Hypertextual Grammar Development *

Luca Dini and **Giampaolo Mazzini**
Centro per l'Elaborazione del Linguaggio e dell'Informazione (CELI)
Via G. Ferraris 109, Palazzo Tartara 13100 Vercelli (Italy)
E-mail: {mazzini,dini}@celi.sns.it

Abstract

We will present a new model of grammar documentation which exploits the potentialities of an hypertextual representation of lingware. As we will show this model insures the following benefits:

- Constant coherence between the documentation and the various pieces of linguistic information.
- Valuable organization of the linguistic knowledge, in order to increase the productivity of the grammar writer.
- Possibility of sharing grammatical resources and related documentation with other sites in a completely intuitive way.

1 Introduction

In recent years a big stress has been put on the view of grammar development as a subtype of software development. The attention reserved to phases like grammar design, testing and validation, as well as the key role assumed by issues such as reusability and portability, is a clear sign of the tendency to narrow the gap between grammar and software development (Nagao 1988, Boitet 1989, Schütz 1995). A further element of similarity is represented by the fact that grammar development is nowadays always involving more than one single grammar developer, the standard situation being one of cooperation among different sites (cf., for instance, the LS-GRAM, ERGO, and Vermobil scenarios). In this respect, it mirrors the same evolution of industrial software systems, which privileged a distributed architecture over a centralized one.

There is a point, however, where methodologies in grammar writing and in software engineering still diverge: documentation. It has been pointed out by many authors (e.g. Metzeger & Boddie 1996) that documentation should take at least 10% of the person-power allocated to a project, and that, in any case, documentation should be taken as seriously as design, programming and testing. For these reasons, different programming styles have developed different techniques in order to guarantee both consistency and user-friendliness of documentation, the most striking example being represented by Computer-Aided Software Engineering, which evolved, originally, as a tool for software documentation.

On the side of grammar engineering, on the contrary, the topic seems to have been underestimated, and even projects where grammar documentation was central (such as LS-GRAM) conceived it in a quite traditional and obsolete way, i.e. as a process of editing huge reports after the end of the implementation phase (cf. the criticisms to this approach made by Booch (1996)).

The present work aims at providing a new model of grammar documentation, by exploiting the potentialities of an hypertextual format. As we will show our model insures the following benefits:

- Constant coherence between the documentation and the various pieces of linguistic information.
- Valuable organization of the linguistic knowledge, in order to increase the productivity of the grammar writer.
- Possibility of sharing grammatical resources and related documentation with other sites in a completely intuitive way.

The program implementing this model (HyperGram) is currently fully compatible with ALEP, and it will be soon integrated with PAGE. In any case,

* We are grateful for suggestions and discussions to Gregor Erbach, Stephan Oepen, Axel Theofilidis.

its extension to other grammar development platforms is quite unproblematic, and in the future we will consider the integration with a broader range of grammar development tools.

2 Desiderata

2.1 Documentation in Software and Grammar Engineering

In Booch (1996) documentation-driven projects are described as a "...degenerate of requirements-driven processes, a case of bureaucracy gone mad in the face of software...". Their most salient feature is the need of producing documentation packages as deliveries for the various phases of the project. What usually happens in these cases is that the implementative work stops a couple of weeks before the deadline, and a massive documentation work is performed, all in once, until the delivery is pushed out of the door. On the contrary, in standard requirements-driven projects there is no temporal gap between code writing and documentation writing (cf. also Metzeger & Boddie, 1996).

This situation can be generalized to grammar writing, where the standard practice seems to confine the writing of the documentation to a kind of leftovers. As a consequence, in many cases, documentation does not reflect the rationale under certain choices of the implementation, but reduces to an informal description of formally represented linguistic structures. Moreover, in successive releases of the same implementation, the links between the documentation and the implementation tend to become weaker and weaker. In big projects it is almost impossible to ensure the coherence between the implementation and the documentation.

This situation is particularly problematic in cases of distributed grammar development, when more sites are involved in cooperative work. Under these circumstances, lack of synchronization between documentation and real code could cause serious communication problems and a general delay in the work flow.

Also, both reusability and usability are affected by poor or incoherent documentation. On the side of *reusability* , the costs for learning and maintaining an undocumented grammar are often comparable to the costs of a development from scratch. On the side of *usability*, grammar documentation is the base for producing final user documentation, without which no natural language system will ever be able to attract any industrial user (cf. Zoeppritz, 1995).

2.2 Documentation in Grammar Engineering

One of the key point of recent developments in Grammar Engineering is represented by the convergence of certain linguistic theories (e.g. LFG and HPSG) and real grammar development (cf. Cole & al 1997, Ch. 3.3). Thus, certain theoretical results can be easily incorporated in actual implementations, and certain computational treatments have proved to be able to provide valuable hints to theoretical research. This mutual relationship constitutes a good rationale for the view of grammar writing *mainly* as documentation writing. Both the phase of grammar design and implementation could be conceived as the production of a set of abstract linguistic generalizations, where the actual implementative platform only plays a role in restricting the power of the tools to express such generalizations. Indeed, as soon as migration tools among different platforms are available (cf. Dini 1997, Bloch 1997, EAGLES 1996) the concrete syntax of the implementation plays a much lighter role than in the past, and the documentation becomes, in a sense, *the grammar.*[1] From the opposite perspective, the availability of clear and well designed documentation would would make grammar reports attractive for theoretical linguists (Cf. Erbach & Uszkoreit 1990).

3 HyperGram

HyperGram (Hypertextual Grammars) is a model for grammar development and documentation inspired to the idea of *literate programming*, which was first proposed by Knuth (1974) (cf. Knuth (1992), for an overview). Actually, the main source of inspiration is the hyper-literate programming approach (Steinman & Yates 1995, 1996), a revision of literate programming stressing the importance of hypertextual connections between pieces of code, in order to increase both the quality of the documentation and the productivity of the programmer. Therefore HyperGram is meant to serve as a tool both for documenting grammars and for facilitating the work of the grammar engineer.[2]

[1]The similarity with the literate programming approach immediately comes to mind. Such a similarity will be emphasized in section 3 where the HyperGram model will be presented.

[2]In a sense, documentation tools need to be tailored with respect to the kind of linguistic organization (or linguistic theory) which is chosen as the basis for the implementation. In the case we are considering in this paper, we have in mind a typed, unification-based user language, which fits very well the hypertextual organization of the lingware. Indeed values of attributes are

The main goals that the model is intended to reach, which, we think, constitute possible answers to real needs of a typical user of systems like ALEP or PAGE in the context of a grammar development project, are the following:

1. It allows to produce an updated printed documentation at any stage of the process of grammar development, avoiding inconsistencies between the real grammar code and the code exemplified in the report; inconsistencies of this sort are frequent in standard reports.
2. It produces an hypertextual version of the documented resources, which can be directly made available for public consultation, e.g. via the Internet;
3. It provides the grammar writer with the possibility of accessing the lingware during the development or debugging work, by means of a unique hypertextual interface, which emphasizes user-friedliness and efficiency. This interface allows the direct interaction with the real grammar modules which can be edited, modified, compiled and so on.

4 HyperGram's Modules

The general organization of the HyperGram system[3] is shown in fig. 1, where the relations between the various modules and the linguistic resources are made explicit. The basic idea is that during the process of grammar production an integrated HTML text containing both the documentation and the links to the lingware is maintained. Such a report will be available at any time either for browsing (using a standard http compliant program) or printing. The coherence between the HTML version of the lingware and the one which is actually compiled is preserved through a set of automatic compilation steps completely transparent to the grammar engineer. Also the distinction between "reporting" and "implementing" looses much of its importance, as relevant pieces of documentation can be accessed and modified in a hypertextual fashion directly during grammar editing. The single conversion steps are described in details in the remaining sections.

easily understandable as links to piece of information contained in the type system. These pieces of information, the types, refer, in turn, to other sets linguistic constraints which can be analougously interpreted in a hypertextual fashion.

[3]The instance of HyperGram that we will describe in the following is centered on documentation of ALEP lingware. Analogous considerations hold for the PAGE version.

4.1 HG-conversion

The module labeled as `HG-conversion` in fig. 1 is a program written in **emacs-lisp** aimed at assigning an hypertextual structure to the lingware files used by the system. The various conversion steps are the following:

- The lingware files (written in plain ASCII) are assigned a basic HTML tagging, in such a way that the original indentation of the code (for instance the one automatically produced by **emacs** modes for grammar editing) is maintained (using the <PRE> tag). The original lingware files are of course left unchanged, while the HTML files (HTML lingware, in the figure) are saved in a directory specified by the user when the HyperGram system is configured.
- Some hypertextual links among linguistic descriptions used in the lingware are expressed by means of the standard anchor mechanism of HTML, by interpreting the grammar formalism. The main idea is to use hypertextual links to express the logical relations holding among the various objects involved in the grammar structure, namely types, phrase structure rules and macros (or templates). For instance, whenever a type or a macro is used as the value of an attribute in a linguistic description of any sort (i.e. a type declaration, a rule, or the body of a macro definition), an HTML anchor is produced, pointing to the definition of the relevant type or macro; when a type is introduced in the type declaration, it is anchored to the fragments of hierarchy where it appears. And so on.

4.2 Integrated Report

In order to produce an integrated HTML version of the documentation, the following preconditions have to be satisfied:

- Every rule, or type declaration or macro definition in the lingware is labeled by means of an unambiguous identifier. This identifier can be expressed either as the value of a specific attribute in the body of the expression, or as an external comment.
- Wherever a particular piece of lingware code is specifically documented in the report, a pointer to its identifier (in the sense specified above) is inserted, rather than a copy of the code itself; let us refer to that pointer as *main pointer*. If the code is referred to in other sections of the report, then a different pointer to the same identifier has to be established (*secondary pointer*).

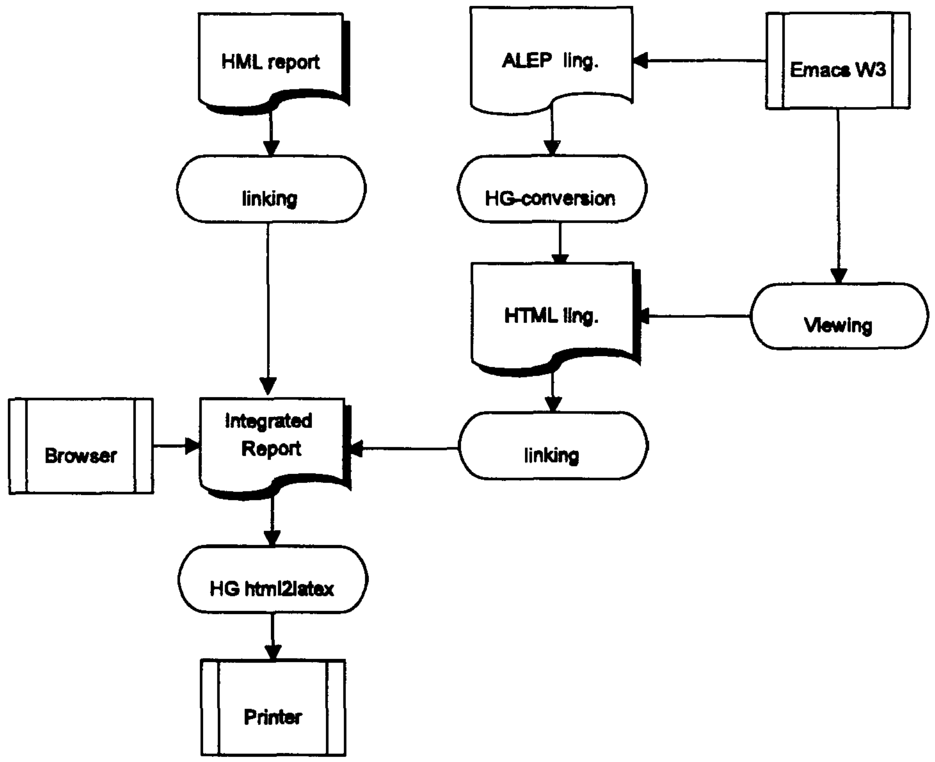

Figure 1: The general work flow of HyperGram

Unlike the main pointer, which must be unique, it is possible to specify many secondary pointers to the same identifier.

Once these relations between the documentation text and the documented code are made explicit by the grammar writer, the integrated hypertextual report is automatically produced by a compiler (the module labeled `linking` in the figure).

The work done by this compiler is rather simple. It converts the pointers and identifiers described above into HTML anchors, with the following general organization:

- The pointers used in those sections of the report where parts of code are documented (i.e. the *main pointers*) are translated into anchors to the appropriate rules (or types or macros) in the **`HTML-lingware`** files containing them;
- Similar anchors are established in all the other points of the report where a rule is referred to by means of a *secondary pointer*;
- In the HTML-lingware, each object is anchored to the section of the report where it is more specifically described: namely, where its *main pointer* is declared.

In this way an updated, standard HTML-based hypertextual version of the whole grammatical module and of the related documentation is in principle available at any time for Intranet/Internet consultation.

4.3 Documentation Printing

In spite of our belief that the best format to deliver grammar documentation is the hypertextual one, there might be case where also printed documentation is required. Thus we developed a module aimed at producing a printable version of the documentation, labeled as **`HG html2latex`** in fig. 1. A set of **`emacs-lisp`** functions is devoted to convert the original hypertextual documentation, which, as described above, is assumed to have been originally written in HTML format, into a printable LaTeXdocument.

The **`HG html2latex`** module interprets the pointers and the identifiers declared in the report and in the grammar files respectively, as described above in section 4.2. As a result, every rule or type or macro is included in the printed report in only one point, namely where the main pointer to it has been previously declared. This is automatically done by the program, which retrieves the parts of code associated to each pointer from the actual grammar files, and includes them in the report at the appropriate place.

In all the other parts of the report where a piece of lingware is mentioned, but not specifically described, a LaTeX internal cross-reference is introduced. This is precisely the reason for the use of different types of pointers in the report (see 4.2 above). Indeed, under this assumption, the point where the code must appear in the printed report is unambiguously specified by means of the unique main pointer. In the hypertextual version of the integrated report this kind of distinction is not relevant, as any reference to the lingware is simply an anchor to a specific part of a lingware file.

4.4 Browsing and Editing the Lingware

The interface chosen in the HyperGram model for the hypertextual navigation within the lingware and the associated documentation is the emacs-internal HTML browser **emacs-w3**.

A set of specific emacs-lisp functions have been added in order to integrate the standard navigation procedures with the possibility for the user to access the source lingware, to edit it and, possibly, to compile it in the relevant grammar development platform. Crucially, the HTML version of lingware files should never be accessed by the grammar developer; it is automatically produced or updated once the lingware has been modified. Moreover, the user friendliness of the navigation through the lingware is enhanced by making explicit the type of relation expressed by an anchor (for instance the relation between a type used as a value in a rule and its definition in the type theory) by means of a special formatting, such as a particular font or color for the text.

Here is how the browsing mechanism within grammar files will look from the point of view of the user:

- having an existing grammatical file (call it **rg_file**), written in the relevant user language, a single command in an **emacs** buffer will allow the user 1) to create an updated HTML file (**hg_file**) bearing all the information described above in terms of internal and external hypertextual links; 2) to invoke the **emacs-w3** browser on that file; and 3) to browse it.
- if an anchor that points to a different grammar file is followed, the relevant hg_file is generated if it does not exist, while if it is already existent, it is updated when necessary (i.e. if the corresponding rg_file has been modified after the date of its creation).
- when browsing an hg_file in a **emacs-w3** buffer, a single command allows to switch to the underlying rg_file, with the cursor located in the same region. A parallel command allows to go back to the hypertext, which is automatically updated if necessary, namely if the rg_file has been modified; also in this case the cursor location is maintained.

The whole mechanism allows the grammar writer to systematically use hypertextual navigation within the grammatical module, taking a possible advantage from the fact that the hypertextual model proposed here makes some relations among linguistic objects explicit. Since it is important to keep in mind these relations when working on a complex grammar, with a highly structured type theory, the hypertextual approach could provide a substantial help to the grammar writer. In many cases, it could represent a preferable alternative to the use of more sophisticated tools for graphical representation of linguistic objects, in that, on the one hand, it is fully integrated with the editing tool, and, on the other hand, it covers in an uniform way all the object used in the grammar module, not only the type theory.

5 Conclusions

The system is oriented towards the need of fast grammar development, easy training for researchers which start working on an existing grammar, and high quality documentation, which are undoubtedly crucial points for the success of a grammar project based on available language engineering platforms and for the reusability of its results. These needs emerge with a particular relevance when considering distributed projects for grammar development where both information sharing among cooperating groups and public dissemination of the results via the World Wide Web become crucial.

Also, the possibility of producing in short time paper versions of the documentation seems to fit the needs of a standard grammar development projects, where many checking points are still based on the evaluation of printed documentation.

References

Boitet, Charles, 1989, Sofwteare and Lingware Engineering in Modern M(A)T Systems. In I.S. Batori, W. Lenders and W. Putschke, *Computational Linguistics - An International Handbook* *on Computer Oriented Language Research and Applications*, Walter de Gruyter, Berlin and New York.

Booch, Grady (1996). *Object Solutions.* Addison-Wesley, Menlo Park, CA.

Cole, Ronald; Joseph Mariani; Hans Uszkoreit; Annie Zaenen; Victor Zue, (1997). *Survey of the State of the Art in Human Language Technology.* Web version: `http://www.cse.ogi.edu/CSLU/HLTsurvey/`.

Dini Luca (1997). The ALEP2PAGE Grammar Translator, In *Proceedings of the 3rd ALEP User Group Workshop.* IAI, pp. 27-33.

EAGLES (1996). *Formalisms Working Group Final Report.* Version of September 1996.

Erbach, Gregor; Hans Uszkoreit (1990). *Grammar Engineering: Problems and Prospects.* CLAUS Report No. 1, July 1990.

Metzeger, Philip; John Boddie (1996). *Managing a Programming Project.* Prentice Hall PTR, New Jersey.

Knuth, Donald E. (1974). Computer Programming as an Art. In *Communications of the ACM* 17, pp. 667-673.

Knuth, Donald E. (1992). *Literate Programming.* CSLI Lecture Notes, no. 27.

Schütz, Jörg (1995). Language Engineering - Fixing Positions. IAI Memo, ME0695

Steinman, Jan; Barbara Yates (1995). Managing Project Documents.In *The Smalltalk Report*, June 1995, also available at `http://www.Bytesmiths.com/pubs/index.html`.

Steinman, Jan; Barbara Yates (1996). Documents on the Web in *The Smalltalk Report*, June/July 1996, also available at `http://www.Bytesmiths.com/pubs/index.html`.

Zoeppritz, Magdalena (1995). Software Ergonomics of Natural Language Systems, in *Language Engineering*, Gerhard Heyer and Hans Haugeneder Eds., Vierweg, Wiesbaden.