ELECTRONIC PUBLISHING, VOL . 1(2), 97–104 (SEPTEMBER 1988)

Parallel processing and document layout

HEATHER BROWN

Computing Laboratory
The University
Canterbury
Kent CT2 7NF
UK

SUMMARY

Interactive editing and layout of high quality multi-media documents is a demanding application that is limited by the processing power available from current workstations. This short paper takes a preliminary look at the opportunities for exploiting parallelism within the document layout process, and suggests that radically new ways of thinking may be needed to take advantage of the enormous parallel processing capabilities offered by a new generation of workstations based on configurable networks of Transputers.

KEY WORDS Structured documents ODA (Office Document Architecture) Document layout Parallel layout algorithms Occam Transputers

THE CHALLENGE

The current state of interactive document processing and, indeed, the whole so-called 'Desk-top Publishing Revolution' has been made possible by the simultaneous emergence of cheap but powerful workstations and laser printers. But with software systems already limited by the processing power available on the current ranges of workstations, it is time to look ahead at the opportunities — and problems — provided by parallel processing techniques and the major leap in workstation performance they can provide. Transputer-based workstations capable of delivering processing power of the order of hundreds of MIPs are already available in research and development environments. The challenge is to develop the software techniques needed to use them effectively.

As a first step in taking up the challenge, this paper attempts to identify areas where parallelism may be exploited within the document layout process. It begins by showing how the layout process fits into the overall document processing model and how it may be divided into high level layout and low level layout. It then concentrates on areas of the model that offer parallel processing potential, and finally suggests that some parts of the layout process might be tackled in a very different manner in a parallel environment.

STRUCTURED DOCUMENTS

The idea that documents conform to an underlying structure has been reflected in document preparation systems for many years. Much recent work has been based on the original ideas of Alan Shaw[1], and has lead to the development of structured systems like GRIF[2]. and Interleaf[3]. and to the proposed international standard for an Office

0894–3982/88/020097–08$05.00
Received 6 November 1987
Revised 3 February 1988

Document Architecture (ODA)[4]. Although there is no real agreement on the form such document structures should take, a number of general trends can be identified. Most document structures are hierarchical and object-oriented, they provide a framework for multimedia documents, and they frequently provide a mechanism for defining document ‘styles’ or generic document definitions. Another important feature is a means of keeping low-level layout details separate from the document structure and content. This allows different ‘views’ of the document to be produced relatively easily.

As the document model defined by ODA conforms to these general trends, it will be introduced very briefly and used to provide the framework of the discussion on parallelism. Readers interested in further details are referred to Part 2 of the ODA standard itself or to Brown[5] for a fuller informal description.

ODA DOCUMENT ARCHITECTURE

The ODA model of a document is tree-like. The *content* of the document is stored entirely in the leaf nodes, while the *structure* of the document is given by the shape of the tree. This separation of content and structure is crucial for multimedia documents as it allows different types of content to co-exist within the same document.

An ODA document is described by two structures: a *logical* structure and a *layout* structure. The logical structure divides and subdivides the document into objects that mean something to the human author or reader. Chapters, sections, titles, diagrams, paragraphs and references are typical examples of logical objects. Only the lowest level objects, such as titles or paragraphs, have content associated with them. The layout structure on the other hand is concerned with a visible representation of the document. It divides and subdivides the content into page sets, pages, and rectangular areas within pages. If these rectangular areas have nested areas defined within them they are known as *frames*. The lowest level areas are known as *blocks*. A frame might be used to represent a column of text, for example, with nested blocks representing individual paragraphs. By definition, only blocks have content associated with them.

The creation of the structures for a particular document is guided and controlled by two sets of object type definitions (one set for logical objects and one for layout objects). These specify the types and combinations of objects allowed and thus provide *generic* structures for a particular class or ‘style’ of document. In ODA terminology the definitions constitute the generic logical and generic layout structures for a document class, while a document belonging to the class is described by its own specific logical and specific layout structures.

In addition to these four structures, ODA uses *content architectures* to define content and the rules used to process it. There are currently three content architectures: character, raster graphics, and geometric graphics. For character content, the content architecture defines details of the positioning and orientation of characters, positioning of lines and words, indentation, tabulation, and the use of different character fonts. (Full details of the content architectures can be found in parts 6, 7, and 8 of the standard.)

The *layout process* decides exactly where each item of the document is to be placed. It uses the specific logical structure, the generic structures, and the content architectures to create the specific layout structure. The process divides cleanly into two. The low level (or content) layout process takes content portions and lays them out into blocks.

The high level (or document) layout process takes the resulting blocks and places them in frames or pages. The content architectures are only used in the low level layout.

An example of high level layout

The high level layout process is influenced by a number of attributes defined in the generic structures. This section introduces the relevant attributes and gives a simplified example of high level layout.

The definition for each (non-leaf) object has an attribute called *generator for subordinates* which describes how the object can be made up from subordinate objects. These attributes effectively define a simple grammar for the document class. They indicate that subordinate objects may be optional (OPT), required (REQ), or repeated (REP), and that groups of subordinates may occur in a given sequence order (SEQ) or as a choice (CHO) where only one of a group occurs.

Figure 1 shows a simple generic logical structure that might be used for a section of a technical document. It indicates that a section consists of a compulsory title, followed by an optional author's name, followed by one or more subsections. Each subsection starts with a subtitle. The 'REP CHO' construct indicates that the subtitle is followed by a series of paragraphs or lists occurring in any order. Lists consist of one or more items.

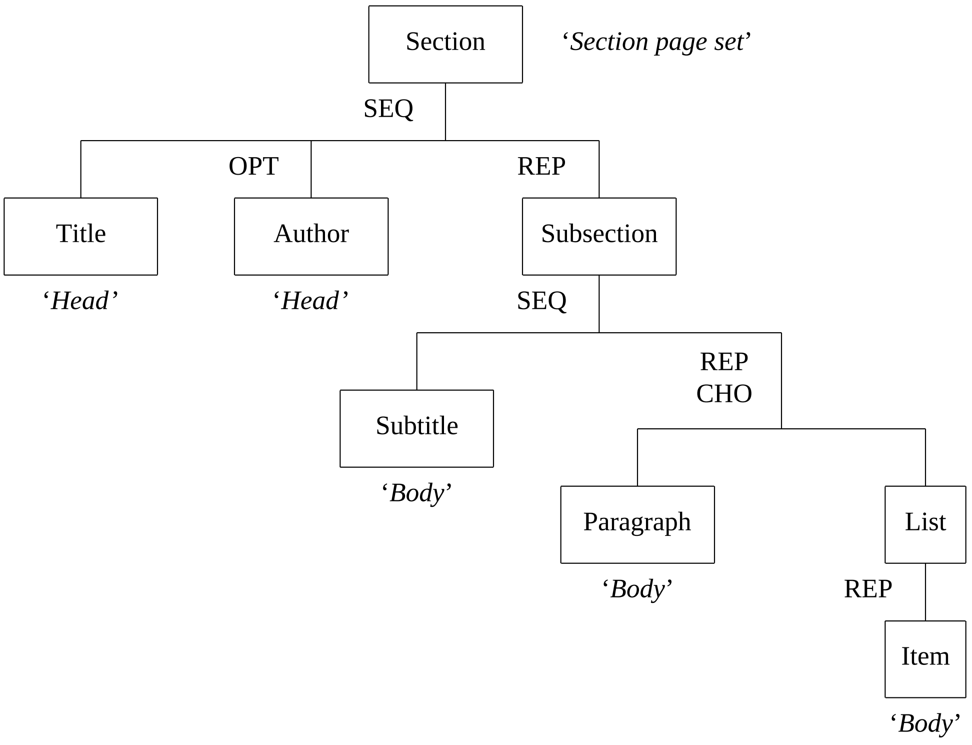

Figure 1. Generic logical structure

A corresponding generic layout structure might define two different page styles — a special ‘Title page’ for the first page of the section, and a ‘Continuation page’ for all subsequent pages. Figure 2 shows the top level of a possible generic layout structure. The title frame represents an area set aside on the first page for the title of the section, and for the author’s name if present. The body frame and continuation body frame represent areas for all the rest of the section.

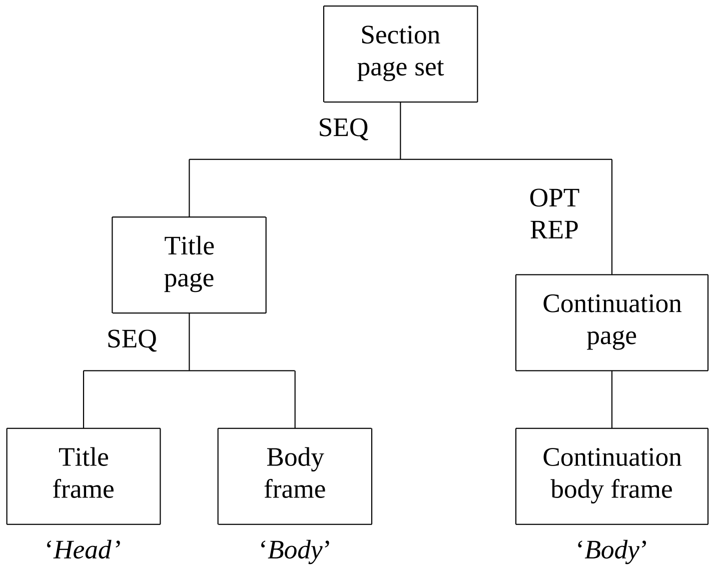

Figure 2. Generic layout structure

A crucial aspect of the high level layout is to decide what layout objects may be used for a given logical object. This is dictated by three attributes (whose values are shown in italics in Figure 1 and 2) called ‘layout object class’, ‘layout category’, and ‘permitted categories’.

‘Layout object class’ is used to indicate that an entire logical object must be placed in a single instance of a particular type of layout object. No other part of the document may share that layout object. The attribute normally refers to a page set or page and is used to direct major logical divisions of the document into separate sets of pages. Within one of these major divisions, ‘layout category’ and ‘permitted categories’ are used to direct logical objects into certain frames. ‘Layout category’ associates a single name with a leaf logical object, and ‘permitted categories’ associates one or more names with a frame. If an object is given a layout category, it can only be placed in a frame that has been given the same name as one of its permitted categories. An appropriate use of this category mechanism can help with complex layout where different content needs to be directed to different areas.

In our simple example we use only one layout object class and two categories. The logical section of Figure 1 has its layout object class defined as ‘*Section page set*’, and each section is thus laid out in a separate instance of the page set shown in Figure 2. When the layout process comes to the section title, it looks for a frame with ‘*Head’* as one of its permitted categories, and thus directs the title (and the author's name if present) to the title frame on the title page. All the other leaf items in the section have '*Body*' as their layout category, so the layout process directs the first subtitle to the body frame on the title page. Subsequent items are sent to that frame until it is full, and they are then sent to continuation body frames on continuation pages. The positioning of blocks within the frames is controlled by further attributes that are not described here.

Figure 3 shows a specific page set that might be created. The frames are labelled with their permitted category names. Note that the continuation body frame on the third page is not necessarily full.

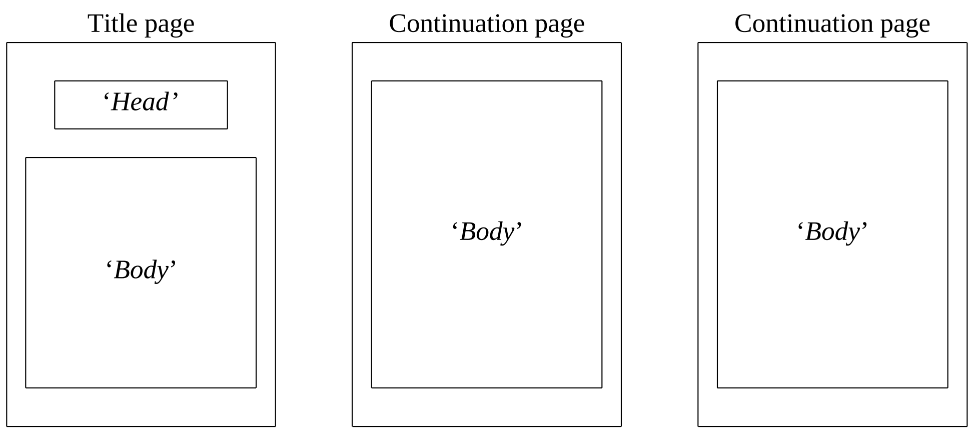

Figure 3. Specific page set

PARALLELISM WITHIN THE ODA MODEL

Two areas within the model provide scope for exploiting parallelism. The first of these is in the low level layout where the processing of individual blocks is essentially independent. The second comes from the one-to-one correspondence between page sets and major divisions of the document as defined by the layout object class.

Putting these two together makes it attractive to think in terms of the low level layout consisting entirely of a large number of independent parallel processes, each working on its own individual block, and the high level layout consisting of a smaller number of mutually independent parallel processes each using the information from the lower level to work on its own page set.

The framework described above makes no assumptions about the standard of layout, but the interface between the high and low level must make suitable provisions for high quality. In particular, if we assume the layout process uses some form of the TEX boxes-and-glue-and-penalties model[6], then some information on the penalties for splitting a block across frames must be passed from the low level to the high level. In most cases it would be sufficient for the low level layout to provide:

- the dimensions of the block;
- a list of places where the block may be split, together with an associated penalty.

In the majority of cases only blocks with character content would be splittable. The positions for splitting would, of course, correspond to the divisions between lines of text, and the associated penalty would take account of widows, orphans, and hyphenation.

This information would normally be sufficient to allow the high level to cope with awkward layout problems. However, in case the high level could still not easily avoid widows and orphans, or place graphics blocks of awkward size, it might be appropriate to allow a reprocessing mechanism where the high level could ask the low level to process a block again with a target size specified. This mechanism might enable the low level to provide a paragraph one line longer or shorter, or to return a scaled graphics block.

The independence of the different parts of the layout process in the basic framework described here has a number of attractions for interactive editing as well as parallel layout. In many cases the effects of editing a logical object are confined to the low level layout of the object itself and the high level layout of its containing page set. Where an edit causes changes to such things as page or section numbers the effects are spread further — which is why some interactive systems keep automatic numbering facilities to a minimum. A reprocessing mechanism, as described in the previous paragraph, widens the effects of edits and is therefore one of the elements of quality that is all too frequently discarded. A parallel workstation should cope easily with the processing overheads of this improvement.

RETHINKING FOR A PARALLEL ENVIRONMENT

It is possible to find a number of promising extensions to the basic framework outlined in the previous section. In particular we could seek to exploit the use of layout categories so that independent parallel processes could cope with different categories. However, this kind of thinking is really only tinkering with the problem. Radical new environments need radical new ideas. It is important to recognise that the ground rules have changed and the basic principles of system design may need to be changed with them.

Occam[7] provides the most flexible environment for exploiting Transputer networks. Programs consist naturally of many small concurrent processes communicating along point-to-point synchronised channels, and the overheads associated with creating and scheduling processes are very small — comparable to the overheads of procedure calls in sequential languages. Designing large software systems in occam is, in some respects, more like hardware design with processes taking the place of simple hardware components and complex systems being built up from large numbers of simple replicated components. Used sensibly, this ‘occam engineering’ approach provides a powerful tool to help us reason about parallel systems[8]. Systems supporting 100,000 concurrent processes are perfectly feasible and manageable. The Transputers themselves provide immense power and speed. One board containing four T800 floating-point Transputers can deliver 40 MIPs, and each of the Transputers has a usable DMA input/output capacity of 1.8 Mbytes/second (in each direction) over four links.

A great deal remains to be learned about the effective use of Transputer networks, particularly about mechanisms for distributing large bodies of data and for balancing processing loads. Three general points are described briefly below.

(i) Multiple copies of data may be preferable to partitioned data. In 3-dimensional graphics systems, for example, it may be profitable to duplicate the data model of a scene in each Transputer, but to arrange that each Transputer performs ray tracing and rendering for its own section of the required view.

(ii) Pipelines of processes, with the data moving through the processes, may be more effective than processes that range over the data. If the data can pass through the pipeline as fast as they can be accessed, then long pipelines of extremely simple processes may outperform complex methods ‘proved’ to be more efficient for sequential computers. (A simple example of this comes from sorting. A pipeline of trivially simple processes — each comparing two numbers and passing the larger one on to the next — can implement a form of bubble sort that is as fast as reading and storing the numbers ready to perform a ‘more efficient’ sort.)

(iii) Processes may replace data structures. Complex links between pieces of data may be replaced by the channels between occam processes. An element of this showed up in the layout framework introduced in the previous section, where the levels of the layout process formed a tree-like pattern similar to the specific layout structure.

TOWARDS PARALLEL PARAGRAPH LAYOUT

This section attempts to apply some of these ideas to a TEX-like paragraph layout algorithm.

At the top level we might think in terms of the pipeline approach. As the paragraph content is accessed it passes through the following pipeline of four major processes.

- *Hyphenation*
  This contains all the data required for the hyphenation algorithm. It identifies possible hyphenation points and inserts information about them into the data stream.
- *Font information*
  This contains all font information. It keeps track of the current font and inserts any necessary information (character widths, space sizes, ligatures, kerning, and so on) into the data stream.
- *Boxes-and-glue-and-penalties*
  This stage turns the data stream into the boxes and glue form needed for the final line breaking process. Some penalties can be dealt with at this stage.
- *Line breaking*
  This is the main layout process which uses the penalty system to decide on the optimum layout. As it does not know about page breaks, it can only place advisory penalties on possible page breaks at each line.

Each of these four main processes would certainly be broken down further. Within the hyphenation process, for example, it might be appropriate to investigate the possibility of replacing TEX’s complex trie structure by processes. (Different words would then be passed through a different pipeline of processes.) Within the final line breaking process, on the other hand, it might be more appropriate to think in terms of spawning separate pipelines of processes to evaluate the different possibilities.

CONCLUSION

Transputer-based systems are already being used for driving high quality laser printers and thus for the imaging of documents. The ideas presented in this short paper are intended to show that there is also an enormous potential for the use of Transputers in the area of interactive document editing and layout, and to stimulate discussion and argument on the ways in which this potential can be realised.

ACKNOWLEDGEMENTS

Many of the ideas on parallelism expressed in this paper come from discussions with Peter Welch. I am grateful to him for infecting me with his enthusiasm for occam and Transputers. I would also like to thank Ian Utting and colleagues from the Alvey Fortune project for many illuminating discussions on parallel layout. Finally, I would like to thank the SERC for their support of several research projects on document processing.

REFERENCES

1. A. C. Shaw, ‘A model for document preparation systems’, *Technical Report 80–04–02*, University of Washington, 1980.
2. V. Quint and I. Vatton, ‘Grif: An Interactive System for Structured Document Manipulation’, in *Text Processing and Document Manipulation*, J. C. van Vliet ed., pp 200–213, Cambridge University Press, 1986.
3. R. A. Morris, ‘The Interleaf User Interface’, in *PROTEXT III: Proceedings of the Third International Conference on Text Processing Systems*, J. J. H. Miller ed., pp 20–29, Boole Press, 1987.
4. ISO/DIS 8613 Information Processing — Text and Office Systems — Office Document Architecture (ODA), Parts 1,2,4–8, 1987.
5. H. Brown, ‘Document Structures for Integrated Text and Graphics’, *Techniques for Computer Graphics*, D. F. Rogers and R. A. Earnshaw ed., pp 469–483, Springer-Verlag, 1987.
6. D. E. Knuth and M. F. Plass, ‘Breaking Paragraphs into Lines’, in *Software — Practice and Experience*, **11**, 1119–1184, (1981).
7. D. May and R. Shepherd, ‘Occam and the Transputer’, in *Concurrent Languages in Distributed Systems*, North-Holland, 1985.
8. P. H. Welch, ‘Emulating Digital Logic Using Transputer Networks (Very High Parallelism = Simplicity = Performance)’, in *Proceedings of the ‘Parallel Architectures and Languages Europe’ International Conference*, pp 357–373, Springer-Verlag Lecture Notes in Computer Science 258, 1987.