Comprendre des Descriptions Simples à l'aide d'un Configurateur

Mathieu Estratat, Laurent Henocque

Comprendre des Descriptions Simples à l'aide d'un Configurateur

Mathieu Estratat **Laurent Henocque**

Université Paul Cézanne,
Laboratoire des Sciences de l'Information et des Systèmes
Avenue Escadrille Normandie Niemen
13397 Marseille cedex 20, France

{mathieu.estratat, laurent.henocque}@lsis.org

Résumé

Le but de l'analyse du langage naturel est de construire des représentations traitables par une machine d'un texte donné en entrée. Nous présentons ici une nouvelle approche de ce problème qui combine l'analyse syntaxique (le parsage) et la construction de la sémantique en une unique tâche de configuration. L'utilisation d'un configurateur pour le parsage est possible en décrivant la grammaire comme un modèle objet sous contraintes. Nous étendons cette approche au traitement de la sémantique en introduisant deux nouveaux modèles objets contraints : un pour représenter la sémantique, et l'autre pour lier la syntaxe et la sémantique via la notion de "schémas". Nous mettons en avant les avantages de cette approche et présentons des résultats expérimentaux dans le cadre encore restreint de textes descriptifs simples, mais qui pourra être étendu pour capturer des sémantiques plus riches. Le système objet contraint sous-jacent est présenté formellement en utilisant le langage relationnel Z.

Abstract

The ultimate goal of natural language parsing is to build machine processable representations of text input. We present here a novel approach to this problem, that treats the combination of grammar parsing and building semantics as a single configuration task.Parsing using configurators is made possible by describing the grammar as a constrained object model. We extend the scope to semantics by introducing two new constrained object models : one for the semantical domain, and another that creates a bridge between syntax and semantics. We highlight some advantages of this approach, and present experimental results, in a restricted case of descriptive texts that can be further extended to capture richer semantics. The underlying constrained object system is formally presented using the Z relational language.

1 Introduction

Pouvoir construire à partir d'un texte en langage naturel décrivant des objets une représentation de ces instances traitable par une machine est une tâche ambitieuse, réalisable grâce à une évolution orientée objet de la programmation par contraintes appelée configuration. Cet article est organisé de la manière suivante : la section 1.1 décrit les approches classiques pour le parsage du langage naturel et présente les travaux existants dans ce domaine. La section 2 présente rapidement la configuration. La section 3 introduit nos objectifs de recherche et les limitations pour lesquelles nous avons opté. La section 4 présente les trois modèles objets sous contraintes : le "monde" ou le domaine du problème (la description d'ordinateurs (PC)), le parseur et dans la section 4.3 les *schémas*, une construction utilisée pour lier le modèle objet de la grammaire et celui de la sémantique. Les résultats expérimentaux dans la section 5 sont obtenus en utlisant le configurateur JConfigurator[8] d'Ilog. La section 6 conclut et présente les résultats à venir et les différentes évolutions possibles.

1.1 Le problème usuel des analyses syntaxico - sémantiques

Il existe plusieurs manières de comprendre le mot "sémantique" lorsque l'on travaille en traitement du

langage naturel. La plus simple, utilisée dans de nombreuses théories linguistiques (comme par exemple GPSG [5] ou HPSG [15]), consiste à considérer la sémantique comme les valeurs prises par des traits sémantiques ad hoc, documentés dans le lexique et permettant de filtrer les phrases incompatibles. Par exemple, le lexique peut préciser le fait qu'un verbe "d'action" (comme "manger") nécessite un sujet de type "agent" (comme "Paul" ou "l'homme"). Dans ces formalismes, les dépendances aident à la propagation de telles notions sémantiques entre des pronoms et leurs référents à travers plusieurs phrases[1].

Une approche plus formelle de la sémantique en linguistique computationelle utilise le parseur pour construire une représentation en logique des prédicats à partir d'une phrase ou d'un texte. Dans [10], la sémantique des phrases est vue comme la composition fonctionnelle d'expressions du lambda calcul associées à chaque mot via le lexique[2]. En utilisant cette technique, un programme peut "lire" un texte et en produire une expression logique[3]. Cette approche est incompatible avec le fait de laisser la sémantique du texte aider à désambigüer l'arbre syntaxique, même si cela est souvent requis[4]. Cette immense contribution a cependant clarifié la sémantique des déterminants aussi bien que celle de la négation et a suscité de forts espoirs quand au fait que la sémantique soit traîtable de cette manière, mais malheureusement elle est apparue comme trop difficile à généraliser.

Dans [11] l'auteur présente un "dialogue" avec une machine, travail qui pose des bases essentielles au traitement automatique des langues par la programmation logique. Prolog a de plus été utilisé dans [3] pour décrire et analyser les "Definite Clause Grammars" (DCG) [14], ce qui montre la faisabilité du traitement de la sémantique d'un "sous-ensemble intéressant du français" [2]. Les expressions du lambda calcul ainsi que l'application fonctionnelle sont facilement représentables par des termes et des règles Prolog. Bien qu'étant opérationnelle, cette approche possède les même inconvénients que ceux évoqués plus haut, puisqu'il est difficile de lier les règles Prolog relatives à la preuve de théorème aux règles spécifiques de l'analyseur afin d'exploiter dynamiquement la sémantique.

La "Discourse Representation theory" (DRT) [9] aborde les difficultés posées par les références pronominales en utilisant une variante de la logique du premier ordre qui aide à contrôler la portée des variables quantifiées. Construire une "boîte" de DRT comme résultat de l'analyse d'un texte pose toujours le problème de travailler avec cette représentation et rend difficile l'exploitation dynamique de l'information sémantique existante lors de l'analyse des phrases suivantes.

Le système ILLICO [12, 13] basé sur Prolog contourne cette difficulté d'une manière intéressante : la grammaire permet d'analyser des textes dont la sémantique est décrite par des réseaux conceptuels [16], construits dynamiquement et de manière concurrente en utilisant une évalutation paresseuse (via le prédicat "freeze"[5]). Les résultats impressionants du sytème ILLICO indiquent clairement que le traitement de la sémantique lors de l'analyse syntaxique est possible, une intuition qui guide notre propre travail.

Il faut également noter que d'une manière totalement orthogonale, Gross [6] s'est interrogé sur l'agencement des compositions fonctionnelles dans la sémantique du langage naturel en énumérant les expressions idiomatiques que nous utilisons dans la vie de tous les jours, insistant sur le fait qu'il est plus probable que la sémantique soit portée par des groupes de mots ou par des combinaisons de mots que par les mots eux-même.

Notre approche essaie de traiter le problème de la sémantique des textes descriptifs en s'appuyant sur une approche de type modèles finis. Lors de la lecture (et du parsage) d'un texte, nous ne construisons pas une formule logique mais un modèle fini (au sens logique) d'un modèle objet sous contraintes, formel et préexistant. Par exemple, notre programme lit la phrase "Le PC a deux disques durs." et la *comprend* comme un monde où au moins trois objets interconnectés coexistent : une instance du type "PC" et deux instances du type "DisqueDur" en plus des autres objets non mentionnés et sous-jacents. Les contributions nouvelles de ce travail sont que :

1. les modèles objets décrivent à la fois l'ensemble des phrases syntaxiquement valides et le domaine sémantique,
2. le lien entre la syntaxe et la sémantique est lui-même un modèle objet contraint,
3. un configurateur exploite en même temps les trois modèles objets. La sémantique aidant ainsi à

[1] Cette manière de prendre en compte la sémantique dans les théories grammaticales ne peut pas traiter facilement la sémantique dans sa totalité. Par exemple, le texte "Paul est aveugle. Il lit ma lettre." est parfaitement valide.

[2] Le lambda calcul permet d'attribuer une sémantique rigoureuse aux expressions logiques *incomplètes*. C'est donc un langage qui permet de décrire comment des fragments (invalides séparément) de la logique du premier ordre peuvent être combinés afin de mener à des formules logiques *syntaxiquement* valides. Nous avons mis en évidence *syntaxiquement* pour souligner le fait que la formule logique résultante nécessite encore d'être exploitée par un prouveur de théorèmes.

[3] Généralement du premier ordre, car les déterminants sont traduits par des quantificateurs, mais potentiellement modale, ou du second ordre ou également linéaire.

[4] Par exemple pour décider à quoi rattacher un complément de nom ou une coordonnée.

[5] Lorsqu'elle est gelée ("freeze"), l'évalutation d'un but est repoussée jusqu'à ce que certaines des variables qu'il contient soient instanciées.

désambigüer la syntaxe,

4. le résultat est un ensemble d'objets interconnectés pouvant être utilisé par un autre programme de raisonnement.

Ainsi, le parsage sémantique sous contraintes possède les propriétés permettant au système de travailler en mode analyse ou en mode génératif tout en permettant aux modèles syntaxique et sémantique de rester indépendants.

2 La Configuration

Configurer consite à simuler la réalisation d'un produit plus ou moins complexe à partir de composants choisis dans un catalogue de types. Le produit peut avoir un caractère concret (tel un PC ou une voiture), ou être abstrait (comme par exemple, un voyage organisé ou une police d'assurance). Résoudre un tel problème peut être envisagé par diverses approches techniques, généralement au moyen d'évolutions de la programmation par contraintes ou de la programmation logique, les CSP "standards" étant inadéquats. [8] fournit un point d'entrée intéressant vers une bibliographie du sujet qui n'est pas appropriée ici. Dans [8], l'évolution des CSP considérée s'appuie sur l'utilisation de variables ensemblistes et de classification, permettant de décrire les relations d'association et d'héritage, et l'expression de contraintes traversant la structure construite.

La structure générale d'un problème de configuration est organisée autour d'un modèle objet définissant quelles compositions sont valides et de contraintes sur les éléments de ce modèle (compatibilité entre éléments via les valeurs de leurs attributs respectifs par exemple). Le modèle objet, appelé également modèle générique, représente l'ensemble des assemblages de composants possibles pour le problème envisagé. La figure 1 présente le diagramme de classes d'un modèle objet simplifié pour des ordinateurs (un exemple très souvent utilisé en configuration). Un tel diagramme pris isolément représente un surensemble de tous les PC potentiellement réalisables. Nous distinguons sur cette figure à la fois des relations d'héritage(les flêches) et des relations d'association(les traits) entre les classes soumises à des restrictions sur les cardinalités. Par exemple, un PC possède une seule carte mère mais peut avoir de un à quatres disques durs. Les relations d'héritage aident à la formalisation des abstractions en factorisant les attributs ou les relations partagées par plusieurs sous-classes, comme par exemple, l'attribut "prix" de la classe *Composants*.

Un tel modèle nécessite des contraintes de bonne

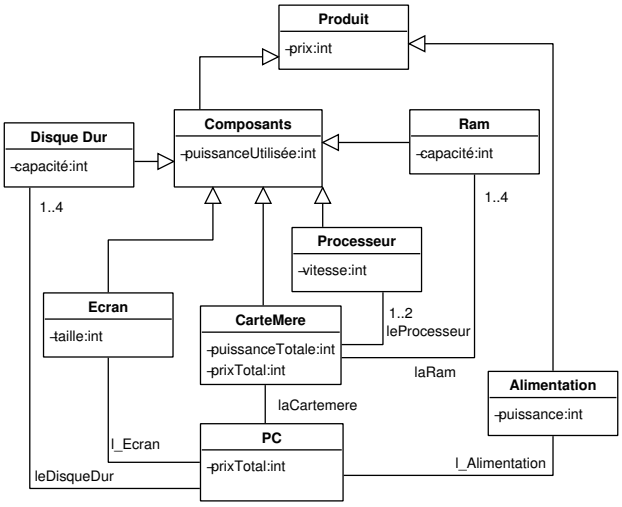

FIG. 1 – Un modèle objet générique pour la configuration d'ordinateurs

formation[6] pour ne décrire que les seuls assemblages valides. Par exemple, pour un PC réellement utilisable, "L'énergie électrique produite par l'alimentation doit dépasser la somme des énergies comsommées par tous les composants intervenants dans la solution" :
$\forall x \in PC, \sum(x.DisqueDur.puissanceUtilisée, x.Ecran.puissanceUtilisée, x.CarteMere.puissanceUtilisée) \leq x.Alimentation.puissance$

Cet exemple[7] illustre bien le type de contraintes devant être adjointes au modèle objet pour définir les seules construction valides. La figure 2 décrit le fonctionnement d'un configurateur de manière très succinte. Le configurateur est constitué d'un précompilateur qui vérifie si les contraintes adjointes au modèle générique sont bien en accord avec celui-ci et si le modèle générique ne contient pas d'erreur non plus. Ensuite ce modèle "compilé" est passé au solveur avec la partie de solution à configurer et *l'objet racine* permettant d'initier la configuration.

2.1 Recherche de solutions

Peu d'éléments sont connus au départ de la recherche de solution exceptés le modèle générique et un fragment de la solution devant être complété par le configurateur. Un objet *racine* permettant d'initier le processus est également disponible. Les élé-

[6] Nous empruntons volontairement ce terme au domaine des grammaires

[7] Dans cet exemple semi-formel, la notation pointée usuelle est utilisée pour traverser les relations et accéder aux attributs des objets.

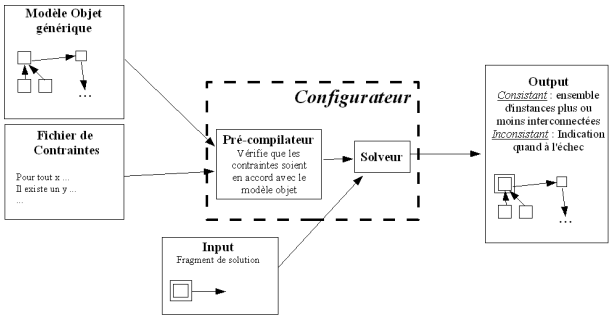

FIG. 2 – Le processus de configuration

ments inconnus sont entre autres le nombre d'objets nécessaires, leurs interconnections (représentant un graphe), leurs types et les valeurs de leurs attributs.

Etant semi-décidable, ce problème capture la complexité inhérente de la logique du premier ordre : les quantifications universelles y sont explicites et les quantifications existentielles sont prises en compte par les cardinalités des relations. Par exemple, la condition spécifiant que chaque carte mère possède au moins un processeur est une variation de l'axiome

$$\forall\, x \in \mathit{CarteMere}\, \exists\, y \in \mathit{Processeur}\ \mathit{possede}(x, y)$$

où "possede" est le prédicat binaire, vrai lorsque x contient/possède au moins un y.

2.2 Les sorties

Le configurateur doit fournir en sortie si l'entrée est consistante un graphe représentant les instances des classes du modèle générique interconnectées conformément au modèle et, si possible, dans le cas où il n'existe pas de solution, une explication de l'échec sous la forme d'un ensemble de contraintes incompatibles. Il est possible de travailler en s'arrêtant à la première solution trouvée, ou d'en produire plusieurs. Les problèmes de configuration sont assez fréquemment présentés comme des problèmes d'optimisation combinatoire : on cherche une solution qui optimise la valeur d'un certain critère.

3 Définition du problème et cadre de l'étude

Dans [4], à partir d'un modèle objet pour les grammaires de propriétés [1] comme présenté sur la figure 3 et d'une phrase, un configurateur est capable de générer un étiquettage syntaxique. Nous nous attachons ici à montrer comment cette approche peut être étendue à l'analyse de la sémantique, représentée elle aussi par

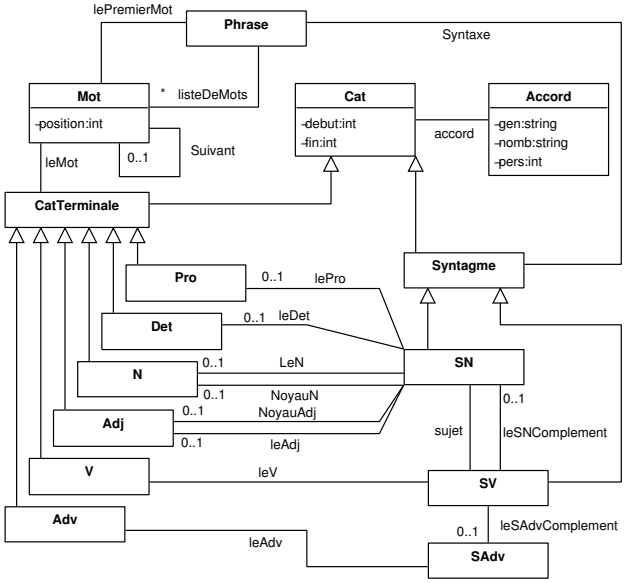

FIG. 3 – Un modèle objet pour le parseur

un modèle générique adéquat. Lorsqu'on traitera de phrases décrivant un PC par exemple, nous utiliserons un modèle générique proche de celui de la figure 1. De plus, nous proposons d'utiliser un modèle objet auxiliaire pour lier le modèle de la grammaire (syntaxique) et celui de la sémantique ou du monde sur lequel on écrit. Ce modèle auxiliaire est constitué d'un ensemble de *schémas sémantiques*. Un schéma est un objet lié d'un côté à des données syntaxiques et à un certain nombre d'objets du domaine (ou du monde) de l'autre. L'utilisation des schémas sémantiques permet de produire parallèlement à l'analyse une représentation orientée objet de l'information portée par chaque phrase.

Par exemple, en lisant la phrase "Le PC a deux disques durs", le programme construit une solution à partir des objets : PC_1, DD_1, DD_2, CM_1, $Ecran_1$, $Alimentation_1$, $Proc_1$, Ram_1 impliquant les liens : $(PC_1, Alimentation_1)$,
$(PC_1, Ecran_1)$, (PC_1, CM_1), (PC_1, DD_1), (PC_1, DD_2), $(CM_1, Proc_1)$, (CM_1, Ram_1) comme illustré par la figure 6.

3.1 Limitations

Ces travaux sont à l'heure actuelle dans une phase de démonstration d'intéret de la méthode. Le problème traité ici est semi-décidable dans le cas général et demeure exponentiel dans les meilleurs cas. Ainsi nous ne tiendrons pas compte des considérations relevant de la

capacité à monter en charge ni de la couverture du programme. Nous nous restreignons à un domaine précis : le monde du PC, dont le modèle objet est présenté sur la figure 1 et détaillé plus tard dans la Section 4.1, ainsi qu'à une syntaxe limitée, dont le modèle objet est présenté sur la figure 3 et détaillé dans la Section 3. Notre grammaire ne prend pas en compte, entre autres : la récursion [8], ni la négation, ni les pronoms et pas non plus la coordination. Le domaine sémantique ne prend pas en compte des ordinateurs réels. Un modèle détaillé d'ordinateur décrirait des classes pour le clavier, la souris, les ports ISA, AGP, PCI, USB ainsi que pour les cartes les utilisant comme adaptateurs pour divers types d'entrée/sortie (audio, video, réseau RJ45, etc...) et surement d'autres composants. Pour résumer, nous imposons une restriction sur la richesse de la sémantique. Nous utilisons des phrases *descriptives* dont le sujet ne représente qu'un seul objet. Toutes les instances d'objets ainsi définis doivent appartenir à un monde d'objets valides possibles défini par le *modèle objet fixé*. Autrement dit :

1. le modèle objet est connu et inchangeable,
2. les sujets des phrases ne peuvent pas représenter plus d'un objet à la fois.

Au delà des perspectives de montée en charge et de couverture, les restrictions sur le modèle ne rendent pas notre travail ad hoc et ce, grâce au méchanisme d'abstraction autorisé par le paradigme orienté objet. Une discussion sur la difficulté relative de prise en compte des extensions prévues de la sémantique est présentée dans la Section 6.

4 Les modèles objets sous contraintes

Cette section présente formellement une partie des modèles objets que nous utilisons. Afin de rendre la présentation indépendante de tout cadre applicatif de la configuration, les contraintes sont présentées en utilisant le langage de spécification Z, comme décrit dans [7][9]. Le modèle objet est divisé en trois sous-modèles distincts : syntaxe, sémantique et schémas.

4.1 Le domaine de la Sémantique

Les instances de ce monde sont des éléments de la classe abstraite "*WorldObject*". Toutes les relations présentées dans le modèle objet de la figure 1 sont factorisées en une unique relation, exprimant la fermeture transitive pour chacune des relations de la figure 1[10] appliquée à la classe *WorldObject* nouvellement introduite. Cette relation est nécessaire pour abstraire les relations entre éléments distants de plus d'une relation. Par exemple, la phrase "Le PC a deux processeurs" nécessite une relation entre les classes *Computer* et *Processeurs* alors que dans le modèle objet original, il existe une carte mère entre ces deux composants. Ainsi la relation "possede" représente la fermeture transitive de l'union de toutes les relations ayant une sémantique de composition. Une implémentation de ce problème typique de configuration requiert de choisir, pour différentes raisons techniques, entre combiner la relation "possede" avec les relations d'origine à l'aide de l'inclusion et des contraintes de classification ou de simuler les relations originales en utilisant des contraintes de cardinalité. Afin de ne pas surcontraindre notre système nous avons opté pour la seconde option. En respectant les notations de [7], les spécifications de classes du modèle objet pour la sémantique sont :

$$class - WorldObject : abstract$$

$$\begin{array}{|l} class - PC : concrete \\ -inherit : WorldObject \\ prixTotal : \mathbb{N} \end{array}$$

$$\begin{array}{|l} class - Produit : abstract \\ -inherit : WorldObject \\ prix : \mathbb{N} \end{array}$$

$$\begin{array}{|l} class - Alimentation : concrete \\ -inherit : Produit \\ puissance : \mathbb{N} \end{array}$$

$$\begin{array}{|l} class - Composant : abstract \\ -inherit : Produit \\ puissanceUtilisee : \mathbb{N} \end{array}$$

$$\begin{array}{|l} class - CarteMere : concrete \\ -inherit : Composant \\ puissanceTotale : \mathbb{N} \\ prixTotal : \mathbb{N} \end{array}$$

[8] Nécessaire notament pour les compléments du nom, comme par exemple dans le syntagme nominal (aussi appelé groupe nominal) "(Le processeur de (l'ordinateur de (mon patron)))" mais également dans les relatives.

[9] Il est difficile de présenter brièvement ce langage et les définitions proposées, ainsi, le lecteur est redirigé vers cet article, accessible électroniquement pour tout détail.

[10] cette relation porte le nom "possede", dénomination qui représente bien la notion de fermeture transitive, exemple : une voiture possède des portes et chaque porte possède des vitres, la fermeture transitive de la relation entre voiture et porte permet d'exprimer que la voiture possède un certain nombre de vitres

$$class - Processeur : concrete$$
$$-inherit : Composant$$
$$vitesse : \mathbb{N}$$

$$class - Ecran : concrete$$
$$-inherit : Composant$$
$$taille : \mathbb{N}$$

$$class - DisqueDur : concrete$$
$$-inherit : Composant$$
$$capacite : \mathbb{N}$$

$$class - Ram : concrete$$
$$-inherit : Composant$$
$$capacite : \mathbb{N}$$

La relation "universelle" *possede* peut être spécifiée par la fonction :

$$possede : WorldObject \longrightarrow \mathbb{F}\ WorldObject$$
$$\forall a, b, c : WorldObject \bullet possede(a, b) \land possede(b, c) \Rightarrow possede(a, c)$$

Les contraintes de cardinalité des relations peuvent être spécifiées de la même façon. Pour chaque PC, la cardinalité du role réflexif hérité de la classe WorldObject sur elle-même varie entre 4 et 7 (en utilisant l'opérateur d'image relationnelle de Z $\_(\!|\_|\!)$).

$$\forall pc : PC \bullet 4 \leq \#possede(\!|\{pc\}|\!) \leq 7$$

ou de manière alternative (en utilisant l'opérateur de déréférencement $\rightsquigarrow$ de [7])

$$\forall pc : PC \bullet 4 \leq \#pc \rightsquigarrow possede \leq 7$$

Chaque ordinateur possède exactement une alimentation, un écran et de un à quatre disques durs ... :

$$\forall pc : PC \bullet \#((pc \rightsquigarrow possede) \cap Alimentation) = 1$$
$$\forall pc : PC \bullet \#((pc \rightsquigarrow possede) \cap Ecran) = 1$$
$$\forall pc : PC \bullet 1 \leq \#((pc \rightsquigarrow possede) \cap DisqueDur) \leq 4$$

et ainsi de suite pour chacune des relations de la figure 1.

4.2 Le modèle objet du parseur

Le modèle objet utilisé pour la syntaxe est présenté sur la figure 3. Les grammaires de propriétés proposent que tout élément syntaxique soit représenté par une catégorie. Une catégorie est une structure de traits (un ensemble de traits). Un trait est un couple <*trait,valeur*>. Les traits sont utilisés en linguistique afin de représenter les caractéristiques des éléments pris en compte. Nous nous concentrons sur trois classes qui apparaîtront dans la liste des contraintes importantes plus tard : le syntagme (ou groupe) nominal (SN), le nom (N) et le déterminant (Det).

$$class - SN : concrete$$
$$-inherit : Syntagme$$

$$class - N : concrete$$
$$-inherit : CatTerminale$$

$$class - Det : concrete$$
$$-inherit : CatTerminale$$

[1] définit toute grammaire à l'aide de sept sortes de propriétés.

- Constituants : relation de composition entre catégories. En Z, les fonctions ($\longrightarrow$) sont des relations et les injections ($\rightarrowtail$) peuvent être utilisées pour imposer des cardinalités unaires.

$$leN : SN \rightarrowtail N$$
$$leDet : SN \longrightarrow Det$$

- Unicité : un nom apparaît au plus une fois dans un syntagme nominal. La déclaration ci-dessus de *leN* et *leDet* permet de définir également qu'au plus un N intervient dans la relation entre le *SN* et le *N* via la relation *theN*.
- Noyaux : élément qui "gouverne" le syntagme. Par exemple, dans un syntagme nominal, le *nom*. Le noyau est obligatoire et unique.

$$leNoyauN : SN \rightarrowtail N$$

- Linéarité : contraintes de précédence linéaire. Par exemple dans une phrase, le verbe doit être placé avant le complément. Toute catégorie, des terminales aux syntagmes, hérite de la classe abstraite *Categorie* qui contient les attributs *debut* et *fin* permettant d'implanter la linéarité.

$$class - Categorie : abstract$$
$$debut : \mathbb{N}$$
$$fin : \mathbb{N}$$

$$class - CatTerminale : abstract$$
$$-inherit : Categorie$$

$$debut : Categorie \rightarrowtail \mathbb{N}$$
$$fin : Categorie \rightarrowtail \mathbb{N}$$
$$\forall c : Categorie \bullet debut(c) = c.debut$$
$$\forall c : Categorie \bullet fin(c) = c.fin$$

Il est maintenant possible de spécifier qu'un syntagme nominal nécessite un déterminant avant le nom :

$$\forall\, sn : SN \bullet fin(leDet(\!|\{sn\}|\!)) \leq debut(leN(\!|\{sn\}|\!))$$

– Nécessité : obligation de cooccurrence de certaines sous-catégories. Dans un syntagme nominal, la présence d'un nom commun implique la présence d'un déterminant (par exemple dans "le chat").

$$\forall\, sn : SN \bullet leN(\!|\{sn\}|\!) \in Commun \Rightarrow \#leDet(\!|\{sn\}|\!) \geq 1$$

– Exclusion : obligation de non-cooccurrence de certaines sous-catégories. Dans un syntagme nominal, lorsqu'un nom propre est présent, il ne peut y avoir de déterminant (par exemple *'Le Paul').

$$\forall\, sn : SN \bullet leN(\!|\{sn\}|\!) \in Propre \Rightarrow \#leDet(\!|\{sn\}|\!) = 0$$

– Dépendance : Aucune méthode de traduction générique ne peut être formulée pour cette propriété[11] et notre grammaire ne requiert pas l'expression de telles contraintes.

4.3 Les schémas sémantiques

Nous proposons comme cadre générique à l'analyse de textes descriptifs de combiner les modèles objets sous contraintes de la grammaire et des domaines sémantiques en utilisant un modèle intermédiaire ou "pont" sous la forme d'un modèle objet pour les *schémas*. Les schémas mettent en relation la syntaxe et la sémantique et laissent la séquence de mots du texte contraindre l'occurrence d'instances du domaine (et inversement). Les schémas sont liés aux constructions syntaxiques, aux éléments du monde et aux autres schémas. Ils peuvent être spécialisés grâce à des relations d'héritage, comme illustré sur la figure 4.

Nous présentons les schémas et contraintes pour des phrases du type "sujet + avoir/posséder + complément possédant une cardinalité" telles que : "Le PC a /possède deux disques durs". Un sous-ensemble important de notre modèle de schéma (présenté sur la figure 5) peut donc être spécifié par les classes :

class − ScSvCard : concrete
$-inherit : ScSv$
$val : \mathbb{N}$
$objetDuMonde : \mathbb{F}\ WorldObject$

[11] Ces contraintes ad hoc permettent notamment de lier des éléments distants (comme par exemple un pronom et son référent) mais aussi de coordonner les genres et les nombres

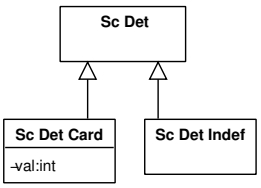

FIG. 4 – Une partie du diagramme de classes pour ScDet

class − ScSnCard : concrete
$-inherit : ScSn$
$val : \mathbb{N}$
$objetDuMonde : \mathbb{F}\ WorldObject$

class − ScDetCard : concrete
$-inherit : ScDet$
$val : \mathbb{N}$

et les associations :

$leWorldObject : ScSvCard \rightarrowtail WorldObject$
$leScDetCard : ScSnCard \rightarrowtail ScDetCard$
$leSn : ScSn \rightarrowtail SN$
$leSv : ScSv \rightarrowtail SV$

Toutes les classes de schémas pour les éléments cardinaux possèdent un attribut de type entier *val*. Le lexique décrit les correspondances entre un mot et une classe ou un nom d'attribut ou une valeur, etc ... Par exemple, grâce au schéma *ScDetCard*, l'attribut "val" du schéma lié au mot "deux" prendra comme valeur l'entier "2" qui est la sémantique de ce mot. Le schéma *ScSn* contient le nom de la classe dénotée par le nom compris dans le SN du schéma. Le schéma *ScSvCard* contraint la cardinalité de la relation entre l'objet représenté par le nom du sujet et ceux représentés par le complément en fonction de la valeur du déterminant cardinal. Pour des raisons de modularité de l'implémentation, les associations entre schémas et catégories peuvent être définies à un niveau d'abstraction[12]. Par exemple, le schéma *ScDet* est une classe abstraite pour tous les schémas de déterminants en relation elle même avec la classe *Det* représentant les déterminants.

$leDet : ScDet \rightarrowtail Det$

[12] JConfigurator réalise le typage dynamique durant la recherche et supporte les contraintes de type (contraintes dites de classification).

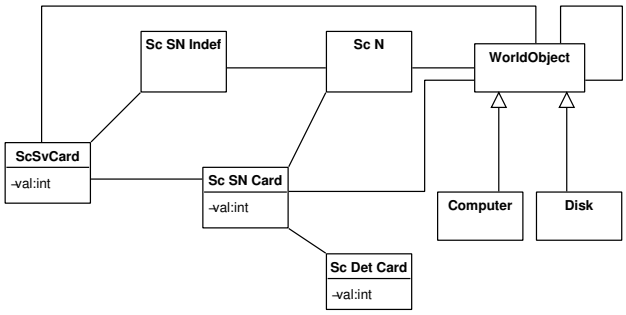

FIG. 5 – Fragment du diagramme de classe pour *ScSvCard*

4.3.1 Les contraintes

Selon les définitions précédentes, une contrainte définit que les valeurs des attributs *val* des catégories précitées sont identiques.

$$\forall x : ScSnCard \bullet x \cdot leScDetCard \rightsquigarrow getVal = x.val$$
$$\forall x : ScSvCard \bullet x \cdot leScSnCard \rightsquigarrow getVal = x.val$$
$$\forall x : ScSnCard \bullet x \cdot leSnCard \rightsquigarrow getObjetDuMonde = x.objetDuMonde$$

Une autre contrainte définit que chaque *ScSvCard* est en relation avec un nombre d'objets du monde égal à la valeur de l'attribut *val*.

$$\forall s : ScSvCard \bullet s.val = \#((s \cdot leWorldObject \cdot possede) \cap (s.objetDuMonde))$$
$$\forall s : ScSnCard \bullet s.objetDuMonde = \{o : WorldObject \mid o.type = s.classname\}$$

Notre modèle suppose que chaque nom commun dénote une classe présente dans le modèle. Les séquences "disque dur" ou "carte mère" sont traitées comme des mots composés (en effet, ils pourraient être également écrits "disque-dur" ou "carte-mère"), et sont ainsi étiquettés par un N(nom). Pour rester le plus général possible, les schémas sémantiques ne peuvent pas être directement liés aux composants du modèle sémantique (PC, CarteMère, etc...). De fait les schémas sont liés à la classe *WorldObject* dont les classes précédemment citées héritent. Ainsi pour parser des phrases décrivant des instances d'un autre "monde" sémantique, il suffit de "charger" le modèle objet contraint et le lexique associés à ce monde.

5 Résultats expérimentaux

Nous avons implémenté un programme de configuration à partir des spécifications précédentes et en utilisant l'outil de programmation par contraintes JConfigurator.

phrase	échecs	points de choix	temps
(1)	2	7	0,81 s
(2)	1	5	0,84 s
(3)	1	5	0,81 s
(1) + (2)	2	21	1,02 s
(1) + (2) + (3)	1946	1979	3,19 s

TAB. 1 – Résultats expérimentaux

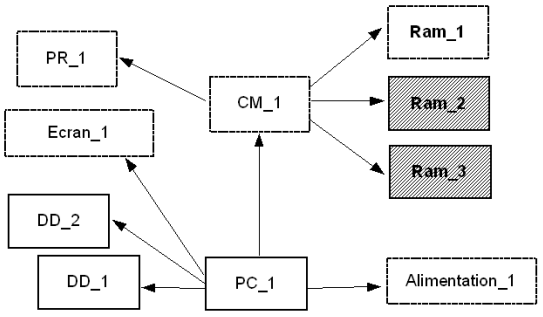

FIG. 6 – Le monde construit lors de l'analyse

Nous présentons les résultats expérimentaux obtenus en utilisant ce programme de configuration pour analyser des phrases dans le domaine sémantique du PC. Les résultats présentés correspondent aux trois phrases :
(1) Le PC a deux disques durs.
(2) La carte mère possède un processeur.
(3) La carte mère a trois barettes de mémoire.

Le résultat de la lecture de ces trois phrases à la suite est présenté sur la figure 6 où les rectangles blancs représentent les objets introduits dans la solution après lecture de la phrase (1), les rectangles Ram_2 et Ram_3 sont adjoints au modèle après lecture de la phrase 3. Les rectangles dont le contour est en pointillés représentent les objets introduits implicitement pour satisfaire les contraintes de cardinalité. Les objets du domaine introduits par les phrases sont réutilisés autant que possible dans la suite du traitement du texte. Par exemple, la carte mère, nécessaire implicitement pour "comprendre" la phrase (1) est la même que celle à qui réfèrent les phrases (2) et (3).

Le tableau 1 liste les statistiques d'éxécution obtenues[13] lors de la résolution du problème pour des

[13]Ordinateur : P4 2.4GHz, 512 Mo RAM, Windows XP SP2, Ilog JConfigurator 2.1

phrases isolées ou en tant que texte. La première colonne décrit quel type de test est effectué tandis que les deuxième et troisième colonnes listent respectivement le nombre d'échecs et le nombre de points de choix. La dernière colonne présente les temps d'éxécution en secondes. Le temps d'éxécution est lié d'une part à la taille du problème de configuration et d'autre part, bien sur, au nombre de points de choix. Ilog Jconfigurator n'implémente pas la "création d'objets à la demande". Cette limitation force le programme à pré-créer autant d'instances de chaque classe abstraite qu'il pourrait en avoir besoin aucours de la recherche (le typage dynamique est réalisé lors de la recherche). L'absence de contrainte brisant les symétries est une cause parmi d'autres du nombre important de points de choix. Dans le cas de la lecture de phrases enchaînées, le nombre important de points de choix est dû à l'impossibilité technique (provisoire) de figer le résultat de l'analyse de chaque phrase en séquence. Il est intéressant de souligner que ces travaux récents n'ont pas permis d'étudier des heuristiques de recherche. Ce que le programme effectue ici est ce qu'il doit calculer par défaut.

6 Conclusion et perspectives

Ces travaux montrent la faisabilité d'utiliser un configurateur pour analyser une séquence de phrases et construire dans le même temps une représentation intelligible par la machine de ce que "veut dire" le texte. Bien sûr le travail présenté est soumis à de nombreuses et rigoureuses restrictions, certaines identifiant parfaitement les travaux à venir, et d'autres nécessitant une étude plus approfondie. Dans la première catégorie, nous pouvons placer la prise en compte de phrases dénotant la valeur d'attributs (comme par exemple, dans les phrases "la fleur est bleue." ou "Le disque dur tourne à une vitesse de 7200 tr/min"), la valeur de nombreux adjectifs ou autres pronoms singuliers. Dans la seconde figurent des éléments relatifs à l'étude du langage naturel dans sa généralité et notament : la prise en compte de la négation, les déterminants indéfinis et pluriels (qui seront traduit en tant que quantifieurs universels) mais aussi les phrases imbriquées (compléments de nom et relatives) , les coordinations, etc ...

Cette approche pourrait être mal interprétée comme étant ad hoc. Tout d'abord, toute prise en compte de la sémantique d'un domaine donné requiert d'implémenter un modèle spécifique, fut-ce en utilisant le lambda calcul ou tout autre système. De plus, le paradigme orienté objet, grâce au mécanisme d'abstraction, offre une possibilité unique de "capturer la généralité". La configuration permet d'affiner le type des objets jusqu'à la sous-classe appropriée durant la recherche. Ainsi, il est suffisant de savoir que chaque phrase est associée à un schéma issu de la plus abstraite des classes "Schéma". Le bon type de schéma sera sélectionné lors de l'exécution par la propagation des contraintes. Par ailleurs, les résultats produits par une telle approche sont des objets interconnectés, facilement sérialisables (en xml par exemple) et donc utilisables dans de très nombreuses applications, des bases de données à des systèmes de raisonnement complexes. Ces travaux préservent une propriété essentielle des parseurs basés sur l'utilisation de Prolog : le programme peut être utilisé aussi bien de manière analytique que de manière générative, pour produire des phrases qui décrivent une situation.

Les recherches en cours s'attachent au traitement des attributs et d'une partie des déterminants quantifiés universellement. Parmi les perspectives les plus saillantes, on peut évoquer la prise en compte des articles indéfinis et des définis pluriels ("Chaque PC possède une carte mère."), dont la sémantique universelle relève de l'extension du monde. Des phrases contenant de telles constructions peuvent être vues comme des sources d'apprentissage du monde pour le programme. Une application de taille réelle de ces travaux dans le cadre de la modélisation déclarative de surfaces est en cours d'implémentation. Cette application attendra en entrée un texte décrivant une surface 3D. A partir d'un modèle prédéfini de la sémantique/du monde, et de l'entrée descriptive, nous proposons la modélisation d'une solution. Ainsi le programme capable de traiter ces informations peut les utiliser pour tracer la surface décrite informellement par l'utilisateur.

Références

[1] Philippe Blache. *Les Grammaires de Propriétés : des contraintes pour le traitement automatique des langues naturelles*. Hermès Sciences, 2001.

[2] Alain Colmerauer. Un sous-ensemble interressant du français. *RAIRO*, 13(4) :309–336, 1979.

[3] Alain Colmerauer. An introduction to prolog 3. *communication of the ACM*, 33(7) :69–90, 1990.

[4] Mathieu Estratat and Laurent Henocque. Parsing languages with a configurator. In *proceedings of the European Conference for Artificial Intelligence ECAI 2004*, pages 591–595, Valencia, Spain, August 2004.

[5] Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan A. Sag. *Generalized Phrase Structure Grammar*. Blackwell, Oxford, 1985.

[6] Maurice Gross. Lexicon-grammar : the representation of compound words. In *Proceedings of*

the 11th coference on Computational linguistics, pages 1–6. Association for Computational Linguistics, 1986.

[7] Laurent Henocque. Modeling object oriented constraint programs in Z. *RACSAM (Revista de la Real Academia De Ciencias serie A Mathematicas), special issue about Artificial Intelligence and Symbolic Computing*, page to appear, 2004.

[8] Ulrich Junker and Daniel Mailharro. The logic of ilog (j)configurator : Combining constraint programming with a description logic. In *IJCAI 2003 Workshop on Configuration*, 2003.

[9] Hans Kamp. *A theory of Truth and Semantic Representation.* groenendijk et al(eds.), Dordrecht : Foris, 1984.

[10] Richard Montague. *Formal philosophy : selected papers of Richard Montague.* Edited by R. Thomason. yale university press, new haven, 1974.

[11] Robert Pasero. *Un essai de communication sensée en langue naturelle.* Université d'Aix-Marseille, Marseille, 1976.

[12] Robert Pasero and Paul Sabatier. Illico : un système générique de compréhension d'un sous-ensemble du français. Technical report, LIM, November 1999.

[13] Robert Pasero and Paul Sabatier. Specifying and processing constraints on formal representations of sentence. In *6th International Workshop on Natural Language Understanding and Logic Programming, NLULP 99*, pages 33–44, 1999.

[14] Fernando CN Pereira and David Warren. Definite clause grammar for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13 :231–278, 1980.

[15] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar.* The University of Chicago Press, Chicago, 1994.

[16] John F. Sowa. Semantics of conceptual graph. In *Proceedings of the 17h Conference on Association for Computational Linguistics*, pages 39–44, La Jolla, California, 1979.