ALEP-Based Distributed Grammar Engineering

Axel Theofilidis
IAI
Martin-Luther-Str. 14
66111 Saarbrücken, Germany
axel@iai.uni-sb.de

Paul Schmidt
University of Mainz, FASK Germersheim
An der Universität 2
76726 Germersheim, Germany
schmidtp@usun2.fask.uni-mainz.de

Abstract

Starting from a clarification concerning the notion of distributed grammar engineering, we present options of distributed grammar engineering as are supported by the ALEP grammar development platform and as were instantiated in the LS-GRAM project. The notion of distributed grammar engineering being grounded in the concepts of grammar modularization and grammar module integration, we focus on ALEP features providing operational support for these two concepts in terms of both data storage and internal classification of grammar code. We conclude with an indication of the major benefits of ALEP facilities for distributed grammar engineering and derive some general desiderata from this.

1 Two Notions of Distributed Grammar Engineering

According to our understanding, two notions of distributed grammar engineering (DGE) should be distinguished: (i) DGE meaning that tasks which jointly contribute to the development and maintenance of a grammatical resource are distributed over different persons working, perhaps, at different sites; (ii) DGE meaning that different domains of linguistic information, different layers of linguistic description are distributed over different processing (sub-)tasks.

Irrespective of this difference in notion, there is a fundamental presupposition common to both notions of DGE: that grammatical resources bear a modular structure, and that grammar modules being distributed over different authors or processing devices can neatly be integrated to form a coherent grammatical resource, respectively to support a coherent chain of processing tasks. Thus, as shown in Figure 1, the idea of DGE is firmly grounded in the concepts of modularization and integration: once grammar modularization is provided and integration of grammar modules is ensured, the option of DGE falls off as a side-effect.

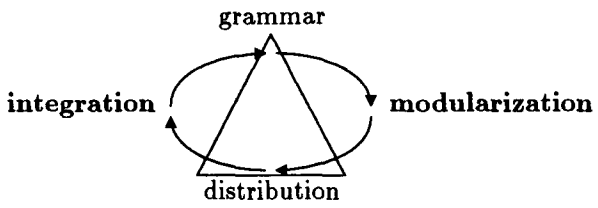

Figure 1: *DGE grounded in the concepts of modularization and integration*

In the following two sections we will present options of grammar modularization and grammar module integration as are supported in the ALEP grammar development platform both in terms of data storage (section 2) and in terms of internal classification of grammar code (section 3). We will indicate how these options were made use of to support DGE in the LS-GRAM project,[1] in the context of which broad-coverage grammatical resources for nine EU-languages were developed. Though no operational concept of DGE has been implemented at the multi-lingual level in this project, each particular grammar was collaboratively written by several authors which, in some cases, were even located at different sites,[2] thus giving rise to a real need for DGE support.

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

[2] The Dutch grammar was developed at SST, Utrecht, and KUL, Leuven; the German grammar at IAI, Saarbrücken, and IMS, Stuttgart; the Italian grammar at DIMA, Torino, and ILC, Pisa; the Spanish grammar at FBG and UPF, both Barcelona.

2 Grammar Modularization and Integration in terms of Data Storage

The ALEP platform realizes an object-oriented environment. As such it assumes storage of data at two levels: the file level and the object level. At the file level, data (grammar code) may be distributed over an arbitrary number of files, each of them containing a particular type of data (e.g. type and feature declarations, lexical entries, phrase structure rules). A phrase structure grammar, for instance, may be distributed over several files each of them containing a set of rules accounting for a particular domain of constituency (e.g. 'S', 'VP', 'NP') or for some particular type of construction (e.g. apposition, coordination, extraposition). Similarly, a lexicon may be distributed over several files along dimensions such as part-of-speech category or sub-language.

At the object level, on the other hand (cf. (Groenendijk, 1994)), an arbitrary number of files constituting a coherent grammar module may be grouped into an object being defined in terms of the ALEP User Language (AUL)[3] and stored in the ALEP Object Repository (AOR). Figures 2 and 3 show sample (but partial) AUL objects representing a phrase structure component and a declarations component of a grammar.

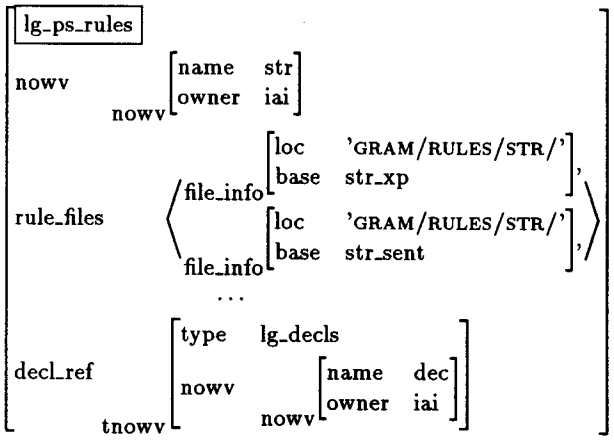

Figure 2: *Sample AUL object representing a phrase structure component*

The object of type lg_ps_rules bears the list-type feature 'rule_files', with each of the 'file_info' feature structures pointing to a file containing a set of phrase structure rules.[4] In addition, an object of type lg_decls is referred to by the feature 'decl_ref'.

[3] A typed feature structure notation similar to the linguistic formalism supplied with ALEP.

[4] A UNIX directory path and a file name are given as the values of the attributes 'loc' and 'base'.

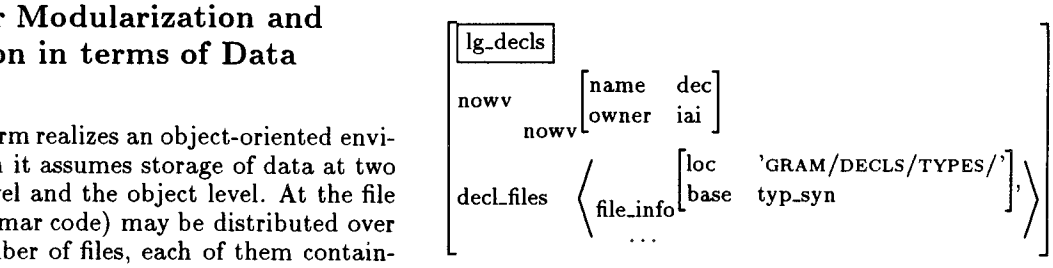

Figure 3: *Sample AUL object representing a declarations component*

This object (shown in Figure 3) represents the declarational basis of the respective phrase structure component. It refers, in its turn, to a list of files containing a coherent set of declarations (e.g. type and feature or macro declarations) upon which a phrase structure component or any other grammar component (e.g. a lexicon component) may be based.

A coherent set of grammar components may, in turn, be grouped (i.e. integrated) into a higher-level object of type lg_lingware_group representing a complete grammar (or sub-grammar). This is shown in Figure 4 illustrating the modular and, at the same time, hierarchical style of data structuring characteristic of ALEP.[5]

Presuming a principled distribution of grammatical data over files (reflecting, for instance, different types of grammatical phenomena or different domains of grammatical description), a whole range of specialized grammars or grammar components may be configured at the object level according to particular grammar development or maintenance tasks. Objects may be defined, for instance, which incrementally extend the core coverage of a grammar by new domains of linguistic description. This is illustrated in Figure 5, where two lg_ps_rules objects are shown, both of which share the files containing the basic phrase structure rules dealing with sentential (file 'str_sent') and non-sentential (file 'str_xp') constructions, but which extend this set of files once by a file accounting for coordinated structure and, in the second case, by a file accounting for paragraph structure.

Based on this kind of definition of specialized grammars or grammar components, grammar development and maintenance tasks being related to particular domains of linguistic decription may easily be assigned to, and distributed over, different persons possibly working at different sites. Physical distribution (exchange) of grammar components is conveniently supported by the ALEP 'Export'

[5] Objects of type lg_lex_rules and lg_tlm_rules represent lexicon, respectively two-level morphology, components.

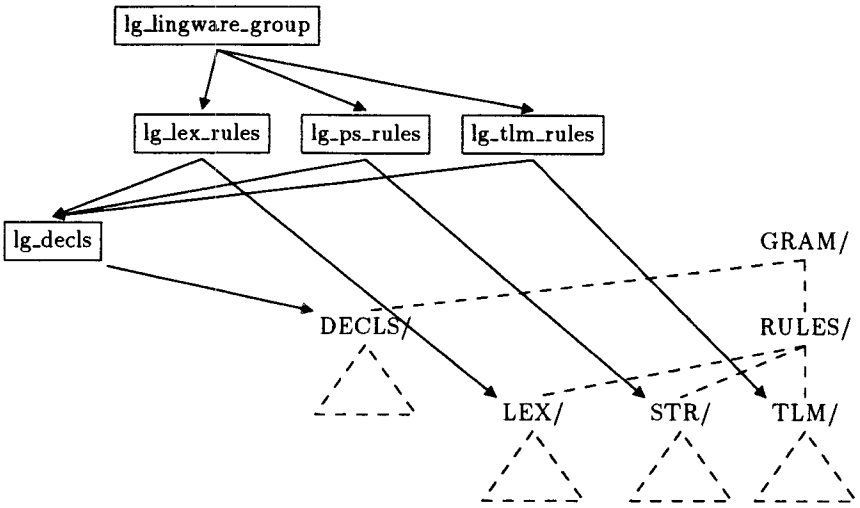

Figure 4: *Data storage at the AOR object level and the UNIX file level*

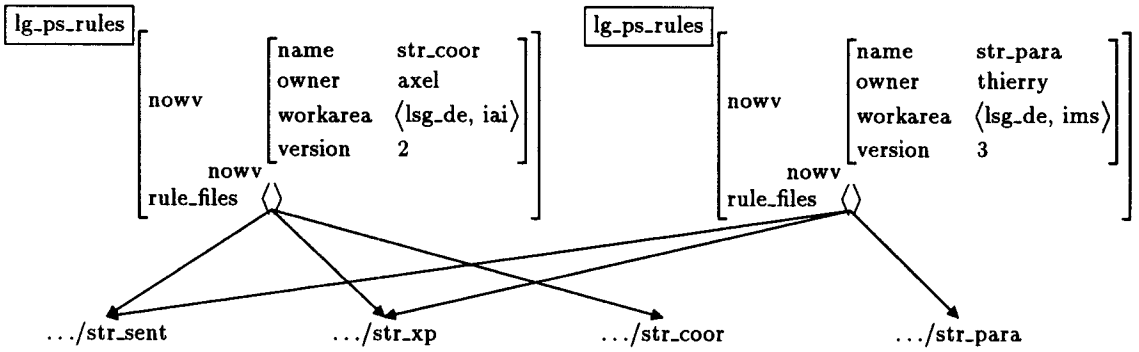

Figure 5: *Object-based definition of specialized grammar components*

and 'Import' functionality, where 'Export' performs a (UNIX) 'tar' operation on a selected set of objects, and 'Import' performs an 'untar' operation on an 'Export'-created tar-file, asserting all respective objects to the AOR as well as creating all directories and files being referred to by these objects (cf. (Groenendijk, 1994), chapter 3).

Management of larger sets of objects being iteratively exchanged between persons or sites is well supported by a number of features being assigned to every object, such as the 'comment' feature, which allows to encode a comment string with every object, and, most importantly, the object identification feature 'nowv' which requires every object to be assigned a unique combination of (object) name, (object) ownership, (object) workarea, and (object) version (cf. (Groenendijk, 1994), chapter 4), thus allowing to keep track of distributedness in grammar writing. In the illustration given in Figure 5, for instance, the 'nowv' feature indicates that Axel obviously is the person who elaborates on the second version of a phrase structure component covering coordinated structure as part of the German LS-GRAM resources developed at IAI, whereas Thierry is the person who elaborates on the third version of a phrase structure component covering paragraph structure as part of the German LS-GRAM resources developed at IMS.

This style of distributed grammar development has been a standard practice in the LS-GRAM project. Thus, for instance, it has been a typical approach to have the morphological and the syntactic grammar components developed by different persons and, as in the case of the Italian or Spanish grammar, even at different sites.

After each cycle of distributed grammar development based on the definition of specialized grammars or grammar components, re-integration of the various grammar modules can easily be performed at the level of data storage by defining respective new

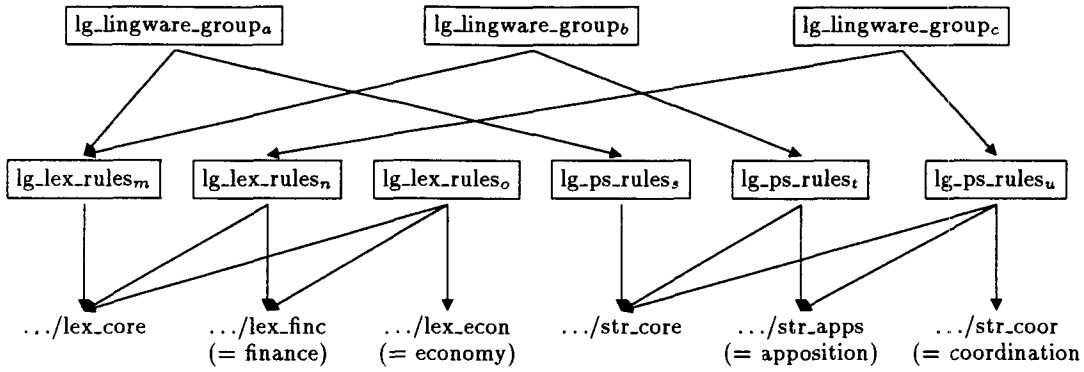

Figure 6: *Object-based sub-grammar configuration*

objects. Grammar modules being established at the file level and spread over different objects representing specialized grammar components can be merged into one object representing a full coverage grammar component. Such full coverage grammar components may, in turn, be grouped into a higher level object of type lg_lingware_group representing a full coverage grammar.[6]

Interesting to note is that the style of modularization and integration of grammars supported in ALEP at the level of data storage not only conveniently supports DGE (as we hope to have shown), but also a high degree of flexibility in configuring (and re-configuring) grammars according to specific (and changing) demands of different application scenarios. This is illustrated in Figure 6, showing how sub-grammars with varying coverage can be configured at the object level based on a fine-grained modularization of grammar components along dimensions such as sub-language for lexicon components, or types of syntactic constructions for phrase structure components.

3 Grammar Modularization and Integration in terms of Data Classification

Besides in terms of data storage, ALEP also supports grammar modularization and integration in terms of internal classification of grammar code based on the notion of specifiers. Specifiers are designated feature structures[7] which serve the purpose of encoding membership of a rule in one (or more than one) class of rules and, thus, realize the notion of a rule classifier. By specifying rules to be members of particular classes of rules, grammars are internally (and, in that, multi-dimensionally) partitioned into sub-grammars (cf. (Simpkins, 1994a), chapters 5 and 7).

Specifier-based grammar partitions may be established along two basic dimensions illustrated in Figure 7: along a vertical dimension, grammar partitions may be established according to different types of processing operations to apply; lexical entries, for instances, may be specified to be applied during word segmentation (= two-level based morphographemic analysis), during analysis (= parsing), or during refinement only (the operation of refinement will be explicated below). Along a horizontal dimension, on the other hand grammar partitions may be established according to different types of structural units being involved in the parsing operation; structure rules may be specified to be applied only when parsing morphemes to words, words to sentences, or sentences to paragraphs.

The main effect that can be obtained by an intelligent specifier-based grammar partitioning is in terms of increased performance efficiency: By specifier-based grammar partitioning, access of rules during execution of some processing (sub-)task can be restricted to a sub-grammar being identified via a particular instance of the specifier feature structure and encoding only as much information as is relevant to the respective processing task.

Irrespective of performance support, however, specifier-based grammar partitions constitute an operational concept of grammar modularization that can be multiply exploited in DGE. In that, the ALEP 'Refine' operation plays a crucial role. 'Refine' is a monotonically operating feature-decoration algorithm which (re-)applies structure rules and lexical entries to consolidated structure trees as are obtained from analysis (or synthesis). Important with regard to DGE, as we will see shortly, is that 'Refine' may be executed an arbitrary number of times in succession.

The set of rules applied by the 'Refine' operation

[6] Though the process of merging objects is not yet functionally supported in the ALEP environment, it is considered a trivial task to integrate a respective functionality.

[7] 'designated' in that they are picked out by a special feature path declaration

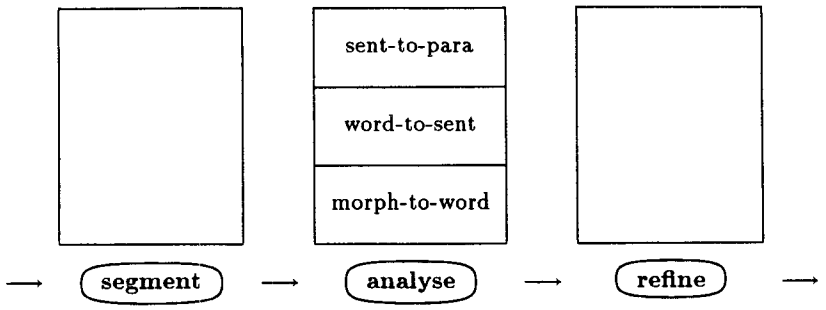

Figure 7: *Grammar partitioning along a vertical and a horizontal dimension*

must constitue a complete grammar which is unifiable (partially identical perhaps) with the grammar that was applied by the preceding operation. In that, however, 'Refine' will produce an effect only by application of rules which add some information (feature decoration!) compared to the corresponding rules that were applied by the preceding operation.

By a systematic distribution (based on a vertical grammar partition scheme) of different domains of linguistic information over an analysis grammar and one, or more, 'Refine' grammars, the presumed modularity of linguistic knowledge is (monotonically) fleshed out at the level of grammar engineering. Thus, for instance, it has been a typical practice in LS-GRAM to distribute syntactic and semantic information over an analysis and a refinement grammar respectively; by this, parsing will not be affected by ambiguities residing in the semantic domain (cf., for instance, (Schmidt et al., 1996a) and (Schmidt et al., 1996b)).

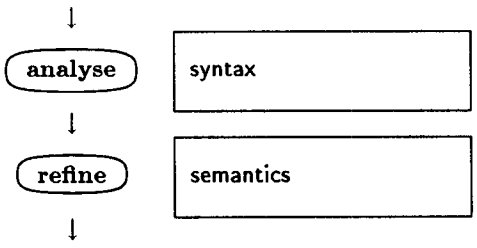

Figure 8: *Grammar partitioning according to different domains of linguistic information*

More importantly with regard to DGE, however, is that grammar modules being distributed over the processing operations of analysis and refinement and being delimited according to different domains of linguistic information, can simultaneously be distributed over different authors according to their specific expertise. The degree of this kind of distributedness can be significantly increased by a still finer-grained modularization of grammatical resources assuming multiple application of the 'Refine' operation, as illustrated in Figure 9. Thus an account of syntax can be distributed over a grammar module supporting shallow parsing and a grammar module performing syntactic filtering based on the refinement operation; different aspects of semantics and pragmatics may futhermore be distributed over distinct grammar modules being successively applied during further refinement stages:

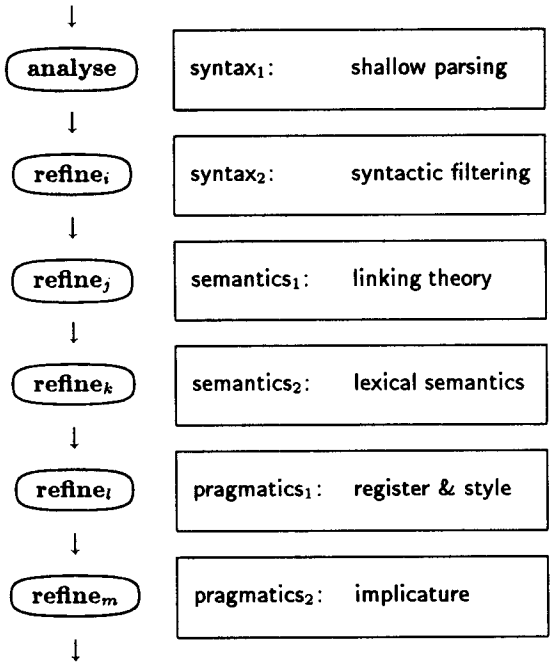

Figure 9: *Grammar partitioning according to different domains of linguistic information*

However, if the only effect of specifier-based grammar partitioning was that of distributing different domains of linguistic information over grammar modules to be applied during successive processing stages, the same effect could also be obtained by simply assuming distinct grammar modules in terms of data storage (i.e. at the file and object level of data storage). But, in terms of both gram-

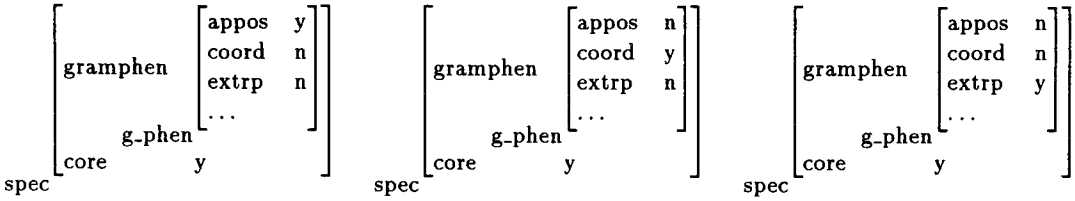

Figure 10: *Specifier feature structures establishing a cross-classificatory grammar partition*

mar development and maintenance tasks it is, in fact, an advantage of the specifier-based approach to grammar modularization that logically related grammar code may be stored in one and the same file, though it will be applied, in effect, at different processing stages. An even bigger advantage is that the specifier-based approach to grammar modularization supports a multi-dimensional, cross-classificatory partitioning of grammars which is not possible in terms of data storage, unless at the cost of redundantly duplicating grammar code. In terms of specifier feature structures as those shown in Figure 10, for instance, grammar rules are simultaneously assigned to the class of rules establishing a core-coverage grammar and to one of several classes of rules accounting for specific grammatical phenomena, such as apposition, coordination or extraposition.

Based on specifier feature structures establishing a cross-classificatory partition scheme for grammars, specialized grammar modules can be both uniquely identified by full specification, and integrated by underspecification, of specifier features. Thus, for instance, by reference to the underspecified feature structure shown in Figure 11, all grammar modules are called except for those dealing with extraposition.

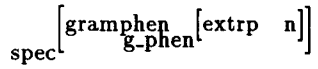

Figure 11: *Underspecified specifier feature structure effecting grammar module integration*

4 Conclusions and Desiderata

The mechanisms and devices described in the previous sections constitute a first step towards removing a decisive bottleneck in large-scale distributed grammar engineering[8]. The bottleneck is that the bigger and the more complex a grammar becomes the more difficult it is to extend it, improve it, or adapt it. Grammars tend to become huge, incomprehensible monolithic blocks which are more and more difficult to maintain, with distributed development becoming impossible. In the following we will make some general, summarizing points about what we think can be derived from the kind of facilities provided by ALEP for DGE in general. In this, we will mainly focus on benefits of specifier-based partitioning, addressing the issues of testing, distribution, maintenance and deployment, as well as touching on the issue of monotonic vs. non-monotonic grammar development. In section 4.2 we conclude with deriving some general desiderata for DGE.

4.1 Benefits for Large-Scale Grammar Engineering

Testing: A major advantage of the specifier facility lies in the fact that testing (and thus development) becomes easier as specific modules (that may be under reconstruction) may be tested separately by plugging together just the relevant modules in terms of reference to the approriate specifiers. Thus, it is possible to plug together a core grammar with a coordination module in order to test coordination while leaving aside other phenomena, such as extraposition, which may be irrelevant at the given stage of grammar development. Then, gradually, more modules may be added (one by one) in order to explore the interaction of these modules.

Distribution: It is obvious that, as far as distribution of grammar development is concerned (in the sense that different linguists develop the same grammar), facilities as described are a minimum. Based on mechanisms of grammar partitioning, grammars can be developed in a distributed fashion in that one linguist may work on developing a treatment of appositions, building on a given core grammar, while another one is working on a treatment of coordination, building on the same core grammar without unwanted interference.

Maintenance: From what has been said about testing and distribution, it is obvious that a grammar existing in a modular form as supported by the facilities described is better maintainable than a monolithic one.

Deployment: In the same way grammar modules can be plugged together for testing and, more

[8] We think that large-scale grammar engineering must be distributed in general.

generally, for development purposes, grammar modules can also be plugged together in order to be deployed in different application scenarios with distinct requirements in terms of linguistic coverage or depth of analysis. That is, one could well envisage that, for particular applications, specific sets of grammar modules are plugged together by reference to the appropriate specifiers.

Monotonic vs. Non-Monotonic Grammar Development: The specifier facility, as introduced so far, suggests that the optimal way of proceeding in grammar development (also independently of this facility) is to proceed monotonically, i.e. by simply adding different modules accounting for specific domains of linguistic description (such as those exemplified by coordination and extraposition) to already available modules. However, this is an unrealistic requirement as the treatment of new phenomena hardly ever consists of simply adding (monotonically) sets of new rules to an existing grammar. It is often required to revise existing modules (even the core grammar) in the light of requirements deriving from the treatment of new phenomena. But even in this case the modularization achieved by specifier-based partitioning of grammars is the condition for performing the required changes efficiently and under optimal control, since the consequences of such changes can be studied module by module.

As a final point, it should be mentioned that partitioning of grammars by approriate specifiers supports adaptation of a grammar to new theoretical insights about the nature of human language that may become necessary by new developments in theoretical linguistics. Specifiers thus contribute considerably to the 'reusability' of grammars.

4.2 Desiderata

The options of grammar modularization provided in ALEP both in terms of file and object-based data storage and in terms of specifier-based classification of grammar code, constitute a good basis for supporting DGE at an operational level. As for support for grammar module integration, however, we still see the need for add-on functionality complementing operational support for grammar modularization.

When thinking of grammar module integration in terms of data storage, one has to bear in mind that, in itself, integration of grammar modules at the level of data storage does not yet entail an integrated grammar. Integration of grammar modules presupposes integration in terms of linguistic specifications meaning that each particular grammar module must be integratable at least wrt. a common declarational basis (type and feature theory, macros etc.) and some implemented notion of a core grammar.

In both respects, problems with integration may occur during the course of distributed grammar development, where different persons simultanously elaborate on different grammar modules. Interferences may occur, for instance, due to the fact that work on a specific grammar module requires parts of the declarational basis to be modified which are shared by other modules. Interference may also occur due to the fact that a specific grammar module being developed by one author feeds in information into modules developed by other authors, with these modules, in turn, being supposed to thread this information to still other modules. To get hold of such interferences, sophisticated versioning control functionality is required providing for automatic creation of version protocols (encoding modifications, authorship etc.), for comparing or merging parallel versions of some grammar module, and for checking information paths across grammar modules.

As for the concept of specifier-based grammar partitioning, we consider it desirable to provide a direct linking to the data storage layer by providing the option of defining objects not only in terms of a reference to files, but also in terms of a reference to specifier information. The idea is that the data being represented by an object can be selected (and automatically stored in appropriate files) based on a full or partial specification of the specifier feature structure. By implementation of this idea, specifier-based grammar partitions of arbitrary grain-size could become physically manifest at the object-level of data storage making them immediately accessible to object-level support of DGE such as the 'Export'/'Import' functionality.

References

Marius Groenendijk. 1994. Environment Tools Guide. ALEP-2 – Guide to the ALEP User Interface Tools. CEC, Luxembourg.

Paul Schmidt, Sibylle Rieder, Axel Theofilidis, Thierry Declerck. 1996a. Final Documentation of the German LS-GRAM Lingware (LRE 61029, Deliverable DC-WP6e (German)). IAI, Saarbrücken (http://www.iai.uni-sb.de/LS-GRAM).

Paul Schmidt, Sibylle Rieder, Axel Theofilidis, Thierry Declerck. Lean Formalisms, Linguistic Theory, and Applications. Grammar Development in ALEP. 1996b. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 286–291, Copenhagen, Denmark.

Neil K. Simpkins. 1994a. ET-6/1 Linguistic Formalism. ALEP-2 – User Guide. CEC, Luxembourg.

Neil K. Simpkins. 1994b. Linguistic Development and Processing. ALEP-2 – User Guide. CEC, Luxembourg.