ELECTRONIC PUBLISHING, VOL. 5(3), 143–155 (SEPTEMBER 1992)

Granularity in structured documents

FRANS C. HEEMAN [1]

Department of Mathematics and Computer Science
Vrije Universiteit
De Boelelaan 1081
1081 HV Amsterdam
The Netherlands

SUMMARY

Structured documents have become a widely accepted concept for document manipulation applications like editing, formatting, and archiving. However, some aspects of structured documents are still not well understood. In particular, the transition in structured documents from logical structure to contents, is a grey area in which different systems use different interpretations.

In this article, we discuss this *granularity* aspect of structured documents. We focus on the underlying concepts of structured documents without referring to any application, so that this discussion is kept clear from aspects that are not related to structured documents.

KEY WORDS Structured documents Granularity Generic logical structure Grif ODA SGML

1 INTRODUCTION

In the definition of structured documents, a document is not just a sequence of characters. Rather, a document consists of a *structure* and of *contents*. The structure of a document is defined by dividing a document into parts: this division into parts and the relations between the parts reflect the *logical structure* of the document. At the lowest level of this structure, the actual contents are to be found. For example, the paper you are now reading can be divided into a title, the author's name and affiliation, an abstract and five sections. Each section can be divided into a section title and a number of paragraphs. A paragraph may contain actual contents, such as the text of this paragraph.

Since the logical structure is based on the 'meaning' of the parts of a document, there is no single, unique, logical structure for a given document. Most people agree on the concept of a logical structure, though different people may define a different logical structure for one and the same document. For the most part these different approaches do not give rise to problems. However, as we approach the vague transition from logical structure to actual contents, these differences give rise to different interpretations of the concepts of structure and content.

In Sections 2 and 3, we give a quick overview of the (usual) terminology involved when discussing structure and contents. More extensive introductions can be found in [1,2,3]. In a way, Section 2, on logical structure, presents a top-down approach of structured

[1] Present address: Centre of Mathematics and Computer Science, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands.

CCC 0894–3982/92/030143–13 *Received 20 September 1990*
Revised 5 June 1992

documents, whereas Section 3, on contents, presents a bottom-up approach. These two approaches should meet somewhere in the middle, but this is precisely where the problems arise. These problems are discussed in Section 4, using examples from three existing systems that handle structured documents: Grif [4], ODA [5,6], and SGML [7,8]. Section 5 summarizes the results.

2 LOGICAL STRUCTURE

We first give the terminology related to the logical structure, and summarize this in Figure 1. Readers familiar with the concept of structured documents may want to glance over the figure and skip the rest of this section.

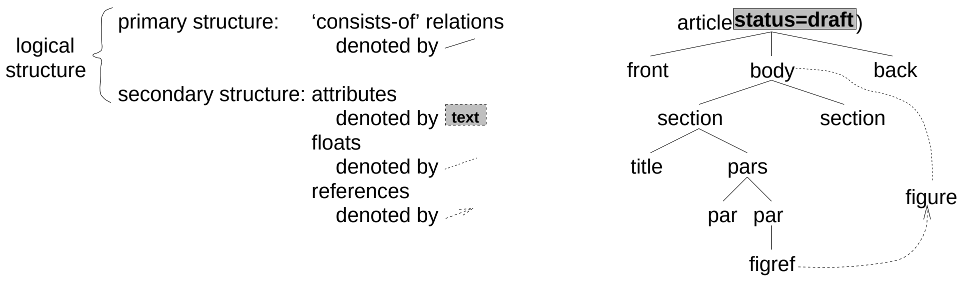

Figure 1. A schematic overview of the terminology on logical structure used in this article. On the left, the terminology on logical structure is listed, together with a legend for the tree-figure on the right. This tree shows an example document. A Document Type Definition, or DTD, is a generic description of a class of documents

As stated in the introduction, a document is defined to consist of a *logical structure* and of *contents*. The structure of a document is defined by dividing a document into parts, subdividing these parts, etc. Each part in this structure defines a part of a document that has some specific meaning. From now on, we define the parts as *objects* that have a *type* [9]. For example, if part of a document is to represent a chapter consisting of an ordered sequence of a chapter title and one or more paragraphs, we have an object of type 'chapter' (or simply an object 'chapter') that may have objects of type 'chaptertitle' and 'paragraph' as descendants. When an object is the root of the structure, its type represents the type of the entire document, i.e. the *document type*. Examples of document types might be 'article', 'memo', or 'manual'.

Up to this point, we have only talked about specific documents and their logical structure. However, the logical structure of similar documents can be described in a generic way. Consider a journal that contains several articles. One article may have three chapters and another article may have five chapters, but in general all articles contain a sequence of one or more chapters. The *generic structure* of these articles can be defined by a title, followed by a sequence of one or more author names, followed by an abstract, followed by a sequence of one or more chapters, etc. It can be said that all articles in a journal fall in the same class of documents, or alternatively, that they are of the same type. Thus, a generic structure describes a type of document. We call this structure the document type definition, or DTD. A DTD contains *definitions* of objects and a document contains *instances* of these objects. A useful analogy for a DTD is the definition of a programming language: in the same way that programs are written in some syntax defined by a grammar, documents are to conform to the definition given by a DTD. In this analogy, a document type (e.g. article, memo) corresponds to a specific programming language (e.g. Pascal, C). This analogy can be extended even further, by comparing software engineering with document engineering [10].

The availability of a generic structure together with documents that conform to this structure offers a new way for devising applications for documents. An example is formatting. It is no longer necessary to include in each document all kinds of formatting directives, since these can be defined once for the generic structure. For instance, if some part of a document is indicated as being the title, the generic structure may contain directives to typeset this part in Times Roman Bold, pointsize 14. Using the generic structure, a *generic layout* can be constructed that defines the layout for all documents conforming to that generic structure. Another application area is databases. To store the title and authors of all articles into a database, an application can be devised that scans the articles for the necessary fields and deposits their contents into that database. Using the generic structure, an application has knowledge of where it can find some specific kind of information in a document.

A model similar to the model of structured documents is that of *nested environments*. An environment is a logical part of a document, like a section or quotation, that can be freely nested within some other environment, i.e. the hierarchy is not restricted by a grammar. This model originates from Scribe [11], and is currently used by the well-known LaTeXsystem [12]. Although LaTeX is often referred to as supporting structured documents, this is not the case, because it does not *enforce* a generic or logical structure.

Within the logical structure, two kinds of structure can be distinguished: the primary and secondary structure. The *primary structure* of a document consists of the relationships used to describe the logical structure of a document, such as ‘consists-of’. The *secondary structure* defines additional supporting relationships, such as ‘refers-to’. The following two subsections describe these structures.

2.1 Primary structure

The most common primary structure is a hierarchy: a document is divided into parts, which are subdivided, etc. The relations between the parts denote that one part *consists of* other parts. An alternative, more general, primary structure is that of a directed acyclic graph (DAG), in which parts can be shared.

The primary structure can be *homogeneous* (all parts use the same relationships) or *heterogeneous* (different parts use different relationships). For example, if a mathematical formula is described using a tree-structure and a table is described using some matrix structure, then documents containing formulae and tables have a heterogeneous primary structure.

In general, the following constructs are offered to define the primary structure of a document:

sequence: an ordered sequence of objects, possibly of different type;
aggregate: an unordered sequence of objects, possibly of different types; the precise order of the objects in a document is not determined by the DTD, any permutation of these objects in a document is correct;

list: an ordered sequence of 0, 1 or more objects of the same type;
choice: one object whose type may be any of a given list of types;
combinations: the above constructs may be combined, for example a choice of two lists.

2.2 Secondary structure

In addition to the primary structure, a secondary structure can be defined. This contains relationships that cannot be expressed within the primary structure. An example is a cross-reference from one part in the primary structure to another, which defines the relation ‘refers-to’ between these parts, in addition to the relation ‘consists-of’ that is defined by the primary structure. The secondary structure is mainly defined using three constructs: attributes, floating objects and references to objects:

attributes: Attributes are used to denote semantic information that is not provided by the primary structure. Examples are the status of a document (draft or final), or the language it is written in (French or English). The use of these attributes is not defined, they merely offer extra facilities that can be used if so desired.

Attributes have a *type* and some common types are ‘numeric’, ‘textual’ and ‘enumeration’. The DTD specifies which attribute(s) are defined for an object and what values are allowed. A specific value is defined in a document. This value may be inherited from another object, or specified relative to the value of some other attribute. Furthermore, systems may define that a value is required, optional, defaultable, assumed to be supplied by the application, etc.

floating objects: A floating object is an object for which the DTD does not determine the exact position in a document. The object is allowed to appear at a more or less arbitrary place in part or all of the document.

As an example, SGML defines a construct called an *inclusion*. The definition of an object in a DTD can have zero or more included objects associated with it. The definition of such an included object appears somewhere else in the DTD and is similar to that of regular objects. Zero or more different instances of an included object may appear anywhere within the subtree that is an instance of the object at which the included object was declared. Thus, in SGML, instances of included objects are contained in the primary structure. As an example, consider the following:

```
<!ELEMENT chapter (title, paragraph*) +(figure) >
```

This defines an object called ‘`chapter`’ as being a ‘`title`’ followed by zero or more (indicated by ‘`*`’) instances of the object ‘`paragraph`’. This ‘`chapter`’ object defines an inclusion (‘`+(figure)`’) which means that anywhere in the subtree starting at object ‘`chapter`’, zero or more instances of the object ‘`figure`’ may appear.

SGML also defines *exclusions*: this are the opposite of inclusions, i.e. certain objects are not allowed to appear in some part of the document. The use of inclusions and exclusions is further restricted because of ambiguities that may arise (see [13]).

references: A final construct is that of a reference to a specific instance of an object. References may be defined among objects within one document, or they may be defined among objects in multiple documents. References can be used to define hypertext documents [9].

In Grif, a reference is a separate construct that refers to an instance of one specific type of object. For example, when a DTD defines an object ‘Figure’, it may define ‘RefFig = REFERENCE(Figure)’ to refer to an object of type ‘Figure’. Using the type ‘ANY’, references to any type of object are possible.

In SGML and ODA, references are defined as a special kind of attribute. In SGML, an attribute of the predefined type ‘ID’ must have some unique value. The value of an attribute of the predefined type ‘IDREF’ must correspond to exactly one value of an ID-attribute. With this mechanism, one can refer to the object for which this ID-attribute was defined. For example:

```
<!ELEMENT figure ... >
<!ATTLIST figure name ID #REQUIRED >

<!ELEMENT figref ... >
<!ATTLIST figref refto IDREF #REQUIRED >
```

The above DTD defines an object ‘`figure`’ that has an attribute ‘`name`’: this attribute is required to have some unique value. The object ‘`figref`’ has an attribute ‘`refto`’, whose value must be an ID-value. The following outline of a document is a valid instance of the above DTD (text between angled brackets indicates the start of the named object):

```
... <figure name=tree> ... <figref refto=tree> ...
```

This document contains an instance of ‘`figure`’ that has the value ‘`tree`’ as its name, and the reference ‘`figref`’ refers to this figure by specifying the same value for its attribute. References in SGML are limited, in that they can only refer to objects in the current document, and may only refer to entire objects that have an ID-attribute. HyTime [14], an ISO/IEC international standard built on SGML, among other things defines an extension of SGML that allows references to objects in other documents or to part of an object, thus facilitating hypermedia.

3 CONTENTS

Following the top-down approach of the previous section on logical structure, we now view structured documents bottom-up, by giving terminology related to contents. Again, readers familiar with the concept of structured documents may want to skip the rest of this section and just glance over Figure 2, that summarizes this terminology.

We distinguish three concepts, namely atoms, basic content and content types:

atoms: By atoms, we mean the most basic recognizable objects that are present in documents, for example a character or a circle. One might argue that a character or a circle is not a basic object because one can distinguish even finer parts within them. For instance a character may have a descender or may have some accent mark. That is why the definition contains the qualifier ‘recognizable’. Although this definition is still not watertight, we may reasonably expect that we have conveyed our intention.

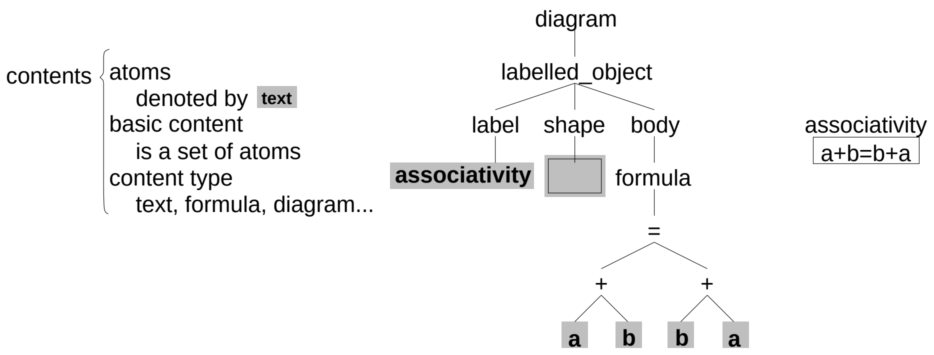

Figure 2. A schematic overview of the terminology on contents used in this article. On the left, the terminology on contents is listed, together with a legend for the tree-figure in the middle. This tree shows an example document, of which a possible layout is shown on the right

basic content: Similar atoms can be grouped into sets, for instance a set of characters, or a set of graphical objects. Such a set is referred to as basic content.

content types: Basic content can be used to build more complex objects, like text, mathematical formulae, tables, and diagrams. These objects are called content types. For example, the content type ‘text’ could be defined to consist of a linear sequence of characters, a content type ‘mathematical formulae’ could consist of a tree-structure of characters and a content type ‘image’ could consist of a two-dimensional matrix of black-and-white values. Another interesting example is the content type ‘diagram’. This could consist of some graph-structure of graphical objects; however, diagrams often contain text as well, so a diagram would also contain instances of the content type ‘text’.

To summarize, content types can be characterized by the components they use (basic content and other objects) and by the internal structure they define between these components. Content types are treated as atomic objects when composing higher-level objects.

4 WHERE LOGICAL STRUCTURE ENDS AND CONTENT BEGINS

In this section, we will discuss some problems regarding the logical structure, the contents, and the grey area in between. Following Sections 2 and 3, we first give a top-down discussion in Section 4.1, followed by a bottom-up discussion in Section 4.2. Both sections use examples from Grif, ODA, and SGML.

The root of the problems lies in the fact that, although logical structure should by definition be expressed within the primary structure, it may also be expressed within the secondary structure or within the contents. Since logical structure is derived from the ‘meaning’ of parts of a document, it is not possible to define an algorithm for defining logical structures, neither has a document one unique logical structure. This leaves the way open for different interpretations by different people.

Besides being an inherently intuitive concept, logical structure is not always separable from physical structure. In [15], logical structure is contrasted with the graphic and visual structure of a document. For example, in text, several conventions and levels of meaning can be distinguished. Aspects such as spelling, semantic attribute encoding (e.g. denoting *emphasis* or even `:-)` irony in some graphic way), punctuation, and graphic relations between blocks of contents, are all levels of graphic encoding of meaning (i.e. logical structure) within contents, blurring the distinction between logical and physical structure, and between logical structure and contents.

4.1 From structure to contents

In this section, we will give some examples of problems with logical structure. To illustrate that there is not one unique way of defining logical structure, we use the earlier example of an article. An article is defined to consist of a sequence of a title, one or more authors, one or more chapters and one or more appendices; a chapter consists of a sequence of a title and one or more paragraphs. However, this same type of document can also be structured in quite another way by grouping objects differently (see Figure 3).

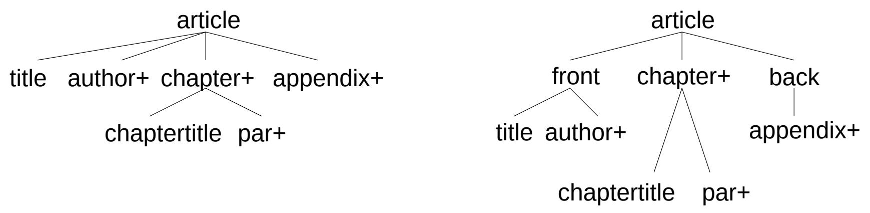

Figure 3. Two different but related document types ('author+' means one or more authors)

Another problem with DTDs is that the logical structure of a document can also be expressed in the secondary structure using attributes, or within contents. This is often encountered in parts that have a fine-grained structure. As an example, take the definition of a variable with a name and a type, as found in many programming languages (e.g. Pascal). This structure can be defined in three ways (see Figure 4). There is no consensus as to which of the three is 'correct'. Probably the definition using attributes is preferred, not because of reasons concerned with the logical structure on its own, but for example because the user-interface for editing then becomes easier.

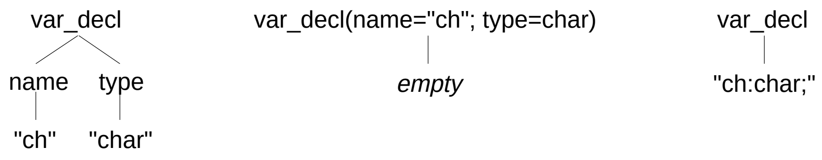

Figure 4. Three ways to define the logical structure of a variable declaration in Pascal: using the primary structure (left), attributes (middle) or a content type for text (right). In this last case, the string is assumed to be in some syntax; this isanalogous to the NOTATION attribute of SGML, as discussed in Section 4.2

The example of Figure 4 leads to the granularity problem, in which the boundary between logical structure and contents begins to blur. What happens is that logical structure is incorporated within contents. This has its effects on defining basic contents and content types, as is shown in the following paragraphs.

In Figure 5, the object called ‘paragraph’ is defined to consist of ‘text’. In this example, a content type ‘text’, consisting of a sequence of characters, is assumed. Alternatively, a ‘paragraph’ could consist of zero or more objects called ‘sentence’, which in turn consist of zero or more objects called ‘word’, and a word consists of one or more objects ‘char’ which represent one single character. Now we have the concept of one character as a content type, instead of a sequence of characters. This example is similar to the one in Figure 3, but as it occurs at the lower levels of the structure, it interacts with the definition of contents.

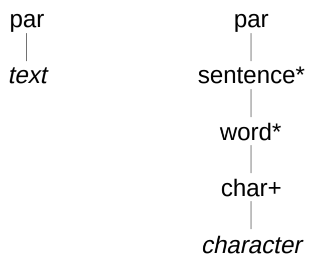

Figure 5. The granularity problem has its effects on content types. At the left, the logical structure stops at the level of a paragraph, and a content type ‘text’ takes over. At the right, the logical structure continues down to the level of a single character, which is defined as a content type

The above example uses a content type for text, one of the more frequently used content types. However, the problem is not confined to only text. As an example, we take mathematical formulae. Since the logical structure of a formula is inherently a tree-structure, and the primary structure of documents also happens to be a tree-structure, most systems use the primary structure to denote the structure of a formula, and combine this with a content type for text. However, we believe this to be an accidental match and not a correct structural approach. At the lower levels of a formula, when we encounter expressions, this leads to two approaches (see Figure 6):

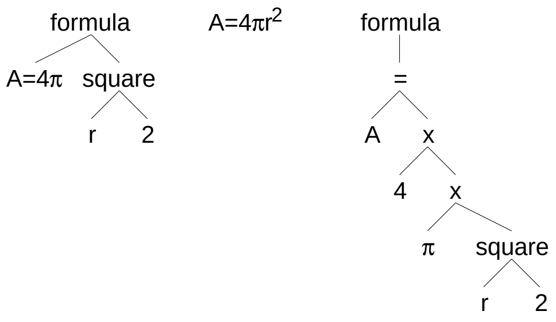

Figure 6. The granularity problem in a mathematical formula. At the left, the ‘logical’ structure of the formula $A = 4\pi r^2$ is shown, in which expressions are regarded as strings. At the right, the (more) logical structure of the formula is shown, that correctly reflects its meaning

- the first approach regards the level of expressions within a formula as basic, i.e. as a linear sequence of characters;

- the second approach considers expressions also to be tree-structured, deriving this structure from the priority of the operators used in the expression.

The first approach is used in Grif. Although the logical structure of expressions is a tree-structure, they are more easily edited as if they were strings. However, this leads to an unnatural grouping, as shown in the left part of Figure 6, which makes interactive manipulation of the formula by the user more cumbersome. If Grif were to use the second approach, editing of expressions would become inconvenient. This is the familiar problem of editing fine-grained structures in a structured editing environment.

An example of a system that uses the second approach is INFORM [16], an interactive editor for mathematical formulae. INFORM does not adapt the definition of the logical structure of expressions so that editing becomes easy. Rather, the editor is designed in such a way, that expressions are internally always represented as trees, while the user edits them as strings [17]. In this way, other applications, such as formula manipulation systems, are still able to use the logical structure of expressions. In our opinion, problems with editing should not influence the definition of logical structure. Instead, systems that manipulate logical structures, such as editors, should be designed to do this gracefully.

Other possible content types, e.g tables, music, chemical diagrams, chess positions, etc, pose even more problems. Unlike mathematical formulae, their structure does not happen to be a tree-structure. For example, in Grif a DTD for tables can be defined as a heading followed by a number of rows that consists of a number of cells, using the list, choice, and aggregate constructs of the primary structure. Using the reference construct of the secondary structure, each cell is linked to the heading of its column (see also [18]).

However, this DTD for tables still allows invalid tables to be expressed that are nonetheless valid with respect to the DTD, since the DTD cannot enforce that a table specification in a document contains the correct number of entries for each row. Grif solves this problem by attaching user-defined functions to some types of components. These functions are called when certain events occur, such as adding or removing columns. Using this mechanism, it is possible to maintain the correct number of cells in each row when column headings are added or removed. However, the structure of tables is still not enforced by the DTD in the same way that the DTD enforces a document structure. Applications external to the DTD are assumed to maintain correctness.

The approach used by Grif to incorporate tables is similar to that of SGML [19]. Incorporating tables and other content types in ODA is still under discussion (see the article of Bormann and Bormann in [6]).

4.2 From contents to structure

We will now look at the content features defined by Grif, ODA and SGML, and use this to approach the granularity problem bottom-up. Grif defines the following content features:

- text: a sequence of zero or more characters;
- symbol: a *single* mathematical symbol that may vary in size, such as the symbol ‘Σ’ used in a summation;
- graphics: a *single* geometric figure, such as a line or a circle;
- picture: a matrix of pixels, i.e. an image. Subtypes are available for EPSF (PostScript), CGM, TIFF, XBitMaps, and other formats.

In our terminology, Grif defines two types of basic content (‘symbol’ and ‘graphics’) and two content types (‘text’ and ‘picture’). The two content types imply two other types of basic content, namely characters and pixels. The basic content type called ‘symbol’ is remarkable. When describing the logical structure of a summation, one could distinguish the lower bound, upper bound and the expression over which the summation iterates, but one does not explicitly refer to the symbol ‘Σ’. The layout of the formula does indeed contain this symbol, but this fact should not be described within the logical structure. The way ‘symbol’ is used in Grif, is by defining a general construct that contains a lower bound, upper bound, and an expression, in which the symbol defines the specific meaning of the general construct. Still, considering only the logical structure and contents of a formula and not considering layout or implementation issues, one would not need to define a basic content item such as ‘symbol’.

In ODA, there are currently three content features defined:

- character content: this defines text, consisting of a sequence of not only characters but also of control functions that specify logical and formatting information for subsequent characters.
- geometric graphics content: this uses CGM [20], which defines entire pictures (i.e. line drawings).
- raster graphics content: this defines images.

Thus, ODA has three content types, which define the three basic content ‘characters’, ‘graphical elements’, and ‘pels’ (picture elements), respectively.

In SGML, just one content feature is defined, called ‘CDATA’ (there are also variants ‘PCDATA’ and ‘RCDATA’ but these are not fundamentally different). This is a simple sequence of zero or more characters. Thus, in our terminology, SGML defines one content type ‘CDATA’ that uses a basic contents for characters. Other contents, e.g. mathematical formulae, raster graphics or geometric graphics are to be expressed using this single content type. This can be done by interpreting the character codes in a different way (e.g. for bitmaps). Alternatively, a special attribute, called the ‘NOTATION’-attribute, can be used to specify some syntax or format within the character data. As an example, we may define an object ‘formula’ that contains character data:

```
<!NOTATION eqn ... >
<!NOTATION tex ... >

<!ELEMENT formula PCDATA >
<!ATTLIST formula format NOTATION (eqn | tex) #REQUIRED >
```

Associated with the object ‘`formula`’ is an attribute ‘`format`’ of type ‘`NOTATION`’, with ‘`eqn`’ or ‘`tex`’ as possible values. These values are declared in the ‘`NOTATION`’ declarations (in these declarations, the dots are to be replaced by a further identification of the notation). Depending on the (required) value of the attribute ‘`format`’, the object ‘`formula`’ is supposed to contain either an *eqn*- or TEX-specification of a mathematical formula.

Grif, ODA, and SGML differ significantly in their content types. Grif defines two content types and two types of basic content. ODA defines three content types, and plans to add other content types, in particular audio, temporal relations and hypermedia (see also the article of Bormann and Bormann in [6]). SGML defines only one content type, and allows several ways to use it in order to support other content types (see for example the HyTime extensions [14]). In general, the support for content types is extensive for text, capable of improvement for mathematical formulae, tables and pictures, and under development for chemical formulae, music, etc. The approaches currently used are suitable for some content types but do not extend to others.

All three systems define one kind of primary structure (hierarchy), and they define content types that internally have their own logical structure. Alternatively, one could define several kinds of primary structure (not only hierarchy, but also matrix, graph, etc.), and use these structures to combine basic contents into content types. This leads to a heterogeneous logical structure, in contrast with the homogeneous structure defined by the systems. The advantage of using a heterogeneous logical structure, is that all logical structure is available within the primary structure in a well-defined way, rather than incorporated within contents in different ways. An example of a system that uses a heterogeneous logical structure is **p**ed**tnt** [21,2].

5 CONCLUSIONS

Structured documents are often explained in connection with formatting systems, but this is only one application. Also, when defining some logical structure, it is tempting to adjust the definition so that implementation of, for instance, a formatter or a user-interface becomes more straightforward (as in the case of expressions in mathematical formulae). However, in order to be able to use many different applications for structured documents, one must strictly separate the concepts of the logical structure from applications concepts and from *ad hoc* practical considerations. This is the approach we took in this article. It leads to a better understanding of the characteristics of logical structure and contents on their own, which are not understood well enough yet, notwithstanding the appearance of standards in this area (see also [22,23]).

The previous sections show the following problems:

- There is no unique logical structure for some given type of document. Since the logical structure of a document is derived from the meaning of parts of that document, logical objects can be grouped differently, or can even be defined differently, by different people.
- As well as defining the logical structure using the primary structure, one can also use the secondary structure (e.g. attributes) for this purpose.
- There is no clear distinction between structure and contents. The granularity, the level at which the structure part stops and the content part takes over, differs among systems. On some occasions, the primary structure is (mis)used to express the logical structure of a content type, on other occasions, logical structure is enclosed within contents. Devising applications that use the logical structure becomes more difficult, since part of the logical structure may reside within content types.
- Content types such as tables, chemical formulae, mathematical formulae, music scores, sound, etc. are not incorporated in current systems, or only partly and in an *ad hoc* way. The current state of the art does not provide for an integrated, uniform way to handle heterogeneous structures in documents.

The first two problems are concerned with the *use* of current systems: these systems all define more or less the same mechanisms to express DTDs, but they do not prescribe the DTDs themselves. Different people will use the mechanisms in different ways, or may even use them in an incorrect way. An example of the latter is the usage of attributes for layout information, which goes against the fundamental concept of structured documents (see for example [24], in which attributes are defined to denote the style for rules that are used as borders of rows and columns in a table).

In general, constructing a DTD cannot be captured in some algorithm, because in order to design a DTD one must understand the meaning of the parts of a particular type of documents. So it seems sensible to accept this problem, and to develop guidelines for designing DTDs (e.g. [19]), and to provide for means to convert documents from one type to another, related type [25].

The other two problems are related and are concerned with the definition of contents. In our opinion, the logical structure of content types, e.g. tables and formulae, should not be defined using the logical structure of a document. These objects are radically different, therefore they should not (and cannot) be forced to be represented in one and the same way. It is tempting to do so with mathematical formulae, but things become difficult for tables and downright impossible for content types such as music scores.

Each content type has its own characteristic structure and set of components. For a document, its structure is a tree-structure combined with references and floating objects; its components are other content types. A mathematical formula also has a tree-structure but is possibly supplemented with concepts like associativity and infix operators; its components are characters.[2] Text has a linear structure; its components are characters and other content types (e.g. mathematical formulae). The challenge is to find a heterogeneous model in which these different structures can be integrated while maintaining their characteristics.

ACKNOWLEDGMENTS

I would like to thank Hans van Vliet, who helped me to structure the content of this article and improved the wording of my English, and Vincent Quint, who answered my questions on Grif and read an earlier version of this text. I am very grateful to Sylvia van Egmond for her invaluable support which gave me the motivation I needed to finish writing this paper. The structure and focus of the presentation of ideas was helped by the useful comments of several anonymous referees.

REFERENCES

1. R. Furuta, ‘Concepts and models for structured documents’, in *Structured Documents*, pp.7–38, Cambridge University Press, Cambridge, (1989).
2. R. Furuta, V. Quint, and J. André, ‘Interactively editing structured documents’, *Electronic Publishing—Origination, Dissemination and Design*, **1**(1), 19–44, (April 1988).
3. R. Furuta et. al., ‘Document formatting systems: Survey, concepts and issues’, *ACM Computing Surveys*, **14**(3), 417–472, (September 1982).
4. V. Quint and I. Vatton, ‘Grif: An interactive system for structured document manipulation’, in *Proc. of the Int. Conf. on Text Processing and Document Manipulation (EP86)*, ed., J.C. van Vliet, pp. 200–213, Nottingham, England, (April 1986). Cambridge University Press.

[2] Note that the components of a mathematical formula do not include mathematical or graphical symbols, since these are only relevant for formatting, not for the logical structure.

5. ISO-8613, *Information Processing—Text and Office Systems—Office Document Architecture (ODA) and Interchange Format*, Geneva, 1989.
6. 'Special issue on Office Document Architecture', *Computer Networks and ISDN Systems*, **21**(3), (May 1991).
7. ISO-8879, *Information Processing—Text and Office Systems—Standard Generalized Markup Language (SGML)*, Geneva, 1986.
8. C.F. Goldfarb, *The SGML Handbook*, Oxford University Press, 1990. (ISBN 0-19-853737-9).
9. R. Furuta and P.D. Stotts, 'Object structures in paper documents and hypertexts', in *Workshop on Object-Oriented Document Manipulation (WOODMAN '89)*, pp. 147–151, Rennes, France, (May 1989).
10. V. Quint, M. Nanard, and J. André, 'Towards document engineering', in *Proc. of the Int. Conf. on Electronic Publishing, Document Manipulation and Typography (EP90)*, ed., R. Furuta, pp. 17–29, Gaithersburg, Maryland, (September 1990). Cambridge University Press.
11. B.K. Reid, 'Scribe: a document specification language and its compiler', Technical Report CMU-CS-81-100, Carnegie-Mellon University, Pittsburgh, (October 1980). (Dissertation).
12. L. Lamport, LaTeX*: A Document Preparation System*, Addison-Wesley, 1985. (ISBN 0-201-15790-X).
13. S. van Egmond and J.B. Warmer, 'The implementation of the Amsterdam SGML parser', *Electronic Publishing—Origination, Dissemination and Design*, **2**(2), 65–90, (July 1989).
14. S.R. Newcomb, N.A. Kipp, and V.T. Newcomb, 'The "HyTime" hypermedia/time-based document structuring language', *Communications of the ACM*, **34**(11), 67–83, (November 1991).
15. R. Southall, 'Visual structure and the transmission of meaning', in *Proc. of the Int. Conf. on Electronic Publishing, Document Manipulation and Typography (EP88)*, ed., J.C. van Vliet, pp. 35–45, Nice, France, (April 1988). Cambridge University Press.
16. S. van Egmond, F.C. Heeman, and J.C. van Vliet, 'INFORM: an interactive syntax-directed formulae editor', *The Journal of Systems and Software*, **9**(3), 169–182, (March 1989).
17. F.C. Heeman, 'Incremental parsing of expressions', *The Journal of Systems and Software*, **13**(1), 55–70, (September 1990).
18. G. Coray, K. Lemone, and C. Vanoirbeek, 'The use of inheritance in document specifications', in *Workshop on Object-Oriented Document Manipulation (WOODMAN '89)*, pp. 165–169, Rennes, France, (May 1989).
19. ISO/TR-9573, *Information Processing—SGML Support Facilities—Techniques for Using SGML*, Geneva, 1988.
20. ISO-8632, *Information Processing Systems—Computer Graphics—Metafile for the Storage and Transfer of Picture Description Information*, Geneva, 1987.
21. R. Furuta, 'An integrated, but not exact-representation, editor/formatter', in *Proc. of the Int. Conf. on Text Processing and Document Manipulation (EP86)*, ed., J.C. van Vliet, pp. 246–259, Nottingham, England, (April 1986). Cambridge University Press.
22. V. Joloboff, 'Document representations: Concepts and standards', in *Structured Documents*, 75–106, Cambridge University Press, Cambridge, (1989).
23. D.M. Levy, 'Topics in document research', in *Proc. of the ACM Conf. on Document Processing Systems*, pp. 187–193, New York, (December 1988). ACM.
24. Association of American Publishers, *Markup of Tabular Material*, April 1986.
25. R. Furuta and P.D. Stotts, 'Specifying structured documents transformations', in *Proc. of the Int. Conf. on Electronic Publishing, Document Manipulation and Typography (EP88)*, ed., J.C. van Vliet, pp. 109–120, Nice, France, (April 1988). Cambridge University Press.