TOWARDS ADAPTIVE TEXT PROCESSING

Ján KOLLÁR
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovakia, tel. 055/602 4179, E-mail: Jan.Kollar@tuke.sk

SUMMARY

A method of adaptive text processing presented in this paper exploits typesetting system LaTeX. Essentially, the goal is to prepare correct LaTeX source files without any knowledge of LaTeX command, using the production tool, being currently developed - an adaptive text editor (ATE). Typographical correctness of target pdf, dvi, or ps documents is left to LaTeX and its style files. However, to prevent repetitive LaTeX translation, the target ATE files are syntactically correct .TEX files. At the same time, ATE is extensible to any new LaTeX command and/or environment. Finally, ATE is able to hide LaTeX commands and environments to a user, depending on the level of his knowledge; in extreme case, no knowledge is required. In this way, typesetting system LaTeX may be used not just by experts, but also by people unfamiliar with it at all. Hence, adaptive text processing may extend to many application areas, supporting them by correct typesetting. ATE is an open language system. Its text source files consists of hidden, visible and modifiable text areas that are terminal symbols of a language, which is a subset of LaTeX. LL(1) grammar of this language is defined using .TeX samples incrementally, in an interactive manner. The defined grammar is a part of ATE scheme. In this paper we concentrate to ATE schemes and the essential approach to their construction.

Keywords: *Text processing, typesetting, LaTeX, open systems, syntax driven editing, text generalization, context-free languages, LL(1) parsing*

1. INTRODUCTION

There is no proof but high deal of evidence that the text editors based on graphic design called WYSIWYGs (What You See Is What You Get) are not sufficiently open with respect to the user requirements in the future. It is simple consideration that careful specification of an editor constructed as a program cannot guarantee the requirements to text processing that are yet unknown. Since a language is a finite representation of infinite number of programs [2,4], the typesetting system LaTeX [5,6,8], being an open language system, rather than a program system, is extensible to satisfy any user requirements in the next.

For example, to get an expression E enclosed in correct semantic brackets, as below

$$[\![E]\!]$$

it is possible to define new commands, named for example `\lp` and `\rp`, in the form

```
\newcommand{\lp}
{\mbox{$[$\hspace{-1.52pt}$[$}}

\newcommand{\rp}
{\mbox{$]$\hspace{-1.52pt}$]$}}
```

Then `$\lp E \rp$` produces the required target text form. Using WYSIWYG, if it does not provide semantic brackets in the set of characters, a user may just think about what is better – to type **[**E**]**, or [[E]], but both forms are formally incorrect.

Using LaTeX, formatting is applied safely to the whole document and *a uniform form is guarranteed*. The document preparation using LaTeX *is flexible*, since no manual reformatting is needed if some other form of a document is required. Since formatting commands are defined in source document and style files exclusively, LaTeX documents *are stable.* Composing separated documents into singleton one never yields a catastrophic scenario, well known when a WYSIWYG is used. Formatting is processed by LaTeX, not by a user.

Finally, *typesetting is the two-dimensional architecture*, i.e. it is a kind of art. Typographically correct fonts, correct spacing, correct mutual relations between font sizes in different parts of document, correct indentation and many other typesetting rules must be considered when a document form is designed by a qualified typograph. It is over the scope of this paper to explain, that formatting rules for a WYSIWYG user are not sufficient to substitute the typesetting rules in typography that have their historical background.

Using LaTeX, a user types his text as an argument of formatting commands and/or environments in a source file .TEX, which is unformatted text file. There are two complications coming out from this approach. First, a user must be familiar with LaTeX commands and environments. Second, the target form (pdf, dvi, or postscript) of a document is invisible until the source file is translated. On the other hand, there is no formatting work left to a user, since LaTeX during compilation is supplied by the typesetting rules given by document style files and possible new definitions in a document. Changing the formatting definitions, the target document is changed uniformly.

For example, if there is a need to produce pdf document with hyperlinks, instead of the command

```
\usepackage[pdftex]{color,graphicx}
```

the next two commands are used

```
\usepackage{graphicx}

\usepackage[plainpages=true,
citecolor=blue,pdfstartview=FitH,
   colorlinks]{hyperref}
```

The target document will be the same, except that all links are alive; clicking on them in Acrobat Reader the corresponding target is approached.

On the other hand, sometimes a user cannot spent his time neither by learning commands nor by new commands and environments definitions. Then it is better to differ the levels of user LaTeX knowledge in which .TEX document is prepared, or exclude such knowledge at all. This is the basic idea leading us to the development of ATE – adaptive text editor. We attend that ATE is not for those familiar with LaTeX or even TeX [3], such as mathematicians, computer scientists, etc. It is addressed rather to administrative areas, providing opportunity to produce documents taken from many sources, preceding both the necessity for LaTeX knowledge and the necessity for manual formatting the documents.

In the past, we have performed experiments just with user communication facilities addressed to display defininition.. In this paper we present the structured nature of ATE, as an open language system, adaptable to any user application area. First we illustrate the unsafe mode, in which LaTeX commands may be used. Then we discuss the form of the ATE scheme and the principles of its construction more precisely. ATE is kind of a syntax driven editor, parametrized by LL(1) language, derived from the LaTeX samples. As a result, LaTeX text is typed correctly, since it is a terminal string belonging to derived language – a subset of LaTeX .

2. UNSAFE MODE

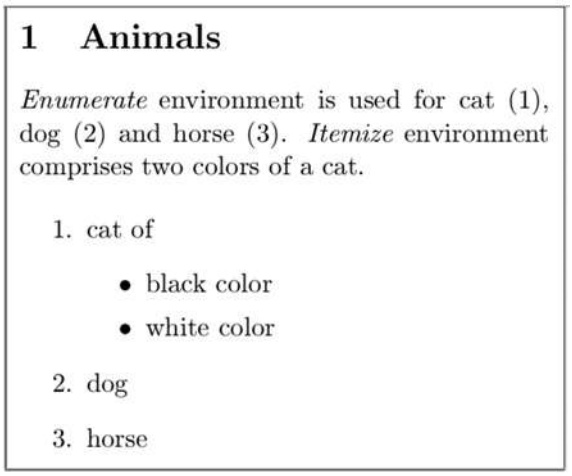
A simple example, introduced in this section, illustrates the unsafe mode of editing, exploited also by currently used shells, such as WINSHELL coming with TeXLive 5.0d distribution [6]. Using ATE, this mode is appropriate for an experienced user able not just to type the text using LaTeX commands, but also to define a scheme for a less experienced user.

Suppose we want to produce a document in final pdf form SAMPLE.PDF as shown in the Fig. 1. This form may be produced from SAMPLE.TEX source LaTeX file introduced in the Example 2.1. Let us briefly comment this text file. The basic font size is 11 points and the basic style is given by `article` style file, as it is defined in `\documentclass` command. We define Eastern (or Middle) European coding of the text (IL2), the usage of pdfLaTeX including colors and graphics, to be able produce target SAMPLE.PDF directly, including graphicx package commands and jpg pictures. We also suppress page numbering. Formatting commands under the comment line serve just to adopt the final document to the required paper and text area. The body of SAMPLE.TEX is introduced in document environment separated by `\begin{document}` and `\end{document}`. It comprises the formatting commands, such as `\section`, `\item` and environments, such as selected by `\begin{itemize}` and `\end{itemize}`.

Example 2.1 SAMPLE.TEX – the source form of the document

```
\documentclass[11pt]{article}

\def\encodingdefault{IL2}
\usepackage[pdftex]{color,graphicx}
\pagestyle{empty}
% comment line
\addtolength\topmargin{-40mm}
\setlength\paperwidth{80mm}
\setlength\textwidth{76mm}
\oddsidemargin=-22mm

\begin{document}

\section{Animals}

{\em Enumerate}  environment is
used for cat (\ref{cat}),  dog
(\ref{dog}) and horse
(\ref{horse}).
{\em Itemize} environment comprises
two colors of a cat.

\begin{enumerate}
\item cat of \label{cat}
\begin{itemize}
\item black color
\item white color
\end{itemize}
\item dog  \label{dog}
\item horse  \label{horse}
\end{enumerate}

\end{document}
```

The result of the translation by pdfTeX (a kind of LaTeX, which produces pdf target files directly) is the file SAMPLE.PDF, which is displayed and/or printed using Acrobat Reader. Typographically correct shapes of letters and correct spacing may be noticed, but also some blur introduced by transformation of displayed document via clipboard into the jpg form included in this Word document.



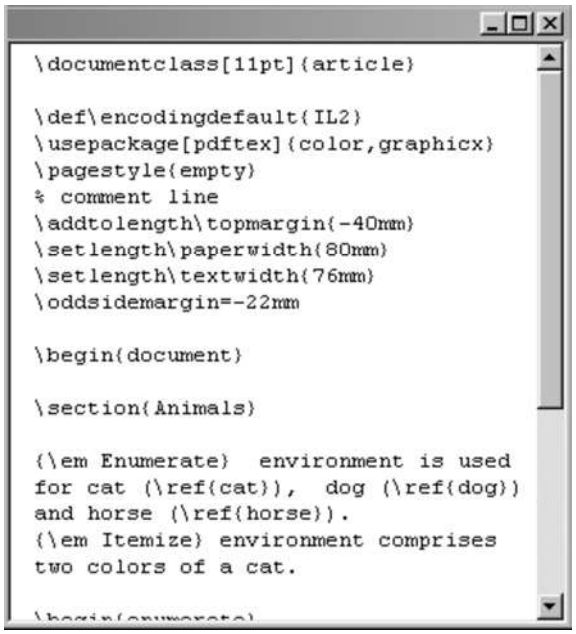
1 Animals

Enumerate environment is used for cat (1), dog (2) and horse (3). *Itemize* environment comprises two colors of a cat.

1. cat of
   - black color
   - white color
2. dog
3. horse

Fig. 1 SAMPLE.PDF - the target form of the document

A user may type his text in unsafe mode using full set of LaTeX commands. Then, the user communication interface looks like in the Fig. 2.



```
\documentclass[11pt]{article}

\def\encodingdefault{IL2}
\usepackage[pdftex]{color,graphicx}
\pagestyle{empty}
% comment line
\addtolength\topmargin{-40mm}
\setlength\paperwidth{80mm}
\setlength\textwidth{76mm}
\oddsidemargin=-22mm

\begin{document}

\section{Animals}

{\em Enumerate} environment is used
for cat (\ref{cat}), dog (\ref{dog})
and horse (\ref{horse}).
{\em Itemize} environment comprises
two colors of a cat.
```

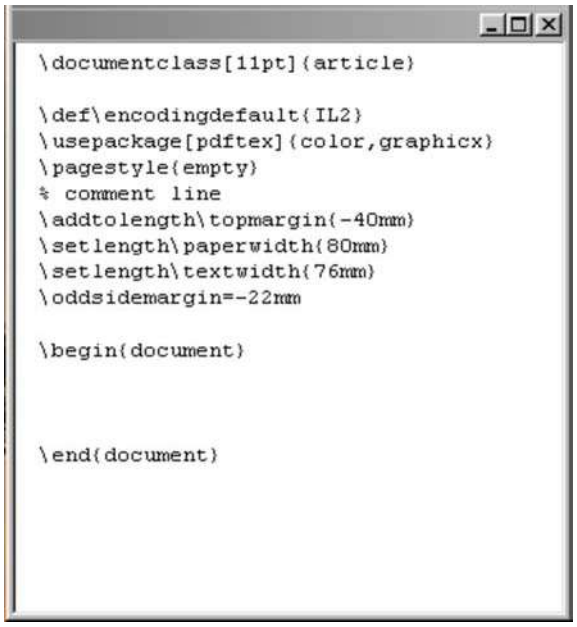
Fig. 2 SAMPLE.TEX - unsafe mode

The aim of safe mode is to produce an unformatted source text file, which is correct when compiled by LaTeX into the target file. Hence we must build a communication interface preserving this safeness.

3. COMMUNICATION INTERFACE

Suppose a text is typed in the unsafe mode, according the Fig. 3.

This text comprises just the definition part, not however a body, which is supposed to be typed by less experienced user.



```
\documentclass[11pt]{article}

\def\encodingdefault{IL2}
\usepackage[pdftex]{color,graphicx}
\pagestyle{empty}
% comment line
\addtolength\topmargin{-40mm}
\setlength\paperwidth{80mm}
\setlength\textwidth{76mm}
\oddsidemargin=-22mm

\begin{document}



\end{document}
```

Fig. 3 SAMPLE.TEX - sample for a scheme

Having been the sampled text typed, it may be used for the definition of a scheme, which is saved into an external file. When reading this scheme by ATE again, let us require the user communication interface will appear, as shown in the Fig. 4.

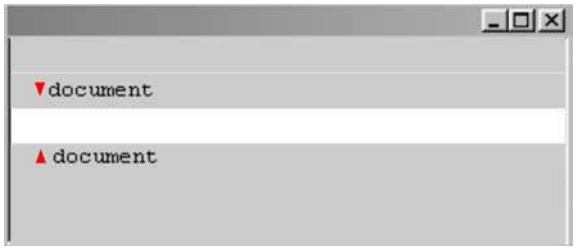


Fig. 4 Simple user communication interface

The communication interface in Fig. 4 is defined by the scheme as follows:

```
% Language Definition
A0 -> T0
A1 -> T1
A2 -> T2
A3 -> T3
S0 -> T0 T1 T2 T3
% Control Buttons
% Display Types
A0 -> H\
A1 -> HR:{\red ▼}{\black document}\
A2 -> M\
A3 -> HR:{\red ▲}{\black document}\
% Displayed Areas
A0 -> /\documentclass[11pt]
      ......................
      \oddsidemargin=-22mm/
A1 -> /\begin{document}/
```

```
A2 -> //
A3 -> /\end{document}/
%
```

The scheme above consists of four parts, which will be discussed later. At this point it may be noticed that the communication interface according to Fig.4 is very poor. Using safe mode, a user is allowed just to type a text without LaTeX commands. Preventing the use of LaTeX commands in safe mode, (even new commands) does not require extremely complicated analysis; suppressing the use of backslash, just a few constructs must be excluded.

On the other hand, the document as required in Fig. 1 can be prepared just in unsafe mode, as shown in Fig. 5. Then all advantages of ATE are lost again, since there is no guarantee that the typed body text is correct. This problem we will solve in this paper by more detailed definition of the first two parts of the scheme, dealing with the language definition and control buttons.

According to the scheme above, we have a language defined just as a sequence of four terminals, generated from starting symbol `S0`, and no control button is available to a user.

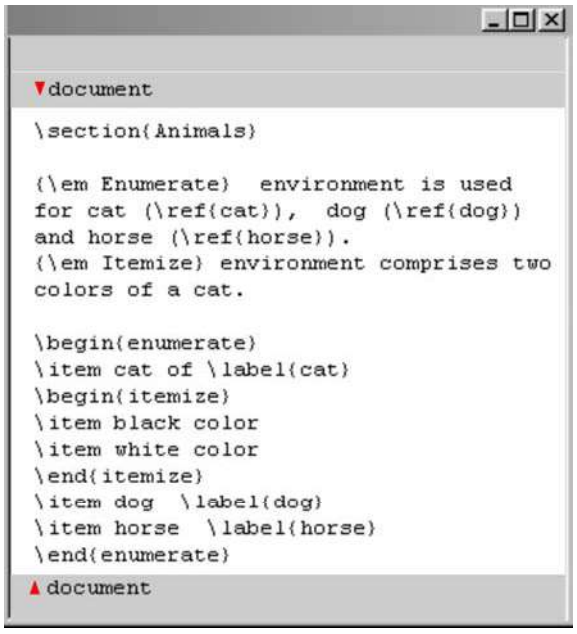


Fig. 5 SAMPLE.TEX - Unsafe mode

4. DISPLAY DEFINITION

Typing a sample, it is used to construct the scheme in the following steps:

1. Selection of Displayed Areas
2. Definition of Display Types
3. Language Definition, and
4. Control Buttons definition

Displayed areas are selected, marking subsequent parts of sampled text, highlighting them (for example using two different background colors). As a result, we obtain a set of areas, `A0,..,An`, associated to the generated buttons of the same names. Hence, Displayed Areas part of the scheme is a mapping of button names to areas (fragments) of source documents. The characters enclosing an area text are the same and such that they do not occur in enclosed text.

Of course, it is supposed that the areas are selected using correct sample, i.e such that has been translated using LaTeX before.

The areas are not necessarily adjacent, since not all parts of text must be associated with buttons, just those required to be included in the scheme. Using just a set of associated buttons it is still impossible to reconstruct the text exactly, since display types are yet undefined. On the other hand, pressing all buttons, initial state of text when ATE is started using the scheme may be checked.

Once buttons are generated, the corresponding area to each button is highlighted and Display Types may be defined. Since the same display type may be used for different areas, the display type is really the type, which unifies the appearance of the different areas, when they are displayed. Pressing a set of the buttons that represent the areas of the same display type, a type generation button is used to define a new type, which may be as follows:

- H … hiden area
- HR:*string* … hiden area represented by a *string* of characters
- H\ … Hiden area terminated by newline, i.e. such that extends to the whole line
- HR:*string*\ … Hiden area represented by a *string* of characters and terminated by newline
- V …visible area
- V\ … visible area terminated by newline
- M … modifiable area
- M\ … modifiable area terminated by newline
- M+ … modifiable area of at least one character
- M+\ … modifiable area of at least one character terminated by newline

For each hidden text area, a string of characters can be defined, which is displayed instead of LaTeX text, or if a LaTeX text area is empty. Both hiden and visible areas are stable, since they are not modifiable. On the other hand, modifiable areas are such that may be affected while editing, hence, it must be distinguished transitive and positive closures for area characters typed. To exclude deleting all characters while editing, the type M+ (or M+\) is used. In this case the change of box cursor to vertical bar cursor is suppressed, indicating that the area cannot be empty. A terminating newline character \ cannot be deleted, of course. In addition to the display types above, it is possible to extend the hidden areas types including the representation

by pictures and aligned horizontal and vertical lines, appropriate especially for displaying the array and tabular-like environments.

Considering just Display Types and Displayed Areas in a scheme, this scheme can be read by ATE from an external file and expanded to a user communication interface. However, to provide a communication interface, which guarantees positive restrictions to a user, enabling him to type his text safely, including LaTeX commands using control buttons, both Language Definition and Control Buttons parts must be defined. Their definition is the most complicated part of the scheme and it is introduced in the next section.

5. LANGUAGE AND CONTROL BUTTONS DEFINITION

Suppose we want to provide a communication interface to a user, which allows him to fill in the white areas according to Fig. 6.

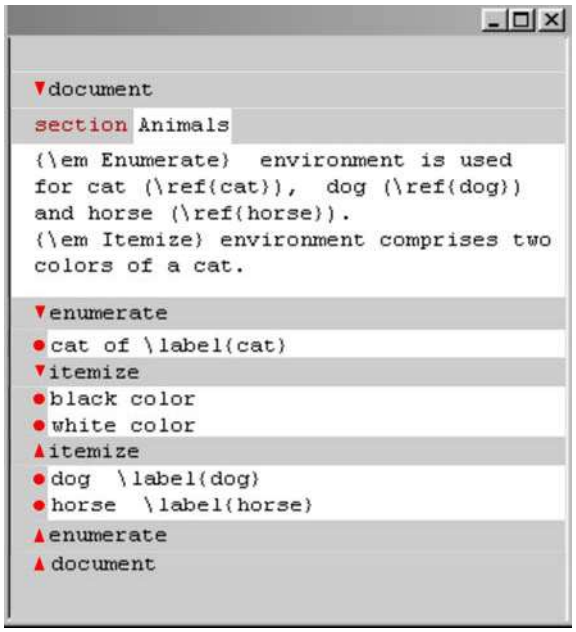


Fig. 6 SAMPLE.TEX - towards no LaTeX knowledge

Omitting the fact that SAMPLE.TEX above is still filled in using unsafe mode, the number of LaTeX commands is reduced. Defining display, we require all white areas be empty except the title Animals. The corresponding display parts of the scheme are as follows.

```
% Display Types
A0,A4 -> H\
A1 -> HR:{\red ▼}{\black document}\
A2 -> HR:{\red section}
A3 -> M+
A5,A8,A11,A13,A16,A18 -> M\
A6 -> HR:{\red ▼}
        {\black enumerate}\
A7,A10,A12,A15,A17 -> HR:{\red ● }
A9 -> HR:{\red ▼}
        {\black itemize}\
A14-> HR:{\red ▲}{\black itemize}\
A19-> HR:{\red ▲}
        {\black enumerate}\
A20-> HR:{\red ▲}{\black document}\

% Displayed Areas
A0 -> /\documentclass[11pt]
      .....................
      \oddsidemargin=-22mm/
A1 -> /\begin{document}/
A2 -> /\section{/
A3 -> /Animals/
A4 -> /}/
A5 -> //
A6 -> /\begin{enumerate}/
A7 -> /\item /
A8 -> //
A9 -> /\begin{itemize}/
A10 -> /\item /
A11 -> //
A12 -> /\item /
A13 -> //
A14 -> /\end{itemize}/
A15 -> /\item /
A16 -> //
A17 -> /\item /
A18 -> //
A19 -> /\end{enumerate}/
A20 -> /\end{document}/
```

LaTeX is a context-free language. Considering that it is open language system, when defining the restrictions for typing, we use rather something as the syntax mining from the script, which was typed and which correctness was proved by LaTeX translation, than whole definition of language syntax.

First, the selection of areas above requires at least an essential knowledge of LaTeX structure. Second, the scheme must be defined with respect of positive restrictions given to a user, not restricting him too much. Third, the scheme designer must be familiar with the essential principles of generalization, which are useful when defining a language from a sample of terminals sequence, producing a grammar for this language `L`. We provide opportunity for LL(1) language, as a subset of LaTeX. Hence the generalization principles are coming from the definition of extended Backus-Naur form EBNF, which (defined in EBNF itself) is as follows:

```
EBNF -> P{P}
P -> N->SE
SE -> SQ{|SQ}
SQ -> SY{SY}
SY -> N|T|(SE)| [SE]|{SE}
```

Informally, `EBNF` is a nonempty sequence of production rules `P`, each in the form comprising

nonterminal N on the left-hand side and syntactic expression SE on the right hand side. A syntactic expression is in the form of non-empty sum of sequences SQ, and syntactic sequence is in the form of non-empty sum of syntactic symbols SY. Each syntactic symbol is a nonterminal symbol N, or terminal symbol T, or a syntactic expression in parentheses (SE), or possible occurrence of a syntactic expression [SE], or a repetitive occurrence of a syntactic expression {SE} including empty symbol ε.

It holds

[SE]= ε|SE, and {SE}= ε|SE|SE SE|..

We use the next equivalent forms, more appropriate for language L definition:

```
SE~= [SE]
SE*= {SE}
SE+= SE{SE}
```

designating explicitly positive closure by SE+.

The aim is to define a language L, such that

$$L^0 \subseteq L \subset LaTeX$$

where L^0 is a language able to generate just a sequence of displayed areas. Hence L may be a superset of L^0 providing more flexibility to a user than just filling in the initial form defined by displayed areas part of the scheme.

Since different areas may be represented by the same terminal symbol, the style in which they are associated with a new terminal is similar as for display types; a subset of buttons associated with areas are pressed, followed by pressing new terminal generation button, which generates a new button designated by new terminal symbol name. For example each areas corresponding to subsequent occurrence of \item command in an environment are mapped into the same terminal symbol. On the other hand, if two occurrences of \item command in different environments are to be displayed differently, then they must be mapped to two different terminals. In general, there is no relation between the number of display types and the number of terminal symbols. To be more concrete, let us introduce Language Definition part, as follows.

```
% Language Definition
A0 -> T0
A1 -> T1
A2 -> T2
A3 -> T3
A4 -> T4
A5,A8,A11,A13,A16,A18 -> T5
A6 -> T6
A7,A10,A12,A15,A17 -> T7
A9 -> T8
A14 -> T9
A19 -> T10
A20 -> T11
S0  -> [] T0 T1 S1+[S1+,S1-]T11
S1+ -> [] T2 T3 T4 [S2*,S2-]S2*
S2* -> [S3,S4,S5]
S3  -> [] T5
S4  -> [] T6 S6+[S6+,S6-]T10
S5  -> [] T8 S6+[S6+,S6-] T9
S6+ -> [] T7 [S2*,S2-]S2*
```

The mappings of areas to terminal symbols of a subset of LaTeX are in the forms, such as follows

```
A7,A10,A12,A15,A17 -> T7
```

The production rules follow them and they comprise the information about the control buttons visibility (such as [S6+,S6-]) in a context of the parse tree which represents a derivation from starting symbol S0, which, at the same time, represents a control button S0.

It may be proved, that the grammar above is equivalent to that as follows

```
S0 -> T0 T1 (T2 T3 T4 (T5 | S1 |
S2)*)+ T11
S1 -> T6 (T7 (T5 | S1 | S2)*)+ T10
S2 -> T8 (T7 (T5 | S1 | S2)*)+ T9
```

where the terminals **T3** and **T5** are emphasized to designate that they express modifiable areas - lexical units of the language. It is easy to see, that reading the Displayed Areas subsequently (introduced below in the top top line), the leftmost derivation will result to the sequence of terminals (in bottom line), as follows:

```
A0  A1  A2  A3  A4  A5  A6  A7  A8
T0  T1  T2  T3  T4  T5  T6  T7  T5

A9  A10 A11 A12 A13 A14 A15 A16 A17
T8  T7  T5  T7  T5  T9  T7  T5  T7

A18 A19 A20
T5  T10 T11
```

When a scheme is expanded, parse tree representing the derivation is constructed. While editing using ATE, it is not just required the parse tree construction, but also its manipulation. Hence, the production rules must be defined in the way, which guarantees each nonterminal on left hand side of a rule be mapped to a button, which, when pressed, causes an incremental generation of corresponding syntactic expression on right hand side. Except that when working in closures, additional buttons, such as for copy, delete, and other actions can be added. We however restrict here just to a button S-, which means exit from closures.

Let us explain the meaning of the first rule of our Language Definition, which is as follows

```
S0  -> [] T0 T1 S1+[S1+,S1-]T11
```

Pressing a button S0, no button is visible ([]) and areas from Displayed Areas part corresponding to terminals T0 and T1 are produced. Then the string of terminals is produced from nonterminal S1+. After that, buttons [S1+,S1-] will appear. Pressing S1+, the production from S1+ is repeated. Pressing S1-, terminal T11 is produced. It may be seen, that button S1+ serves to include multiple sections of the defined structure in a document. In this matter, a user may press just the visible buttons that, surely, must be mapped to some pictures from a given set. The goal of Control Buttons part is to provide such mapping to a user. This is simple, for example, as follows.

```
% Control Buttons
S0  -> pic1
S1+ -> pic2
S2* -> pic3
S3  -> pic4
S4  -> pic5
S5  -> pic5
S6+ -> pic6
```

Finally, we illustrate the style in which the document may be written sequentially pressing the sequence of buttons S0 S1+ S2* S3 S2* S4 S6+ .. S1-. Bold-faced terminals mean switching to the edit (lexical) mode, which, when exited, cannot be never repeated in the single session in this case, since we have not defined a buttons for tracing the parse tree. The derivation controlled by pressing the buttons is as follows

```
S0 T0 T1 S1+ <1> S1- T11

<1>=T2 T3 T4 S2* S3 T5
              S2* S4 <2> S2-

<2>=T6 S6+ T7 S2* S3 T5
               S2* S5 <3> S2-
       S6+ T7 S2* S3 T5  S2-
       S6+ T7 S2* S3 T5  S2-
       S6-
    T10

<3>=T8 S6+ T7 S2* S3 T5 S2-
       S6+ T7 S2* S3 T5 S2-
       S6-
    T9
```

We have used the substitutions <1>, <2>, and <3> to make the derivation more readable. As a result, the text of hidden areas comprising LaTeX commands is mixed with user text, producing correct LaTeX document, provided that user works in safe mode, of course. The language defined is a superset of a minimal language required for scheme areas expansion, being stil a subset o LaTeX. A user in safe mode is allowed to repeat sections, items of environments and to include itemize environment in enumerate environment and vice versa. However, emphasizing and referencing is suppressed and may be used just in unsafe mode sofar.

6. CONCLUSION

As shown in this paper, typesetting is a two-dimensional architecture, considering the final form of document. At the same time, the documents are highly structured and this fact is exploited when a user communication interface is defined by a scheme. A scheme comprises the information about the initial state of a source document, and the information about its possible restrictions and/or extensions, given by a LL(1) language – a subset of LaTeX. In this way, it is possible to define less or more restrictive schemes, according to user application areas. The schemes are stable, since they cannot be destructed when editing a document.

Sofar we have implemented just display definition part, not language definition parts of ATE. Since ATE will be implemented in framework of a master thesis, it was time to formulate our aims more precisely, and still not too formally. Theoretically, the task is simple: it is necessary to maintain LaTeX text, related to a parse tree [2,4]. The implementation is exploits well known methods [1], but it is more complicated than the theoretical background, since we require ATE be an open system, which is able to define each potential user communication interface and to use it while editing.

Special attention must be paid to array and tabular environments, since the numbers of columns is defined in constructs \begin{tabular} and \begin{array}. That is why columns in the tabular and/or array body, separated by & character, cannot be constructed using positive closures, but they must be restricted to the number of iterations. Although ATE is not a WYSIWYG, vertical lines separating columns must be aligned. Also emphasizing may be performed in a WYSIWYG fashion. It is also necessary to think about the method of automatic generation of labels, if appropriate. The composition of many documents must prevent the clash of labels and bibitems.

Syntax driven text processing is simple, especially for LL(1) language. Moreover, the language syntax is derived manually, using LaTeX constructs that have semantics determined. It may be noticed however, that the derived language may be a macro language over LaTeX, being still a subset of LaTeX. It is so, since hidden areas may comprise multiple language elements of LaTeX.

An interesting problem is to mine the syntax from samples of database or even natural language records, associating the semantics to production rules (of course not restricted to context-free grammar), and to use this semantic information when building the language incrementally [7]. In this way a new user requirement would be simply generalized and the maintenance would not require unreliable affecting the implementation. This, however, is the future.

REFERENCES

[1] Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms. Addison-Wesley, 1985

[2] Aho, A.V., Sethi, R., Ullman, J.D.: Compilers - Principles, Techniques, and Tools. Addison Wesley, 1988

[3] Knuth, D.E.: The TeXbook. Addison-Wesley, Reading, MA, 1986. ISBN 0-201-13447-0. 483 pp.

[4] Kollár, J., Havlice, Z.: Language Systems Technology. Elfa, s.r.o., Košice, 2001. ISBN 80-89066-12-7, 186pp. (in Slovak: Technológia jazykových systémov)

[5] Lamport, L.: LaTeX – A Document Preparation System. Addison-Wesley, Reading, MA, 1985. ISBN 0-201-15790-X. 242 pp.

[6] LaTeX3 Project Team: LaTeX2e for authors. 33pp. TeXLive 5.0d distribution.

[7] Novitzká, V.: Semantics of programs. Elfa, s.r.o., Košice, 2001. ISBN 80-88964-59-8, 145 pp.

[8] Olšák, P.: Typesetting System TeX. Brno-Konvoj, 2000. ISBN 80-85615-91-6. 300. pp. (in Czech: Typografický system TeX)

BIOGRAPHY

Ján Kollár (Assoc. Prof.) was born in 1954. He received his MSc. summa cum laude in 1978 and his PhD. in Computing Science in 1991. In 1978-1981 he was with the Institute of Electrical Machines in Košice. In 1982-1991 he was with the Institute of Computer Science at the University of P.J. Šafárik in Košice. Since 1992 he is with the Department of Computers and Informatics at the Technical University of Košice. In 1985 he spent 3 months in the Joint Institute of Nuclear Research in Dubna, Soviet Union. In 1990 he spent 2 month at the Department of Computer Science at Reading University, Great Britain. He was involved in the research projects dealing with the real-time systems, the design of (micro) programming languages, image processing and remote sensing, the dataflow systems, the educational systems, and the implementation of functional programming languages. Currently the subject of his research is process functional paradigm and its application to aspect oriented programming.