ELECTRONIC PUBLISHING, VOL. 7(3), 163–178 (SEPTEMBER 1994)

Why use HyTime?

L. A. CARR, D. W. BARRON, H. C. DAVIS AND W. HALL

Department of Electronics & Computer Science
University of Southampton
Southampton SO9 5NH
UK

SUMMARY

The Hypermedia/Time-based Structuring Language (HyTime) is a recently adopted International Standard (ISO/IEC 10744:1992). The paper presents the need and potential for HyTime, provides a brief explanation of its various facilities and shows how it may be applied to good effect in various situations, with particular reference to hypertext interchange from Microcosm (an open hypertext system). It then goes on to explore several alternatives to HyTime and compare their relative strengths and weaknesses.

1 INTRODUCTION

The aim of the Hypermedia/Time-based Structuring Language, or HyTime [1,2] is to preserve information about the scheduling and interconnection of related components of a hypermedia document (e.g. audio, music score and libretto in a CD-ROM version of an opera) that would otherwise be embedded inside application-specific 'scripts'. It grew out of the work of the ANSI X3V1.8M committee on a Standard Music Description Language, and became a separate standards activity in 1989, achieving draft status in 1991 and final International Standard status in April 1992. HyTime is a standard for expressing document architectures in SGML and so to understand HyTime it is necessary to understand the particular importance of generic markup as embodied in SGML.

2 MARKUP

Traditionally, markup was the process of marking a manuscript with instructions to the compositor for rendering the manuscript in print. More recently the manuscript has become a computer file and the compositor a computer program; markup now comprises the codes inserted into the text to control the composition program. These codes may be explicitly inserted by the author (in the case of typesetting systems like LaTeXor troff) or added 'behind the scenes' as a consequence of the author choosing a particular style from a menu (in word processor systems like Microsoft Word or WordPerfect). These codes control printed (physical) attributes of the document, such as the fonts and spaces used to render the text (Figure 1), and mimic the pre-existing technology used by printers, so the models which both the above kinds of programs manipulate is that of a book, magazine, memo, or letter—i.e. any printed item.

Because of the tedious and repetitive nature of this 'physically oriented' low-level typographic manipulation, markup languages adopt procedural abstractions (macros) (Figure 2) which mirror higher-level physical document constructs like display paragraphs, hanging indents, bulleted lists and headings, and reflect a document's logical or abstract composition, such as its construction from chapters, sections and subsections, figures and tables.

CCC 0894–3982/94/030163–16
Received 20 June 1993
Revised 21 January 1994

```
.ce 1
.ft B
A title, a title, my kingdom for a title
.ft R
.sp 0.5i
In this chapter we look at the possible
```

Figure 1. Nroff physical markup for a section heading

```
.H 1 "A title, a title, my kingdom for a title
In this chapter we look at the possible
```

Figure 2. Nroff mm logical markup for a section heading

Emphasized text is no longer marked up with prescriptive physical commands to "change the font to italic", but with an abstract declaration that "the following text is emphasized". It is now the responsibility of the composition program to know how to suitably render emphasized text—in other words its role has expanded from dealing with page-imaging semantics to dealing with document semantics. The advantage of this style of markup is that the author can concentrate on expressing ideas within an appropriate logical framework without worrying about issues of presentation that the compositor should deal with. Markup systems which adhere to this philosophy (troff+mm, LaTeX, GML) emphasize the logical nature of their markup, especially the facilities for expressing the document's overall hierarchical structure. However, a closer inspection reveals that such markup is still implicitly tied to describing the physical layout of a printed document. In fact mm and LaTeX markup for a 'section' or 'chapter' is defined in terms of lower-level primitives for changing fonts and leaving vertical space, and so is still a physically oriented markup, rather than a truly logical one. Document structures like 'sections' are catered for in name only; in fact one is really marking up the section heading alone. Another apparently separate component of the logical document structure, the footnote, only has any meaning in a paginated environment and may need to be re-interpreted as a marginal paragraph or an endnote in an on-line text presentation system. Logical markup has been taken to its (logical) extreme by SGML, the Standard Generalized Markup Language [3,4], which defines a regime for document markup without any predefined processing operations or any built-in document structure semantics. This lack of built-in semantics is both SGML's greatest strength and greatest weakness. The strength is that SGML forces abstraction from the eventual document delivery medium and allows a content-based approach, imposing a cognitive discipline that brings benefits beyond the immediate publishing requirements. The weakness is that the immediate requirement is usually for printing! SGML specifies each document architecture with a DTD (Document Type Definition) defining the hierarchy of structures which may compose the document. This architecture may be used by an interactive document editor to check the structure of the document being created, or by a document formatter to process the entire document. There is a strict syntax associated with the architecture which may be understood and verified by any SGML-compliant application, but each application is responsible for interpreting the 'meaning' of the document structure, according to its requirements.

```
<entry>
<biographand><name>John Smith</></>
<dob><day>12<month>June<year>1934</dob>
<dod><day>1<month>Feb<year>1987</dob>
John was born in <birthplace><place>Edinburgh</birthplace> and
studied <subject>English</> at <education><place>Southampton</>
University</>, graduating in
<graduation><date><yr>1956</graduation>. He married
<spouse><name>Emma Jones</name></spouse> in 1962 and became
<profession>MP</> for Southampton in 1975 until his death.
</entry>
```

Figure 3. SGML markup for an entry in a biographical dictionary

For example, a biographical dictionary may be marked up as in Figure 3. To produce a printed document it may only be necessary to specially highlight the start of the entry, the name and dates of birth and death of the individual. The other tags may be completely ignored during formatting, with the text set as if they were not there. However, when forming a biographical database from the same document it may be deemed important to identify all the information marked above so that the database can be used to determine everyone who was educated at a particular university. Furthermore, in an on-line hypertext version, any of the names that appear in the body of an entry may require a button to appear over them if that person is further described in their own entry. Without the extra markup it would be impossible to pick out these details that make the data useful for many purposes apart from printing. This is especially pertinent since many published works (particularly encyclopaedias and dictionaries) are becoming available in printed and electronic versions. (See [5] for a description of the construction of an SGML DTD for coding literary texts.)

In a modern hypermedia document environment, the models manipulated by the computer programs are no longer those of traditional printing technology, with common interfaces and operational semantics. A document may consist of a collection of video sequences, audio clips and computer animations as well as text. Abstract but physically based markup can no longer be used to define ‘how to’ present each piece of information because there is as yet no standard practice to follow for presenting non-textual information. In any case instructions such as “leave 2 seconds of space and then show this video clip in the top-left corner of the screen with that text next to it” leave little room for true hypermedia which necessitates user-directed interaction.

Although physically oriented markup has a limited role in a truly multimedia environment, it has a crucial role to play in describing each different component of the document, describing its representation and its purpose (especially for non-textual information). The markup can therefore be used in two ways

1. To encode or represent the various document objects themselves.
2. To describe meta-information about the objects or their intended use.

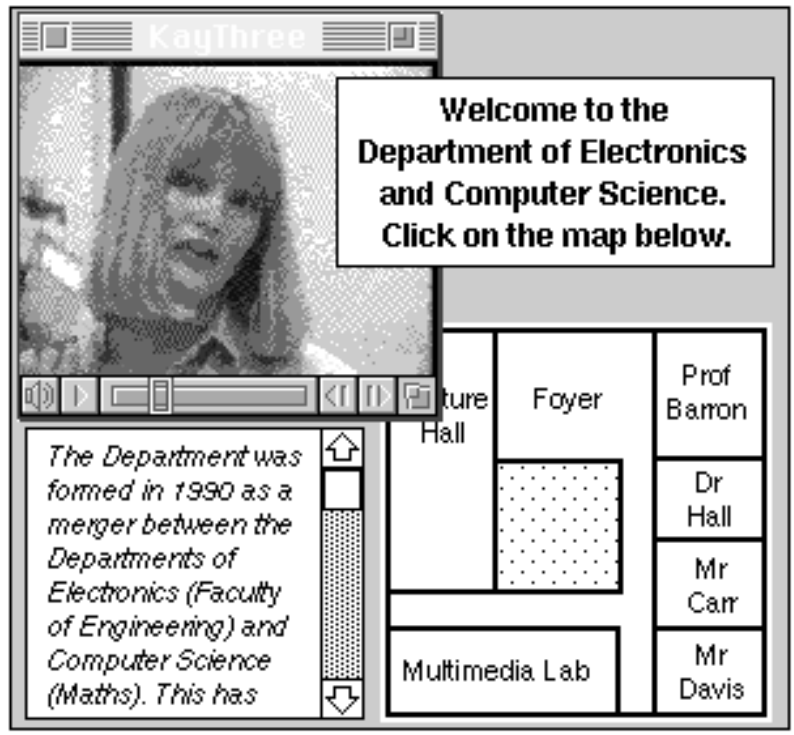

Figure 4. A multimedia document

2.1 Using markup to express document objects

For example, Figure 4 demonstrates a document which consists of some text, a diagram and a video sequence. Figure 5 shows how it might be coded according to an SGML DTD, with the document structure used as a container to hold the various encodings of the document media (text, picture and video). The contents of the text objects are also coded according to the DTD, although the diagram object is coded according to some external scheme and the video object as a mixture of SGML markup and external scheme. All the document objects (text, diagram and video) are coded using SGML markup and also have special tags which give information about the objects.

2.2 Expressing inter-object relationships

Having established that we can use a markup environment based on an SGML DTD to code the basic objects and embedded data structures that make up a multimedia document, we turn our attention to the interaction between the various objects within a document, and to the interaction of a reader with those objects. Multimedia information comes in many guises: movies can be embedded in a word-processed office memo, maintenance manuals can use video sequences to demonstrate exactly how to replace a faulty component, and a music CD-ROM could contain the orchestral score and libretto for a musical as well as the choreography and lighting instructions. The first example can be easily accomplished on

```
<mmdoc>
<element type=text>Welcome to the Department of
     Electronics and Computer Science. <p>Click on the map
     below.</>
<element type=diag size=7x8 rendering=winmetafile>
     AA145367382A5</>
<element type=text>The Department was formed in 1990 as
     a merger between the Departments of Electronics
     (Faculty of Engineering) and Computer Science
     (Maths). This has resulted in a successful
     partnership of hardware and software expertise.</>
<element type=video size=3x3 rendering=frames>
     <frame num=1 timecode=002701>AA145367382A5...
     <frame num=2 timecode=002702>AA145367382A5...
     <frame num=3 timecode=002703>AA145367382A5...
     </>
</mmdoc>
```

Figure 5. Representing the document with SGML

a multimedia PC and is often seen in product advertisements but has dubious utility. The second example requires more resources to create and a good deal of information design, but is well within the technical grasp of a publisher to produce on CD-ROM. The third example is much more problematic: a number of different media sources are related to each other in time. The connections between the objects in the first example is minimal (a video can have a rectangular bounding box for placement within the document, but will usually only be activated when the reader invokes some 'play' command). There is more complexity of relationship between the objects in the second example, but this can probably be represented as a tree of objects with explicit relationships between any text object and the other text and video objects to which it refers. The third example, however, is composed not of independent information units but of continuous streams of information which are closely tied at all points in each information source. HyTime was particularly designed for expressing these relationships between different information objects. Although SGML or any similar encoding scheme may provide rudimentary cross-referencing facilities for implementing the connections in the first two examples, it is the complex situations which the third example demonstrates where HyTime comes into its own.

3 MISCONCEPTIONS ABOUT HYTIME

3.1 HyTime is not an application program

HyTime is a methodology for describing document features and the relationships between different parts of documents, but it does not prescribe the *meaning* of these features or relationships. It uses terms like 'hyperlinking' without defining what happens when a link is followed, or even how a link is activated. HyTime is not a system that can be executed to display multimedia documents and jump between document objects using hyperlinks. An

application would need HyTime added-value to interpret a HyTime-compliant document and render it for display. It is in fact anticipated that the main use of HyTime will be for encoding documents for interchange between various proprietary systems, and although the HyTime standard provides various facilities to speed up native rendering of a HyTime document, HyTime is not necessarily the most suitable format for coding multimedia material.

3.2 HyTime is not a document architecture

Although HyTime is used to mark up hypermedia documents in conjunction with SGML it is not a single document architecture (i.e. it is not a DTD). Early versions of the draft standard did in fact define a HyTime DTD, but this was abandoned as being too restrictive. Instead HyTime is often referred to as a *meta-DTD* since it provides a set of standard components (or architectural forms) which can be used to construct document architectures. As such, HyTime defines a (very large) family of document architectures, and rules for constructing their DTDs.

4 HYTIME DEFINED

HyTime is neither an application that implements hypertext or multimedia facilities nor a single document architecture that requires document objects to be expressed in a particular fashion. Instead it provides a set of standard abstract facilities that can be built into any document architecture that is expressed in SGML. HyTime is both abstract and specific: it provides abstractions of facilities that are useful in building document architectures, but is very specific about how these abstract facilities must be coded. All HyTime facilities are expressed in SGML and use SGML as a starting point for document representation; in fact the HyTime standard includes many miscellaneous features such as 'glossaries' which are intended to make SGML authoring easier. HyTime constructs are expressed as combinations of SGML elements and attributes which have to be interpreted by a HyTime engine subsequent to their parsing as SGML elements. Although such a HyTime engine may appear to play the role of a postprocessor for SGML files, a more cooperative role is needed, since the SGML parser may be required to provide access to any objects in external entities which the HyTime engine needs to interpret. In fact, both HyTime and SGML processing engines may be components of a larger document environment.

4.1 HyTime extends SGML

HyTime is primarily concerned with documenting the relationships between different parts of documents. SGML already has facilities for making references between elements of a document: elements may be *labelled* with an *id* attribute and then referred to by that label in another element's *idref* attribute. This facility can be used to implement cross-references, hypertext jumps, object class systems, style sheets or many other constructs; however, it is quite restrictive for a number of reasons. Firstly, only whole elements may be addressed, and so document objects are rigidly defined with quite a coarse granularity—it is not possible to quote a reference to a relevant fragment of a paragraph. Secondly, every element which is to be addressed must be explicitly labelled (conversely, only elements which the author has bothered to label may be addressed). This is not a worrying restriction to the originator

of a document, who is free to make whatever labelling additions may be desired, but an author who is trying to 'link in' to an existing work (a standard reference resource such as a dictionary, or a seminal academic paper) may have great problems expressing an arbitrary link using just an idref. The third problem with SGML idrefs is that they may only refer to labels within the *same* document. This makes linking to external reference works impossible without including them in their entirety through an entity reference. Thus in order to allow flexible linking of documents, one of HyTime's major functions is to extend SGML's object addressing model.

4.2 What HyTime provides

HyTime enhances SGML with extra capabilities with regard to the representation and addressing of objects, so that they do not have to correspond one-to-one with labelled elements. Object addresses may be constructed from a combination of sub-addressing techniques, starting from a well-known object, such as an SGML named external entity or a previously labelled SGML element (or HyTime object). From such a starting place it is possible to repeatedly narrow down the address by taking a linear offset from one of the ends of the object, or by specifying a hierarchical position within a tree-structured object. Object addresses (or a part of an object's address) may also be specified as the result of a query on the document (its structure or data content). This flexible addressing mechanism may be used, for example, to allow a literature student to refer to a specific word or phrase buried inside a paragraph of a read-only document that is not even marked up in SGML. HyTime also provides some facilities for describing hypertext links. These are nothing that plain SGML could not do, given the extra addressing capabilities of HyTime, but it is very useful to have an agreed standard for representing marked up links. One of HyTime's most-publicized enhancements is the ability to represent complex temporal-based information, and a major part of it is to control the sequencing and coordination of the rendition of the various objects.

5 A BRIEF TOUR OF HYTIME

HyTime is a modular standard, with the document designer free to choose only those facilities which will be needed. The *base module* is always required and provides facilities using SGML constructs for object representation and addressing, as well as miscellaneous facilities for other HyTime modules. The *measurement module* defines the concept of addressing document objects according to a measurement along some abstract dimension (e.g. words 3 to 27 could be a measurement within a paragraph). Various standard units are defined for familiar temporal and spatial measurements. The *location address module* allows reference to be made to document objects which cannot be addressed with the normal SGML facilities of the base module: these objects can be referenced by name, position or query. A *location ladder* can be built up of gradually more and more specific location addresses (e.g. the draft chapter's fourth heading's third word's second letter). The *hyperlinks module* provides methods for representing link objects (based on the various object addressing and representation methods provided above) and the semantics associated with traversing the link. The *scheduling module* provides events which are objects positioned within a multidimensional coordinate space. The *rendition module* provides ways for describing the modifications that can be made to an object within an event and the ways that events can be projected from one coordinate system into another.

6 THE MANY FACES OF HYTIME

HyTime's facilities are all to do with locating and linking information. The location module enables arbitrary information to be addressed, the hyperlinks module allows this information to be associated with other information, the scheduling module allows different information objects to be coordinated in some way. Only the rendition module provides any rudimentary facilities for altering objects. Because of the very general nature of these facilities, HyTime has application in many different situations: here are some examples of its use in the fields of text processing, presentations and hypertext interchange.

6.1 Text processing

Text processing is mostly concerned with the production of printed documents and requires information to be moved (to produce footnotes), copied (to produce tables of contents) and collated (to produce indexes). HyTime is useful in this situation because it can describe the relationship between individual text items and the larger document structure. For example, in order to produce an index a list of terms has to be decided on, and then all the relevant occurrences of each term (or its synonyms) must be referenced in the text. Each index entry catalogues the relationship between its term and a number of occurrences in the document and can thus be modelled by a hyperlink (despite its name a hyperlink does not necessarily have anything to do with hypertext, only the connection of two document objects). A hyperlink encodes a connection between several document objects called 'anchors' of the link, and assigns a 'role' to each of the anchors. For an entry in an index there could be two anchors—the term to be indexed and the set of its occurrences within the document. Figure 6 shows such an index entry. It connects a 'term' element to an 'occurrences' element whose instantiations have ids 't1' and 'o1' respectively. Both elements are declared to occur inside the *indexentry* element. *Indexentry* is not part of HyTime, it is simply defined in the DTD (as shown in Figure 7) with HyTime standard attributes. It is the *HyTime* attribute that identifies the *indexentry* as being an example of an independent link (*ilink*) to the HyTime engine. The HyTime engine can then handle the value of the *linkends* attribute to find the various anchors for the text processing application to use. (More likely the value of the *anchroles* attribute would be fixed in the DTD and so not given in the document instance itself.) In text processing environments, index terms are frequently given special markup in the body of the text. If this is the case, HyTime may locate the term's use by referring to the markup's id. If this is not the case, or the indexer does not have write access to the documents text, then HyTime may locate the index entries by using a *dataloc* (data location) element. A dataloc element identifies an anonymous span of data within another named object (called the location source, or *locsrc*, perhaps an element with an id or a named entity) by giving an offset from one end of that object and an extent. For example, if this section (entitled 'Text processing') had been marked up with an id of *textp*, the following examples of a dataloc element could address the word 'production', either by counting characters or words from the start of the section. (The *dimlist* element treats its numbers as a measurement along an abstract dimension, in this case the data content of a section element.)

```
   <dataloc locsrc=textp quantum=str><dimlist>45 10</></dataloc>
or <dataloc locsrc=textp quantum=word><dimlist>8 1</></dataloc>
```

```
<indexentry anchrole="term occurrences" linkends="t1 o1">
        <term id=t1>multimedia
        <occurrences id=o1>oc1 oc2 oc3</>
</indexentry>
```

Figure 6. An entry in an index

```
<!ELEMENT indexentry - - (term, occurrences?)>
<!ATTLIST indexentry   HyTime NAME #FIXED ilink
                       anchrole NAMES #REQUIRED
                       linkends IDS #REQUIRED>
```

Figure 7. Defining an IndexEntry construct in the DTD

```
<!ATTLIST occurrences  HyTime NAME #FIXED nmlist
                       nametype NAME #FIXED element>
```

Figure 8. Defining an occurrences construct in the DTD

Since each term appears numerous times within the document the ‘occurrences’ anchor is a HyTime *multloc* or multiple location, which consists of a list of ids, each resolving eventually (perhaps indirectly through a dataloc) to a word in the document. By use of the HyTime-based indexentry document structure given above, we have enabled the document designer to express connections between a specific document object (here a piece of text) and numerous places in the document. This allows the index to refer not just to occurrences of a particular word, but to whole paragraphs of text, or pictures and diagrams. It is the responsibility of the index creator to decide how to represent each of these connections.

6.2 Presentations

Often there is a requirement to make a presentation based on the information drawn from a collection of books. This presentation not only imposes a temporal ordering on the information but also allocates a time-span to particular pieces of information, based not on the length of the content, but on its perceived importance. An educational course syllabus is an example of such a presentation. HyTime can be used to represent such a course syllabus by using a *finite coordinate system* (fcs) to represent a timeline, and then mapping each component of the course onto the appropriate position on that timeline.

Figures 9 and 10 show the definition and use of such a timeline. In Figure 10 we see that a semester contains a course schedule which contains a number of lectures, each of which contains a set of contents and refers to a duration for the lecture. The contents themselves are references to the contents of a textbook, perhaps indirectly through a dataloc. Figure 9 shows how this is defined using HyTime’s constructs. The semester is an example of a finite coordinate system whose axes are defined by a *timeaxis* structure (not shown here). In fact there is just one axis here (the time axis) which would be measured in ‘teaching blocks’ for convenience. The *courseschedules* which it contains are examples of HyTime’s

```
<!ELEMENT semester - - (courseschedule)+ >
<!ATTLIST semester
            HyTime NAME #FIXED fcs
            axisdefs NAME #FIXED timeaxis>
<!ELEMENT courseschedule - - (lecture)+ >
<!ATTLIST courseschedule
            HyTime NAME #FIXED evsched>
<!ELEMENT lecture - - (content)+ >
<!ATTLIST lecture
            HyTime NAME #FIXED event
            exspec IDREFS #REQUIRED>
<!ELEMENT duration - O (#PCDATA)
            -- LexModel(snzi, s+, snzi) -->
<!ATTLIST duration
            HyTime NAME #FIXED extlist
            id ID #REQUIRED>
<!ELEMENT content - O (#PCDATA)>
<!ATTLIST content
            HyTime NAME #FIXED nmlist>
```

Figure 9. Defining a timeline in a DTD

```
<semester><courseschedule>
        <lecture exspec=single>
                  <content>chap1</>
        <lecture exspec=dbl>
                  <content>chap3 chap4 sect6</>
        <lecture exspec=single2>
                  <content>chap2</>
</courseschedule></semester>
<duration id=single>26 1</>
<duration id=dbl>37 2</>
<duration id=single2>78 1</>
```

Figure 10. Using a timeline

event schedules. Each schedule contains many *events* (lectures in this example) which tie a document object (the content elements) to a position and extent in the coordinate system (place the content along the time axis). The purpose of the duration elements (HyTime *extlist*) is to specify the start and extent of the event in the units of the coordinate system. This example uses particularly opaque measurements, so to make it more useful to a human it would be better to project the events in this coordinate system onto a natural calendar by using the *event projector* facility of the rendition module.

6.3 Hypertext interchange

Exchanging documents between word processors is a common problem for which one solution lies in the manufacturers of each program making translators available for importing documents native to all the other (commercially successful) programs. An alternative

```
\DocID history.intro \Offset 246 \Selection Mihailovich
```

Figure 11. Microcosm address tuple

```
<nameloc id=histDoc>
        <namelist nametype="entity">history.intro</></>
<dataloc id=mihail quantum=str locsrc=histDoc>
        <dimspec>246 11</dimspec></>
```

Figure 12. Address tuple as a HyTime location ladder

approach is to define a common 'document interchange language' (such as Microsoft's Rich Text Format) for which each program only needs to provide an 'export' and 'import' facility. A similar problem exists for exchanging hypertext documents between hypertext systems. Microcosm [6] is an open hypermedia system developed at the University of Southampton. One of its chief features is that no information concerning links is held in documents; instead all link information is held in external linkbases which contain the required details about the source and destination anchors of the links. It comprises independent components (document viewers and link managers) which communicate by passing messages. Working in such an open environment means that the system response may be suboptimal and so hypertexts developed in Microcosm may be translated to a cut-down but optimized delivery environment (such as Microsoft Help). One of the major problems inherent in such a translation is that the linking facilities of the two systems may not directly map onto each other. The rich nature of HyTime's linking capabilities make it possible to translate hypertext semantics into a HyTime representation without loss of information and it is therefore useful to use HyTime to form an intermediate representation (a kind of 'Rich Hypertext Format') as a midway stage in mapping between two hypertext systems. The translation process then divides into a sub-process that converts a native Microcosm dataset into a HyTime-based representation, and then further translation process to convert (possibly a subset of) this HyTime representation into another hypermedia format [7].

The most common Microcosm addressing mechanism is the *(document id, offset, extent)* tuple. The Microcosm address specification tuple in Figure 11 references a string of (implicit length) eleven characters starting at character offset 246 of a document whose id is *history.intro*. It could be expressed as the two-stage HyTime location ladder in Figure 12, in which the first (nameloc) element associates an SGML id *histDoc* with the document, and the second (dataloc) element locates the string within the identified document. Any reference to the name *mihail* will now resolve to the requested object. HyTime links may have more than two anchors, and the document designer has to provide semantics for each of the anchors. By contrast, Microcosm links have only two anchors (*source* and *destination*), but a destination anchor may be composed of many documents objects (the equivalent of a HyTime multiple location). HyTime links can take two forms—*contextual* links, whose definitions appear at one of the sites of the link anchors (i.e. in context), and *independent* links, whose definitions are given at some other place in the hyperdocument. Microcosm links are always of the latter type, since link definitions are stored in separate linkbases, referring to their anchor positions through the addressing mechanisms above. A Microcosm linkbase can now be modelled as a collection of HyTime independent links:

```
<mcmlink anchrole="source destination"
     linkends="srcid dstid" endterms="linkdisp1 linkdisp2">
```

where the multiple destination may be specified as a simple list of destinations as follows:

```
<nameloc id="dstid"><namelist nametype=element>
             destid1 destid2 destid3</></>
```

This example is similar to the index example given previously, except that the information given by the *link endterms* is intended to specify how the link source and destination are to be portrayed—here the source is formatted as a button and provides a short preview of each component of the multiple destination. This is achieved using elements of the following form:

```
<displayinfo id="linkdisp1"> <anchorformat>button</></>
<displayinfo id="linkdisp2"> <anchorformat>normaltext</></>
```

which are referred to by references to their unique identifier (id) within the mcmlink element. A Microcosm link may completely specify its source anchor (in terms of *document*, *offset* and *content*) in which case it is known as a *specific* link. But by leaving the offset or document unspecified the content acts as a source anchor for this link anywhere that it appears in any document. This is a *generic* link which no longer contains explicit connections to a source document location. HyTime makes provision for locations to be specified as the result of a *query* performed on the content or structure of a document, defining a standard query notation (HyQ) for this purpose and it is possible to express the source locations of a generic link with such a query. This can be done by replacing the explicit dimension specification (dimspecs) in Figure 12 with an *axis marker query* which represents a matching operation against the required texts. Any query notation (e.g. regular expression searches) is allowed in this context. For specific links, the source specification *srcid* resolves (through a dataloc) to a single location. For generic links, srcid resolves to a multiple location through a query which returns a dataloc for each occurrence of a particular piece of text, where the query domain is either a single document (local link) or the entire hyperdocument (generic link).

7 ARGUMENTS FOR AND AGAINST HYTIME

“One man’s meat is another man’s poison” is certainly true in the world of electronic publishing. Religious wars are fought over the use of different word processors, and the features which endear HyTime to one community of users are likely to alienate a different community. Certainly HyTime is a standard which provides added value for SGML, and as such is likely to be adopted with some enthusiasm by users of SGML. However, both SGML and HyTime have a significant emphasis towards information interchange and therefore are frequently hidden from the end-user and visible only to application programs. Since HyTime and SGML are so closely related, any of the arguments brought to bear against SGML are likely to apply to HyTime. Barron [3] cites as one of the major obstacles to the take-up of SGML the need for changes in working practices and the development of new software. These still remain potent arguments against the use of SGML, though the number of popular commercial products which support it is slowly increasing. HyTime, as a very recent standard, is in a worse commercial position. However, SGML is making significant

inroads into major military and commercial documentation systems with concomitant changes in working practice, thus preparing the way for HyTime-based approaches to information handling and interchange. Another apparent weakness that is shared with SGML is its lack of inbuilt 'default' structure. A standard set of tags (such as the British Library Starter Set) are required to convey standard document semantics in SGML; standard document architectures incorporating hyperlinks and event schedules are similarly required for documents to be fully shared. One of the overwhelming arguments in favour of HyTime is the changing nature of publishing. When SGML was proposed as a standard it was becoming more commonplace for authors to exchange individual documents electronically and the requirement was for a common medium for expressing these documents. In recent years the development of international networks has enabled sharing on a wider scale, with repositories of documents and multimedia information being set up across continents. One of the important needs is to be able to tie these information resources together, linking to or citing other works published on a remote server. Many common applications do now provide hypertext facilities, enabling the linking of information. However, most of them do this as a product of an internal scripting language: the links are hidden and exist as a consequence of the execution of a program rather than explicitly declared data objects, making it difficult to exchange the data between applications.

8 WHO USES HYTIME?

HyTime is a very recent standard, and so there are as yet no commercial products which are based on it although it has been used in research environments [7,8]. In the commercial world TechnoTeacher, one of the lead players in the HyTime standardization effort, are developing a set of object-oriented classes for building HyTime applications called 'HyMinder'. A product can be made to conform to a subset of HyTime facilities without providing a complete HyTime engine: for example DynaText, from Electronic Book Technologies, can build a HyTime location ladder, but cannot use one fully. IBMs IBMIDDoc language conforms to HyTime in its use of the base module but no other features. HTML (see below) and the Open System Foundations standard DTD both make use of HyTime-style hyperlinks.

9 ALTERNATIVES TO HYTIME

HyTime is one possible solution for encoding hypermedia documents for interchange. There are a number of alternative approaches, some of which are existing or forthcoming international standards, others of which are popular commercial or academic solutions. See [9] for information about the relationships between these and other international standards for text and hypermedia.

9.1 MHEG

MHEG is a forthcoming international standard for interchanging hypermedia objects [10]. It is a container architecture which allows media objects to be represented according to an appropriate (external) standard along with instructions for their presentation and behaviour. MHEG objects are to be encoded according to ASN.1 or SGML. It is intended as a practical interchange standard for industrial-strength hypermedia applications requiring real-time

interchange and addresses the problem of exchanging multimedia objects *for presentation*. In this way it is very different from HyTime: display and control semantics are a part of the MHEG standard, whilst in HyTime these are devolved to the controlling application.

9.2 HyperODA

HyperODA is a set of proposed extensions to the Open Document Architecture standard (ISO 8613). ODA is a container architecture which represents text and graphics (each expressed according to an appropriate external standard), providing logical (abstract) and layout (physical) views of the document in parallel. HyperODA extends this model with extra content architectures for audio and images, as well as the link objects and temporal layout. HyperODA is similar to MHEG in that it associates presentation semantics with the document objects, but is more prescriptive than MHEG since it constrains the representation of the component multimedia objects to a small number of international standards. This has the advantage that two HyperODA-compliant applications can always completely understand any document that they exchange, but has the disadvantage that new kinds of media object (movies, for example) cannot be represented without a new version of the standard being defined. HyperODA's similarity to HyTime comes from its association of logical structure with the document components.

9.3 HTML

HTML (HyperText Markup Language) is an SGML-based document architecture used by the academic 'World-Wide Web' (WWW) project [11]. It is designed for simply structured textual documents with embedded graphics and provides hypertext links. As well as providing HTML's architecture for expressing documents, the project defines a universal addressing scheme and a transport protocol for locating and retrieving networked documents. HTML's links are similar to HyTime's contextual links and can be expressed in a HyTime-compliant fashion. Since WWW provides both a document architecture with rendering semantics and an application environment for viewing its documents it would seem more immediately useful than HyTime. However, HyTime's advantage is its extensibility: it can be applied to many document architectures, and a single HyTime 'application engine' can provide the hypertext facilities for all of them.

9.4 Acrobat

Adobe's new 'Acrobat' product [12] is interesting to compare with HyTime. It defines a platform-independent multimedia document encoding called PDF (portable document format) which treats a document as a collection of objects. A document consists of a set of page objects, each of which refer to a number of textual, graphical or pictorial objects. Link, structure, annotation and page preview objects are also supported, along with video and audio objects in a forthcoming release. All objects are coded in PDF's 7-bit text representation, although they may be decoded into various standard media types (e.g. JPEG). Acrobat is quite unlike HyTime in many ways since all the objects are preformatted for presentation onto pages of a specific size. There is no abstract information held with any of the objects, and it is even a non trivial task to extract the text from its formatting. Fundamentally PDF is an architecture based on a hierarchy of objects which

has implicit semantics for document and page objects. This architecture could easily be extended to allow alternative representations for each object, for example to provide a variety of image formats (to support a variety of display software), image resolutions (to support differing speeds of communications link), text renditions (to support language dependencies) or structure abstractions (logical or physical representations to support both formatted display and textual searches). Work is in fact currently underway to try to meld both SGML abstract and PostScript physical representations which would provide facilities similar to ODA. An Acrobat link has a source which is a rectangle in a pages coordinate space and a destination which is a view of another page within the same document. What Acrobat lacks is the ability to fix link anchors into the content and the ability to address objects from other documents. The former arises because there is no easy way to address an object's internal structure, the latter because there is no universal document addressing mechanism on the software platforms on which Acrobat depends. Both of these concerns are addressed by HyTime through its extensions to SGML's simple addressing capabilities.

10 CONCLUSION

Markup can express important information about documents: about their structure and the way they should be presented. This information is added value. It allows a document to be reused and interchanged between systems for many purposes and therefore is an economic consideration. The benefits of generalized markup (as exemplified by SGML) for representing document structure are increasingly appreciated, especially in commercial and military organizations which have to deal with large volumes of information. Projects such as the Oxford English Dictionary [13,14] illustrate the benefits of this approach both for the production of different versions of the dictionary in printed form and for the production of a CD-ROM-based version with advanced searching capabilities. The next wave of development in electronic document handling will be in the field of hypermedia documents, and facilities akin to those provided by SGML for simple text-based documents are required to describe the more complex structures of multimedia hyperdocument collections and their inter-relationships. HyTime extends the SGML model to meet these requirements. There are alternatives to the use of HyTime, each with different strengths and weaknesses. MHEG and Acrobat provide good presentation features, but ignore the information's structure. HTML provides one simple logical document architecture, but is not extensible. HyperODA accommodates both presentation and structure but restricts the kind of multimedia objects that can be used. Many alternatives exist for the simple handling of multimedia documents; many alternatives exist for the simple handling of hypertexts. However, there is no alternative to HyTime for applications in which it is necessary to preserve the structure of the document and express the relationships between its structured components.

ACKNOWLEDGEMENTS

Particular thanks are due to Martin Bryan for his explanation of HyTime.

REFERENCES

1. *Hypermedia/Time-based Structuring Language (HyTime), ISO/IEC Standard 10744*, International Standards Organization, 1992.

2. S. Newcomb, N. Kipp and V. Newcomb, 'The hytime hypermedia/time-based document structuring language', *Communications of the ACM*, **34**(11), 67–83, (November 1991).
3. D. Barron, 'Why use SGML?', *Electronic Publishing: Origination, Dissemination and Design*, **2**(1), 324, (1989).
4. *Standard Generalized Markup Language (SGML), ISO Standard 8879*, International Standards Organization, 1986.
5. L. Burnard, 'Rolling your own with the tei', *Information Services and Use*, **13**, 141–154, (1993).
6. H. Davis, W. Hall, I. Heath, G. Hill and R. Wilkins, 'Towards an integrated information environment with open hypermedia systems', in *Proceedings of the ACM Conference on Hypertext*, ACM Press, New York (1992).
7. L. Carr, H. Davis and W. Hall, 'Experimenting with hytime architectural forms for hypertext interchange', *Information Services and Use*, **13**, 111–119, (1993).
8. J.F. Koegel, L.W. Rutledge, J.L. Rutledge and C. Keskin, 'Hyoctane: a Hytime engine for an MMIS', in *Proceedings of the First International Conference on Multimedia*, pp. 129–136, ACM Press, New York (1993).
9. M. Bryan, 'Standards for text and hypermedia processing', *Information Services and Use*, **13**, 93–102, (1993).
10. R. Price, 'MHEG: an introduction to the future international standard for hypermedia object interchange', in *Proceedings of the First International Conference on Multimedia*, pp. 121–128. ACM Press, New York (1993).
11. T.J. Berners-Lee, R. Cailliau and J.-F. Groff, 'The world-wide web', *Computer Networks and ISDN Systems*, **24**(45), 454–459.
12. D. Brailsford, Adobe's Acrobat—the Electronic Document Catalyst, Computer Science Technical Report, Nottingham University (1993).
13. D. Raymond and F. Tompa, 'Hypertext and the Oxford English Dictionary', *Communications of the ACM*, **31**(7), 67–83, (1988).
14. E. Weiner, 'The electronic English dictionary', *Oxford Magazine*, 6–9, (1987).