Grammar Modularization for Efficient Processing: Language Engineering Devices and Their Instantiations

Axel Theofilidis, Paul Schmid and Thierry Declerck
{axel;paul}@iai.uni-sb.de and declerck@dfki.de

Abstract: This paper describes how unification-based grammar development benefits from a modularization methodology resulting in significant efficiency gains without giving up recognized advantages of unification grammars. Gains in efficiency are of such a scale that an industrial deployment of unification grammars seems to become realistic. We introduce a number of devices provided by the ALEP system which support grammar modularization at an operational level, and we illustrate how these devices have been instantiated in the grammatical resources developed in the LS-GRAM project. We conclude with drawing a prospective scenario of modular grammars serving industrial applications.

1 Unification Grammars for Industrial Applications?

There is a well-known gap between the scientific concept of unification grammar and its deployment in industrial contexts. Only few and limited examples of unification grammars are in fact known to be used in industrial NLP applications.

Being typically grown in academic contexts, unification grammars often lack an orientation towards coverage requirements of real-life documents. They are rather oriented towards the world of phenomena dealt with in the linguist's textbooks, aiming at most general models of language competence while at the same time becoming entangled with what is currently fashionable in the (computational) linguistics scientific community (both in terms of scientific paradigm and in terms of language data).

Against the background of the computationally rather expensive concept of unification and in combination with a failure to anticipate efficiency, cost and maintenance requirements of industrial NLP applications, this attitude in developing unification grammars results in an exponential decrease of performance features and, thus, in disqualification of unification grammars from real industrial deployment. On the other hand, unification grammars are known to have a number of higher level properties, such as declarativity and, by this, maintainability and extensibility, which are desirable for large-scale commercial NLP systems in terms of quality standards and, in the end, commercial success.

The LS-GRAM project funded by the Commission of the EU under the LRE program[1] aimed at narrowing the gap between the scientific concept of unification grammar on the one hand, and the requirements of industrial NLP applications on the other hand. To achieve this, the design of grammars was based on an investigation of coverage requirements for real-life documents of a specific domain (newspaper articles on economy) and on the exploitation of grammar engineering techniques supporting optimal performance (cf. [Schmidt et al.1996]).

2 Modular Grammar Design – Devices and Instantiations

The Advanced Language Engineering Platform (ALEP) that served as the implementation framework of the LS-GRAM project provides a range of devices supporting modularization of grammars, and thus efficiency-oriented grammar engineering in the paradigm of unification

[1] LRE 61029: *Large-Scale Grammars for EU Languages*

grammars. In this section we will introduce these devices one by one and illustrate how they have been employed in the LS-GRAM project to the end of producing efficient unification grammars. Though, for these grammars, there is still a distance to go, useful experiences have been acquired in the project which pave the way for future improvements and, in the end, the industrial deployment of unification grammars.

2.1 Integration of Frontends

Not every bit of a text requires intelligent linguistic processing, say, in terms of parse trees or compositional semantics. It is very important to draw a clear line between those phenomena which call for "linguistic intelligence" implementable in terms of unification grammar and those which are best handled by, say, pattern matching algorithms or finite state technology.

The open architecture of ALEP allows easy integration of frontends such as texthandling modules or part-of-speech taggers. In LS-GRAM, various tagging procedures have been developed for the recognition and markup of all kinds of so called "messy details" occurring in the LS-GRAM corpora such as names, dates, numbers, currency expressions, fixed multi-word units etc. (cf. [Bredenkamp et al.1996]):

```
<USR LEVEL="W" VAL="30 Millionen Dollar">
   <USR LEVEL="M" TYPE="CURRENCY">
      30_millionen_dollar
   </USR>
</USR>
```

Figure 1: *User-defined markup of German currency expression* '30 Millionen Dollar'

The flow of information from frontends to linguistic processing (or from linguistic processing to backends) is modelled by special declarations – `ts_ls_rule` declarations – establishing correspondences between feature-based linguistic descriptions and the markup of text, i.e. (i) a particular tag, such as `USR` marking user-defined regular expressions, (ii) a list of attribute-value pairs annotated with this tag, and (iii) the tagged string itself:

$$
\begin{array}{l}
\text{ts\_ls\_rule(} \\
\quad {}_{ld}\left[\begin{array}{ll} \text{SPEC}\,|\,\text{USR\_TYPE} & \boxed{1} \\ \text{SIGN}\,|\,\text{STRING}\,|\,\text{FIRST} & \left\langle \boxed{2} \;|\; \_ \right\rangle \end{array}\right], \\
\texttt{USR},\ [\ \texttt{LEVEL="M"}\ ,\ \texttt{TYPE=}\boxed{1}\ ]\ ,\ \boxed{2}\ ).
\end{array}
$$

Figure 2: *Correspondence established between partial linguistic description and markup of text*

Based on such declarations, user-defined markup is carried over to match generic lexical entries covering all instances of a given type of regular expression:

2.2 Specifier-Based Grammar Partitioning

Specific processing sub-tasks do not require simultaneous access to all layers of linguistic information. Co-presence of too many dimensions of information may, in fact, spoil processing

$$
{}_{ld}\left[\begin{array}{ll}
\textsc{spec}\,|\,\textsc{usr\_type} & \texttt{CURRENCY} \\
\textsc{sign}\,|\,\ldots\,|\,\textsc{cat} & {}_{cat}\left[\begin{array}{ll}
\textsc{head} & {}_{n}\left[\begin{array}{ll}\textsc{agr} & (\mathrm{pl}\&\mathrm{p3})\end{array}\right] \\
\textsc{subcat} & \langle\rangle
\end{array}\right]
\end{array}\right]
$$

Figure 3: *Generic lexical entry for currency expressions (partial)*

behaviour by introducing spurious ambiguities in a chain of processing tasks. A reflex of morpho-syntactic, or even syntacto-semantic, reading distinctions in the lexical resources being consulted during the processing task of morpheme recognition (i.e. word segmentation) will unavoidably result in multiple solutions between which it may not be possible to decide at this very stage of processing. The same holds for syntacto-semantic, or purely semantic, reading distinctions (e.g. frame variants, or instances of polysemy) during syntactic parsing. This is particularly true of any non-deterministic parser.

To avoid spurious ambiguities in a chain of processing tasks, the ALEP linguistic sub-system foresees the use of specifiers. Specifiers are designated feature structures which serve the purpose of encoding class membership of rules. By specifying rules to be members of particular classes, grammars are internally partitioned into sub-grammars.

One dimension along which grammar rules may be distributed over different classes is the dimension of processing tasks by which grammar rules are applicable:

$$
{}_{ld}\left[\begin{array}{ll}
\textsc{spec}\,|\,\textsc{apply} & \left[\begin{array}{ll}\textsc{morph\_recog} & + \\ \textsc{morph\_parse} & -\end{array}\right] \\
\textsc{sign} & \left[\;\ldots\;\right]
\end{array}\right]
$$

Figure 4: *Grammar partitioning according to applicable processing operations, e.g. morpheme recognition and morphological parsing*

On the basis of respective specifier assignments, an account of some dimension of linguistic information can be postponed to that processing sub-task by which it is most appropriately and, at the same time, most efficiently handled. Thus, the range of interpretations of some inflectional affix may be accounted for by lexical entries to be applied during morphological parsing, but not during morpheme recognition having only access to a generic encoding of this affix.

Specifier-based partitioning of grammars and distribution of sub-grammars over processing tasks bears a rich potential of efficiency-oriented grammar engineering. Especially when combined with the refinement operation provided by ALEP, this approach opens the perspective of integrating rather shallow parsing with the efficient computation of higher-level linguistic properties.

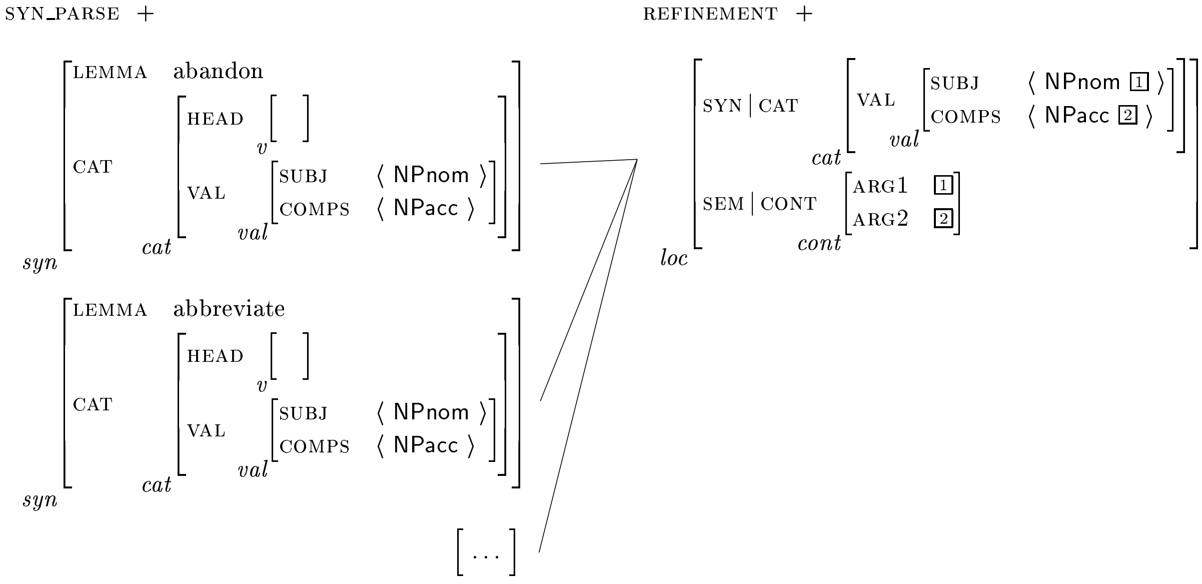

Figure 5: *Valency to argument structure linking postponed to refinement*

2.3 Employment of Monotonic Stratification (Refinement)

The refinement operation is a monotonically operating feature-decoration algorithm which (re-)applies designated (in terms of specifiers) structure rules and lexical entries to consolidated tree structures obtained from parsing. By adding information in terms of features that were left unspecified or underspecified during parsing, the refinement operation may produce two basic effects supplementary to mere feature decoration:

- The effect of **disambiguation** may follow from the fact that features being added and propagated during refinement result in a unification failure, such that some input structure may not survive the refinement operation.
- The effect of **ambiguation** may follow from the fact that the code being applied during refinement introduces some sort of disjunctive information, such that more than one output data structure is obtained on the basis of a single input data structure.

The most interesting aspect of employing refinement is that of postponing an account of some layer of linguistic information which does not contribute substantially to constraining the process of parsing.

The borderline is not easy to draw, of course. Lexical semantics, for instance, may well contribute to the resolution of attachment ambiguities and, thus, to constraining parsing. At the same time, introducing lexical semantic features typically increases non-determinism by disjunctive specification of lexical items. Experience in LS-GRAM has shown that it was less costly to have attachment ambiguities during parsing which may be resolved at the refinement level, than to start out with lexical disjunction during parsing and claim resolution to take place at this very stage of processing.

Compositional semantics, on the other hand, proved to be a clear candidate for being handled in refinement, where generic lexical entries take care of, say, linking valency frames with argument structure (cf. Figure 5). The advantage of such an approach lies not only in the significant reduction of the size of data structures to be dealt with during parsing; the clear separation of syntax and compositional semantics in terms of distinct grammar modules supports the plugging in and out of different types of semantic modules on the basis of the same syntactic module, thus enabling grammar configuration according to the specific demands of a particular application.

Grammar configuration relative to applications is further supported by the fact that refinement may apply an arbitrary number of times in succession, such that other higher-level linguistic properties, like presupposition and implicature, or register and style, may be handled by distinct modules to be accessed on demand.

2.4 Head Scheme Based Parsing

Two of the ALEP built-in parse algorithms are based on head schemes, i.e. structure rules are accessed by a selected head daughter. Parsing then proceeds head outwards.

Head daughter selection is performed by special declarations specifying (to an arbitrary degree of abstraction) a mother node and the particular daughter node to be used as parsing head. On the basis of such declarations, the transitive reflexive head closure is computed against which structure rules are tested during parsing in attempting to reach the top node, i.e. the axiom of the grammar.

Interesting to note is that head selection is an implicit way of modularizing a grammar. Applied to a given rule, head selection partitions the grammar into those linguistic descriptions which match the respective head daughter and those which do not. Trivially, the larger the set of mother node specifications which match a selected head daughter, the more often the respective rule may be accessed and the less efficient performance will be. Thus, head selection becomes an important device of efficiency-oriented grammar design. It has been shown in LS-GRAM that the selection of functional categories as parsing heads (rather than substantive categories) significantly improves performance by reducing the number of transitive relations across structure rules, that is, by establishing a finer-grained modularization of grammars in terms of head selection.

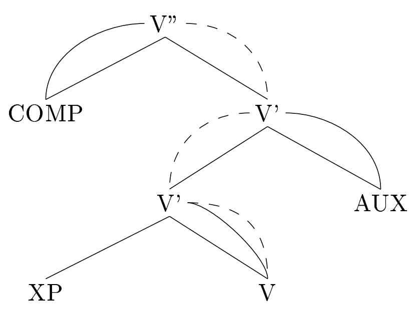

Figure 6: *Computed head relations for functional categories (non-dashed curves) and substantive categories (dashed curves) serving as parsing heads*

To illustrate the point made, imagine the effect which an account of coordination typically has on the behaviour of a simple left-to-right bottom-up parser. With the head scheme based parsers of ALEP, the selection of coordinating conjunctions as parsing heads ensures that the set of rules accounting for conjunct structure and, subsequently, coordinated structure will only be accessed if a coordinating conjunction is actually met in the string to be processed. Thus, the parse time for a complete paragraph taken from the German LS-GRAM corpus and comprising sentences not containing coordinated structures only slightly increases after the integration of an elaborate account of coordination.

3 Prospects of Modular Grammar Design

In the previous section we illustrated the use of various devices supporting grammar partitioning in ALEP. However, the potential of theses devices and, particularly, of specifier-based grammar partitioning is much larger.

So far, we have mentioned only one dimension along which grammars developed in the ALEP framework may be partioned by employing the notion of specifiers: the dimension of processing tasks by which instances of grammar code are applicable. Beyond this, specifiers may be used to establish a multi-dimensional, cross-classificatory partitioning of grammars even in a multilingual setting. Further dimensions of partitioning are:

- **diversity of languages**, where modules shared between languages are distinguished from language-specific modules
- **sorts of linguistic information**, where modules are distinguished according to the layer of information they contribute (e.g. morphological, syntactic, semantic, pragmatic)
- **types of structure units**, where modules are distinguished according to the structural units they account for, both linguistic (e.g. word forms, phrases, sentences, paragraphs) and textual (e.g. titles, text body)
- **domains of coverage**, where modules are distinguished according to the particular grammatical phenomena (e.g. apposition, coordination, extraposition) or the sublanguage they account for (e.g. administrative, economic, financial)

By multidimensional partitioning of grammars, a pool of integratable grammar modules will be obtained which supports easy configuration of sub-grammars according to the particular demands of specific applications and, at the same time, allows grammar modules to be reused across different applications. The configuration of sub-grammars relative to the demands of particular applications and, by this, to the end of optimal performance, may functionally be supported in various modes: input-parameter-driven and, in that, either interactive or non-interactive, or pre-processor-driven and, in that, either statically or dynamically:

Besides configurability and reusability, the maintainability and extensibility of grammar modules in a distributed grammar engineering set-up, where grammar development and maintenance tasks are distributed over different persons working, perhaps, at different sites, is optimally supported by this architecture.

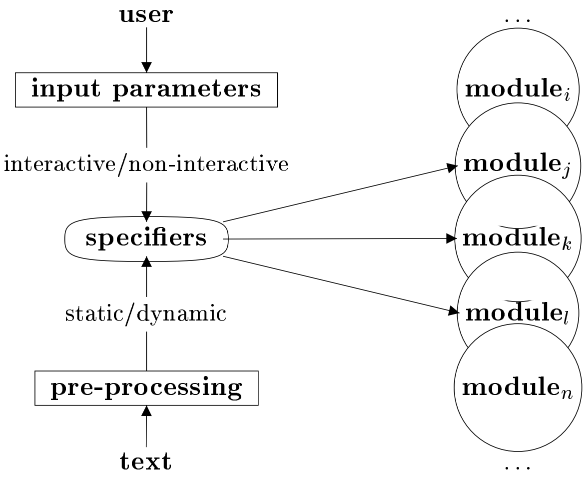

Figure 7: *Sub-grammar configuration out of a pool of grammar modules*

References

[Bredenkamp et al.1996] Andrew Bredenkamp, Thierry Declerck, Frederik Fouvry, and Bradley Music. 1996. Efficient integrated tagging of word constructs. In *Proceedings of the 16th International Conference on Computational Linguistics, COLING-96*, volume 2, pages 1028–1031, Copenhagen.

[Bredenkamp et al.1997] Andrew Bredenkamp, Thierry Declerck, Frederik Fouvry, Bradley Music, and Axel Theofilidis. 1997. Linguistic engineering using ALEP. In *Proceedings of the RANLP'97*, volume 1, Tzigov Chark, Bulgaria.

[Declerck and Maas1997] Thierry Declerck and Heinz Dieter Maas. 1997. The integration of a part-of-speech tagger into the ALEP platform. In *Proceedings of the 3rd ALEP User Group Workshop*, pages 19–26, Saarbrücken.

[Fouvry and Bredenkamp1997] Frederik Fouvry and Andrew Bredenkamp. 1997. Partial parsing in alep. In *Proceedings of the 3rd ALEP User Group Workshop*, pages 33–36, Saarbrücken.

[Music1995] Bradley Music. 1995. Tagging messy details in ALEP. Technical report, Center for Sprogteknologi, Copenhagen. LS-GRAM (LRE 61029) Supplement to Deliverable E-D5-DK.

[Schmidt et al.1996] Paul Schmidt, Axel Theofilidis, Sibylle Rieder, and Thierry Declerck. 1996. Lean formalisms, linguistic theory and applications. Grammar development in alep. In *Proceedings of the 16th International Conference on Computational Linguistics, COLING-96*, volume 1, pages 286–291, Copenhagen.

[Theofilidis and Schmidt1997] Axel Theofilidis and Paul Schmidt. 1997. ALEP-based distributed grammar engineering. In *Proceedings of the ENVGRAM Workshop (in conjunction with ACL-97/EACL-97)*, volume 1, Madrid.