# Parsing Languages with a Configurator

**Mathieu Estratat** and **Laurent Henocque** [1]

**Abstract.** Recent evolution of linguistic theories heavily rely upon the concept of constraint. Also, several authors have pointed out the similitude existing between the categories of feature-based theories and the notions of objects or frames. We show that a generalization of constraint programs called configuration programs can be applied to natural language parsing. We propose here a systematic translation of the concepts and constraints introduced by property grammars to configuration problems representing specific target languages. We assess the usefulness of this translation by studying first a recursive (context free) language with semantics, then a natural language subset with lexical ambiguities. Our proposal improves over property grammars because the search procedure in our case is generic and does not rely upon an ad-hoc solver. Configuration techniques also extend constraint programming through object orientedness and set variables. Even though dependency grammars exploit set variables themselves, we foresee that the possibility of coupling the parser with constrained object models describing the word semantics is at the advantage of configuration. It also provides natural (if not easy) parser integration of natural language semantics. Our experiments show the practical efficiency of this approach, which does not require the use of ad hoc algorithms and can be freely used in analysis, generative, or hybrid mode.

## 1 Introduction

Recent evolution of linguistic theories heavily rely upon the concept of constraint [12, 2, 3, 4]. According to these formalisms, the validity of a syntactical construct is achieved when feature constraints are satisfied. Two approaches at least make explicit claims of being purely constraint based. The implementation of dependency grammars in [4] exploits a generic CSP (Constraint Satisfaction Problem) framework (within a concurrent constraint programming paradigm) with set variables and selection constraints. The property grammars [2, 3] are presented together with a specific parsing algorithm. Although being very different in nature, these two viewpoints argue that constraint propagation is an efficient tool for disambiguating natural language. Furthermore, the general properties of feature grammars [7] lead several authors to point the resemblance between these notions and frames or objects, and also the need for multiple inheritance [12]. Simultaneously, an evolution of constraint programming in the direction of configuration problems has favored the emergence of efficient configurators with potential AI applications [9, 14]. A tool like JConfigurator [9] heavily relies upon constrained set variables, as does the implementation of dependency grammars in [4]. There, parsing is explicitly referred to as a configuration task. It is worth noting that the Mozart implementation of the Oz language comes with built in feature constraints, with immediate application to natural language processing [16], which further confirms the proximity of the concepts.

Configuring consists in building (a simulation of) a complex product from components picked from a catalog of types. Neither the number nor the actual types of the required components are known beforehand. Components are subject to relations (this information is called "partonomic"), and their types are subject to inheritance (this is the taxonomic information). Constraints (also called well formedness rules) generically define all the valid products. A configurator expects as input a fragment of a target object structure, and expands it to a solution of the problem constraints, if any. This problem is undecidable in the general case. Such a program is well described using an object model (as illustrated by the figures 3 and 5), together with well formedness rules. Technically solving the associated enumeration problem can be made using various formalisms or technical approaches : extensions of the CSP paradigm [10, 6], knowledge based approaches [15], terminological logics [11], logic programming (using forward or backward chaining, and non standard semantics) [14], object-oriented approaches [9, 15]. Our experimentations were conducted using the object-oriented configurator Ilog JConfigurator [9].

Configurators have proved their capacity to handle complex constrained object models in many industrial situations. By doing so, they address the general goal of dealing with the semantics of some field of knowledge. The dominating approach to natural language parsing views syntax and semantics as two separate issues, addressed by different formalisms (e.g. HPSG[2] for syntax and lambda calculus plus logic for semantics). We relate here experiments conducted with a radically different viewpoint, an answer to the question : can a configurator deal with both the semantics and the syntax of a given knowledge field? Our leading intuition is that the process of building a parse tree is very similar to a configuration activity (since new constructs are introduced to group words in adequate categories like the verb or noun phrases, and these constructs are linked by relations). Among potential benefits are the fact that we may foresee to handle exact semantics for languages dealing with specific knowledge, and also that efficient cooperation between syntactic and semantics constraints can be obtained.

We present here an application of configuration to the problem of parsing languages, a work that originates from [5]. To this end, and because we aim at parsing natural languages, we do not propose an ad hoc formulation of the problem, but propose a systematic translation of property grammars [3] in terms of a configuration model. As a means of assessing the validity of the proposed approach, we detail two working examples : one is the parsing of the archetypal context free grammar $a^n b^n$, and the other is the parsing of a simple natural language subset. The $a^n b^n$ grammar is recursive, which

[1] LSIS - UMR CNRS 6168, Faculté des Sciences et Techniques de Saint-Jérôme, Avenue Escadrille Normandie-Niemen, 13397 Marseille cedex 20, France, Université d'Aix-Marseille III, email : mathieu.estratat@lsis.org, laurent.henocque@lsis.org

[2] Head-driven Phrase Structure Grammar [12]

despite its apparent simplicity addresses an inherent difficulty of natural language (noun phrases are recursive). In this example, a single constrained object model describes both the syntax and the underlying simple semantics. The other example is a lexically ambiguous subset of natural language. In both cases, we show that the configurator used for parsing can be exploited in a mixed analysis/generative fashion, and that object constraint propagation allows for solving the parsing problems in little or no search.

There are several motivations for placing ourselves in the context of property grammars (instead of dependency grammars for instance), which highlight the original contribution of this work. Property grammars use categories as do most CL theories since GPSG[3] and HPSG. This gives access to a wide corpus of research and existing grammars. As such, [3] presents a grammar of french rich enough for many serious applications. The seven kinds of constraints in property grammars are very easy to translate as configuration constraints, and are also much easier to read or understand than some of their dependency grammar equivalents. Furthermore, some of the power of a CSP based implementation of dependency grammars has to do with the fact that no intermediate categories are needed, hence that the total size of the problem is known at start. This turns out to be a limitation when it makes the grammar constraints awkward to formulate. Also, standard CSPs are long known as too limited for real configuration tasks [10, 9, 8, 14, 1]. In particular, the dynamic nature of configuration problems must be accounted for, at least with activity variables as in dynamic CSPs. In our case, when the semantics of sentences refer to objects newly introduced in the discourse, no CSP based approach will be suitable for dealing with them. Thus by using a configuration viewpoint over both the semantics and syntax of a language, we gain a unique possibility of intermixing both aspects of language parsing in a single framework. From a constraint programming standpoint, this leads to potentially optimal constraint propagation between both sub-problems, as well as built in mixed analytic/generative parser operation.

## 1.1 Property grammars

Property grammars [2, 3] are a constraint-based linguistic formalism. [3] both proposes a classification of feature constraints called *properties*, and a parsing algorithm that attempts to exploit constraint propagation (although in a rather "ad hoc" fashion since no "standard" constraint system is used) so as to solve syntactic ambiguities as early as possible. Algorithms left aside, property grammars involve two important notions : *categories* represent all recognized syntactic units (either linked with an isolated word or a word group), and *properties* (a synonym for constraints) apply to these categories, so as to specify well formedness grammar rules and phrase cohesion rules. Categories are mapped to both the words and the phrases of a sentence. For example, the figure 8 shows that "la porte" is a $noun\text{-}phrase$ where "la" is a $determiner$ and "porte" a $noun$. To each of this three words or word groups is associated a category, $NP$, $Det$ and $N$ respectively.

## 1.2 Categories and constraint programming

Categories are feature structures. A feature structure is a set of $(attribute, value)$ pairs used to label a linguistic unit, as figure 1 illustrates, where *son* is a *masculine noun*, at the *singular*, $3^{rd}$ *pers*. This definition is recursive : a feature value can be another feature structure, or a set of features.

Functionally, a feature can be mapped to a CSP variable, and a feature structure can be seen as a binding of values to an aggregate

[3] Generalized Phrase Structure Grammar [7]

$$\left[\begin{array}{lll} \textit{Cat:} & \text{N} & \\ \textit{Phon:} & \text{son} & \\ \textit{Agreement:} & \left[\begin{array}{ll} \textit{Gen :} & \text{masc} \\ \textit{Num :} & \text{sing} \\ \textit{Per :} & \text{3rd} \end{array}\right] & \\ \textit{Case :} & \{\text{Common}\} & \end{array}\right]$$

**Figure 1.** An instance of the category N

of feature variables. A feature value can be a constant from a specific domain (for instance an enumeration as $\{Singular, Plural\}$, or an integer as $\{1(st), 2(nd), 3(rd)\}$). A feature value can also be a (set of, list of) feature structure(s) (as $Agreement$ in figure 1). Hence standard finite domain CSP variables cannot be used to model features, and a notion of relations, or set variables must be used (as in [9, 15, 4]). It is worth pointing that feature structures are available as a language construct in Oz [13] and support feature constraints.

## 1.3 Properties vs. Constraints

Properties [3] are constraints that apply to categories, and specify syntax well formedness rules and phrase cohesion rules. There are seven kinds of properties : *constituency*, *heads*, *unicity*, *requirement*, *exclusion*, *linearity* and *dependency* detailed in section 2.2. That such linguistic constraints can have a counterpart in a constraint system seems obvious. For instance, linearity (or precedence) can be implemented using an order relation among integers. Some constraints, like constituency or heads, deserve a more object oriented translation involving at least set variables.

## 1.4 Plan of the article

The section 2 describes a mapping from property grammars to configuration problems. Sections 3 and 4 present examples. Section 5 concludes, and presents ongoing and future research.

# 2 From property grammars to configuration

## 2.1 An object model for categories

The structural elements of linguistic formalisms straightforwardly map to configuration problems. A feature corresponds to a *CSP variable*. Feature structures are aggregates well modeled using *classes* in an object model. A category naturally maps to a *class* in an object model, inserted in a class hierarchy involving *inheritance* (possibly multiple [12]). For instance, the category in figure 1 is translated into a class in an object model, as illustrated by figure 2.

Many features have (sets of) feature structures as their values, which can be adequately modeled using *relations* in an object model. (When the configurator used is itself object-oriented [9], such relations are implemented using set variables.) For instance, a noun phrase may have a noun as its head[4] : this relation between categories can be adequately modeled using a relation in the corresponding object model as illustrated in figures 3 and 5.

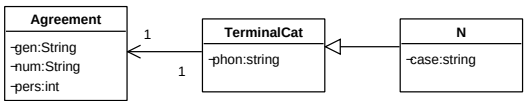


**Figure 2.** An object model for the category N

[4] Informally speaking, the head is the core element in a phrase, the one that governs its linguistic properties

## 2.2 Object model constraints for properties

Properties define both object model relations and constraints, adjoined to the object model built from a given property grammar. We use uppercase symbols to denote categories (e.g. $S, A, B, C \ldots$). We also use the following notations : when an anonymous relation exists between two categories $S$ and $A$, we denote as $s.A$ the set of $A$s linked to a given $S$ instance $s$, and as $|s.A|$ their number. For simplicity, and wherever possible, we will use the notation $\forall SF(S)$ (where F is a formula involving the symbol S) rather than $\forall s \in SF(s)$. Class attributes are denoted using standard dotted notation (as e.g $a.begin$ that represents the $begin$ attribute for object $a$). $I_A$ denotes the set of all available indexes for the category $A$.

- **Constituents** : $Const(S) = \{A_m\}_{m \in I_A}$ specifies that an S may only contain elements from $\{A_m\}$. This property is described by using relations between $S$ and all $\{A_m\}$, as shown in the object models presented in figures 3 and 5.
- **Heads** : The $Heads(S) = \{A_m\}_{m \in I_A}$ property lists the possible heads of the category $S$. The heads element is unique, and mandatory. For example, $Heads(NP) = \{N, Adj\}$. The word "door" is the *head* in the $NP$ : "the door". The $Head$ relation is a subset of $Const$. Such properties are implemented using relations as for constituency, plus adequate cardinality constraints.
- **Unicity** : The property $Unic(S) = \{A_m\}_{m \in I_A}$ specifies that an instance of the category $S$ can have at most one instance of each $A_m, m \in I_A$ as a constituent. *Unicity* can be accounted for using cardinality constraints as e.g : $\forall S | \{x : S.Const \mid x \in A_m\}| \leq 1$ which for simplicity in the sequel, we shall note $|S.A_m| \leq 1$. For instance, in an $NP$, the determiner $Det$ is unique.
- **Requirement** : $\{A_m\}_{m \in I_A} \Rightarrow_S \{\{B_n\}_{n \in I_B}, \{C_o\}_{o \in I_C}\}$ means that any occurrence of all $A_m$ implies that all the categories of either $\{B_n\}$ or $\{C_o\}$ are represented as constituents. As an example, in a noun phrase, if a common name is present, then so must a determiner ("door" does not form a valid noun phrase, whereas "the door" does). This property maps to the constraint

$$\forall S(\forall m \in I_A \, |S.A_m| \geq 1) \Rightarrow \\ ((\forall n \in I_B \, |S.B_n| \geq 1) \vee (\forall o \in I_C \, |S.C_o| \geq 1))$$

- **Exclusion** : The property $\{A_m\}_{m \in I_A} \nLeftrightarrow \{B_n\}_{n \in I_B}$ declares that two category groups mutually exclude each other, which can be implemented by the constraint :

$$\forall S, \begin{cases} (\forall m \in I_A \, |S.A_m| \geq 1) \Rightarrow (\forall n \in I_B \, |S.B_n| = 0) \\ \wedge \\ (\forall n \in I_B \, |S.B_n| \geq 1) \Rightarrow (\forall m \in I_A \, |S.A_m| = 0) \end{cases}$$

  For example, a $N$ and a $Pro$ can't cooccur in a $NP$. (Note that in the formulation of these constraints, $\Rightarrow$ denotes logical implication, and not the requirement property.)
- **Linearity** : The property $\{A_m\}_{m \in I_A} \prec_S \{B_n\}_{n \in I_B}$ specifies that any occurrence of an $\{A_m\}_{m \in I_A}$ precedes any occurrence of an $\{B_n\}_{n \in I_B}$. For example, in an $NP$ a $Det$ must precede an $N$ (if present). Implementing this property induces the insertion in the representation of categories in the object model of two integer attributes $begin$ and $end$ that respectively denote the position of the first and last word in the category. This property translates as the constraint :

$$\forall S \, \forall m \in I_A \, \forall n \in I_B, \\ max(\{i \in S.A_m \bullet i.end\}) \leq min(\{i \in S.B_n \bullet i.begin\})$$

- **Dependency** : This property states specific relations between distant categories, in relation with text semantics (so as to denote for instance the link existing between a pronoun and its referent in a previous sentence). For instance, in a verb phrase, there is a dependency between the subject noun phrase and the verb. This property is adequately modeled using a relation.

Properties can therefore be translated as independent constraints. It is however often possible to factor several properties within a single modeling construct, most often a relation and its multiplicity. For instance, constituency and unicity can be grouped together in some models where one relation is used for each possible constituent (we made this choice in the forthcoming examples, in figures 3 and 5).

## 3 Application to the context free language $a^n b^n$

We now present an application of the previous translation to the description of the language $a^n b^n$, archetypal representative of context free grammars. In [3] the langage $a^n b^n$ is defined using the following properties :

$$\begin{cases} Constituency : & Const(S) = \{S, a, b\}; \\ Heads : & Heads(S) = \{a\}; \\ Unicity : & Unic(S) = \{S, a, b\}; \\ Requirement : & a \Rightarrow b; \\ Linearity : & a \prec b; a \prec S; S \prec b; \end{cases}$$

We implemented this grammar using the object model listed in figure 3 and its associated model constraints. In this model, the classes $S$,

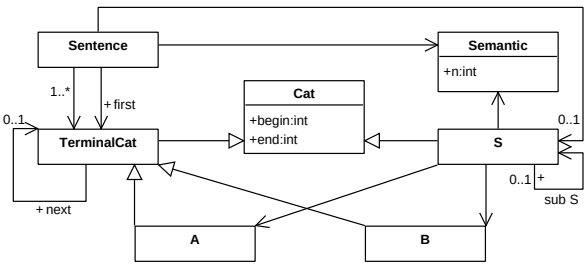


**Figure 3.** An object model for $a^n b^n$

$A$, and $B$ correspond to the categories introduced in [3]. The class $Cat$ is an abstraction for all the categories. It provides the attributes $begin$ and $end$ required for the proper statement of linearity constraints. The class $TerminalCat$ is an abstraction for the terminal categories, here $A$ and $B$. The class $Sentence$ is related to its list of "words" : instances of specific subclasses of $TerminalCat$. It also relates to the first word in this list. The word list is implemented using an extra attribute $next$ in the class $TerminalCat$. $Sentence$ is related with its $Semantic$ and with an $S$ (each sentence is linked with both its syntax and semantic representation). $S$ is related with $Semantic$ : each non terminal category has an associated semantic. Linearity properties translate to additional constraints :

$$\begin{cases} \forall S \, , \, S.A.begin < S.B.begin; \\ \forall S \, , \, (|S.S| == 1) \Rightarrow (S.A.end \leq S.S.begin); \\ \forall S \, , \, (|S.S| == 1) \Rightarrow (S.S.end \leq S.B.begin); \end{cases}$$

### 3.1 Semantics

The semantics one can bind to a valid sentence of the language $a^n b^n$ is obviously the number $n$ of $a$s and $b$s that can be counted. To this end, the model class $Semantic$ has an integer attribute $n$. The number $S.Semantic.n$ denotes the number of $A$s in an $S$. The constraints that bind the syntax and the semantic are :

$$\begin{cases} \forall S\ (|S.S| == 1) \Rightarrow S.Semantic.n = 1 + S.S.Semantic.n \\ \forall S\ (|S.S| == 0) \Rightarrow S.Semantic.n = 1 \\ \forall Sentence\ Sentence.Semantic = Sentence.S.Semantic; \end{cases}$$

These axioms recursively define the semantic of an $S$ as the number of its enclosed $A$s, and the semantic of a sentence as that of its toplevel non terminal category.

### 3.2 Parsing

The configurator is exploited as follows : it expects as input a group of partially known objects (for example some instances of the category $TerminalCat$), that are partially interconnected (via the relation $next$). It tries to complete this input by adding further objects and interconnections, and by deciding for the actual type of all objects as well as for the actual value of all attributes so as to satisfy all the model constraints.

The figure 4 illustrates several possible sessions with the parser. In this table, the system states are described as triples $\langle words, syntax, semantic\rangle$ ("?" denotes an unknown object, and "$\diamond$" an unknown word), and the system behaviour is denoted using $input\ state \mapsto output\ state$ rules. The first two lines correspond to pure analysis use cases. In the second line, the sentence does not belong to the language. The last two lines in figure 4 illustrate generative or hybrid uses of the the program.

$$\begin{cases} \langle aaabbb, ?, ?\rangle \mapsto \langle aaabbb, S(a, S(a, S(a, null, b), b), b), 3\rangle \\ \langle abbb, ?, ?\rangle \mapsto false \\ \langle \diamond\ a\ \diamond\ b, ?, ?\rangle \mapsto \langle aabb, S(a, S(a, null, b), b), 2\rangle \\ \langle ?, ?, 2\rangle \mapsto \langle aabb, S(a, S(a, null, b), b), 2\rangle \end{cases}$$

**Figure 4.** Some parser sessions

### 3.3 Experimental results

Table 1 lists results obtained[5] for both consistent and inconsistent entries of various sizes. The first column lists the configurator input and the sixth column the elapsed time in seconds. These results

**Table 1.** Experimental results

| $p$ | $\#fails$ | $\#cp$ | $\#csts$ | $\#vars$ | $\#secs$ |
|---|---|---|---|---|---|
| $aaabbb$ | 0 | 29 | 370 | 143 | 0.48 s |
| $\diamond\ a\ \diamond\ b$ | 0 | 22 | 469 | 119 | 0.39 s |
| $a(10)\ b(10)$ | 0 | 92 | 1672 | 430 | 0.77 s |
| $a(20)\ b(20)$ | 0 | 182 | 4892 | 840 | 1.33 s |
| $a(50)\ b(50)$ | 0 | 52 | 24152 | 2070 | 4.74 s |
| $a(51)\ b(49)$ | 1 | 0 | 1687 | 1417 | 3.92 s |
| $\diamond\ a(50)\ b(49)$ | 1 | 0 | 1684 | 1416 | 5.12 s |

show the efficiency of constraint propagation which allows to reach the solution with a very limited number of fails, specially in analysis mode. That the program exits with no choice point when the entry is inconsistent ($a(51)\ b(49)$ and $\diamond\ a(50)\ b(49)$) stems from the fact that constraint propagation alone detects failure. This is so because most relations have a multiplicity of 1, which triggers efficient propagations. Natural like languages as illustrated in the forthcoming example do not generally share this property. Note that this behavior can be obtained thanks to a straightforward symmetry breaking constraint that removes symmetries artificially introduced in the JConfigurator model[6]: we make sure that the values of the *begin* attributes of all $S$ instances are strictly increasing.

Computation times have been listed for information but are strongly impacted by the cost of building the constrained object model in Java.

[5] PC used : P4 2,4GHz - 512 Mo DDR - Windows XP SP1 - Java 2 V.1.4.2 - Ilog JConfigurator 2.1

## 4 Parsing a lexically ambiguous natural language

The figure 5 represents a fragment of the constrained object model for a subset of french, where constituency and unicity are made explicit. The figure 6 illustrates some well formedness constraints. In the figure 7 we define a small example lexicon.

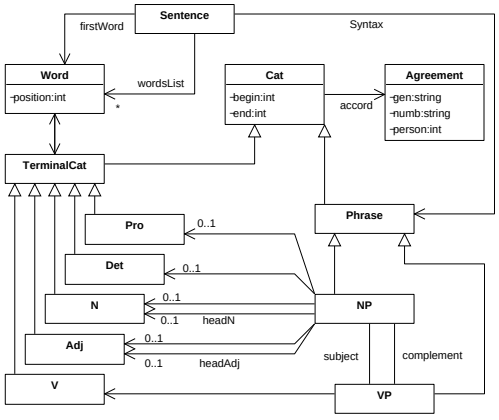


**Figure 5.** Object model used to parse our language

The language accepted by this constrained object model is made of phrases constructed around a subject, a verb and a complement, where both the subject and the verb are mandatory, and both the subject and the complement are noun phrases.

$Head : |NP.headN| + |NP.headAdj| = 1;$
$Linearity : Det < N;\ Det < Adj;$
$Exclusion : (|NP.N| >= 1) \Rightarrow (|NP.Pro| = 0)$ and
$(|NP.Pro >= 1) \Rightarrow (|NP.N| = 0);$
$Requirement : (|NP.N| = 1) \Rightarrow (|NP.det| = 1)$

**Figure 6.** Some NP constraints

Both constraints are stated straightforwardly within the object model via relations and their cardinalities (as can be seen in the figure 6). More constraints are required however, like the constraints stating that a Head is a constituent, or the constraints ruling the value of the $begin$ and $end$ attributes in syntagms.

[6] the current version of JConfigurator does not implement creation by necessity, and requires the preliminary creation of interchangeable "S" instances

### 4.1 Experimental results

We tested the object model with sentences involving variable levels of lexical ambiguity, as from the lexicon listed in figure 7.

| WORD | CAT | GEN | NUM | PERS |
|---|---|---|---|---|
| ferme | N | fem | sing | 3 |
| ferme | Adj | - | sing | - |
| ferme | V | - | sing | 1,3 |
| la | Det | fem | sing | 3 |
| la | Pro | fem | sing | 3 |
| mal | N | masc | sing | 3 |
| mal | Adj | - | - | - |
| porte | N | fem | sing | 3 |
| porte | V | - | sing | 1,3 |

**Figure 7.** A lexicon fragment

Sentence $(1)$, "la porte ferme mal" (*the door doesn't close well*) is fully ambiguous. In this example, "la" can be a *pronoun* (i.e. *it* as in "give **it** to me !") or a *determiner*(like *the* in "**the** door"), "porte" can be a *verb* (i.e. *to carry*) or a noun (*door*), "ferme" can be a *verb* (*to close*), a *noun* (*farm*) or an adjective (*firm*) and "mal" can be an *adjective* (*badly*) or a *noun* (*pain*). Our program produces a labeling for each word and the corresponding syntax tree (figure 8).

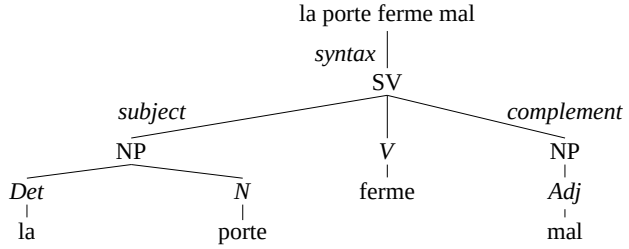


**Figure 8.** Syntax tree for the french sentence "la porte ferme mal"

Sentence $(2)$ is "la porte bleue possède trois vitres jaunes" (*the blue door has three yellow windows*). Here "bleue" and "jaunes" are *adjectives*, "vitres" is a *noun* and "trois" is a *determiner*. The last sentence $(3)$, "le moniteur est bleu" (*the monitor is blue*) involves no ambiguous word. The table 2 presents results obtained for the sentences (1), (2) and (3). These results show that the correct labeling is

**Table 2.** Experimental results for french phrases

| $p$ | $\#fails$ | $\#cp$ | $\#csts$ | $\#vars$ | $\#secs$ |
|---|---|---|---|---|---|
| $(1)$ | 4 | 40 | 399 | 220 | 0.55 s |
| $(2)$ | 3 | 50 | 442 | 238 | 0.56 s |
| $(3)$ | 1 | 35 | 357 | 194 | 0.52 s |

obtained after very few backtracks ($fails$). The number of fails depends on number of ambiguous words in the sentence. The execution time also depend upon the sentence length. The remaining fail in $(3)$ stems form the fact that a default minimization search is performed.

## 5 Conclusion

We have described a translation of property grammars into a configuration problem. We also showed that a generic search procedure, combined with built in constraint propagation not only allows one to solve the problem very efficiently (the Java program can parse in just a few seconds sentences up to a hundred words), but offers all the possibly expected interaction modes. The capacity to achieve sentence completion, or generation, has many practical applications.

Our proposal improves over property grammars because the search procedure in our case is generic and does not rely upon an ad-hoc solver. Configuration techniques also extend constraint programming through object orientedness and set variables. Even though dependency grammars exploit set variables themselves, we foresee that the possibility of coupling the parser with constrained object models describing the word semantics is at the advantage of configuration. Last, the truly logical formulation of configuration constraints avoids the difficulties inherent to backward or forward chaining rule based constraint programming. We expect our forthcoming results to confirm those intuitions. Ongoing research involves the implementation of parser for a natural language subset of french dealing with the semantics of three dimensional scene descriptions.

## REFERENCES


[1] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis, 'Consistency restoration and explanations in dynamic csps—-application to configuration', *Artificial Intelligence*, **135**(1-2), 199–234, (2002).

[2] P. Blache, 'Property grammars and the problem of constraint satisfaction', in *ESSLLI-2000 workshop on Linguistic Theory and Grammar Implementation*, (2000).

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

[4] Denys Duchier, 'Axiomatizing dependency parsing using set constraints', in *Sixth Meeting on Mathematics of Language, Orlando, Florida*, pp. 115–126, (1999).

[5] Mathieu Estratat, *Application de la configuration à l'analyse syntaxico sémantique de descriptions*, Master's thesis, Faculté des Sciences et Techniques de Saint Jérôme, LSIS équipe InCA, Marseille, France, submitted for the obtention of the DEA degree, 2003.

[6] G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner, 'Configuring large-scale systems with generative constraint satisfaction', *IEEE Intelligent Systems - Special issue on Configuration*, **13(7)**, (1998).

[7] G. Gazdar, E. Klein, G.K. Pullum, and I.A. Sag, *Generalized Phrase Structure Grammar*, Blackwell, Oxford, 1985.

[8] Andreas Günter and Christian Kühn, 'Knowledge-based configuration - survey and future directions', in *5th Biannual German Conference on Knowledge Based Systems, Würzburg, Germany, Lecture Notes in Artificial Intelligence LNAI 1570*, pp. 47–66, (March 1999).

[9] D. Mailharro, 'A classification and constraint based framework for configuration', *AI-EDAM : Special issue on Configuration*, **12(4)**, 383 – 397, (1998).

[10] Sanjay Mittal and Brian Falkenhainer, 'Dynamic constraint satisfaction problems', in *Proceedings of AAAI-90*, pp. 25–32, Boston, MA, (1990).

[11] B. Nebel, 'Reasoning and revision in hybrid representation systems', *Lecture Notes in Artificial Intelligence*, **422**, (1990).

[12] C. Pollard and I.A. Sag, *Head-Driven Phrase Structure Grammar*, The University of Chicago Press, Chicago, 1994.

[13] Gert Smolka and Ralf Treinen, 'Records for logic programming', *The Journal of Logic Programming*, **18**(3), 229–258, (April 1994).

[14] Timo Soininen, Ilkka Niemela, Juha Tiihonen, and Reijo Sulonen, 'Representing configuration knowledge with weight constraint rules', in *Proceedings of the AAAI Spring Symp. on Answer Set Programming: Towards Efficient and Scalable Knowledge*, pp. 195–201, (March 2001).

[15] Markus Stumptner, 'An overview of knowledge-based configuration', *AI Communications*, **10(2)**, 111–125, (June 1997).

[16] Peter Van Roy, Per Brand, Denys Duchier, Seif Haridi, Martin Henz, and Christian Schulte, 'Logic programming in the context of multiparadigm programming: the Oz experience', *Theory and Practice of Logic Programming*, (2003). To appear.