THE
UNIVERSITY
OF HULL

What Is A Document?

Darren Stephens
Centre for Internet Computing
University of Hull
Draft 2

June 2001

Typesetting

This document was typeset using LaTeXversion 3.14159 in Windows NT4 - (MikTeXdistribution) and RedHat Linux 7.1 in draft forms between April and June 2001. Postscript graphics have been produced using; Adobe Photoshop, Jasc PaintShopPro and Dia.

The document was printed using a Hewlett Packard DeskJet 710C for draft copies and a Hewlett Packard LaserJet 4M for the final version, both printing from Windows NT.

Contents

List of Figures

1 Introduction

1.1 The Definition of A Document

The SGML specification document [ISO8879] provides following two definitions within its glossary:

- **Document :** A collection of information that is processed as a unit. A document is classified as being of a particular document type.
- **Document Type :** A class of documents having similar characteristics; for example, journal article, technical manual or memo.

The first of these definitions describes the salient characteristics of a document. The definition is a very broad one and could encompass a very wide range of entities. The key words to understand within the definition are, however, *collection* and *unit*. The former implies that a document can be a composite entity, consisting of a number of parts. The latter implies a sense of ordering or structure which can be used to describe the whole.

The second definition states that documents may be classified depending upon features or properties that a particular class of documents may have. The definition further implies a type or commonality for classes of documents and that this can be represented in an ordered and consistent way. This will be discussed later, in section 2.3.

Documents may take many forms, both electronic and otherwise. For the purposes of this paper, however, discussion will be restricted to documents that can be represented and transferred in electronic formats.

1.2 Collections of Documents

It is more usual for documents in areas other than hypermedia to be self-contained but also to reference other documents, and in addition share common elements by means of inclusion,generally by the use of bibliographies and footnote systems. The problems of consistent addressing and referencing are interesting within themselves but will not be discussed within this document except in their most general sense. This issue is already addressed within web hypermedia by use of URI system (which is discussed briefly in section 2.4).

Hypermedia documents (more often called "nodes" by hypermedia practitioners), however, have associative relationships (links) between them which enable users to move freely between nodes. It is possible to consider a number of nodes as comprising a single interconnected, composite, distributed document entity.

Even then it is difficult to ascertain the extent or boundaries of such composite entities. A number of people (Bray [Bray1996], amongst others) have discussed this problem. This relationship between nodes or groups of nodes has been examined in work done in the area of hypermedia research.

1.3 A Definition of Mark-Up

Documents can be constructed using what is known as a *mark-up language*. Such a language is used to identify and show elements of document structure within the document itself. The term was first used by Charles Goldfarb during the development of GML (see Section 2), the embryonic mark-up language developed by IBM in late 1960s.

Marking-up can take a number of forms and could be done both by machines and by human authors. This does not however, strictly speaking, have to be case, although one of stated aims of SGML standard was to create document mark-ups that were both human readable and editable . For instance, HTML is easily written and read by both human and mechanical readers but a Microsoft Word document is not. In the latter case, the underlying structure is not obviously apparent to a document author, although such a structure does exist.

A document is marked up by means of "tagging". and this tagging may (or may not) be human readable. Sections of document that require description are enclosed by tags to describe an *element*. The tag itself is enclosed by delimiters and the choice of such delimiters is arbitrary. An example of HTML markup, showing the nesting of elements and the use of tagging can be seen in Figure 1.3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type"
      CONTENT="text/html; CHARSET=iso-8859-1">
<TITLE>A Test Document</TITLE>
</HEAD>

<BODY>
<H1>Test Document</H1>
<P>Testing, testing, 1, 2, 3!</P>
</BODY>
</HTML>
```

Figure 1: An example of SGML style mark-up in HTML

The particular style shown become increasingly popular because of rise in use of

mark-up languages like HTML and XML. Both mark-ups descended from the SGML standard that Goldfarb had defined. Elements do not have to be atomic and can contain others, although this is dependent upon the mark-up language used and the DTD (q.v) that defines it.

2 A Short Taxonomy Of Documents

In order to understand various forms of documents, it is perhaps useful to provide a short review of such items to better facilitate classification. Graphically, this strucutre is shown in figure 2. Items at bottom levels are shown as examples only.

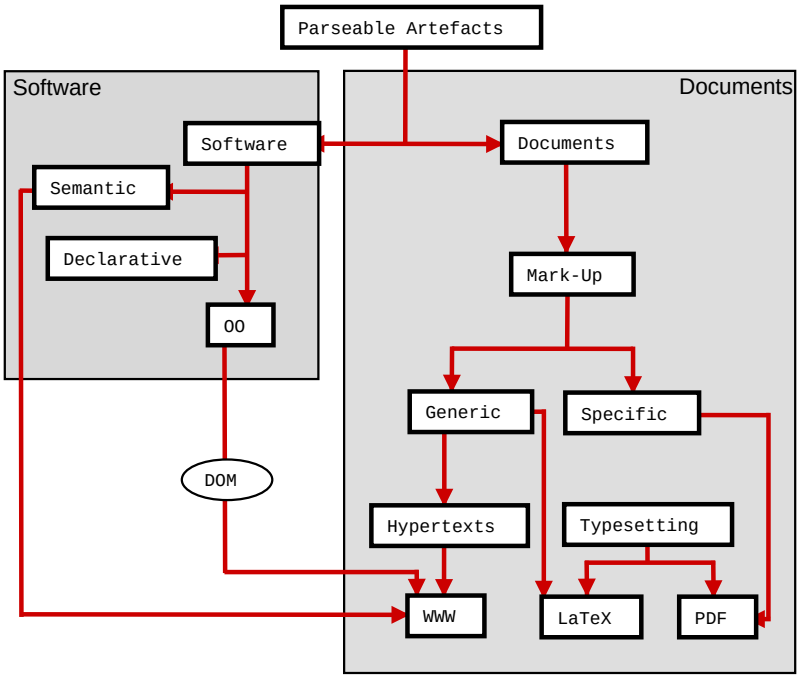

Figure 2: A General Document Taxonomy

Bryan[Bryan1997a] states that any device that stores formatted text for later retrieval and use must use some form of mark-up to do so. This category therefore includes program source code. The mark-up may only be machine readable in some cases however. In general terms the following types of document exist:

- **Unformatted Text**
  The simplest form of document is the unformatted, raw text document, The data contained inside is not subject to any formal structure although it is actually to provide a (trivial) DTD for such a document. It can also be argued that,without such a DTD, a document would, in fact, conform to a notional type definition, especially in the case of natural language

documents. This notional definition would have to be determined by the reader for the document to have any meaning, however.

- **Specific Mark-Up**
  Specific mark-up is used in systems such as PostScript or RTF processing in word Processors like Microsoft Word or Corel WordPerfect. The mark-up is usually application or task specific and is not designed to be transferable between environments or applications. This can also include styles such as delimiter separated lists, where each field's meaning may only be known by the parsing application. As such, specific mark-ups are commonly used in typesetting systems, where the major degree of formatting control is exterted by the machine.

- **Generic Mark-Up**
  One of the stated goals of generic mark-ups is to allow encoded information to be exchanged within a number of environments and to allow the resultant documents to be viewed and edited by both machines and people. Generalised mark-ups are commonly stored in plain text files, making them highly portable systems as tools to convert text file formats are effectively universal. LaTeXis an exception to the specific mark-up situation that usually occurs with typesetting languages. The mark-up used in the TeXfamly is actually generic and is easily transformable into HTML or PDF, for example. [1]

The move to develop generic mark-ups began in the late 1960s as a result of first Gencode, and later GML (both at IBM), led by Goldfarb [Goldfarb1970]

Such documents *do not* need to be entirely textual. They are able to contain references to (or have embedded within them) other objects such as multimedia content. Implications of this in hypermedia documents are discussed in section 3.2

2.1 The Structure of Mark-Up Documents

The description of generic mark-up documents is generally presented in one of two ways: Presentational vs. Logical

- **Presentational Structure** describes how a document will *ppear* when used. It is specifically concerned with the description of visual elements and how a document will appear when viewed. As far as version 3.2, HTML had begun to evolve along this route. HTML 4 and its variants attempts to reverse this trend.

[1] Many of these tools are part of the standard TeXdistributions

- **Logical Structure** is more concerned with the layout of document in terms, not of its visual appearance, but how it may be interpreted by machines or how its meaning may be interpreted by a human user.

Jelliffe prefers to extend the presentational versus logical division into six sub categories which uses to describe the structure of a document more fully. These he defines as:

1. **Page Layout**, describing the high level visual layout of the document, as most people would use when working with a desktop publishing environment.
2. **Page Objects**, describing individual items within the document such as images, figures, tables, text blocks, etc.
3. **Glyphs**, are another visual indicator: the type faces and sizes used for type within a document.
4. **Characters**, the meaning associated with a particular glyph [ISO9541], such as the choice of language, environment or character set.
5. **Editorial Structure** - word, sentence and language structure within the document.
6. **Topic Structure**, the arrangement of content at a higher, conceptual level.

Jelliffe further states that there is a flow of dependence, progressing in reverse order up this list. Unlike so-called traditional software, the topic structure of a document may affect its page layout. For some documents of course, some of these categories have little or limited meaning. A document made up solely of a sequence of images may require or exhibit little or no glyph information, for instance.

The simple linear flow of dependence may not, then, be sufficient to describe the structure of documents. It is perhaps, more realistic to describe feedback mechanisms (akin to lifecycle models in software engineering) to model the document design process and the artefact it produces. This is espcially true because documents are likely to undergo more constant, greater and varied re-engineering throughout their lives.

Document mark-up languages can be further classified into another two broad categories:

- **Procedural Mark-up**, which describes how a document WILL be displayed. This is most commonly used in formatting languages such as PostScript or Word Processing systems (most especially WYSIWYG ones)

or where presentational cues *must* be adhered to maintain a correct and consistent visual appearance.

- **Descriptive Mark-up** only provides suggestions to a rendering agent as to how the marked-up document should be displayed (i whatever context that may be). The user agent does not have to abide by these instructions, they are merely hints. "Classic" HTML and the TEXlanguages were designed to work in this way, allowing individual user agents or rendering agents to decide on the exact presentation used, either internally or by the introduction of an external entity defining presentational characteristics. In a web context, this commonly takes the form of a *stylesheet*

To illustrate this point, we can consider the example of the typesetting language TEX(and its variants,including LATEX). LATEXis described at the web site of the LATEXproject [LaTEX2000] in the following way:

> "LATEXis a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents, but it can be used for almost any form of publishing.
>
> "LaTEXis not a word processor! Instead, LaTeX encourages authors not to worry too much about the appearance of their documents, but to concentrate on getting the right content."
>
> "To produce this in most typesetting or word-processing systems, the author would have to decide what layout to use, so would select (say) 18pt Times Roman for the title, 12pt Times Italic for the name, and so on. This two results: authors wasting their time with designs, and a lot of badly designed documents!"

The key paragraph is the second. It is an almost complete re-iteration of the principles of a generic, descriptive mark-up language.

Generic document mark-up is derived from the concept of the physical mark ups (note the difference in spelling) used by typesetters. Mark up of that type is used to show compositors the physical layout of manuscripts prior to printing. Hence it is a physical procedural mark-up.

LATEXis not a hypermedia environment, It does not have the capability to hyperlink. [2] It can however,contain embedded links to other resources, acting like the inclusion mechanisms mentioned in section 3.2.

[2]Extensions are avaialble to embed hyperlinks within documents, to be opened by external agents, however

2.2 Document Type Definitions

The concept of classification is a highly important one. as is apparent from the earlier definitions of what documents are. The mechanism of the Document Type Definition or **DTD** exists to allow such classification and to define document structures consistently, The DTD describes what elements of a particular type a document can contain and further, how the grammar of aggregations of these elements is defined.

In the area of XML this concept been further extended to allow creation of XML Schemas [XML2001]. XML Schemas define a DTD using XML syntax and are themselves XML documents. Proponents of schemas say that they are easier to build and maintain than the more opaque DTD. They are also easier to generate and change dynamically.

2.3 Document Validity

What constitutes a valid document? Stated simply, a valid document is one that conforms to its specified DTD. However, it is apparent on Web at large that a great number of documents are not valid HTML but still manage to be viewable by a range of user agents. This is not entirely due to user agents themselves more generous in their document parsing. Clearly, validity is not a complete description of a document's ability to be used, or to be "complete", in some way.

Documents may also be *well formed* . A well-formed document is one that conforms to a grammar defined by DTD. By definition a valid document is also well formed. However, a well-formed document may also contain elements or attributes not defined within the DTD. Typically, in an HTML context, such items not within the DTD will be ignored but the document will still be processed if structured correctly i.e.: they are syntactically correct. This is discussed further in section 3.3

For mainly historical reasons, HTML browsers tend to be "sloppy" when parsing. This is to allow user agents to display nodes in at least some form, however ugly, and not to burden end users with error messages. Browsers such as W3C's Arena and Amaya browsers do allow users to view errors in badly-formed HTML if they wish.

In web browsers, display of HTML is done in three stages:

1. **Retrieval**, where document is fetched, whether from the network or from a local source.
2. **Parsing**, where the document is read and arranged into a document tree

containing document elements. Whether this parsing is done by a validating parser or not is an issue of implementation.

3. **Rendering**, where the parsed tree is then formatted for display by a display manager.

Sloppy parsing of HTML was a key reason for its rapid take-up by amateur users, allowing badly formed mark-up to be written and displayed. However, an XML or SGML parser is more strictt. Syntactical and semantic errors are reported, especially because in many cases, visual rendering is not required. In XML this job should be done by formatting systems such as XSL (eXtensible Style Language).

2.4 Hypermedia

A major difference between hypermedia systems and more general document systems is the ability to directly follow links from one "node" to another, navigating through some space defined by its developer. The path through this space is not necessarily a linear one and, as is discussed later, links do not fall into a single, homgeneous, category.

The idea of hypermedia is not a new one. As early as 1945 Vannevar Bush [Bush1945] had suggested construction of an aid to human memory which he called "memex". Such an aid was designed to link data items, providing associations between them, which could then be navigated by the user.

Others, such as Nelson with his Xanadu project [Nelson1965] and Englebart's NLS (oN Line System) [Englebart1995] also provided means for navigating through data spaces in non-linear ways.

Notwithstanding this, hypertext systems are still systems of documents.

As Hall and Lowe point out, however, until the early 1990s many hypermedia systems were closed [Lowe1999b] and locked into proprietary standards. In the hypermedia community, however, the definition of a closed system used in such a context a different meaning to that used in more general computing. A closed hypermedia system is one where there are a fixed number of encapsulated applications integrated with mechanisms used to link them. In effect, without the facility to easily create or edit nodes during navigation, a system will be closed. Any mark-up being used in these systems was very likely to be specific and perhaps also procedural.

Tim Berners-Lee [Berners-Lee2000] specifically identified a generic document mark-up standard as one of the cornerstones of the development of the World Wide Web, in tandem with a resource identification system (URI) and a transport protocol to deliver content (HTTP). His envisaged system was an open one,

although the ability to edit nodes in real-time via use of the HTTP PUT method (included in his original web client software) not been widely implemented at either client or server end. Two notable exceptions are W3C's Amaya and Netscape's Navigator Gold 2.x and 3.x browsers. The apache web server also includes support for using the HTTP PUT mechanism in this way.

3 Relationship Documents and Software

3.1 Nature of Software

At this stage it is wise to narrow the area of investigation of artefacts a little further. Up until this point the consideration of documents has been mainly as generic entities. In order to concentrate specifically on relevant properties of documents in web-based systems, discussion of document systems will now be limited to the area of hypermedia. This is possible because hypermedia documents are merely a subset of entire set of documents. Any assumptions or properties stated will apply only to that class of artefacts.

3.1.1 Documents As A Software Type

It is possible to extend Bobrow and Stefik's software taxonomy to include so-called "semantically oriented" entities. These entities could also be described as declarative constructs, putting them into same category as languages like Prolog, which merely present relationships entities and expect executing agent to analyse predicate calculus defined to provide meaningful results. Mark-up documents do conform to definition of software provided in that paper: they specify a transformation a raw state and marked-up state which may be interpreted by some user agent and they do so within a specified context (that of user agent that displays them). Such a relationship is illustrated in Figure 3.1.1.

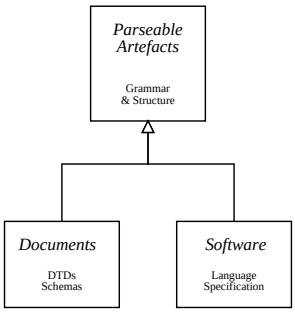

Figure 3: The Relationship between Documents & Software

Warren [Warren2001] identifies that web-based systems follow Lehman's Laws of Software Evolution[Lehman1985], providing another indication that web systems and software systems are somehow related.

Walker [Walker1997] identifies a number of parallels software engineering and what he calls IEW (Information Engineering for the Web). In addition this, most web systems are of a mutable nature and can thus be classified, using

Lehamn's taxonomy [Lehman1999], as a *type E* system; such systems may have emergent properties that arise through real world use.

Further, it is also possible for hypertexts to be considered as an example of a declarative programming language. we can treat the realtionships between elements and mark-up generally as examples of clauses - statements about the structure of a document. Such a system would be broadly in line with Kowalski's view of a purely declarative programming language, wholly concentrated upon a predicate calculus relating elements within a document. Such a predicate calculus would vary with each document but would still be governed by rules laid down using the document grammar described in its DTD. Such a document would also have object-oriented characteristics, due to the nature of the Document Object Model [W3C2001a], It would also show correspondence with the work of Eisenstadt and Brayshaw [Eisenstadt1988], who comment on the implicit connection with object-orientation that logic programming systems display.

Producing a LaTeXdocument (like this one) shares some of properties of the process of writing a program in a compiled language. Firstly, a source (.tex) document must be written. and is then parsed by a compiler, producing an object (.dvi) file if syntactically correct. The document may contain other embedded objects or references to them. In other words, a document can contain either statically or dynamically linked elements - just like a software program.

The process of authoring and viewing documents in formats such as HTML and variants of XML can be better compared to scripting environments (e.g. Perl, Unix shell scripting) where no explicit object file is created and lexical, syntactic and semantic analysis takes place at run-time, every time the "program" is executed. In this case a 'transient ' artefact may be produced.

3.1.2 Compilation Within Software Systems

In a "traditional" software system, the process of compilation occurs in three stages discussed by [Bennett1996]:

1. **Lexical Analysis** determines whether correct the 'words' have been used. In other words, whether the language used is correct.
2. **Syntactical Analysis** determines whether words are arranged in a way consistent within some prescribed grammar. For a piece of software source code this prescribed grammar may be in form of a language specification or of a DTD or schema for a document.
3. **Semantic Analysis** attempts to produce meaningful constructions from syntactically analysed structures produced in previous stage. There is no guarantee that a pass on syntactical analysis will produce code that will

satisfy a semantic analysis, in much same way as a grammatically sound piece of prose may make no sense.

3.1.3 Compilation Within Document Systems

Bryan makes a comparison between document and programming language processors. Using Bryan's analogy [Bryan1997a], an SGML parser can be seen to perform the functions of a lexical, syntactical and semantic analyzer with respect to SGML documents. This could be treated like compilation or interpretation (although Barron does make a distinction between the two [Barron2000a]) in the document context. The treatment of errors in this process is essesntially an implemenation issue. Commonly, with HTML browsers, a user agent will attempt to display a presented document as best it can with no concept of a "compile time" error to halt execution. Malformed documents may not display as the author intended or indeed may produce no output at all, in effect generating sometimes fatal run-time errors.

A document parser is therefore comparable to a compiler (in the case of TEX because it produces object files) or an interpreter, as is the case with HTML in a web browser.

3.2 Links Within Software & Hypermedia

The presence of the *link*, in its various forms, is the characteristic that defines the difference between "normal" and hypermedia documents. The link can be considered in a number of forms, from GOSUB or GOTO constructions within programming languages to acting like a `fork()` (as is used within Unix programming environments).

3.2.1 Classification of Links

Sucha s classification does not fully address the nature of links, unfortunately, as some kinds of link need not be followed at all. A link can also be seen to behave like a dependency, on one hand like the HTML LINK or BASE elements in HTML) or else like A or AREA elements. The resulting linkages fall into the following categories:

1. **Dynamic**
   Items such as the IMG element in HTML or the use of LINK to retrieve externally linked stylesheets are able to be classified as dynamic linkages. The resource is requested and (possibly) retrieved at run-time. They are held separately from the calling node until run-time.

2. **Static**
   Static linkages occur when external elements are linked *before* before run-time The principle mechanism for this is the WWW context is the use of server-parsed documents to enable server-side inclusion. This mechanism effectively acts as a static linker.

3. **Strong**
   Links of this type (such as LINK,BASE or SCRIPT elements) are *structural* and describe externally linked objects to the node. Such links are ususally used to contextualise and orientate links and resources relative to the chosen node (BASE) or to include embedded or external content (e.g. LINK, used mainly with stylesheets or SCRIPT, which can utilise externally stored script content). Items such as IMG elements can also be placed within this category as the ability of a user agent about whether to render externally linked items is one of user agent configuration (usually an implementation issue). This type of link is not limited to hypermedia, however, and occurs in other mark-ups and programming languages (such as the C preprocessor).

4. **Weak**
   These links are *elective*, such as A HREF (typical links) or AREA HREF (used in image maps) in as much as users can choose whether these links are followed at the time of viewing. The items to which they point are external to the system at run time. This link type is indicative of hypermedia.

In this case, *run-time* is defined as the time when the resource is rendered by a user agent. The first two items have a temporal dependency - the time of linkage is important. The second two items are not temporally dependent but describe the electivity of the

Unlike conventional programming it is quite possible for a statically linked entity to contain dynamic references to others.

It is possible to argue that even the so-called strong links have an element of electivity. An end-user can choose to disable functions of the client to disable these features. However, such functionality is not within the control or scope of the document, merely an implementation issue connected with the user agent and its configuration.

Warren proposes that Lehamn's laws of software evolution miss an important category of maintenance, that of *speculative maintenance*. However, he does not specify how this property manifests itself in a software context. In hypermedia systems, URLs, being pointers can sometimes "break". The prevalence of the HTTP 404 response code demonstrates this. Warren proposes that speculative maintenance can be perfomed on web systems by link checking.

This problem also exists for software systems. Consider a software system that makes use of dynamically linked libraries (i.e. the library code is located and executed at run-time). In the C language, for example, the library code would be referenced by using a function prototype. The calling code would then look in a known list of locations until it finds a match. This match order performed determined by environment variables in different environments (such as PATH in DOS or LD_PATH in some variants of Unix). If a linked library is missing or damaged then execution will be impaired or perhaps even impossible. This is a very similar situation to that which one finds in web based systems. Tools do now exist within operationg systems to perform a relatively limited form of maintenance of this type.

It is possible to consider external function prototypes and URLs to be equivalent, as they both identify named external resources. Internal anchors are a special case of this, in essence being recursive calls to the same resource, merely choosing a separate entry point.

3.3 Document Validity Applied to Software

In Section 2.3, the concept of validity and well-formedness of documents was briefly discussed. The concept of well formedness is slightly more difficult to visualise within a traditional software context. (Even extending a language by use of functions adds the function to the lexicon thus dynamically extending the DTD). There is a parallel however: undefined variables within programs could mean, for instance, that such programs could be classified as well-formed but not valid. This idea is demonstrated briefly in figures 3.3 and 3.3

```
public class Test
{
        int first=1;
        int second=2;

        int theAnswer = first + second + third;

        System.out.print("The answer is: " + theAnswer + "\n");
}
```

Figure 4: A well formed but invalid Java program

The Java and Perl programs shown in Figures 3.3 and 3.3 are similar but have very different outcomes. The former will not compile but the second *will* excute, although possibly with unpredictable consequences. Although neither of these programs is valid (undefined variables exist in both) they are both syntactically correct. In other words both programs are not valid but they are *well formed*. The only difference is one of implementation, the treatment of errors in semantic

analyis. Such a similarity constitutes another shared property between software and documents.

```
#!/usr/bin/perl

$first=1;
$second=2;

$theAnswer = $first + $second + $third;
print "The answer is: $theAnswer\n";
```

Figure 5: A well formed but invalid Perl script

Software programming languages do, in fact, conform to DTDs of a sort. However, from the programmer's point of view they are largely notional. It is interesting to note the structure of the definitions of languages such as C, written using the Backus-Naur Form (BNF)[Crocker1997] as a comparison in this context.

Such similarities can also extend to looking for deeper patterns within software and document strucure. One possible way of doing this is by using a *Pattern Language*. The concept of a pattern language was first discussed by the architect Christopher Alexander [Alexander1977]. Its application to software systems has been dicussed in many quarters, most notably by Richard Gabriel [Gabriel1996a] and has also been dicussed in relation to mark-up languages, although not in the depth to which Gabriel has done. An example of the discussion of an HTML pattern language is by Orenstein [Orenstein1996].

Pattern Languages attempt to look for underlying abstractions in "concrete" systems (i.e. ones that have been built) in order to understand deeper structure. Gabriel has attempted to apply Alexander's work in a software context, most specifically in the area of functional programming.

Gabriel also describes a property of well-engineered code that also applies to hypermedia systems: habitability [Gabriel1996b]. This is an extension of usability in that it applies to *all* who use such a system - including those who create and develop it. Extensibility and flexibility of systems are hugely important in the context of hypermedia systems. The principle that an environment should also be mutable and extensible for its makers is another key principle that should underpin the development of high quality web environments. It is one that is frequently forgetten.

3.4 Document and Software Systems

Document and software systems are composites, made up of many components of a number of different types, from textually based mark-up to multimedia

and even programmatic content, whether this is scripted or otherwise. The heterogeneous nature of hypermedia systems introduces complications into the management and construction of such as system.

In traditional document media the relationship between items is essentially linear. Cross-referencing is possible, however, using items such as footnotes or indices of various types.

However, in hypermedia systems, the relationships between nodes can be far more complex, These relationships range from the linear or hierarchical structure to fully non-linear webs. These structures and the design strategies underpinning them are discussed in more detail in the Yale/CAIM Style Guide [Yale1999]. Design guidelines of this nature are user-centred in that the major design consideration is how the user will navigate thorough the information space. Little consideration appears to be given about the mapping between this conceptual map and its physical implementation - a major issue to developers in terms of maintenance and performace.

Walker [Walker1997] also identifies parallels in the design of software systems and prose authoring. These similarities extend to the task of creating web based content as opposed to the structure and framework in which it is contained. The steps he defines are roughly the following:

A common thread throughout these systems is that they are all being considered as systems of objects. This appears to suggest that methodologies based on the object-oriented paradigm will be useful when builking them.

4 Object-Oriented Hypermedia Document Systems

4.1 Properties of Objects

As one looks into a hypermedia system containing nodes connected by the link dependencies discussed earlier, it is very easy to envisage such a system as a collection of inter-related objects in a similar way as one would consider any other object-oriented space in the process of software engineering. rather like in figure 4.1.

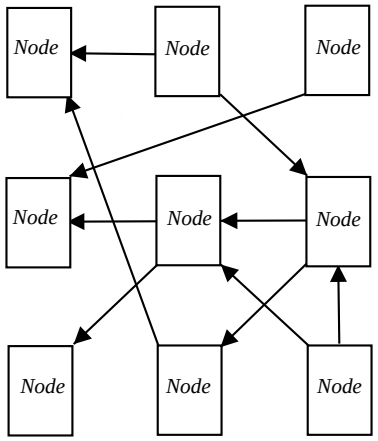

Figure 6: Hypermedia Nodes as Objects

In order to describe an object oriented view of hypertext systems it is perhaps useful to reiterate some of the important properties of software objects:

- **Encapsulation** describes the containment of the object. An object should describe itself fully, including all of its attributes and the methods it can implement.
- **Abstraction** describes how an object should show itself to the outside world. Any ways of communication with an object should be consistent, well-defined and ideally independent of the underlying implementation.
- **Inheritance (and Polymorphism)** describes the logical structure of an object hierarchy. An object can be classified in terms of other types, allowing for increasingly more specialised types to be defined. This mechanism has great power due to its extensibility.

- **Message Passing** occurs when two objects communicate with each other in some way. For a peice of OO software to function there *must* be some degree of inter-object communication taking place.

In a hypermedia system, a degree of encapsulation clearly exists - each node exhibits an element of self-containment. (notwithstanding the links or dependencies it has on others). Each node contains attributes (content) and methods (links) which provide communication with other objects, however simplistic that may be. The inheritance mechanism is a little less easy to describe but the use of tempate-based nodes and the Cascading stylesheet system can be used as examples.

Further, a node presents a consistent interface or API to the world via its node URI and any references it contains. This forms the basis of message passing between nodes.

4.2 Hypermedia Design Methodologies

The object structure also exists more generally in other document systems. Samtinger, when talking about the creation and maintenance of documentation systems touches on the idea of object relations between documentation for specific classes [Sametinger1994]. A similar idea has also been discussed by Nelson as part of the Xanadu project as "transclusion" [Nelson1965], being able to reuse and reference parts of doucments as individual components which could be aggregated into a greater whole. This would involve documents that could be broken down and stored separately, prior to recombination if required. This is becoming more practicable now, due to use use of XML and transform systems to change mark-ups form one form into another in a way that can be automated. The automation of such procedures is an important part of the World Wide Web Consortium's vision of the "Semantic Web" [W3C2001b].

The objective approach to deisgn has been considered in hypermedia, by people such as Schwabe & Rossi in the OOHDM system [Schwabe1995] and in the RMM system, developed by Isakowitz et. al. [Isakowitz1995].

Isakowitz states that RMM is best suited to systems were regular structure exists, such as catalogues and systems where front end data is assembled from relational databases. Many large web projects are constructed for these (or similar) purposes so it seems that useful crossover exists.

4.2.1 OOHDM

OOHDM is essentially a four step process consisting of the following stages:

1. **Conceptual design**, where the application domain is mapped out. This can be done with a standard OO design technique such as UML.
2. **Navigational Design**, where the navigational structure is planned, allowing for differing user profiles and tasks.
3. **Abstract Interface Design**, where metaphors are chosen for navigation and layouts are considered.
4. **Implementation**, where testing and fitness for purpose are evaluated.

4.2.2 RMM

The stages involved in the RMM design process are broadly similar:

1. **ER Design** describes the "Information domain" or the conceptual space. This technique was chosen because it was well-known in conventional data analysis.
2. **Slice Design** defines how information is shown to users and how they access it. The closest analogue to this in more usual OO systems is defining the encapsulation of objects.
3. **Navigational Design**, being the conceptual linkages between nodes and the possible paths thorugh the document system.
4. **Conversion Protocol Design**, which specifies the implementation of the elements so far defined. In an XML based system this could involve the design of a transformation of XML into some other form viewable by the end user (XML through XSL to HTML for instance).
5. **User Interface Design**, describing screen layouts and positioning of elements within nodes.
6. **Run time Behaviour**, where implementation decisions about navigation are decided, concering such things as backtracking and hisotry mechanisms.
7. **Construction and Testing**,which Isakowtz compares directly to traditional software projects.

4.3 Stages of Design

Like software design methods, both systems divide design into three main stages; *conceptual design*, *specification* and *implementation* although the degree to which each are done varies. RMM places more emphasis upon user-centric behaviour,

hence the explicit definition of both construction and run-time testing stages. Isakowitz states that the final four RMM stages are not concerned with access mechanisms. They are discussed in the final two stages of the OOHDM process and some correspondence does exist between these phases and the later phases of RMM.

A common technique for many designers is to construct an approximation of the first two steps using some kind of "site plan". Such a plan can also be used to describe a rudimentary navigation scheme through a site. Tools of this kind are available in software such as Microsoft's FrontPage although their usefulness is questionable in large-scale production design scenarios.

As Lowe remarks, the principal focus in such methodologies is to model information and navigation [Lowe1999a]. These are issues pertinient to the user's experience but not necessarily to other inhabitants of the system (i.e. its developers and maintainers). This is an interesting parallel to the situation in software engineering in the 1970s where structured design was starting to be considered but what analysis techniques existed were little used. The issue is further complicated by the fact that, at present, few hypermedia analysis methodologies exist, and where they do, they are relativley immature [Lee1998]

Unlike more traditional software development,the emphasis on current hypermedia development is primarily user focused. Both of the methodologies described above do not discuss the build quality of systems in great detail. For large web-based systems this is a cause for serious concern. Soon, beacuse of the burgeoning size of such projects, web systems may themselves enter a time of crisis similar to that described for software by Djikstra. [Djikstra1972].

5 Conclusions

The basic determinants of a mark-up document are:

- **Is there a grammar?**
  The grammar defines allowable elements within a document space and their form.
- **Is there a structure?**
  The structure defines how these elements are related to each other.
- **Is it readable?**
  Readability underlines how generalised the mark-up is and how it well can be processed by both machines and humans. Items of specific mark-up may count as documents but are generally only readable by machines.

The principal item that documents require in order to be considered as such is a grammar. This princlple extends beyond the considered range of documents all the way to those written in natural languages. Furthermore, such a grammar is usually defined in the form of either a Document Type Defintion (DTD) or, in the case of XML documents, as a schema. These items are defined in ISO8879. However, such a DTD does not have to explicitly stated within the context of the document.

Document grammars can be compared very closely with programming language specifications, particularly those defined using the Backus-Naur Form. Both machine documents and programimng languages require parsing to allow output or transformation into another state, usually visual for documents and in he form of an executable artefact for program code.

The products of both program and document source are produced by a parsing process of some type. Discrete elements within a document can be parsed into a system of related objects and these objects have relationships with each other. This is very similar to describing the the way clauses are constructed in a declarative progamming language.

The relationship between documents and software becomes more complex still when considering hypermedia, where the linkage between entities in the execution space resembles the linkage of elements in more conventional software systems. Unlike more usual software systems, however, the relationship between entities in a hypermedia system is far more mutable and can change in real time.

References

[Alexander1977] Alexander C, *A Pattern Language*, Oxford University Press, 1977. ISBN 0-19-501-919-9.

[Barron2000a] Barron D, *The World of Scripting Languages*, Wiley, 2000 pp5-6

[Bennett1996] Bennett JP, *Introduction To Compiling Techniques A First Course Using ANSI C, Lexx and Yacc*, Second Ed, McGraw Hill,1996

[Berners-Lee2000] Berners-Lee T, *Weaving The Web*, TEXERE, 2000, pp38-46

[Botafogo1992] Botafogo RA, Rivlin E, Shneiderman B, *Structural Analysis of Hypertexts: Identfying Hierarchies and Useful Metrics*, ACM Transactions on Information Systems 10(2), April 1992, pp142-180

[Bray1996] Bray, T. *Measuring the Web*, in Proc. 5th WWW Conference, Paris, France., 1996

[Bryan1997a] Bryan M, *SGML and HTML Explained (Second Edition)*, Addison-Wesley, 1997, p32

[Bush1945] Bush V, *As We May Think*, The Atlantic Monthly, July 1945

[Conkin1987] Conklin, J *Hypertext: An Introduction and Survey*, IEEE Computer, 20(9), September, 1987, pp17-41

[Crocker1997] Crocker, D ed., *RFC2234 - Augmented BNF for Syntax Specifications: ABNF*, Avaialble online at: http://www.ietf.org/rfc/rfc2234.txt?number=2234 (Accessed: June 22, 2001)

[Djikstra1972] Djikstra EW, *The Humble Programmer (1972 ACM Turing Award Lecture)*, CACM 15(7) July 1972 p859-866

[Eisenstadt1988] Eisenstadt M and Brayshaw M, *The Transparent Prolog Machine (TPM): An Execution Model and Graphical Debugger for Logic Programming*. Journal of Logic Programming, 5(4), 1988.

[Englebart1963] Englebart DC, *A Conceptual Framework for the Augmentation of Man's Intellect*, In Vistas in Information Handling,Spartan Books, pp1-29, 1963.

[Englebart1995] Englebart, DC, *Boosting Our Collective IQ - Selected Readings*, Bootstrap Institute/BLT Press, 1995

[Gabriel1996a] Gabriel RP, *Patterns of Software*, Oxford University Press, 1996 ISBN 0-19-510269-X

[Gabriel1996b] *ibid.* pp9-16

[Goldfarb1970] Goldfarb CF, Mosher EJ, Peterson TI, *An Online System for Integrated Text Processing*, Proc, American Society for Information Science,July 1970, pp 147-150

[Isakowitz1995] Isakowitz T, Stohr EA, Balasubramanian P *RMM: A Methodology for Structured Hypermedia Design*, CACM 38(8) August 1995, pp34-44

[ISO8879] *SGML (ISO 8879:1986)*, International Standards Organisation, 1986

[ISO9541] *Font Information Exchange (ISO/IEC9541-1:1991)*, International Standards Organisation, 1991

[Jelliffe1998] Jelliffe R, *The XML and SGML Cookbook: Recipes For Structured Information*, Prentice Hall 1998

[Knuth1984] Knuth DE, *The TEXBook*, Addison-Wesley, 1984

[Lamport1986] Lamport L, *LATEX: A Document Preparation System*, Addison-Wesley, 1986

[LaTeX2000] *The LaTEXProject Home Page*, Available on-line at: http://www.latex-project.org/,2000
(Last modification date August 10, 2000, Accessed May 15, 2001)

[Lee1998] Lee HS, Suh WJ, *A Workflow Based Hypermedia Developing Methodology*, Graduate School of Management, Korean Advanced Institute of Science and Technology, Available online at: http://kms.kaist.ac.kr/re_center/paper/1998-200.pdf (Accessed: June 25, 2001)

[Lehman1985] Lehman MM, *Program Evolution*, Academic Press, 1985

[Lehman1999] Lehman MM, *Software System Maintenance and Evolution in an Era of Reuse, COTS and Component Based Systems*, IEEE International Conference on Software Maintenance, Oxford, England, August 30 - September 3, 1999,

[Lowe1999a] *Web Development Methodologies: Help Or Hindrance?* WebNet Journal 1(3), July-September 1999, pp9-10

[Lowe1999b] Lowe D, Hall W, *Hypermedia And The Web: An Engineering Approach*, Wiley, 1999, p337

[Nelson1990] Nelson T, *Literary Machines*, Mindful Press, 1990

[Nelson1965] Nelson T, *A File Structure for the Complex, the Changing, and the Indeterminate*, 20th National Conference, ACM, 1965

[Orenstein1996] Orenstein R, *An HTML 2.0 Pattern Language*, Available online at: http://www.anamorph.com/docs/patterns/default.html (Last modified: May 16, 1996, Accessed: May 21 2001)

[Sametinger1994] Sametinger J, *Object-Oriented Documentation*, The Journal of Computer Documentation, 18(1), January 1994, pp3-14

[Schwabe1995] Schwabe D, Rossi G, *The Object Oriented Hypermedia Design Model*, CACM 38(8) August 1995, pp45-46

[W3C2001a] Le Hégaret P, Wood L eds., *Document Object Model (DOM)* Available on-line at: http://www.w3c.org/DOM/ (Viewed: June 25, 2001. Last modified: June 21, 2001)

[W3C2001b] Miller E et al. eds. *Semantic Web*, Avaiable on-line at: http://www.w3c.org/2001/sw/ (Accessed: June 21, 2001. Last modified: April 17, 2001)

[Walker1997] Walker R, *Information Engineering on the World-Wide Web: Drawing Analogies with Software Engineering*, In Proceedings of the Eighth Western Computer Graphics Symposium (SKIGRAPH '97; Whistler, BC, Canada; 6–9 April 1997), 1997, pp. 97–107 Also avaiable on-line at: http://www.cs.ubc.ca/ walker/papers/walker.1997a.pdf (Accessed: June 6, 2001)

[Warren2001] Warren PJ, *PhD thesis - unpublished at time of writing.*, University of Durham, 2001

[Weiss1994] *Isomorphism Between OOP and Documentation: Reflections on Samtinger's Object Oriented Documentation"* The Journal of Computer Documentation, 18(2), May 1994, pp8-12

[XML2001] *XML Schema Part 0: Primer*,David C. Fallside ed. Available on-line at: http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/ (Accessed: May 17, 2001)

[Yale1999] Lynch PJ, Horton S *Web Style Guide : Basic Design Principles for Creating Web Sites*, Yale University Press, Also available online at:

http://info.med.yale.edu/caim/manual/contents.html
(Accessed: June 24, 2001)