Customizable Modular Lexicalized Parsing

R. Basili, M.T. Pazienza, F.M. Zanzotto
Dipartimento di Informatica, Sistemi e Produzione,
Universita' di Roma Tor Vergata (ITALY)
{basili,pazienza,zanzotto}@info.uniroma2.it

Abstract

Different NLP applications have different efficiency constraints (i.e. quality of the results and throughput) that reflect on each core linguistic component. Syntactic processors are basic modules in some NLP application. A customization that permits the performance control of these components enables their reuse in different application scenarios. Throughput has been commonly improved using partial syntactic processors. On the other hand, specialized lexicons are generally employed to improve the quality of the syntactic material produced by specific parsing (sub)process (e.g. verb argument detection or PP-attachment disambiguation). Building upon the idea of grammar stratification, in this paper a method to push modularity and lexical sensitivity, in parsing, in view of customizable syntactic analysers is presented. A framework for modular parser design is proposed and its main properties are discussed. Parsers (i.e. different parsing module chains) are then presented and their performances are analyzed in an application-driven scenarios.

1 Introduction

NLP applications require efficient NLP core components both in terms of linguistic quality and throughput. Different NLP applications have different efficiency constraints and this reflects on each component. Several text processing applications include syntactic parsers as core components. Customizing parsing processors enable the reuse of these components in different application scenarios.

Let us consider as an example a real time application like a front-end question-answering for on-line services. Here fast are preferred to accurate parsing processors. Target sentences are rather simple and structures are recurrent. For example, booking train tickets is often expressed by sentences like: *At what time is the next train from Rome to Paris?*. A parsing processor able to produce partial structures like *At what time* and *from Rome to Paris* is sufficient to support a deductive machinery that answers the question. Complex analysis, e.g. clause boundary recognition, is not relevant as very short sentences (with high expectation about the discourse domain) are always used.

On the other hand, an event recognition (ER) task in an Information Extraction (IE) [20, 21] scenario asks for accurate syntactic material over complex sentences. As an example, let us consider the Penn Tree-bank [19] sentence #1692(9):

(*wsj_*1692(9)) *As part of the agreement, Mr. Gaubert contributed real estate valued at $ 25 million to the assets of Independent American.*

The focus here is on the extraction of the event mainly suggested by syntactic relations established by the verb *to contribute*. Clause embedding, i.e. *valued at $ 25 million*, plays here an important role. As a consequence, deeper parsing, relying on a more expressive grammar, is mandatory.

It is also worth noticing that applications may differ in the *type of necessary syntactic relations*. In order to limit the time complexity, underlying grammars should thus be designed to efficiently cover specific phenomena of interest. *All and only* the information necessary to cover the specific target phenomena with the suitable quality should be used.

A key issue is that limited coverage (i.e. low time complexity) and high confidence are conflicting requirements for the kind of grammatical competence available to the parser. Application developers search for technologies for the largest coverage and confidence of specific phenomena. *Shallow parsing* [1, 2, 9, 4], introduced in the perspective of improving time performances, is, alone, inherently weak and often lexical sensitivity has been suggested as successful approach. In order to increase accuracy, syntactic parsing processors usually exploit lexical information; lexicalized grammar formalisms have been widely proposed (e.g. HPSG [22], LTAG [16], LFG [12]) at this scope, although in frameworks (i.e. linguistic theories) targeted to full syntactic analysis [13]. On the contrary, applications require effective methods for specific phenomena. Flexibility is thus the crucial factor for the success of parsing technologies in applications. It is our opinion that the integration of lexicalized approaches in frameworks for shallow parsing [7] is a relevant area of research.

Building upon the idea of grammar stratification [1], we propose a method to push modularity and lexical sensitivity in parsing in view of customizable syntactic analysers.

Supporting modularity within the parsing process requires:

- a formal and homogeneous definition/representation for the partial parsing results able to support information sharing among subcomponents;
- principles for coherent composition of parsing subcomponents able to ease the design of application specific parsers;
- methods for the systematic control of ambiguity within as well as among the components(i.e. throughout a chain of interactions);
- the detection of specific language phenomena where lexical information is relevant to the control of ambiguity, so that specific lexicalized components can be designed to reflect it.

In this paper a framework for modular parsing is presented. The principles for the stratification of the grammatical analysis and their implications on modularity are defined in the next section. In Section 3 the notion of syntactic module is introduced and a classification according to basic grammatical properties of the different modules is given. In the same section an annotation scheme useful for information exchange among modules is defined as a combination of a dependency and constituency based formalism. Implications on grammatical properties and parsing architectures are then discussed. Finally, section 4 discusses the evaluation of some parsing architectures within a typical application scenario.

2 Fitting parsing performance through stratification and modularization

The interest of NLP application developers is in customizing a parser in order to meet application quality and time constraints. Final performances depend on the adopted trade-off between the two.

It is widely accepted that computational lexicons increase the quality of the syntactic information produced by a parser [8]: this improvement is tightly dependent on the specific language level to which lexical information refers. The stratification of a grammar resulting from a modular decomposition of the parser should facilitates the use of lexical information specific to each level.

Let us again consider the (*wsj_1692(9)*) example and suppose that verb *subcategorization* information is available. The verb *to contribute* would be associated to a `direct object` and to a `recipient` (or `beneficiary`) argument as well. This would result in a frame like `contribute-NP-PP(to)`[1]. The other verb in the sentence, *to value*, would be associated to its `object` (i.e. the evaluated entity) and to a prepositional phrase expressing the "degree/amount" (usually ruled by the preposition *at*), i.e. `value-NP-PP(at)`.

A strategy using a combination of clause boundary recognition and a verb argument detection algorithms could decide that: (i) *valued* is linked to *at $ 25 million*; (ii) *contributed* is linked to *to the assets*. At the level of PP-attachment, most of the ambiguities in the sample sentence disappear since they are resolved by lexical information. Firstly, links derived on lexical basis (i.e. attachment of verb argumental modifiers) have important effects on the remaining ambiguities: other potential attachment sites of argumental PPs like *at $ 25 million* and *to the assets* are discarded. Secondly, persistent ambiguity is reduced. The $(of\ Independent\ American)_{PP}$ structure is no longer allowed to attach to nouns like *real estate* or *million* as illegal bracket crossing of the clause related to *contribute* would be generated: as a result the only allowed attachments are those with the verb *contribute* itself or with the noun *assets*.

The search space of the parser during the above lexicalized process depends on the number of sentence words. If an early parsing phase, i.e. *chunking* [1], is applied, later parsing steps (e.g. the detection of verb modifiers) deal with a much lower amount of ambiguity. *Chunking* is widely adopted to recognize sentence fragments whose boundaries are independent from the verb grammatical projections. In the example sentence (*wsj_1692(9)*), noun phrases (e.g. *Mr. Gaubert*, *real estate*) and modifiers (e.g. *to the assets*, *at $ 25 million*) are simple examples of these segments. The detection of verb modifiers is disburdened since it has to deal only with the representative elements of the recognized structures.

The above example is a simple instance of a phenomenon (i.e. verb subcategorization) that plays a relevant role in the control of the ambiguity propagation throughout the search space of the parser. The level (i.e. after chunking) in which this algorithm is applied and the used lexical knowledge are crucial for optimizing the derived advantage:

- the use of chunks provide an optimal representation as the search for verb arguments is limited to chunk heads;
- the adopted lexical knowledge (i.e. subcat frames) in this specific process is a well focused components of a lexical KBs;
- the verb argument detection suggested by the example strongly interact with other parsing activities (e.g. detection of non-argumental and nominal modifiers), with positive side-effects on the reduction of ambiguity.

The above properties are not specific to this kind of modular decomposition (i.e. chunking + verb_phrase_parsing) but can be generalized to a variety of other potential decompositions. The

[1] Note that the `subject` is missing as mandatory in syntax, although it can be omitted.

effects of lexical information within each component increase the accuracy with respect to each target specific (sub)problem. Modularity thus optimizes the lexical effects on the control of ambiguity throughout chains of specific parsing steps.

The adoption of a modular view in parsing supports a more flexible design (via composition of simpler subcomponents in different parsing architectures) and the throughput control is explicit.

First throughput constraints can be met via simplification (i.e. removing not crucial subcomponents) of the overall architecture. If a modular design is adopted, functionalities of modules and functional dependencies are well-defined. Eliciting processing capabilities consists in removing modules from the parsing architecture.

Moreover, modularity again helps in the control of losses in accuracy over the target phenomenon due to the removal of modules. As an example, let us consider a parser aiming to determine NP boundaries in order to detect candidate terms within a Terminology Extraction process. The removal of a verb argument recognition module would increase the parser throughput, by reducing also the resulting precision. In the example ($wsj\_1692(9)$), the lack of verb subcategorization information provides, as a potential NP, the wrong excerpt *$ 25 million to the assets of Independent American*. It is only by means of a well-defined notion of verb argument detection component that a systematic measure of the trade-off between accuracy and throughput can be controlled and employed as a design principle.

In the next section, a method for designing modular parsing systems is introduced able to support principles of lexicalization and decomposition.

3 A modular approach to parsing

A syntactic processor SP, according to the classification given in [3], is a *linguistic processing module*. It is a function $SP(S, K)$ that, exploiting the syntactic knowledge K, produces a syntactic representation of the input sentence S.

The stratification of the grammar induces modularization of the syntactic processor. The general module component P_i takes the sentence at given state of analysis S_i and augments this information in S_{i+1} exploiting the knowledge K_i. The parser SP is thus a cascade of this modules.

It is crucial to define how the syntactic information produced and processed is represented. The stratification of the parsing activity requires that the representation scheme adopted satisfies some requirements. In fact, on the one hand, stratified parsing techniques require the handling of partially parsed structures (cf. Sec. 2). On the other, lexicalized approaches require that the heads of some types of phrases are accessible during the analysis. In sec. 3.1, classical representations are discussed from the point of view of a modular perspective. Then, we propose, in sec. 3.2, an annotation scheme that satisfies the two requirements, some properties of the annotation scheme are discussed and some restrictions, i.e. *planarity constraints*, proposed. Finally, a classification of the modules is given in Sec. 3.3 according to the kind of information K used and to the typical actions they perform in augmenting the syntactic knowledge gathered for the input sentence.

3.1 Modularity vs. annotation scheme

Modularization and lexicalization impose strict requirements on the annotation scheme used to describe the syntactic information that the processors gather for a target sentence.

In a modularized approach, a stable representation of partially analyzed structures is crucial. In particular, it is required to handle the representation of long-distance dependencies. For instance, considering the example (*wsj_1692(9)*), at a given state of the analysis could be necessary to express that *contributed* is linked to *to the assets*. In a constituency-based framework [11], it is quite hard to express the above relation without specifying the role of the excerpt *real estate valued at $ 25 million*. Furthermore, in the same framework, the relation between contiguous constituents can not be expressed if the constituent captured is not completely formed. In the excerpt of the example sentence (*wsj_1692(9)*) *contributed real estate valued at $ 25 million*, the relation between *contributed* and *real estate* can be expressed only if the constituent *real estate valued at $ 25 million* has been fully recognized. Extensions of constituency-based theories such as TAG [17] and D-Trees [23] allow to express discontinuous links and partial trees. From this point of view, a dependency-based syntactic [24] representation is preferable, since constituency-based approaches in the annotation are not naturally conceived for the representation of distant dependencies without specifying the role of inner structures. On the other hand, a fully dependency-based syntactic approach generally considers the words of a sentence as basic constituents. Thus, each analyzing step has to deal with the same simple constituents: no packing of information is allowed. However, packing is important in a modular approach. A processor using verb subcategorization frames as suggested in section 2, would be enhanced by looking at the candidate complements as single structures. For instance (*wsj_1692*), the analysis of the complements of the verb `contribute-NP-PP(to)` is disburdened if the candidate excerpt of the sentence were factorized in its chunks *[real estate][valued][at $ 25 million][to the assets][of Independent American]*. In fact, the argument *PP(to)* can be easily filled with the chunk *[to the assets]*.

In a lexicalized approach, it is crucial to determine the *potential governor* [14] of a given structure that is its semantic head [22] and activates lexicalized rules. For instance, given the structure *[has widely contributed]*, the annotation scheme should allow to express that the lexical item governing its behavior is *contribute*.

3.2 Extended dependency graph

To satisfy the requirements imposed by the modularization and the lexicalization, the adopted annotation scheme is a combination of the constituency-based and the dependency-based formalisms. Basically, the syntactic information associated to a given sentence is gathered in a graph, i.e. $g = (n, a)$. The typed nodes (i.e. elements of n) of the graph g are the basic constituents of the sentence, while the typed and oriented arcs (i.e. elements of a) express dependencies between constituents (an head and a modifier). Since the order of constituents is important, the set n is an ordered set. For the purposes of the syntactic parsing, nodes can represent sequences of words, i.e. constituents, that can degenerate in a single word. To satisfy the constraint arisen by the lexicalization, a function h that spot the head of each constituent has been introduced. The representation should allow to express the type of each constituent and each arc. The possible types, elements, respectively, of the sets $NTAG$ and $ATAG$, depend on the underlying grammar model. In the following, we refer to those representation graphs as eXtended Dependency Graph (XDG) that is defined as follows:

Def. 1

An XDG is a tuple $XDG =< n, a, Ntag, Atag, h >$ where n are the nodes, a are the arcs, $Ntag$ is the function that relates n with the set of $NTAG$, $Atag$ is the function that relates a with the set of $ATAG$, and h is the function that elects for each node a representing head.

For sake of simplicity, we introduce a compact version $G = (N, A)$ of the XDG. The compact version is a transcription of the XDG defined as follows:

Def. 2
$G = (N, A)$ related to XDG is such that $N = \{(node, tag, head) | node \in n, tag = Ntag(n), head = h(n)\}$ and $A = \{(arc, tag) | arc \in a, tag = Atag(arc)\}$.

The proposed XDG allows to model the grammatical information, i.e. the detected relation and persisting ambiguity, in an efficient way. In an XDG alternative interpretations coexist. In general, more than one interpretations projected by the same nodes are expressed by the same representation graph that, by itself, do not allow multiple interpretations of the nodes. The ambiguity at this level can be modeled with an inherent proliferation of the interpretation graphs. This limitation is an inheritance of the dependency-based theory. Generally, in these theories, words in an interpretation representation belongs to exactly a single word class (cf. [10]).

The XDG represents a single syntactic interpretation only if it is a *dependency tree* (defined in [10]). In term of constraints on the XDG, the requirement translates in the property that forbids multi-headed nodes [24]:

Prop. 1: *Single headed nodes*
if $\exists (a, b) \in A$ then $\forall a' \in N$ then $\nexists (a', b) \in A$.

For instance, in the example ($wsj\_1692(9)$), an interpretation willing to be a single unambiguous syntactic representation of the sentence can not include both the relations *([valued],[at $ 25 million])* and *([contributed],[at $ 25 million])*.

In order to preserve the compatibility in the proposed representation with the constituency based approach, the property (*Prop. 1*) is not enough. Not enabling *crossing links* may be required. *Crossing links* are defined as follows:

Def. 3: *Crossing links*
Two links, $(w_h, w_k), (w_m, w_n) \in A$ where $min\{h, k\} < min\{m, n\}$, are *crossing* **iff** $min\{m, n\} < max\{h, k\} < max\{m, n\}$.

The *planarity* property [15] can, thus, be introduced:

Prop. 2: *Planarity*
$\forall l_1, l_2 \in A . l_1, l_2$ are not crossing.

The two properties, *Prop. 1* and *Prop. 2*, are called *planarity constraints* and make a XDG that satisfies them a *planar graph*. An XDG satisfying *planarity constraints* is a single (partial) syntactic interpretation.

Consequently, since a viable single interpretation of the sentence must be a *planar graph*, an interpretation in which crossing links coexist is ambiguous. In the example, if both the relations *([valued],[of Independent America])* and *([contributed],[to the assets])* coexist, the interpretation is ambiguous.

3.3 Parsing modules

A component P of the modular syntactic parser is a processor that, using a specific set of rules R, adds syntactic information to the intermediate representation of the sentence. Formally, a processor P is a function $P(R, G)$ where R the knowledge expressed in a specific set of rule, and G the input graph. The result $P(R, G) = G'$ is still an XDG.

The syntactic parser modules are classified according to the actions they perform on the sentence, and to the information they use to perform these actions.

The actions that modules perform on the input XDG can be *conservative* or *not-conservative*. In the case of *conservative modules*, all the choices contained in the input graph are preserved in the

output. The property is not true for the *not conservative modules*. A *conservative* module results in a *monotonic* function of the module. A *not-conservative* module is a *not-monotonic* function. A syntactic processor is a cascade of processing modules. Note that the composition of modules preserve, where it exists, the monotonicity.

Furthermore, since the representation of the syntactic information is an *XDG*, the ability of the modules refers to: (i) **constituent** gathering; (ii) **dependency** gathering. Under this distinction, processors are:

- **constituent processors**, P_c, that are purely constituent gatherer;
- **dependency processors**, P_d, that are purely dependency gatherer;
- **hybrid processors**, P_h, that perform both dependency and constituent gathering.

Starting by a model of the process as previously described, i.e. $P(R,G) = G'$, where $G = (N, A)$ and $G' = (N', A')$, and by the distinctions introduced, a description of the typology of processing modules used in the whole parsing processor will be provided. The description is in term of the action they perform on the syntactic graph.

The main characteristic of the processors of the typology P_c is that, in the changing of the constituents (i.e. nodes of the representation) the arcs between constituents are coherently translated, i.e. for each arc in A, there is the correspondent arc in A' if it connects different nodes. For this typology of modules, a *monotonic* processor P_c^M preserves the property of not crossing-brackets between the input and the output, i.e. N' is a partition of N or vice-versa. A *not-monotonic* processor P_c^{NM} does not satisfy this property. In the *monotonic* processors, we distinguish:

$[P_{c.1}^M]$ N' is a partition of N and $A' = \{(a,b) | a \neq b, a = (a_1, \ldots, a_n), b = (b_1, \ldots, b_m), (a_j, b_i) \in A\}$

$[P_{c.2}^M]$ N is a partition of N' and $A' = \{(a',b') | a' = h(a), b' = h(b), (a,b) \in A\}$

We now analyze how, according to this taxonomy, a tokenizer T and a chunker [1] can be classified. The aim of a tokenizer is to split a sentence $S = c_1 c_2 \ldots c_m$ represented by a stream of characters in its composing words $S' = w_1 w_2 \ldots w_n$. It is a $P_{c.1}^M$ module. In fact, the input is a graph whose set of node represents the stream of characters, i.e. $G = (\{(c_1, char, c_1), \ldots, (c_m, char, c_m)\}, \emptyset)$, while the output G' models the words, i.e. $G' = (\{(w_1, token, w_1), \ldots, (w_n, token, w_n)\}, \emptyset)$. The relation between A and A' satisfies the constraint of the module typology. A chunker [1] falls in the typology $P_{c.1}^M$. As, a *Chunker* is a rewriting device of input sentences, according to the available *chunk prototype*(CP) [6]. The objective of the chunker function is to build the chunk representation $cs = ch_1 \ldots ch_m$ corresponding to each input sentence $ws = w_1 \ldots w_n$. Each chunk ch_i is the instance of a chunk prototype in CP and is a sequences of words that does not overlaps other chunk of the sentence. Then, in the proposed framework, the chunker transforms $G = (\{(w_1, m_1, w_1), \ldots, (w_n, m_n, w_n), A)\}$ in $G' = (\{(ch_1, cht_1, h_1), \ldots, (ch_n, cht_n, h_n), A)\}, A')$, where m_i is the pos-tag of the word w_i, cht_i and h_i are respectively the type and the head of the chunk.

The main characteristic of the processor of the type P_d is that in the processing the property $N = N'$ is met. For the *not-monotonic* processors P_d^{NM} of this type no additional property is required. *Monotonic* processor we adopt in the architecture are defined as follows:

$[P_{d.1}^M]$ $A \subseteq A'$, G and G' meet planarity constrains (G and G' represent a single interpretation of the sentence)

$[P^M_{d.2}]$ $A \subseteq A'$, where for each $a \in A'$ the graph $G'' = (N, A \cup \{a\})$ meets planarity constrains, this means that in general a module of this type introduces ambiguity

According to these definitions, a $P^M_{d.1}$ is a $P^M_{d.2}$. Generally, $P^M_{d.1}$ processors gather unambiguous information and are used to trigger $P^M_{d.2}$ processors. These latter are thought to complete the partial information given by $P^M_{d.1}$ processors. Under this taxonomy, a *link parser* [15] is of the $P^M_{d.1}$. In fact, starting from a representation $G = (ts, \emptyset)$ where ts is the set of ordered tokens representing the target sentence, produces a $G' = (ts, A')$ that is a *planar graph*, i.e. meets planarity constraints.

Another classification may be done considering the knowledge that a processor uses to produce modification in the syntactic graph. Here the lexicalization of the grammatical rules plays a crucial role. In this classification, processors are: (i) **lexicon-driven** processors; (2) **grammar-driven** processors. A mildly lexicalized approach is also possible when grammars are only adopted if lexical information is not available. A lexicalized approach usually depends on the availability of accurate information, and it is usually domain dependent. Examples of the lexicalized modules will be given in the next section.

4 Chaos: a modular lexicalized syntactic parser

The major result of the proposed parsing methodology is the possibility of customization given both by the modular and by the lexicalized approaches. Given a set of syntactic processing modules, this results in a range of possible parsers that differ in term of produced syntactic material and performance. In the following sections, we will introduce *Chaos*, (*Chunk Analysis Oriented System*), a customizable parser based on a pool of four modules: the *Chunker*, the *Verb Shallow Analyzer*, the *Shallow Analyzer*, and the *Projector*. We will discuss its adaptability to different applications through the analysis of performance obtained on the standard Penn Treebank [19].

4.1 Linguistic modules and parsing architectures

The stratification of *Chaos* and its parsing processor modules (Fig. 1), reflect the idea that verbs are crucial in controlling the ambiguity at the level of PP-attachment and are important for applications. Thus, the *Chunker* is especially conceived for packing the ambiguity not relevant at the level of PP-attachment. This rely on syntactic categories and on the relative position between words. It processes a POS tagged sentence $ams = (ws, \emptyset)$ and produces a chunked sentence $chunks = (cs, \emptyset)$ using as rules the *chunks prototypes*(see Fig. 1.(1)). According to the classification given in Sec. 3.3, this processor is a *grammar-based constituent gatherer* ($P^M_{c.1}$).

The verb subcategorization structures that play a disambiguating role are exploited by the verb-driven analysis processor *VSP*(Fig. 1.(3)). It is conceived to efficiently extract dependencies that involve verbs as heads ($V - icd$s, i.e. verb inter-chunk dependencies). This processor is a *dependency lexicalized processor*, that can work at different level of lexicalization, of the class $P^M_{d.1}$. *VSP* demands a syntactic graph whose node are chunks, and it works correctly if those chunks are conceived to pack the ambiguity not controlled by verb connections. The module architecture exploits a clause hierarchy approximation (H) via the loop Clause Boundary Recognition (*CBR*) and Verb Shallow Analyzer (*VSA*).

The module of shallow analysis *SP*(Fig. 1.(2)) is designed to express all the syntactic links that are compliant with a particular configuration of the input. It is a *grammar-based dependency module* of

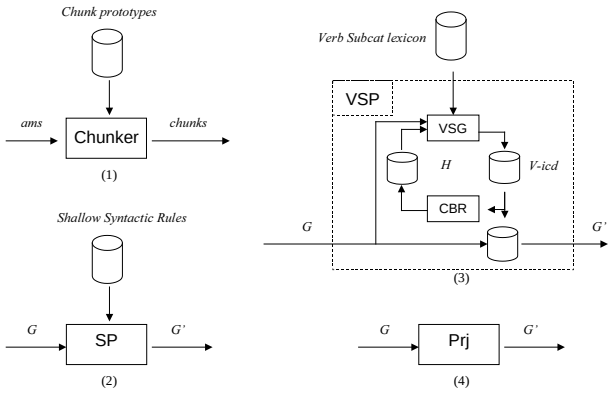

Figure 1: The pool of Chaos processors

the class $P^{M}_{d.2}$.

The module of unambiguous projection *Prj*(Fig. 1.(4)) aims to project a given XDG on the unambiguous subgraph removing colliding arcs. This is a *grammar-based dependency module* of the type P^{NM}_{d}.

To meet the requirements of an application, different chains of analysis can be arranged. Note that in the present configuration, the *Verb Shallow Analyzer* and *Shallow Analyzer* modules work at the higher level of performance if the specific chunker is used.

A chain *Chunker-Verb Shallow Processor* can be sufficient for an IE application devoted to extract events from sentences if the events prototypes are well described by the verb subcategorization frames. On the other hand, for Lexical Acquisition applications such as verb subcategorization frames acquisition that requires a high coverage of the phenomena [5], a parser composed by the *Chunker* and the *Shallow Analyzer* is sufficient. For an application as Terminology Extraction focussed on Noun Phrase boundary recognition, from the point of view of typology of the phenomena covered a chain composed by the *Chunker* and the *Shallow Processor* is enough, but the performance are not sufficient for the task. Thus, a chain *Chunker-Verb Shallow Processor-Shallow Processor* is required to augment the performance.

4.2 Task oriented parser design

We here analyze how to choose parsing chains for given application scenarios through the investigation of their performances. The examined applications are event recognition in an IE context, and candidate term boundary detection in a Terminology Extraction framework. Performances in term of quality of the syntactic material are evaluated through the metrics of Recall, Precision and F-measure. Given a grammatical relation τ (e.g. $NP-PP$), metrics defined as follows:

$$(a) \qquad R^{\tau} = \frac{card((A^{\tau}_{o} \cap A^{\tau}_{s}))}{card(A^{\tau}_{o})} \quad (b) \qquad P^{\tau} = \frac{card((A^{\tau}_{o} \cap A^{\tau}_{s}))}{card(A^{\tau}_{s})} \quad (c) \qquad F^{\tau}(\alpha) = \frac{1}{(\alpha \frac{1}{P^{\tau}} + (1-\alpha)\frac{1}{R^{\tau}})} \qquad (1)$$

A^{τ}_{o} are the correct syntactic relations of type τ for the sentence, and A^{τ}_{s} are the syntactic relations of type τ extracted by the system. The *oracle* used is obtained via a translation from the Penn Treebank [19]. The translation of the PTB constituency-based to the dependency-based annotation scheme, compliant with the evaluation requirements, is a crucial problem. Translation algorithms

have been settled in previous works [18, 6]. In the present work the adopted translation algorithm left untranslated about 10% of the *oracle trees*(i.e. reference corpus trees). The resulting evaluation test-set consists of nearly 44,000 sentences.

For the event recognition, three parsing chains have been tested: two light and one lexicalized. The first composes the chunker, the shallow analyzer and the disambiguator, i.e. *Chunker-SP-Prj*, the second remove the disambiguator, i.e. *Chunker-SA*, and the third introduces the lexicalized verb shallow analyzer, i.e. *Chunker-VSP-SP-Prj*. The interest here is in extracting relations whose verb is the head (V-Sub, V-Obj, and V-PP).

Parsing chain	**Link Type**	R	P	$F(\alpha = 0.5)$
Chunker-SP	V-Sub	0.75	0.89	0.82
	V-Obj	0.90	0.65	0.75
	V-PP	0.82	0.58	0.68
Chunker-SP-Prj	V-Sub	0.75	0.89	0.82
	V-Obj	0.90	0.66	0.76
	V-PP	0.58	0.94	0.72
Chunker-VSP-SP-Prj	V-Sub	0.76	0.89	0.82
	V-Obj	0.90	0.69	0.78
	V-PP	0.70	0.86	0.77

Table 1: verb arguments

Analyzing the table 1, from the point of view of the coverage of the phenomena, a better architecture appears to be *Chunker-SP*, but it guarantees a low level of precision compared to the other two. In case the interest of event extraction is in populating a database of facts, the most suitable process is the chain that guarantees the higher precision degree: the chain *Chunker-VSP-SP-Prj*. While, if the developer will feed an information retrieval system, the chain *Chunker-SP-Prj* is more appropriate.

In the case of NP recognition that Terminology Extraction (TE) requires, the application is interested in the relation typed NP-PP. Experimental evidence shows that the coverage of the phenomena is assured by a chain *Chunker-SP*, but the quality of the syntactic material is improved through the use of triggers provided by verb subcategorization lexicon in the chain *Chunker-VSP-SP*. The trade off between the cost of the system in term of subcategorization lexicon production and the performance required is another factor to be considered. The table 2 shows experimental results.

Parsing chain	**Link Type**	R	P	$F(\alpha = 0.5)$
Chunker-SP	NP-PP	0.85	0.65	0.73
Chunker-VSP-SP	NP-PP	0.82	0.75	0.78

Table 2: noun phrases-prepositional phrases attachment

In a TE chain where the filtering is based upon statistical methods, the chain *Chunker-SP* is light and assures an higher coverage of the phenomena. While in a TE chain where the filtering is done manually, an high degree of precision disburden the work of the terminologists. The improvement with respect to the precision from *Chunker-SP* to the chain *Chunker-VSP-SP*, even if there is a loss in the recall, may justify the cost in term of time complexity of choosing the *Chunker-VSP-SP* instead of the *Chunker-SP*.

5 Conclusions

A framework for modularization of the parsing process that eases their customization to the applications has been here described. The notion of syntactic module has been introduced and a classification according to basic grammatical properties of the different modules has been provided. Particular attention has been given to the syntactic annotation scheme. A useful syntactic information "holder" for the exchange among modules has been defined as a combination of a dependency and constituency based formalisms. An application of the given framework has been proposed. It has been shown and measured how different NLP applications may select an appropriate parsing chain according to their requirements.

References

[1] Steven Abney. Part-of-speech tagging and partial parsing. In G.Bloothooft K.Church, S.Young, editor, *Corpus-based methods in language and speech*. Kluwer academic publishers, Dordrecht, 1996.

[2] Salah Aït-Mokhtar and Jean-Pierre Chanod. Incremental finite-state parsing. In *Proceedings of ANLP97*, Washington, 1997.

[3] Roberto Basili, Massimo Di Nanni, and Maria Teresa Pazienza. Engineering of ie systems: An object-oriented approach. In Maria Teresa Pazienza, editor, *Information Extraction. Towards Scalable, Adaptabe Systems*, number 1714 in LNAI. Springer-Verlag, Heidelberg, Germany, 1999.

[4] Roberto Basili, Maria Teresa Pazienza, and Paola Velardi. A shallow syntactic analyser to extract word association from corpora. *Literary and linguistic computing*, 7:114–124, 1992.

[5] Roberto Basili, Maria Teresa Pazienza, and Michele Vindigni. Corpus-driven unsupervised learning of verb subcategorization frames. Number 1321 in LNAI, Heidelberg, Germany, 1997. Springer-Verlag.

[6] Roberto Basili, Maria Teresa Pazienza, and Fabio Massimo Zanzotto. Evaluating a robust parser for italian language. In *Proc. of the Workshop on the Evaluation of Parsing Systems, held jointly with 1st International Conference on Language Resources and Evaluation*, Granada, Spain, 1998.

[7] Roberto Basili, Maria Teresa Pazienza, and Fabio Massimo Zanzotto. Lexicalizing a shallow parser. In *Proc. of the TALN99*, Cargese, FR, 1999.

[8] Branimir Boguraev and James Pustejovsky, editors. *Corpus Processing for Lexical Acquisition*. The MIT Press, Cambridge, Massachusetts, US, 1996.

[9] Eric Brill. A simple rule-based tagger. In *Proc. of 3rd Applied Natural Language Processing Conference*, Trento, IT, 1992.

[10] Norbert Broker. A projection architecture for dependency grammar and how it compares to lfg. In *Proc. of LFG98 Conference*, Brisbane, US, 1998.

[11] Naom Chomsky. *Aspect of Syntax Theory*. MIT Press, Cambridge, Massachussetts, 1957.

[12] Mary Darlymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zeanen, editors. *Formal Issues in Lexical-Functional Grammar*. CSLI Publications, US, 1995.

[13] Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. Xtag system - a wide coverage grammar for english. In *Proc. of 15th International Conference on Computational Linguistic, COLING'94*, Kyoto, Japan, 1994.

[14] S. Federici, S. Montemagni, and V. Pirrelli. Shallow parsing and text parsing: a view in under-specification in syntax. In *Proc. of Workshop on robust parsing ESSLLI*, Prague, 1996.

[15] D. Grinberg, J. Lafferty, and D. Sleator. A robust parsing algorithm for link grammar. In *Proc. of 4th International workshop on parsing tecnologies*, Prague, 1996.

[16] A. Joshi and Y. Shabes. Tree-adjoining grammars and lexicalized grammars. In M. Nivat and A. Podelsky, editors, *Definability and Recognizability of Sets of Trees*. Elsevier, 1991.

[17] A.J. Joshi, L. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Science*, 1975.

[18] D. Lin. A dependency-based method for evaluating broad-coverage parsers. In *Proc. of the 14th IJCAI*, pages 1420–1425, Montreal, Canada, 1995.

[19] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19:313–330, 1993.

[20] Maria Teresa Pazienza, editor. *Information Extraction. A Multidisciplinary Approach to an Emerging Information Technology*. Number 1299 in LNAI. Springer-Verlag, Heidelberg, Germany, 1997.

[21] Maria Teresa Pazienza, editor. *Information Extraction. Towards Scalable, Adaptabe Systems*. Number 1714 in LNAI. Springer-Verlag, Heidelberg, Germany, 1999.

[22] C. Pollard and I.A. Sag. *Head-driven Phrase Structured Grammar*. Chicago CSLI, Stanford, 1994.

[23] Owen Rambow, J. Vijay-Shanker, and David Weir. D-tree grammars. In *Proc. of ACL'95*, 1995.

[24] L. Tesniere. *Elements de syntaxe structural*. Klincksiek, Paris, France, 1959.