Model-Based Conversions of LaTeX Documents

Dennis S. Arnon
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304 USA
415-812-4425; FAX: 415-812-4241
Internet: arnon@parc.xerox.com

Isabelle Attali
INRIA Sophia Antipolis
Route des Lucioles
06565 Valbonne Cedex, France
Internet: Isabelle.Attali@sophia.inria.fr

Paul Franchi-Zannettacci
University of Nice
CERISI
Sophia Antipolis
06561 Valbonne Cedex, France
Internet: pfz@essi.cerisi.fr

Abstract

We are creating a document conversion system based on modelling the logical structures of two broad categories of document types: *human-oriented* and *machine-oriented*. Each human-oriented, or *user*, type is a genre of documents that is well defined and well known to human authors and readers; for example, "scientific articles", "mathematical formulae" (e.g., $\frac{x^3+x^2-1}{x^2+1}$), or "limericks". Each machine-oriented, or *agent* type, is the set of legal data objects of a document processing tool, for example, "LaTeX documents" or "troff/EQN documents". Our models specify the abstract, rather than the concrete (i.e., surface) syntax, of particular documents. Thus we work with documents as tree data structures, whose concrete presentations have a surface syntax of no intrinsic interest. By so proceeding we hope to best utilize the computational sciences, and sidestep what we believe to be the red herring of "markup language".

Introduction

In contemporary document management systems, documents are often characterized chiefly by the format in which they happen to be represented at some instant of time. A *format* is simply the document encoding convention of a particular document processing agent; we thus prefer to refer to formats as *agent* document types. Formats are often classified into a rough hierarchy of "levels": image formats (TIFF, SunRaster, Fax, etc.) are said to be the lowest level, next are page description level formats (PostScript, InterPress, etc.), and finally are high level, or structured, formats (TeX , SGML DTD (Goldfarb, 1990), ODA DAP (Rosenberg, 1991), etc.).

For some classes of people, such as the implementor of such a document management system, the format-centric approach to document taxonomy is reasonable. These are people who in all likelihood have no contact with either the creators or the consumers of the documents, and their role is to be a "middleperson", or broker, between a creator whose output is in one format, and a consumer who wants the document presented in a different format. The document broker probably neither knows nor cares what the document is about, how it is organized, or what its actual content is; instead, accepting a charter to render instances of one format in the other, as "faithfully" as possible.

Document brokering is not an appropriate view of another large class of document transactions, however. A collection of individuals who use different document processing tools may be co-authoring a single document; or, an author and a sophisticated reader; e.g., a colleague, may wish to share a document. In these instances, the "sender" and the "recipient" may have a good deal of common knowledge about a document's organization and content. Indeed, for optimal communication among themselves, the individuals in such groups want their respective document processing tools to fade into the background. They want instead to focus clearly on the "abstract" document that is the object of their interaction; i.e., an abstract intellectual entity that is independent of the particular software tools used to process concretizations of it. Furthermore, they undoubtedly have a common model of this abstract document (which we would say belongs to a *user* document type) in their respective mind's eyes. To convert the document from one format to another in these situations, it may be advantageous to know what the model is, and use it to "direct" or "govern" the format conversions. In these model-centric contexts, there are likely to be aspects of one format that do not map well to another format, in general. However, there will most likely be *some* way of encoding the relevant features of the abstract document in the destination format. By having the underlying abstract model in hand, and only in this way, we can maintain "knowledge" of which model features are encoded which way in which format, and thereby accomplish higher-quality format conversions for a community of model-centric document users.

In this paper we focus on a single user document type, that of "technical articles" (which may include mathematical formulae). We give a general model-building methodology, then specialize it to create models for the Article user type, and for agent types. Currently our system deals with two agent types, LaTeX documents and Tioga documents. (Tioga is a WYSIWYG editor for structured documents in the Cedar programming environment at Xerox PARC; see Swinehart, et al., 1986). We have created a software tool that supports conversions in which we are given a LaTeX or Tioga incarnation of a document asserted to be a technical article, and we want to convert it to the other agent type. We utilize distinct models for each user and agent type, and precisely specify the rules of interconversion for each $(agentType, userType)$ pair. An $agentType \rightarrow agentType$ conversion is then carried out by $agentType \rightarrow userType \rightarrow agentType$ mappings. Thus our tool converts a document from one format to another, while explicitly seeking to preserve its "technical-article-ness" as much as possible. This is what we mean by a *model-based conversion* of a document: a document believed to be a valid instance of some user type, and presented as an instance of some agent type, is converted to an instance of another agent type, in a manner as faithful to its user type as possible. The choice of particular agent types is secondary; the crux of our methodology is the "direction" of document format conversions by user type models.

The following section presents our general model-building methodology, based on the notion of a *formalism*, and our model for the Article user type. Our model-based approach to format conversion is divided into two steps: *analysis*; i.e., creating a valid user type model for the input document, and *synthesis*; i.e., rendering the model instance in the desired output format. We also call these steps *parsing* and *unparsing*. In subsequent sections, we discuss the parsing and unparsing of Tioga documents, and the parsing and unparsing of LaTeX documents. We concentrate on parsing Tioga documents to Articles and unparsing Articles to LaTeX , rather than the opposite direction. This reflects the fact that at the moment the least developed link in our system is our LaTeX to Article parser, a situation we are in the process of rectifying.

Currently our system is implemented in the Cedar programming environment (Swinehart et al., 1986), which in turn runs on top of UNIX and X Windows. Although we do not currently make actual use of the Centaur system (Borras et al., 1988), we are heavily influenced by the architecture and concepts of Centaur, and of attribute grammars (e.g., see P. Franchi-Zannettacci & D. Arnon, 1989, and A. Brown, H. Blair, 1990).

Modelling Articles with Formalisms

Terms. The first key component of our system is a single, universal data structure and surface syntax for labelled, n-ary, attributed trees. This we accomplish with a general tree manipulation package we have written, called Scrimshaw. "First order terms", "abstract syntax trees", "functional forms", and "expression trees" are additional equivalent names for the basic entities of Scrimshaw; we refer to all of them simply as *terms*. Atomic terms are identifiers, 32-bit integers, "reals" (i.e., IEEE standard floating point numbers), and character strings. Composite terms are "expression trees",

whose "operator names" are identifiers, and whose "arguments" are (recursively) terms.

We implement document processing operations (e.g., parsing, unparsing, conversion) as *term rewriting* or *tree transformation* operations on Scrimshaw terms. The particular "function", from terms to terms, that any such operation represents, is not a part of Scrimshaw itself. Rather, each document type gives certain "interpretations", i.e., meanings, to certain terms of interest, and its operations map them to certain other terms which also have intended interpretations. All document types can equally make use of the general term data structures and operations that Scrimshaw provides, for what it considers to be uninterpreted terms.

Here is a portion of a Scrimshaw term representation of the Tioga form of this paper, which we hope will suffice to illustrate the nature of Scrimshaw Terms, in lieu of a full definition:

```
internalNode[
 nodeList[
  leafNode[
    format[title],
     contents[
       runList[
        text["Model-Based Conversions
              of LaTeX Documents"]
             ]
            ]
           ],
  leafNode[
    format[authors],
     contents[
      runList[
       text["Dennis S. Arnon ..."]
            ]
           ]
        ],
  leafNode[
    format[abstract],
     contents[
      runList[
       text[
        "Abstract: We are creating a
         document conversion system
         ... formulae\'\' (e.g., "],
       text["X",
        prop[
         $MathNotation,
         "quotient[diff[sum[power[name[x],
          number[3]], power[name[x],
          number[2]]], number[1]],
          sum[power[name[x], number[2]],
          number[1]]]"
           ] ],
       text[
         "), or ``limericks\'\'.
          Each machine-oriented ...  ."]
           ]
         ]
       ]
     ]
  ]
```

Formalisms: abstract vs. concrete syntax. A document *type* is a certain family of labelled, n-ary trees, or in other words, some subfamily of Scrimshaw terms. There is some finite set of label names (each of which is a Scrimshaw identifier) for nodes; a node's label is called its "operator". The set of label names is non-disjointly partitioned into subsets called *phyla*; thus, each operator belongs to one or more phyla. In addition, each leaf node has a *value* that is either an identifier, an integer, a character, a string, a term of some other formalism, or empty. For each operator, we specify the number (*arity*) and phyla (*types*) of its descendants. An operator is either *atomic*; i.e., it is a leaf node with zero descendants and a value, or a *fixed arity* operator, with some fixed number $n \geq 1$ of descendants, whose root operators belong respectively to phyla $P_1, ..., P_n$, or a *list* operator, capable of having zero or more descendants whose root operators all belong to a single phylum P. Atomic operators in one formalism, whose values are terms of another formalism, are our means of providing for the nesting of document types one within another, e.g., mathematical formulae within technical articles. The terms of a document type may be attributed, and the attribute (i.e., property) values may be crucial to the specification of the type model; in the limited space of this paper, however, we have little to say about attributes.

A family of terms specified as above is called a *formalism*, and its constituent terms are called the *abstract syntax trees* of the formalism. We implement each abstract type of documents (e.g., Article, math formula) as a formalism. Thus the "canonical" abstract representation of a document in our methodology is as an abstract syntax tree of some formalism.

Let us now illustrate these concepts with excerpts from the Article formalism we use to model technical articles. Owing to the limited space in this paper these excerpts will stand in lieu of a full definition. Here are the specifications of the `article` and `header` operators in the Article formalism:

`article` → `HEADER BODY END ;`

```
header → TITLE AUTHORS ABSTRACT KEYWORDS;
```

is a 4-ary operator; the first descendant of a header must be of phylum `TITLE`, its second of phylum `AUTHOR`, etc. These phyla happen to contain only one operator each:

```
TITLE ::= title ;
AUTHORS ::= authors ;
ABSTRACT ::= abstract ;
KEYWORDS ::= keywords ;
```

so we are effectively requiring a single, 4-part structure in the header of an article. (We write operators in lower case and phyla in upper case throughout this section).

Our model of "text" in Articles is that it is a "list of text items". Thus we have a list operator "paragraph" defined by:

```
paragraph → TEXTITEM * ...  ;
```

which says that a `paragraph` node has zero or more descendants, each belonging to the phylum `TEXTITEM`. That phylum is defined by the lines:

```
TEXTITEM ::= word specialChar formula ...;
word → value is string ;
specialChar → value is string ;
formula → value is MathNotation.ANY ;
```

where `MathNotation` is a separate formalism for mathematical formulae. These lines define our "abstract", "logical", model of text. They say what kinds of things we believe "text" to be comprised of. The `word`, `specialChar`, and `formula` operators are atomic, with their value types specified by the object of the phrase `value is`.

We attach text to structural components of a document with lines such as the following:

```
subsubsection → TITLE PARAGRAPHLIST ;
paragraphList → PARAGRAPH + ...  ;
PARAGRAPHLIST ::= paragraphList ;
PARAGRAPH ::= paragraph ;
title ::= PARAGRAPH ;
```

These say that subsubsection is a binary operator, whose first child is a `TITLE`, and whose second child is a `PARAGRAPHLIST`, i.e., a list of paragraphs. A `TITLE` node must have a `title` operator, which is unary: its child is the `PARAGRAPH` that comprises the text of that title.

Let us briefly consider the `MathNotation` formalism itself (c.f., D. Arnon, S. Mamrak 1991, and D. Arnon, et al., 1988). There, e.g., we define the quotient notation with the lines:

```
quotient → FORMULA FORMULA;
FORMULA ::= quotient sum power ...  name ;
```

Thus quotient is a binary operator.

Thus we use the same definitional mechanism for mathematical formulae as for Articles. As a complete example of the Article formalism, here is an excerpt of a representation of the present paper in it. A more detailed version of this excerpt can be found in the Appendix. Note that we see here how the value of the `formula` operator, which is atomic in the Article formalism, is a term of the `MathNotation` formalism.

```
article[
 header[
  title[
   paragraph[
           word[
"Model-Based Conversions of LaTeX
 Documents "     ]
             ]
        ],
  authors[
     displayItemList[
     paragraph[
       word[ "Dennis S. Arnon
               ... Palo Alto,
                    CA 94304 USA"
                  ]
             ],
                   ]
        ],
  abstract[
    paragraphList[
    paragraph[
        word["Abstract: We are creating"]
         ...
        word[``formulae'' (e.g., " ],
        formula[ quotient[
          diff[ sum[ power[ name[x ],
          number[ 3 ] ], power[ name[ x ],
          number[ 2 ] ] ], number[ 1 ] ],
        ... ] ] ]
]  ]  ]
```

Validation. Validation is the task of deciding whether a given term belongs to a given formalism. Obviously this is a fundamental consistency check that we make frequently when using our system. For example, we take it as a basic criterion for parsing and unparsing steps that the output document, however perturbed it may seem from the input, should always be valid for the target formalism to which it is supposed to belong.

We may define validation of a term for a formalism by means of a slightly more general notion: the validation of a term t for a phylum P. This we

define recursively as follows: let `rootOp` be the operator of the root node of t. If `rootOp` is atomic, then t is valid for P if `rootOp` belongs to P, and if the value of `rootOp` is valid (this involves checking the syntactic correctness of an integer, real, identifier, or string, or recursively checking the validity of its term value for the expected formalism). If `rootOp` is n-Ary, then t is valid for P if `rootOp` belongs to P, if the actual number m of children of `rootOp` is equal to n, and if for $i = 1, ..., n$, $childTerm_i$ is valid for $childPhylum_i$. If `rootOp` is a list operator the definition is similar. Finally, we say that t is valid for the formalism if t is valid for `ROOTPHYLUM`, where `ROOTPHYLUM` consists of the allowable operators of root nodes of terms in this formalism (`ROOTPHYLUM` is analogous to the "start symbol(s)" of a grammar). Often a formalism permits any operator to be a root operator, i.e., its `ROOTPHYLUM` is `ANY`.

Converting Tioga Articles to LaTeX

In this section we first describe the main features of the Tioga editor's document model, and then briefly examine the actual formalism we currently use to model Tioga documents. We then describe the tree pattern matching functions we use to convert Tioga documents to Articles, and finally we outline the actual Tioga-To-Article converter we have written using the tree pattern matching functions. For general background on Tioga, and the Cedar programming environment of which it is a part, the reader may consult Swinehart et al., (1986).

The Tioga document model. Tioga is a true structured document editor in the sense that its internal data structure for any document is a tree of nodes, each of which has character string content. Both entire nodes, and individual characters within a node, can have properties, i.e., attributes. Each node is labelled by an identifier that Tioga calls a *format*. Typical formats are `title`, `abstract`, `head`, `block`, `reference`, etc. Tioga documents are formatted by associating a collection of *style rules* with them. In particular, there should be a style rule for each format that specifies, in a PostScript-like language, how to graphically render nodes of that format. Multiple fonts are provided via both special character properties called *looks*; e.g., `bold`, `italic`, `greek`, plus a mechanism for using the full multinational range of the Xerox character code standard. Mathematics and imbedded illustrations are accommodated via character and node properties.

There is no standard surface syntax for Tioga documents. Virtually all authors of Tioga documents use a small number of standard styles.

A formalism for Tioga documents. As an intermediate step in Tioga to Article conversion, we have defined a formalism which directly expresses the Tioga document model. We call this the *MediumTioga* formalism; we saw an example of it in the subsection on Terms above. We have written Tioga-to-MediumTioga, and MediumTioga-to-Tioga converters. Thus we reduce the Tioga-Article conversion problem to the MediumTioga-Article conversion problem, which we can approach from completely within the Scrimshaw world. Thus, for example, we implement Tioga-to-Article conversion by appropriate tree transformations of Scrimshaw MediumTioga terms to Scrimshaw Article terms.

Tree pattern matching functions. At present our term rewriting capabilities are built on simple tree pattern matching, in which patterns are just literal terms, possibly containing (any number of instances of) pattern variables of two kinds. First, a variable which matches any term (which we call ANYTERM). Second, a variable which matches any list of one or more terms (which we call ANYTERMLIST). For example, here is a list of the patterns we use to search for `title` nodes in MediumTioga documents:

```
leafNode[format[title], ANYTERM]
internalNode[format[title], ANYTERMLIST]
```

Having matched patterns such as the ones above, e.g. having found all the `title` nodes in a MediumTioga document, we then perform a "rewriting" action, e.g. to construct the (unique) title node we must have in the output Article. For this example, the action is to concatenate the text content of all `title` nodes, and their descendant nodes, to make the text content of the Article's title.

Tioga-to-article converter. Our goals are to not fail on any legal Tioga document as input, and to always produce a valid Article as output. Hence, using the pattern matching functions, we traverse the MediumTioga form of an input document and apply a succession of rules to build a valid Article.

Unparsing Articles to LaTeX. An Article term can be unparsed to a LaTeX source file through a straightforward recursive descent tree traversal. A sample of a LaTeX unparsing of an Article representation of this paper is shown in the appendix.

Converting LaTeX Articles to Tioga

The Article document model has been designed to be highly compatible with typical LaTeX representations of documents that are in fact technical articles. Thus at the moment it works well for us to

simply define the LaTeX agent model as being identical to the Article user model. Hence at present we "convert" a LaTeX document to an Article by simply "parsing" its LaTeX source file and "recognizing" the Article that we consider to be encoded there. As before, our goals are to not fail on any valid LaTeX input, and to always produce a valid Article as output. Once we have an Article, it is straightforward to unparse it to a MediumTioga document. At present, we ignore unknown control sequences, including macros. We are in the process of developing a separate LaTeX agent model, and upgrading our LaTeX parser to use it.

Bibliography

Arnon, D., R. Beach, K. McIsaac, and C. Waldspurger. "Caminoreal: An Interactive Mathematical Notebook." Pages 1 – 18 in *Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography*, J.C. van Vliet, ed. Cambridge: Cambridge University Press, 1988.

Arnon, D., and S. Mamrak. "On the Logical Structure of Mathematical Notation." *TUGboat* 12(2), pages 479 – 484, 1991.

Borras, P., D. Clément, T. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. "Centaur: The system." *Proceedings of the* SIGSOFT'88, *Third Annual Symposium on Software Development Environments*. Association for Computing Machinery, Boston, Massachusetts, 1988.

Brown, A., and H. Blair. "A Logic Grammar Foundation for Document Representation and Document Layout." Pages 47 – 64 in *Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography*, R. Furuta, ed. Cambridge: Cambridge University Press, 1990.

Franchi-Zannettacci, P., and D. Arnon "Context-Sensitive Semantics as a Basis for Processing Structured Documents." Pages 135 – 146 in *Proceedings of WOODMAN'89*. Workshop on Object-Oriented Document Manipulation. J. André and Jean Bézivin, editors, (BIGRE 63-64), IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France, Mai 1989, ISSN 0221-5225.

Goldfarb, C. *The SGML Handbook*. Oxford: Clarendon Press, 1990.

Rosenberg, J., M. Sherman, A. Marks, and J. Akkerhuis. *Multi-Media Document Translation: ODA and the EXPRES Project*. New York: Springer-Verlag, 1991.

Swinehart, D., P. Zellweger, R. Beach, and R. Hagmann. "A Structural View of the Cedar Programming Environment." *ACM Transactions on Programming Languages and Systems* 8(4) 419 – 490, 1986.

Appendix

Examples of Conversions

Here is the beginning of the actual Article representation of the Tioga form of this paper.

```
article[
        header[
                title[
                        paragraph[
                                word[
                                        "Model-Based Conversions of LaTeX
Documents "
                                        ]
                                ]
                        ],
                authors[
                        displayItemList[
                                paragraph[
                                        word[
                                                "Dennis S. Arnon"
                                                ],
                                        specialChar[
                                                "(000|012)"
                                                ],
                                        word[
                                                "Xerox PARC"
                                                ],
                                        ....
                                        ]
                                ]
                        ],
                abstract[
                        paragraphList[
                                paragraph[
                                        word[
                                                "Abstract: We are creating a
document conversion .., for example,
                                                ‘‘scientific articles\’\’,
‘‘mathematical formulae\’\’ (e.g., "
                                                ],
                                        formula[
                                                quotient[
                                                        diff[
                                                                sum[
                                                                        power[

name[

x

],

number[
```

```
3

]

],
                                                        power[

name[

x

],

number[

2

]

]
                                                                ],
                                                        number[
                                                                1
                                                                ]

                                                                ,
                                                                ],
                                                sum[
                                                        power[
                                                                name[

x

],
                                                                number[

2

]
                                                                ],
                                                        number[
                                                                1
                                                                ]
                                                        ]
                                                ]
                                        ],
                                word[
                                                "), or ``limericks\'\'.  Each
machine-oriented, ... sidestep

                                                what we believe to be the red
herring of ``markup language\'\'."
                                                ]
                                ]
                        ]
```

```
                        ],
                keywords[
                        paragraph[
                                word[
                                        "Keywords: Structured Documents,
Electronic Documents, Document Conversion "
                                        ]
                                ]
                        ]
                ],
        body[
                sectionList[
                        sectionIntroOnly[
                                title[
                                        paragraph[
                                                word[
                                                        "Introduction"
                                                        ]
                                                ]
                                        ],
                                paragraphList[
                                        paragraph[
                                                word[
                                                        "In contemporary
document management systems, ...
```

Example of Unparsing

Here is the beginning of the LaTeX unparsing of the document that our system actually produces:

```
\documentstyle[12pt]{Article}
\title{Model-Based Conversions of LaTeX Documents }
\author{
Dennis S. Arnon
%Article% specialChar["(000|012)"]
\\
Xerox PARC
%Article% specialChar["(000|012)"]
\\
3333 Coyote Hill Road
%Article% specialChar["(000|012)"]
\\
Palo Alto, CA 94304 USA
\and
Isabelle Attali
%Article% specialChar["(000|012)"]
\\
INRIA Sophia Antipolis
%Article% specialChar["(000|012)"]
\\
Route des Lucioles
%Article% specialChar["(000|012)"]
\\
06565 Valbonne Cedex, France
```

```
\and
Paul Franchi-Zannettacci
%Article% specialChar["(000|012)"]
\\
CERISI
%Article% specialChar["(000|012)"]
\\
Sophia Antipolis
%Article% specialChar["(000|012)"]
\\
06561 Valbonne Cedex, France }
\begin{document}
\maketitle
\begin{abstract}

Abstract: We are creating a document conversion system based on
...
 for example, ``scientific articles'', ``mathematical formulae'' (e.g.,
${\frac{{{{{{{{x}}^{{3}}}}
}+{{{{x}}^{{2}}}}
}}
}-{{1}}}
} {{{{{{x}}^{{2}}}}
}+{{1}}}
}}
$
), or ``limericks''.  Each machine-oriented, or {\em agent} type, is the set
...
 sidestep what we believe to be the red herring of ``markup language''.
\end{abstract}
\begin{keywords}
Keywords: Structured Documents, Electronic Documents, Document Conversion
\end{keywords}
\section{Introduction}



In contemporary document management systems, ...
```