Document Structure and Multilingual Authoring

Caroline Brun **Marc Dymetman** **Veronika Lux**
Xerox Research Centre Europe
6 chemin de Maupertuis
38240 Meylan, France
{brun,dymetman,lux}@xrce.xerox.com

Abstract

The use of XML-based authoring tools is swiftly becoming a standard in the world of technical documentation. An XML document is a mixture of structure (the tags) and surface (text between the tags). The structure reflects the choices made by the author during the top-down stepwise refinement of the document under control of a DTD grammar. These choices are typically choices of meaning which are independent of the language in which the document is rendered, and can be seen as a kind of interlingua for the class of documents which is modeled by the DTD. Based on this remark, we advocate a radicalization of XML authoring, where the semantic content of the document is accounted for exclusively in terms of choice structures, and where appropriate rendering/realization mechanisms are responsible for producing the surface, possibly in several languages simultaneously. In this view, XML authoring has strong connections to natural language generation and text authoring. We describe the IG (Interaction Grammar) formalism, an extension of DTD's which permits powerful linguistic manipulations, and show its application to the production of multilingual versions of a certain class of pharmaceutical documents.

1 Introduction

The world of technical documentation is forcefully moving towards the use of authoring tools based on the XML markup language (W3C, 1998; Pardi, 1999). This language is based on grammatical specifications, called DTD's, which are roughly similar to context-free grammars[1] with an arbitrary number of non-terminals and exactly one predefined terminal called `pcdata`. The `pcdata` terminal has a special status: it can dominate any character string (subject to certain restrictions on the characters allowed). Authoring is seen as a top-down interactive process of step-wise refinement of the root nonterminal (corresponding to the whole document) where the author iteratively selects a rule for expanding a nonterminal already present in the tree and where in addition s/he can choose an arbitrary sequence of characters (roughly) for expanding the `pcdata` node. The resulting document is a mixture of tree-like *structure* (the context-free derivation tree corresponding to the author's selections), represented through tags, and of *surface*, represented as free-text (PCDATA) between the tags.

We see however a tension between the structure and surface aspects of an XML document:

- While structural choices are under system control (they have to be compatible with the DTD), surface choices are not.[2]
- Surface strings are treated as unanalysable chunks for the styling mechanisms that render the XML document to the reader. They can be displayed in a given font or moved around, but they lack the internal structure that would permit to "re-purpose" them for different rendering situations, such as displaying on mobile telephone screens, wording differently for a specific audience, or producing prosodically adequate phonetic output. This situation stands in contrast with the underlying philosophy of XML, which emphasizes the separation between content specification and the multiple situations in which this content can be exploited.
- Structural decisions tend to be associated with choices of meaning which are independent of the language in which the document is rendered. Thus for instance the DTD for an aircraft maintenance manual might distinguish between two kinds of risks: `caution` (material damage risk) and `warning` (risk to the operator). By selecting one of these options (a choice that will lead to further lower-level choices), the author takes a decision of a semantic nature, which is quite independent of the language in which the document is to be rendered, and which could be exploited to produce multilingual versions of the

[1] But see (Wood, 1995; Prescod, 1998) for discussions of the differences.

[2] With the emergence of *schemas* (W3C, 1999a), which permit some typing of the surface (`float`, `boolean`, `string`, etc.), some degree of control is becoming more feasible.

document. By contrast, a PCDATA string is language-specific and ill-suited for multilingual applications.

These remarks point to a possible radical view of XML authoring that advocates that surface strings be altogether eliminated from the document content, and that author choices be all under the explicit control of the DTD and reflected in the document structure. Such a view, which is argued for in a related paper (Dymetman et al., 2000), emphasizes the link between XML document authoring and multilingual text authoring/generation (Power and Scott, 1998; Hartley and Paris, 1997; Coch, 1996): the choices made by the author are treated as a kind of interlingua (specific to the class of documents being modelled), and it is the responsibility of appropriate "rendering" mechanisms to produce actual text from these choices in the different languages[3] under consideration.

For such a program, existing XML tools suffer however from serious limitations. First, DTD's are too poor in expressive power (they are close to context-free grammars) for expressing dependencies between different parts of the document, an aspect which becomes central as soon as the document *micro-structure* (its fine-grained semantic structure) starts to play a prominent role, as opposed to simply its *macro-structure* (its organization in large semantic units, typically larger than a paragraph). Second, current rendering mechanisms such as CSS (Cascading Style Sheets) or XSLT (XLS transformation language) (W3C, 1999b) are ill-adapted for handling even simple linguistic phenomena such as morphological variation or subject-verb agreement.

In order to overcome these limitations, we are using a formalism, *Interaction Grammars* (IG), a specialization of Definite Clause Grammars (Pereira and Warren, 1980) which originates in A. Ranta's Grammatical Framework (GF) (Ranta; Mäenpää and Ranta, 1999; Dymetman et al., 2000), a grammatical formalism based on Martin-Löf's Type Theory (Martin-Löf, 1984) and building on previous experience with interactive mathematical proof editors (Magnusson and Nordström, 1994). In this formalism, the carrier of meaning is a *choice tree* (called "abstract tree" in GF), a strongly typed object in which dependencies between substructures can be easily stated using the notion of *dependent types*.

The remainder of this paper is organized as follows. In section 2, we give a high level overview of the Multilingual Document Authoring (MDA) system that we have developed at XRCE. In section 3, we present in some detail the formalism of Interaction Grammars. In section 4, we describe an application of MDA to a certain domain of pharmaceutical documents.

[3]The word "language" should be understood here in an extended sense that not only covers English, French, etc., but also different styles or modes of communication.

2 Our approach to Multilingual Document Authoring

Our Multilingual Document Authoring system has the following main features:

First, *the authoring process is monolingual, but the results are multilingual.* At each point of the process the author can view in his/her own language the text s/he has authored so far, and areas where the text still needs refinement are highlighted. Menus for selecting a refinement are also presented to the author is his/her own language. Thus, the author is always overtly working in the language s/he nows, but is implicitly building a language-independent representation of the document content. From this representation, the system builds multilingual texts in any of several languages simultaneously. This approach characterizes our system as belonging to an emerging paradigm of "natural language authoring" (Power and Scott, 1998; Hartley and Paris, 1997), which is distinguished from natural language generation by the fact that the semantic input is provided interactively by a person rather than by a program accessing digital knowledge representations.

Second, *the system maintains strong control both over the semantics and the realizations of the document.* At the semantic level, dependencies between different parts of the representation of the document content can be imposed: for instance the choice of a certain chemical at a certain point in a maintenance manual may lead to an obligatory warning at another point in the manual. At the realization level, which is not directly manipulated by the author, the system can impose terminological choices (e.g. company-specific nomenclature for a given concept) or stylistic choices (such as choosing between using the infinitive or the imperative mode in French to express an instruction to an operator).

Finally, and possibly most distinctively, *the semantic representation underlying the authoring process is strongly document-centric and geared towards directly expressing the choices which uniquely characterize a given document in an homogeneous class of documents belonging to the same domain.* Our view is document-centric in the sense that it takes as its point of departure the widespread practice of using XML tools for authoring the *macro-structure* of documents, and extends this practice towards an account of their *micro-structure*. But the analysis of the micro-structure is only pushed as far as is necessary in order to account for the variability inside the class of documents considered, and not in terms of the ultimate meaning constituents of language. This micro-structure can in general be determined by studying a corpus of documents and by

exposing the structure of choices that distinguish a given document from other documents in this class. This structure of choices is represented in a choice tree, which is viewed as *the* semantic representation for the document.[4] One single choice may be associated with text realizations of drastically different granularities: while in a pharmaceutical document the choice of an ingredient may result in the production of a single word, the choice of a "responsability-waiver" may result in a long stereotypical paragraph of text, the further analysis of which would be totally counter-productive.

3 Interaction Grammars

Let us now give some details about the formalism of Interaction Grammars. We start by explaining the notion of choice tree on the basis of a simple context-free grammar, analogous to a DTD.

Context-free grammars and choice trees Let's consider the following context-free grammar for describing simple "addresses" in English such as `"Paris, France"`:[5]

```
address --> city, ",",
            country.
country --> "France".
country --> "Germany".
city --> "Paris".
city --> "Hamburg".
city --> "the capital of",
         country.
```

What does it mean, remembering the XML analogy, to author a "document" with such a CFG? It means that the author is iteratively presented with partial derivation trees relative to the grammar (partial in the sense that leaves can be terminals or non-terminals), and at each given authoring step both selects a certain nonterminal to "refine", and also a given rule to extend this non-terminal one step further; this action is repeated until the derivation tree is complete.

If one conventionally uses the identifier $\texttt{nonterminal}_\texttt{i}$ to name the i-th rule expanding the nonterminal `nonterminal`, then the collection of choices made by the author during a session can be represented by a *choice tree* labelled with rule identifiers, also called *combinators*. An example of such a tree is `address1(city2,country2)` which corresponds to choices leading to the output `"Hamburg, Germany"`.[6] In practice, rather than using combinator names which strictly adhere to this numbering scheme, we prefer to use mnemonic names directly relating to the meaning of the choices. In the sequel we will use the names `adr, fra, ger, par, ham, cap` for the six rules in the example grammar. The choice tree just described is thus written `adr(ham,ger)`.

Making choice trees explicit As we have argued previously, choices trees are in our view the central repository of document content and we want to manipulate them explicitely. Definite Clause Grammars represent possibly the simplest extension of context-free grammars permitting such manipulation. Our context-free grammar can be extended straightforwardly into the DCG:[7]

```
address(adr(Co,C)) --> city(C), ",",
                       country(Co).
country(fra) --> "France".
country(ger) --> "Germany".
city(par) --> "Paris".
city(ham) --> "Hamburg".
city(cap(Co)) --> "the capital of",
                  country(Co).
```

What these rules do is simply to construct choice trees recursively. Thus, the first rule says that if the author has described a city through the choice tree `C` and a country through the choice tree `Co`, then the choice tree `adr(Co,C)` represents the description of an address.

If now, in this DCG, we "forget" all the terminals, which are language-specific, by replacing them with the empty string, we obtain the following "abstract grammar":

```
address(adr(Co,C)) --> city(C), country(Co).
country(fra) --> [].
country(ger) --> [].
city(par) --> [].
city(ham) --> [].
city(cap(Co)) --> country(Co).
```

which is in fact equivalent to the definite clause *program*:[8]

[4] This kind of semantic representation stands in contrast to some representations commonly used in NLP, which tend to emphasize the fine-grained predicate-argument structure of sentences independently of the productivity of such analyses *for a given class of documents*.

[5] For compatibility with the notations to follow, we use lowercase to denote nonterminals, and quoted strings to denote terminals, rather than the more usual uppercase/lowercase conventions.

[6] Such a choice tree can be projected into a derivation tree in a straightforward way, by mapping a combinator $\texttt{nonterminal}_\texttt{i}$ into the nonterminal name `nonterminal`, and by introducing terminal material as required by the specific rules.

[7] According to the usual logic programming conventions, lowercase letters denote predicates and functors, whereas uppercase letters denote metavariables that can be instanciated with terms.

[8] In the sense that rewriting the nonterminal goal `address(adr(Co,C))` to the empty string in the DCG is equivalent to proving the goal `address(adr(Co,C))` in the program,

```
address(adr(Co,C)) :- city(C), country(Co).
country(fra).
country(ger).
city(par).
city(ham).
city(cap(Co)) :- country(Co).
```

This abstract grammar (or, equivalently, this logic program), is language independent and recursively defines a set of well-formed choice trees of different *categories*, or *types*. Thus, the tree `adr(ham,ger)` is well-formed "in" the type `address`, and the tree `cap(fra)` well-formed in the type `city`.

Dependent Types In order to stress the type-related aspects of the previous tree specifications, we are actually using in our current implementation the following notation for the previous abstract grammar:

```
adr(Co,C)::address --> C::city,
                       Co::country.
fra::country --> [].
ger::country --> [].
par::city --> [].
ham::city --> [].
cap(Co)::city --> Co::country.
```

The first rule is then read: "if `C` is a tree of type `city`, and `Co` a tree of type `country`, then `adr(Co,C)` is a tree of type `address`", and similarly for the remaining rules.

The grammars we have given so far are deficient in one important respect: there is no dependency between the city and the country in the same address, so that the tree `adr(ham,fra)` is well-formed in the type `address`. In order to remedy this problem, dependent types (Ranta; Martin-Löf, 1984) can be used. From our point of view, a dependent type is simply a type that can be parametrized by objects of other types. We write:

```
adr(Co,C)::address --> C::city(Co),
                       Co::country.
fra::country --> [].
ger::country --> [].
par::city(fra) --> [].
ham::city(ger) --> [].
cap(Co)::city(Co) --> Co::country.
```

in which the type `city` is now parametrized by objects of type `country`, and where the notation `par::city(fra)` is read as "`par` is a tree of the type: city of `fra`".[9]

which is another way of stating the well-known duality between the rewriting and the goal-proving approaches to the interpretation of Prolog.

[9] In terms of the underlying Prolog implementation, '::' is simply an infix operator for a predicate of arity 2 which relates an object and its type, and both simple and dependent types are handled straighforwardly.

Parallel Grammars and Semantics-driven Compositionality for Text Realization We have just explained how abstract grammars can be used for specifying well-formed typed trees representing the content of a document.

In order to produce actual multilingual documents from such specifications, a simple approach is to allow for parallel realization English, French, ..., grammars, which all have the same underlying abstract grammar (program), but which introduce terminals specific to the language at hand. Thus the following French and English grammars are parallel to the previous abstract grammar:[10]

```
adr(Co,C)::address --> C::city(Co), ",",
                                Co::country.
fra::country --> "France".
ger::country --> "Germany".
par::city(fra) --> "Paris".
ham::city(ger) --> "Hamburg".
cap(Co)::city(Co) --> "the capital of",
                                Co::country.

adr(Co,C)::address --> C::city(Co), ",",
                                Co::country.
fra::country --> "la France".
ger::country --> "l'Allemagne".
par::city(fra) --> "Paris".
ham::city(ger) --> "Hambourg".
cap(Co)::city(Co) --> "la capitale de",
                                Co::country.
```

This view of realization is essentially the one we have adopted in the prototype at the time of writing, with some straighforward additions permitting the handling of agreement constraints and morphological variants. This simple approach has proven quite adequate for the class of documents we have been interested in.

However, such an approach sees the activity of generating text from an abstract structure as basically a compositional process on *strings*, that is, a process where strings are recursively associated with subtrees and concatenated to produce strings at the next subtree level. But such a direct procedure has well-known limitations when the semantic and syntactic levels do not have a direct correspondence (simple example: ordering a list of modifiers around a noun). We are currently experimenting with a powerful extension of string compositionality where the objects compositionally associated with abstract subtrees are not strings, but syntactic representations with rich internal structure. The text

[10] Because the order of goals in the right-hand side of an abstract grammar rule is irrelevant, the goals on the right-hand sides of rule in two parallel realization grammars can appear in a different order, which permits certain reorganizations of the linguistic material (situation not shown in the example).

itself is obtained from the syntactic representation associated with the total tree by simply enumerating its leaves.

In this extended view, realization grammars have rules of the following form:

```
a1(B,C,...)::a(D,...)-Syn -->
    B::b(E,...)-SynB,
    C::c(F,...)-SynC,
    ...
    {constraints(B,C,...,D,E,F,...)},
    {compose_english(SynB, SynC, ..., Syn)}.
```

The rule shown is a rule for English: the syntactic representations are language dependent; parallel rules for the other languages are obtained by replacing the `compose_english` constraint (which is unique to this rule) by constraints appropriate to the other languages under consideration.

Heterogeneous Trees and Interactivity Natural language *authoring* is different from natural language *generation* in one crucial respect. Whenever the abstract tree to be generated is incomplete (for instance the tree `cap(Co)`), that is, has some leaves which are yet uninstanciated variables, the generation process should not proceed with nondeterministically enumerating texts for all the possible instanciations of the initial incomplete structure. Instead it should display to the author as much of the text as it can in its present "knowledge state", and enter into an interaction with the author to allow her to further refine the incomplete structure, that is, to further instanciate some of the uninstanciated leaves. To this purpose, it is useful to introduce along with the usual combinators (`adr`, `fra`, `cap`, etc.) new combinators of arity 0 called *typenames*, which are notated **type**, and are of type `type`. These combinators are allowed to stand as leaves (e.g. in the tree `cap(country)`) and the trees thus obtained are said to be *heterogeneous*. The typenames are treated by the text generation process as if they were standard semantic units, that is, they are associated with text units which are generated "at their proper place" in the generated output. These text units are specially phrased and highlighted to indicate to the author that some choice has to be made to refine the underlying type (e.g. obtaining the text "la capitale de PAYS"). This choice has the effect of further instanciating the incomplete tree with "true" combinators, and the generation process is iterated.

4 An Application to Pharmaceutical Documents

4.1 Corpus selection

Our corpus consists in drug notices extracted from "Le VIDAL®de la Famille" (Éditions du Vidal, 1998), a practical book about health made for the general public. Le VIDAL® includes a collection of notices for around 5 500 drugs available in France. As the publisher, *OVP-Éditions du Vidal* has taken care of homogeneity across the notices, reformatting and reformulating source information. The main source are the New Drug Authorizations (Autorisation de Mise sur le Marché), regulatory documents written by pharmaceutical laboratories and approved by legal authorities.

Relative to multilingual document authoring, this corpus has three features which we considered highly desirable: (1) it deals with a restricted semantic domain (for which various terminological resources are available), (2) it is a homogeneous collection of documents all complying to the same division in sections and sub-sections, (3) there is a strong trend in international bodies such as the EEC towards making drug package notices (which are similar to VIDAL notices) available in multilingual versions strictly aligned on a common model.[11]

4.2 Corpus analysis

An analysis of a large collection of notices from Le VIDAL® de la famille, describing different drugs, from different laboratories was conducted in order to identify:

- the structure of a notice,
- the semantic dependencies between elements in the structure.

For this task, all the *meta-information* available is useful, in particular: explanations provided by Le VIDAL® de la famille and help of a domain expert. Corpus study was a necessary preliminary task before modeling the notices in the IG formalism presented in section 2.

4.2.1 Structure

Notices from Le VIDAL® are all built on the same model, including a title (the name of the drug, plus some general information about it), followed by sections describing the main characteristics of the drug: general description, composition, indications, contraindications, warnings, drug interactions, pregnancy and breast-feeding, dosage and administration, possible side effects. This initial knowledge about the semantic content of the document is captured with a first, simple context free rule, such as:

```
vidalNotice(T,D,C,I,CI,W,DI,PaBF,DaA,PSI)::notice
      -->
      T::title,
      D::description,
      C::composition,
```

[11] A similar but less extended corpus was previously built by the third author as the basis for a prototype of multilingual document authoring using GF.

```
        I::indications,
        CI::contraindications,
        W::warnings,
        DI::drugsInteraction,
        PaBF::pregnancyAndBreastFeeding,
        DaA::dosageAndAdmin,
        PSI::possibleSideEffects.
```

Each section is associated with context-free rules that describe its internal structure:

```
vidalTitle(N,AP, ..., ...)::title
        -->
        N::nameOfDrug,
        AP::activePrinciples,  ...  .

vidalDescription(N,PF,P...)::description
        -->
        ['DESCRIPTION'],
        N::nameOfDrug,
        PF::pharmaceutForm,
        P::package, ...  .

vidalDosageAndAdmin(D,A)::dosageAndAdmin
        -->
        ['DOSAGE AND ADMINISTRATION'],
        D::dosage,
        A::administration.

tablet::pharmaceutForm  --> ['tablet'].
eyeDrops:::pharmaceutForm  --> ['eye drops'].
```

At this point, we allow parallel realizations for French and English. So, in addition to the English grammar given above, we have the French grammar:

```
vidalTitle(N, AP, ..., ...)::title
        -->
        N::nameOfDrug,
        AP::activePrinciples,  ...  .

vidalDescr(N,PF,P...)::description
        -->
        ['PRÉSENTATION'],
        N::nameOfDrug,
        PF::pharmaceutForm,
        P::package, ...  .

vidalDosageAndAdmin(D,A)::dosageAndAdmin
        -->
        ['MODE D'EMPLOI ET POSOLOGIE'],
        D::dosage,
        A::administration.

tablet::pharmaceutForm  -->  ['comprimé'].
eyeDrops:::pharmaceutForm  -->  ['collyre'].
```

This first grammar is fully equivalent to a XML DTD that describes the structure of a notice, though it distinguishes finer-grained units than traditional DTDs tends to do.

4.2.2 Modeling dependencies

But IG goes further than XML DTDs with regard to the semantic control of documents: it enables us to express dependencies which may arise in different parts of a document, including long-distance dependencies, through the use of dependent types presented in section 2.
Identification of the dependencies to be modeled was done in a second stage of the corpus study. For example, we identified dependencies between:

- the pharmaceutical form of a given drug (concept *pharmaceutForm*) and its packaging (concept *package*),
- particular ingredients given in the section *composition* and warning instructions given in the section *warnings*,
- categories of patients the drug is intended for in the section *description* and posology indicated for each category in the section *indications*.

To illustrate the modeling task, we now give more details about one particular dependency identified. Intuitively, it appears that there is a strong link between the pharmaceutical form of a given drug and the way it should be administered: tablets are swallowed, eye drops are put in the eyes, powder is diluted in water etc. In our first grammar, the pharmaceutical form concept appears in the *description* section, since the administration way is described in the *dosage and administration* section. The use of dependent types permits to link these sections together according to the pharmaceutical form. The parts of the (English) grammar involved become:

```
vidalNotice(T,D,C,I,CI,W,DI,PaBF,DaA,PSI)::notice
       -->
       T::title,
       D::description(PF),
       C::composition,
       I::indications,
       CI::contraindications,
       W::warnings,
       DI::drugsInteraction,
       PaBF::pregnancyAndBreastFeeding,
       DaA::dosageAndAdmin(PF),
       PSI::possibleSideEffects.

vidalDescription(N,PF,P,...)::description(PF)
        -->
        ['DESCRIPTION'],
        N::nameOfDrug,
        PF::pharmaceutForm,
        P::package, ... .

vidalDosageAndAdmin(D,A)::dosageAndAdmin(PF)
        -->
        ['DOSAGE AND ADMINISTRATION'],
        D::dosage,
```

```
A::administration(PF).
```

The *administration* section should now be described according to the pharmaceutical form it presupposes, several administration ways being compatible with each form:

```
tabletsAdmin1::administration(Tablet)
        -->
        ['Swallow the tablets without
          crunching them.'].

tabletsAdmin2::administration(Tablet)
        -->
        ['Let the tablets melt under
          the tongue.'].

eyeDropsAdmin::administration(EyeDrops)
        -->
        ['Pull the lower eyelid down while
          looking up and squeeze the eye drops,
          so that they fall between the eyelid
          and the eyeball.'].
```

The consequence of such a modeling is a better control of the semantic content of the document in the process of being authored: once the user chooses *tablet* as pharmaceutical form in the section *description*, his choice is restricted between the two concepts *tabletsAdmin1* and *tabletsAdmin2* in the *administration* section. If he chooses *eye drops* as the pharmaceutical form, there is no choice left if the *administration* section: the text fragment corresponding to the concept *eyeDropsAdmin* will be generated automatically in the document.

This example illustrates how dependencies are propagated into the macro-structure, but they can be propagated into the micro-structure as well: for example, in the *description* section, we can express that the packaging of the drugs is also dependent of their form: tablets are packaged in boxes, eye drops in flasks, powder in packets, etc.:

```
vidalDescription(N,P,...)::description(PF)
        -->
        ['DESCRIPTION'],
        N::nameOfDrug,
        PF::pharmaceutForm,
        P::package(PF),  ... .

box::package(Tablet) --> ['Box'].
flask::package(EyeDrops) --> ['Flask'].
```

This example shows that the granularity degree of the linguistic realization can vary from full text segment (administration ways) to single words (forms like tablet, eye drops, powder, etc.). This is highly related to the reusability of the concept: references to specific forms may appear in many parts of the document, while the administration ways are more or less frozen segments.[12]

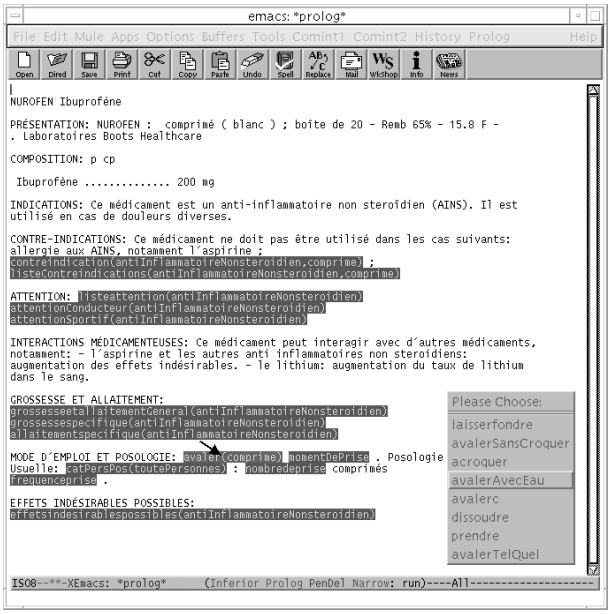

Figure 1: A stage in the authoring of a notice, with French text shown.

The level of generality of dependencies encoded in the grammar needs to be paid attention to: one has to be sure that a given dependency is viable over a large collection of documents in the domain. If a choice made by the grammar writer is too specific, the risk is that it may be not relevant for other documents. For this reason, an accurate knowledge of the corpus is necessary to ensure an adequate coverage of documents in the domain.

4.3 An Example

Screen copies of the IG interface during an authoring process of a VIDAL notice are given on figures 1 and 2. Figure 1 represents the notice authored in French at a given stage. The fields still to be refined by the user appear in dark. When the author wants to refine a given field, a pulldown menu presenting the choices for this field appears on the screen. Here, the author chooses to refine the field avaler in the *administration (mode d'emploi et posologie)* section: the corresponding menu proposes the list of administration ways corresponding to the pharmaceutical form *tablet* he has chosen before. Figure 2 shows the parallel notice in English but one step further, i.e. once he has selected the administration way.

[12] For a discussion of some of the issues regarding the use of templates in natural language generation systems, see (Reiter, 1995).

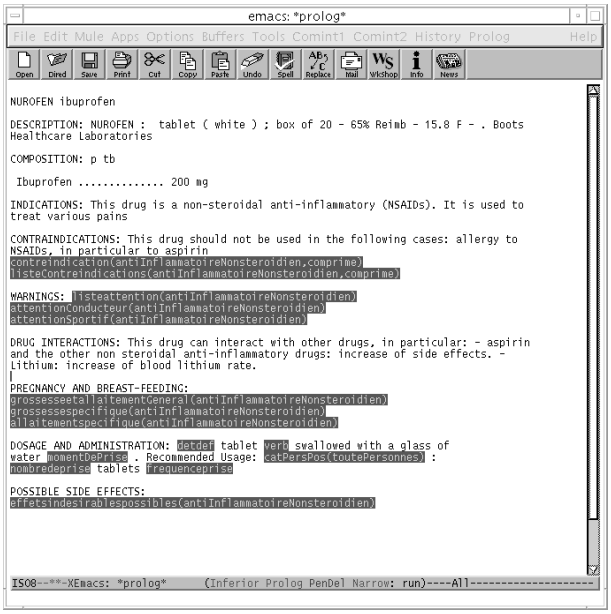

Figure 2: The parallel English notice one authoring step later.

5 Conclusion

XML-based authoring tools are more and more widely used in the business community for supporting the production of technical documentation, controlling their quality and improving their reusability. In this paper, we have stressed the connections between these practices and current research in natural language generation and authoring. We have described a formalism which removes some of the limitations of DTD's when used for the production of multilingual texts and presented its application to a certain domain of pharmaceutical documents.

Acknowledgements Thanks to Jean-Pierre Chanod, Marie-Hélène Corréard, Sylvain Pogodalla and Aarne Ranta for important contributions, discussions and comments.

References

J. Coch. 1996. Evaluating and comparing three text production techniques. In *Proceedings of the 16th International Conference on Computational Linguistics*.

OVP Éditions du Vidal, editor. 1998. *Le VIDAL de la famille*. HACHETTE.

M. Dymetman, V. Lux, and A. Ranta. 2000. XML and multilingual document authoring: Convergent trends. In *Proceedings Coling 2000*, Saarbrücken.

A. Hartley and C. Paris. 1997. Multilingual document production: from support for translating to support for authoring. In *Machine Translation, Special Issue on New Tools for Human Translators*, pages 109–128.

L. Magnusson and B. Nordström. 1994. The ALF proof editor and its proof engine. In *Lecture Notes in Computer Science 806*. Springer.

P. Martin-Löf. 1984. *Intuitionistic Type Theory*. Bibliopolis, Naples.

P. Mäenpää and A. Ranta. 1999. The type theory and type checker of GF. In *Colloquium on Principles, Logics, and Implementations of High-Level Programming Languages, Workshop on Logical Frameworks and Meta-languages*, Paris, September. Available at `http://www.cs.chalmers.se/~aarne/papers/lfm1999.ps.gz`.

W. Pardi. 1999. *XML in Action*. Microsoft Press.

Fernando C. N. Pereira and David H. D. Warren. 1980. Definite clause grammars for language analysis. *Artificial Intelligence*, 13:231–278.

R. Power and D. Scott. 1998. Multilingual authoring using feedback texts. In *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics*, pages 1053–1059.

P. Prescod. 1998. Formalizing SGML and XML instances and schemata with forest automata theory. `http://www.prescod.net/forest/shorttut/`.

A. Ranta. Grammatical Framework work page. `http://www.cs.chalmers.se/~aarne/GF/pub/work-index/index.html`.

E. Reiter. 1995. NLG vs. templates. In *Proceedings of the 5th European Workshop on Natural Language Generation (EWNLG '95)*, pages 95–106, Leiden.

W3C, 1998. *Extensible Markup Language (XML) 1.0*, February. W3C recommendation.

W3C, 1999a. *XML Schema - Part 1: Structures, Part 2 : Datatypes* -, December. W3C Working draft.

W3C, 1999b. *XSL Transformations (XSLT)*, November. W3C recommendation.

D. Wood. 1995. Standard Generalized Markup Language: Mathematical and philosophical issues. *Lecture Notes in Computer Science*, 1000:344–365.