouté la conclusion de R et les arêtes adéquates qui lient cette conclusion au graphe initial. La* fermeture *d'un GC G par un ensemble de règles $\mathfrak{R}$ est obtenue par application des règles de $\mathfrak{R}$ sur G jusqu'à ce que toute information qui est ajoutée par une règle soit déjà présente dans G.*

EXEMPLE. La Figure 7 présente deux règles. Suivant la notation dans (Baget et al., 1999), les sommets 0-colorés sont sur fond blanc et les sommets 1-colorés sur fond noir. Une règle est repérée par $\boxed{\Rightarrow}$. La règle R_1 permet de déduire que "Si un Directeur dirige une Équipe, alors cette Équipe est dirigée par ce Directeur. En d'autres termes, la relation être_dirigé est l'inverse de la relation diriger. La règle R_2 exprime la transitivité de la relation composer.

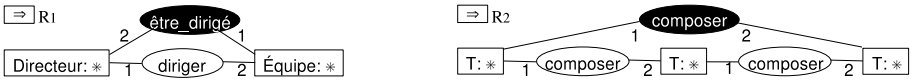

FIG. 7 – *Règles : être_dirigé inverse de diriger (R_1), transitivité de composer (R_2)*

Vérifier des connaissances. Les *contraintes* permettent de vérifier la cohérence sémantique d'une modélisation au cours de son cycle de vie. Elles peuvent être positives ou négatives. Une contrainte positive est de la forme "Si *condition*, il est nécessaire que *obligation*". Une contrainte négative est de la forme "Si *condition*, il ne faut pas que *interdiction*".

Contraintes. *Les contraintes (Baget et al., 1999; Baget et Mugnier, 2002) sont des GCs bicolores. Une* contrainte positive *est composée d'une* condition *et d'une* obligation*. Une* contrainte négative *est composée d'une* condition *et d'une* interdiction*. La condition d'une contrainte est formée de l'ensemble des sommets 0-colorés, l'obligation ou l'interdiction est formée de l'ensemble des sommets 1-colorés. Un GC G* satisfait *une contrainte positive, si toute projection de la condition de C dans G peut être étendue à une projection de C (dans sa globalité) dans G. G* satisfait *une contrainte négative, si aucune projection de la condition de C dans G ne peut être étendue à une projection de C (dans sa globalité) dans G.*

EXEMPLE. La Figure 8 présente deux contraintes positives et une négative. Une contrainte positive est repérée par $\boxed{+}$, une contrainte négative par $\boxed{-}$. La contrainte positive C_1 oblige que tout Directeur doit diriger (au moins) une Équipe. La contrainte positive C_2 oblige que toutes les Équipes doivent participer à tous les Rallyes. La contrainte négative C_3 interdit qu'un Pilote puisse piloter une Auto et une Moto. Notons qu'une contrainte négative peut être ramenée à "Il ne faut pas *interdiction*", en incluant la condition dans l'interdiction (Baget et Mugnier, 2002).

Outils pour les GCs. Nous présentons quelques outils pour la manipulation et l'édition de GCs. Cogitant (http://cogitant.sourceforge.net/) est une bibliothèque de classes C++, pour le développement de logiciels basés sur les GCs. Cette bibliothèque permet de manipuler des GCs

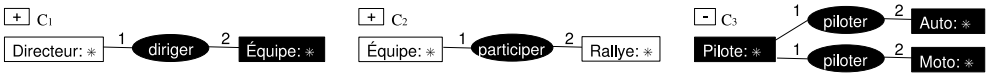

FIG. 8 – *Contraintes : deux contraintes positives, une contrainte négative*

emboîtés typés avec liens de coréférence, de vérifier qu'un graphe conceptuel est correctement construit, d'utiliser la projection, d'appliquer des règles et de vérifier des contraintes. Cogitant intègre aussi un éditeur minimal de GCs emboîtés typés. Amine (Kabbaj, 2006) est une plateforme Java pour le développement de systèmes basés sur les GCs. Notio (Southey et Linders, 1999) est une bibliothèque Java pour la manipulation de GCs. CoGUI (Gutierez, 2005) est une interface graphique en Java, pour l'édition de GCs, de règles et de contraintes, et offre une connexion avec la bibliothèque Cogitant. CharGer (Delugach, 2001) est un éditeur Java de GCs pour la construction de GCs, et peut utiliser conjointement Notio.

3.2 Manipuler des connaissances avec OWL

Les LDs permettent de faire l'hypothèse du monde ouvert, où les connaissances d'un domaine peuvent à tout moment être complétées. Cette souplesse de modélisation nécessite de vérifier que l'ensemble des descriptions localement structurées et "éparpillées" ont un sens dans leur globalité. La *classification* et l'*instanciation* des connaissances sont à la base des raisonnements en LDs pour répondre à ce besoin de cohérence, on parle d'*inférence*. Le langage OWL étant basé sur les LDs, il bénéficie de ces raisonnements.

Classification et Instanciation. *La classification est un processus qui consiste à placer un concept ou un rôle dans leurs hiérarchies respectives, on parle de* classification de concepts *et de* classification de rôles*. L'instanciation est un processus qui consiste à déterminer les concepts (notament le plus spécifique) dont un individu donné peut être une instance.*

3.2.1 Opérations de bases en LDs et outils

Les quatre principales opérations d'inférence pour la classification et l'instanciation sont le *test de subsomption*, le *test de satisfiabilité de concept*, le *test d'instanciation* et le *test de satisfiabilité d'une base de connaissances*. Des deux premiers tests dérivent le *test d'équivalence de concepts* et le *test de disjonction de concepts* (voir Figure 10).

Test de subsomption. *Pour une base de connaissances $\mathcal{K}$ = (T-Box,A-Box), un concept C_1 (resp. un rôle r_1) subsume un concept C_2 (resp. un rôle r_2) par rapport à $\mathcal{K}$, si et seulement si pour tout modèle $\mathcal{I}$ de $\mathcal{K}$, $C_2^{\mathcal{I}} \subseteq C_1^{\mathcal{I}}$ (resp. $r_2^{\mathcal{I}} \subseteq r_1^{\mathcal{I}}$).*

Test de satisfiabilité (de concept). *Pour $\mathcal{K}$ = (T-Box,A-Box), un concept C est* satisfiable *si et seulement si il existe au moins un modèle $\mathcal{I}$ de $\mathcal{K}$ tel que $C^{\mathcal{I}} \neq \emptyset$, c-à-d C admet des instances.*

Test d'instanciation. *Pour $\mathcal{K}$ = (T-Box,A-Box), une assertion de concept a est instance d'un concept C (notament pour C le plus spécifique), notée $\mathcal{K} \models C(a)$, si et seulement si pour tout modèle $\mathcal{I}$ de $\mathcal{K}$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$. On dit que $C(a)$ est* satisfiable *par $\mathcal{I}$.*

Test de satisfiabilité (de $\mathcal{K}$). *$\mathcal{K}$ = (T-Box,A-Box) est satisfiable si et seulement si il existe au moins un modèle $\mathcal{I}$ tel que $\mathcal{I}$ satisfasse tous les concepts et toutes les assertions de $\mathcal{K}$.*

EXEMPLE. Après avoir distingué les individus deux à deux et "fermé" le monde, puis classifier et instancier le couple (T-Box,A-box) en Figures 3 et 4, il n'existe pas d'interprétation qui satisfasse ce couple. En effet, l'équipe Mitsubishi n'est composée que d'un membre, Alphan. Mais si l'on précise que le rôle diriger est une spécialisation de composer, en modifiant la ligne (t7) en Figure 3 par diriger $\sqsubseteq$ composer, alors ce couple (T-Box,A-box) est satisfiable : la Figure 9 présente un modèle pour cette T-Box et cette A-Box. Les éléments en gras ont été déduits par les opérations de classification et d'instanciation. En effet, les opérations d'inférence au niveau terminologique ont inféré que Membre $\sqsubseteq$ Personne, que Directeur $\sqsubseteq$ Membre et que Pilote $\sqsubseteq$ Membre (cf. (t12) à (t14)). Les opérations d'inférence au niveau assertionnel ont inféré : être_dirigé(Mitsubishi, Serieys), Directeur(Serieys) et composer(Serieys,Mitsubishi) puisque diriger(Serieys,Mitsubishi) (cf. (a8)).

$\Delta_{\mathcal{I}}$ = {Serieys, Alphan, Mitsubishi, pajero, parisdakar, Nador}	
Serieys$^{\mathcal{I}}$ = Serieys	Pilote$^{\mathcal{I}}$ = {Alphan}
Alphan$^{\mathcal{I}}$ = Alphan	Directeur$^{\mathcal{I}}$ = {**Serieys**}
Mitsubishi$^{\mathcal{I}}$ = Mitsubishi	Membre$^{\mathcal{I}}$ = {**Serieys, Alphan**}
pajero$^{\mathcal{I}}$ = pajero	Personne$^{\mathcal{I}}$ = {**Serieys, Alphan**}
parisdakar$^{\mathcal{I}}$ = parisdakar	Auto$^{\mathcal{I}}$ = {pajero}
Nador$^{\mathcal{I}}$ = Nador	Moto$^{\mathcal{I}}$ = $\varnothing$
participer$^{\mathcal{I}}$ = {(Mitsubishi,parisdakar)}	Véhicule$^{\mathcal{I}}$ = {**pajero**}
piloter$^{\mathcal{I}}$ = {(Alphan,pajero)}	Équipe$^{\mathcal{I}}$ = {Mitsubishi}
conduire$^{\mathcal{I}}$ = {**(Alphan,pajero)**}	Rallye$^{\mathcal{I}}$ = {parisdakar}
diriger$^{\mathcal{I}}$ = {(Serieys,Mitsubishi)}	Étape$^{\mathcal{I}}$ = {Nador}
être_dirigé$^{\mathcal{I}}$ = {**(Mitsubishi,Serieys)**}	
composer$^{\mathcal{I}}$ = {**(Serieys,Mitsubishi)**, (Alphan,Mitsubishi), (Nador,parisdakar)}	

FIG. 9 – *Un modèle $\mathcal{I}$ pour la base de connaissances (T-Box,A-Box) des Fig. 3 et 4*

Remarquons que ces tests à la base des raisonnements en LDs sont dépendants les uns des autres. La Figure 10 présente les dépendances entres ces tests.

Il est possible d'exprimer, avec le test de subsomption :	**avec le test de satisfiabilité :**	**avec le test d'instanciation :**
C est satisfiable $\Longleftrightarrow C \not\sqsubseteq \bot$	$C_1 \sqsubseteq C_2 \Longleftrightarrow C_1 \sqcap \neg C_2$ insatisfiable	$\mathcal{K}$ est satisfiable $\Longleftrightarrow \mathcal{K} \not\models \bot(a)$
C_1 et C_2 sont équivalents $\Longleftrightarrow C_1 \sqsubseteq C_2$ et $C_2 \sqsubseteq C_1$	C_1 et C_2 sont équivalents $\Longleftrightarrow \neg C_1 \sqcap C_2$ et $C_1 \sqcap \neg C_2$ sont insatisfiables	C est satisfiable $\Longleftrightarrow \{C(a)\}$ est satisfiable
C_1 et C_2 sont disjoints $\Longleftrightarrow (C_1 \sqcap C_2) \sqsubseteq \bot$	C_1 et C_2 sont disjoints $\Longleftrightarrow C_1 \sqcap C_2$ est insatisfiable	$C_1 \sqsubseteq C_2 \Longleftrightarrow \{C_1(a)\} \models C_2(a)$
	$\mathcal{K} \models C(a) \Longleftrightarrow \mathcal{K} \cup \{\neg C(a)\}$ est insatisfiable	

FIG. 10 – *Dépendances entre les tests de subsomption, de satisfiabilité et d'instanciation*

Les moteurs d'inférence actuels en LDs profitent de ces dépendances entre les tests. Deux familles d'algorithmes de raisonnement pour les LDs cohabitent. La première famille, issue du système KL-ONE (Brachman et Schmolze, 1985), utilise des algorithmes de type *normalisation/comparaison* (abrégé par NC), qui sont basés sur la subsomption (voir (Nebel, 1990) pour un exemple d'un tel algorithme). Ces algorithmes ont l'avantage d'être simples à mettre en œuvre, mais sont efficaces seulement pour des LDs très simples comme $\mathcal{FL}^-$ voire $\mathcal{AL}$. Des systèmes comme CLASSIC (Patel-Schneider, 1991) ont fait le choix de ne pas tout représenter, afin d'adopter un comportement déductif contrôlable. À l'inverse, des systèmes comme LOOM (MacGregor et Bates, 1987) et BACK (Peltason, 1991) offrent plus d'expressivité.

La seconde famille utilise des algorithmes basés sur la satisfiabilité, avec l'utilisation d'algorithmes appelés *tableaux* (Baader et Sattler, 2001). Ces algorithmes peuvent s'appliquer sur des LDs plus expressives, comme celles basées sur $\mathcal{SH}$. Des systèmes comme KRIS (Baader et Hollunder, 1991), FaCT (Horrocks, 1998), ou d'autres plus récents comme FaCT++ (version en C++ de FaCT), Pellet (Sirin et al., 2004) et Racer (Haarslev et Müller, 2001) se basent sur de tels algorithmes, tout en possédant des optimisations sophistiquées. FaCT++, Pellet et Racer sont orientés pour la manipulation de documents OWL DL, et sont capables de raisonner sur la T-Box et la A-Box (uniquement la T-Box pour FaCT++).

3.2.2 Langages et outils associés à OWL pour manipuler les connaissances

La manipulation de documents OWL passe d'abord par l'utilisation des opérations en LDs pour classifier et instancier les connaissances. La recherche d'appartenance d'un individu à sa(ses) classe(s) est un exemple d'application directe du test d'instanciation.

Contrairement au modèle des GCs, LDs/OWL ne dispose pas de *variable* (Borgida, 1996), au sens des variables en logique classique. Ceci s'explique de manière intuitive, où après les processus de classification et d'instanciation, tous les individus doivent avoir "trouvé leurs places" parmi les ensembles de concepts. L'interrogation ou l'application de règles en OWL s'avèrent donc limitées d'une façon générale. La prise en compte des règles est d'ailleurs considérée comme "la prochaine étape du Web Sémantique". En complément des raisonnements en LDs, d'autres langages et outils sont nés et proposent l'application de règles et l'interrogation de documents OWL dans un cadre comparable aux traitements en logique classique. Notons que certains outils RDF(S)/OWL empruntent à la logique classique des mécanismes d'inférence : chaînage avant avec CWM, arrière avec Euler, les deux types de chaînage avec Jena2 ou Jess. (Horrocks et al., 2005) appliquent de règles en LPO sur des connaissances OWL.

Déduire des connaissances. SWRL - Semantic Web Rule Language - (Horrocks et al., 2004) est un langage pour appliquer des règles sur des documents OWL, et qui tend à devenir une recommandation W3C (http://www.w3.org/). Il s'agit d'une véritable extension de OWL avec prise en compte des variables. La manipulation de OWL conjointement avec des règles SWRL peut se faire avec la plateforme SweetRules ou la bibliothèque KAON2. SweetRules (http://sweetrules.projects.semwebcentral.org/) regroupe plusieurs outils Java, basés sur le plugin OWL (Knublauch et al., 2004) de Protégé (http:protege.stanford.edu/). Ces outils sont les moteurs d'inférence Jena2 (http://jena.sourceforge.net/) ou Jess (http://herzberg.ca.sandia.gov/jess/), et l'éditeur SWRL Editor (http://protege.stanford.edu/plugins/owl/swrl/). KAON2 (http://kaon2.semanticweb.org/) utilise des raisonneurs en LDs, tels que FaCT++, Pellet ou Racer[8], et intègre un moteur d'inférence de règles (Motik et al., 2004). KAON2 est implémenté en Java, et peut communiquer avec Protégé via une interface DIG (http://dig.sourceforge.net/).

Interroger des connaissances. Parmi les langages d'interrogation en passe de devenir des recommandations W3C ou des standards de fait, nous pouvons citer RQL (Karvounarakis et al., 2002) et SPARQL (Prudhommeaux et Seaborne, 2006) pour l'interrogation de documents RDF(S) avec une syntaxe "à la SQL". OWL-QL (Fikes et al., 2004) est un langage d'interrogation spécifique à OWL, avec une syntaxe qui étend OWL.

L'exploitation de ces langages d'interrogation peut se faire avec les outils suivants. Sesame (http://sourceforge.net/projects/sesame/) et Jena2 proposent l'interprétation de requêtes en RQL.

[8] dans sa version Pro, http://www.racer-systems.com/products/racerpro/users-guide-1-9.pdf

CWM (http://www.w3.org/2000/10/swap/doc/cwm.html), Euler (http://www.agfa.com/w3c/euler/) et Jena2 permettent l'interprétation de SPARQL. Racer constitue une application pour utiliser OWL-QL. Enfin, les langages d'interrogation de données XML, comme XQuery, peuvent servir pour l'interrogation de documents OWL mais leurs utilisations en restent difficiles.

Vérifier des connaissances. La notion de contraintes, au sens de celles présentées avec les GCs et non de celles venant des restrictions d'utilisation de constructeurs en LDs, ne semble pas actuellement préoccuper la communauté du Web Sémantique. Citons le travail (McKenzie et al., 2004) qui étend SWRL pour exprimer des contraintes, et le langage PAL (http://protege.stanford.edu/plugins/paltabs/) dédié à Protégé pour définir contraintes et requêtes sous forme de règles.

3.3 GCs *versus* OWL, aspect raisonnement

Aspect logique. D'un point de vue logique, (Borgida, 1996) prouve que $\mathcal{ALCNR}^+$ a le même pouvoir expressif qu'un sous-ensemble de la LPO ayant des prédicats unaires ou binaires et n'autorisant que trois variables maximum, dont deux libres au plus. (Horrocks et Patel-Schneider, 2003; Baader et al., 2003) montrent que OWL DL est un sous-ensemble de la LPO.

Complémentarité. Nous avons vu que les raisonnements faisables dans le modèle des GCs et ceux en OWL sont assez différents les uns des autres, avec d'un coté des raisonnements basés sur la projection et de l'autre basés sur la classification/instancition. Il est donc intéressant d'aborder ces deux approches comme complémentaire l'une de l'autre. En effet, une difficulté importante en modélisation des connaissances est de trouver un bon compromis entre *expressivité* et *efficacité* du modèle de représentation : plus le modèle est expressif, plus la complexité des raisonnements est élevée et donc son efficacité diminuée. Ainsi, plutôt que de d'essayer d'accroître l'expressivité d'une approche de modélisation, il peut être préférable d'exploiter ce pour quoi une autre est efficace. Ici,

- Le modèle des GCs permet d'appliquer des règles et/puis d'interroger ou de vérifier les connaissances d'un domaine, par des opérations intrinsèques au modèle.
- En amont de cette phase d'exploitation des connaissances, l'environnement LDs/OWL permet de capitaliser les connaissances en offrant une grande souplesse dans cette démarche. En effet, les raisonnements propres à cet environnement permettent de réorganiser et même de compléter les connaissances de manière pertinente.

Certains travaux ont déjà été effectués dans cette perspective. Citons CARIN, combinant logique classique et LDs au sein d'un même langage et outil (Rousset et Levy, 1998). Tim Berners-Lee proposa en 2001 de rapprocher RDF et GCs (http://www.w3.org/DesignIssues/CG.html). D'autres travaux ont suivi, par exemple (Raimbault et al., 2006) propose une traduction de OWL en GCs afin d'en exploiter les outils, et la plateforme Corese (Corby et al., 2000) utilise les GCs pour annoter et traiter les ressources de documents RDF(S).

Un autre aspect de complémentarité peut être le suivant. Dans la définition de rôles avec OWL, certaines limites apparaissent et ne peuvent être exprimées en LDs. Il s'agit d'une part de la difficulté bien connue à représenter des quantifications sur les rôles, comme "tous les élèves assistent à tous les cours". De telles spécifications peuvent s'exprimer par des contraintes en GCs, comme la contrainte positive C_2 en Figure 8. D'autre part, il s'agit d'exprimer des relations entre rôles, comme "deux personnes qui ont pour résidence des pays différents" (exemples tirés de (Napoli, 1997)). De telles spécifications ne sont pas directement exprimables en GCs, qui ne disposent pas de liens permettant d'indiquer que deux sommets sont différents. Ils peuvent cependant être simulés par des relations, comme la relation diff utilisée dans (Baget et al., 1999).

Complexité. Pour le modèle des GCs et le langage OWL, la principale limite est une limite de temps calculatoire. La projection pour la manipulation de GCs utilise des homomorphismes de graphes, ainsi la recherche de projections ou l'application de règles sont de façon générale des problèmes NP-complets (Chein et Mugnier, 1992). (Baget et Mugnier, 2002) indique que la vérification de contraintes est un problème co-NP-complet pour les contraintes négatives et co-NP^{NP}-complet pour les positives. Pour les LDs et OWL, cette dualité expressivité/efficacité a donné lieu à deux courants. D'une part les approches qui ne permettent pas de tout représenter, mais qui ont un comportement déductif contrôlable, comme le système CLASSIC ou les langages OWL Lite et OWL DL ; d'autre part, les approches très expressives comme les systèmes LOOM et BACK, ou OWL Full. Des travaux (Tobies, 2001; Horrocks et Patel-Schneider, 2003) ont montré que les traitements de classification et d'instanciation avec OWL Lite sont dans la classe de complexité EXP-Complet, dans la classe NEXP-Complet pour OWL DL, et dans la classe NP-Complet pour RDFS ; quant aux traitements avec OWL Full ils sont indécidables.

Valeurs numériques. En plus de ces limites de temps calculatoires, le modèle des GCs et le langage OWL souffrent de ne pouvoir traiter des valeurs numériques en général, ou alors par des procédures externes ad-hoc. Ceci s'explique par le choix logique de ces approches de représentation qui sont dépourvues de symboles fonctionnels.

Connaissances structurelles. Nous avons vu que le support en GCs est moins expressif qu'une T-Box en LDs. Cependant l'utilisation de règles et/ou de contraintes viennent compléter ce manque d'expressivité. Les règles et les contraintes en Section 3.1.2 en sont des exemples. La définition récursive de concepts ou de rôles est peu conseillée en GCs ou en OWL. Par exemple, la définition Entier = Entier $\sqcap \forall$successeur.Entier provoque un circuit terminologique ; il n'est donc plus possible de substituer le concept Entier par sa définition, notament dans un processus de normalisation d'un algorithme NC. En GCs, cette définition peut s'exprimer avec un type défini ou une règle ; mais dans ce dernier cas, la fermeture du graphe est infinie.

4 Conclusion

Issus des réseaux sémantiques, le modèle des GCs d'une part et les LDs à travers OWL d'autre part permettent de modéliser de façon formelle des connaissances. Les connaissances d'un domaine sont représentées sur deux niveaux conceptuels qui procurent une vision structurelle et une vision factuelle des connaissances. GCs et OWL sont des approches concrètes de modélisation dans la messure où il existe des outils logiciels efficaces pour modéliser et exploiter les connaissances.

Modéliser des connaissances par le modèle des GCs constitue une approche de modélisation intensionnelle : en premier lieu est créer et organiser le support, puis la modélisation des faits par les graphes se fait de façon relativement intuitive et visuelle. Pour interroger, déduire et vérifier des connaissances, ce modèle dispose de la projection, adéquate et complète vis à vis de la déduction en LPO, de règles et de contraintes. Les raisonnements se font dans l'hypothèse d'un monde fermé. Le langage OWL relève d'une approche de modélisation extensionnelle. Suivant cette approche plus souple de modélisation, les concepts, les rôles et les individus qui constituent les connaissances sont décrits et organisés localement. Des raisonneurs en LDs permettent de classifier et d'instancier les connaissances dans leur globalité, et en trouver les interprétations valides. Le sous-langage OWL DL assure la décidabilité de tels raisonnements,

qui sont fait dans l'hypothèse d'un monde ouvert. En contrepartie de cette souplesse de modélisation, l'utilisation de requêtes ou de règles y est, de façon générale, difficile et limitée. Cependant, des outils RDF(S)/OWL ont été récemment développés pour interroger et déduire des connaissances.

Références

Baader, F., D. Calvanese, D. L. McGuiness, D. Nardi, et P. F. Patel-Schneider (2003). *The Description Logic Handbook: Theory, Implementation, Applications*. Cambridge Univ. Press.

Baader, F. et B. Hollunder (1991). KRIS: Knowledge Representation and Inference System. *SIGART Bulletin 2*(3), 8–15.

Baader, F. et U. Sattler (2001). An Overview of Tableau Algorithms for Description Logics. *Studia Logica 69*, 5–40.

Baget, J. F., D. Genest, et M. L. Mugnier (1999). Knowledge Acquisition with a Pure Graph-Based Knowledge Representation Model. In *Proc. of KAW'99*, Volume 2, pp. 7.1.1–7.1.20.

Baget, J. F. et M. L. Mugnier (2002). Extensions of Simple Conceptual Graphs: the Complexity of Rules and Constraints. *JAIR 16*(12), 425–465.

Berners-Lee, T., J. Hendler, et O. Lassila (2001). The Semantic Web. *Scientific American 279*(5), 34–43.

Borgida, A. (1996). On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence 82*(1-2), 353–367.

Brachman, R. J. et H. J. Levesque (1984). The Tractability of Subsumption in Frame-Based Description Languages. In *Proc. of AAAI'84*, pp. 34–37.

Brachman, R. J. et J. G. Schmolze (1985). An overview of the KL-ONE knowledge representation syste. *Cognitive Science 9*(2), 171–216.

Bray, T., J. Paoli, C. M. Sperberg-McQueen, E. Maler, et F. Yergeau (2004). Extensible Markup Language (XML) 1.0 (Fourth Edition). http://www.w3.org/TR/xml/.

Brickley, D. et R. V. Guha (2004). RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/.

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

Chein, M. et M. L. Mugnier (1997). Positive Nested Conceptual graphs. In *Proc. of ICCS'97*, Volume 1257 of *LNAI*, pp. 95–109. Springer.

Chein, M. et M. L. Mugnier (2004). Concept types and coreference in simple conceptual graphs. In *Proc. of ICCS'04*, Volume 3127 of *LNAI*, pp. 303–318. Springer.

Chein, M., M. L. Mugnier, et G. Simonet (1998). Nested Graphs: A Graph-based Knowledge Representation Model with FOL Semantics. In *Proc. of KR'98*, pp. 524–534. Morgan Kaufmann Publishers.

Corby, O., R. Dieng, et E. Hébert (2000). A Conceptual Graph Model for W3C Resource Description Framework. In *Proc. of ICCS'00*, Volume 1867 of *LNAI*, pp. 468–482. Springer.

Dean, M., D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, et L. A. Stein (2004). OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/.

Delugach, H. S. (2001). CharGer: A Graphical Conceptual Graph Editor. In *Proc. of CG Tools Workshop of ICCS'01*.

Eklund, P. W., N. Roberts, et S. P. Green (2000). OntoRama: Browsing an RDF Ontology using a Hyperbolic-like Browser. In *Proc. of CW'2002*, pp. 405–411. IEEE press.

Fikes, R., P. Hayes, et I. Horrocks (2004). OWL-QL - a language for deductive query answering on the Semantic Web. *Journal of Web Semantics 2*(1), 19–29.

Gruber, T. R. (1993). A translation approach to portable ontologies. *Knowledge Acquisition 5*(2), 199–220.

Gutierez, A. (2005). In *Proc. of ICCS Tools Workshop*. http://www.lirmm.fr/cogui/.

Haarslev, V. et R. Müller (2001). Racer system description. In *Proc. of IJCAR'2001*, Volume 2083 of *LNAI*, pp. 701–705. Springer.

Horrocks, I. (1998). Using an Expressive Description Logic: FaCT or Fiction ? In *Proc. of KR'98*, pp. 636–649.

Horrocks, I. et F. P. Patel-Schneider (2003). Reducing OWL entailment to description logic satisfiability. In *Proc. of ISWC'03*.

Horrocks, I., P. F. Patel-Schneider, S. Bechhofer, et D. Tsarkov (2005). OWL Rules: A Proposal and Prototype Implementation. *Journal of Web Semantics 3*(1), 23–40.

Horrocks, I., P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, et M. Dean (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/.

Kabbaj, A. (2006). Development of Intelligent Systems and Multi-Agents Systems with Amine Platform. In *Proc. of ICCS'06*, Volume 4068 of *LNCS*, pp. 286–299. Springer.

Karvounarakis, G., S. Alexaki, V. Christophides, D. Plexousakis, et M. Scholl (2002). RQL: A Declarative Query Language for RDF. In *Proc. of WWW'02*, pp. 592–603. ACM.

Klyne, G. et J. J. Carroll (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/rdf-concepts/.

Knublauch, H., R. W. Fergerson, N. F. Noy, et M. A. Musen (2004). The Protege OWL plugin: An Open Development Environment for Semantic Web Applications. In *Proc. of ISWC'04*.

Lehmann, F. (1992). *Semantic Networks in Artificial Intelligence*. Pergamon Press.

MacGregor, R. M. et R. Bates (1987). The LOOM knowledge representation language. In *Proc. of Knowledge-Based Systems Workshop*.

McKenzie, C., P. Gray, et A. Preece (2004). Extending SWRL to Express Fully-Quantified Constraints. In *Proc. of RuleML'04*.

Minsky, M. (1980). A Framework for Representing Knowledge. In D. Metzing (Ed.), *Frame Conceptions and Text Understanding*, pp. 1–25. Berlin : de Gruyter.

Motik, B., U. Sattler, et R. Studer (2004). Query Answering for OWL-DL with Rules. In *Proc. of ISWC'04*.

Mugnier, M. L. et M. Chein (1996). Représenter des connaissances et raisonner avec des graphes. *Revue d'intelligence artificielle 10*(1), 7–56.

Napoli, A. (1997). Une introduction aux logiques de descriptions. Tech. Report 3314, INRIA.

Nebel, B. (1990). *Reasoning and Revision in Hybrid Representation Systems*. LNAI. Springer.

Patel-Schneider, P. F. (1991). The CLASSIC knowledge representation system: Guiding principals and implementation rationale. *SIGART Bulletin 2*(3), 108–113.

Peltason, C. (1991). The BACK system - an overview. *SIGART Bulletin 2*(3), 114–119.

Prudhommeaux, E. et A. Seaborne (2006). SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/.

Raimbault, T., H. Briand, R. Lehn, et S. Loiseau (2006). Interrogation et Vérification de documents OWL dans le modèle des Graphes Conceptuels. In *EGC'06*, Volume E-6 of *RNTI*, pp. 187–192. Cépaduès Éditions.

Rousset, M. C. et A. Y. Levy (1998). Combining Horn rules and description logics in CARIN. *104*, 165–209.

Salvat, E. (1998). Theorem Proving Using Graph Operations in the Conceptual Graph Formalism. In *Proc. of ECAI'98*.

Schmidt-Schauss, M. et G. Smolka (1991). Attributive concept descriptions with complements. *Artificial Intelligence 48*(1), 1–26.

Sirin, E., B. Parsia, B. C. Grau, A. Kalyanpur, et Y. Katz (2004). Pellet: A practical owl-dl reasoner. Submitted for publication to Journal of Web Semantics.

Southey, F. et J. G. Linders (1999). Notio - A Java API for Developing CG Tools. In *Proc. of ICCS'99*, Volume 1640 of *LNAI*, pp. 262–271. Springer-Verlag.

Sowa, J. F. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. Addison Wesley.

Storey, M., M. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, et N. Noy (2001). Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in protege. In *Proc. of Workshop of K-CAP'2001*.

Studer, R., R. Benjamins, et D. Fensel (1998). Knowledge Engineering: Principles and Methods. *Data Knowledge Engineering 25*(1-2), 161–197.

Tobies, S. (2001). Complexity Results and Practical Algorithms for Logics in Knowledge Representation. PhD thesis, RWTH Aachen, Germany.

Wermelinger, M. (1995). Conceptual Graphs and First-Order Logic. In *Proc. of ICCS'95*, Volume 964 of *LNAI*, pp. 323–337. Springer.

Summary

We present and compare two formal and concrete approaches for modeling and handling knowledge of a domain. The model of the conceptual graphs is to intensionally model knowledge with a support and graphs. It has a semantics in first order logic, and makes closed world assumption for reasoning. OWL is a language to extensionally describe ontologies and facts in the Web, has a semantics from description logics and makes open world assumption.