ELECTRONIC PUBLISHING, VOL. 8(2 & 3), 51–62 (JUNE & SEPTEMBER 1995)

Web applications and SGML

JACCO VAN OSSENBRUGGEN, ANTON ELIËNS AND BASTIAAN SCHÖNHAGE

Vrije Universiteit
Faculty of Mathematics and Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands

e-mail: {jrvosse,eliens,bastiaan}@cs.vu.nl

SUMMARY

This article advocates the use of SGML technology for the creation, dissemination and display of Web documents. It presents a software architecture that allows for defining the operational interpretation of arbitrary document types by means of style sheets, written in a scripting language. Our approach has been motivated by a desire to extend the functionality of the Web with support for multimedia and active documents. Although growing in complexity, HTML is still lacking in functionality. We prefer a more flexible and generic approach, as enabled by the employment of SGML. After a brief introduction to SGML, we will illustrate how our approach accommodates (extensions of) HTML as well as arbitrary SGML documents containing multimedia data such as video and audio. We will then briefly sketch the software components used in the realization of our approach and discuss some topics for further research.

KEY WORDS SGML WWW Style sheets Active documents

1 INTRODUCTION

SGML (Standard Generalized Markup Language) [1] is an ISO standard which uses tagging to encode the logical structure of a document. The set of tags needed to describe a specific document structure depends on the type of documents involved. For example, the tags needed to mark up a mathematical article are in general not suitable for describing the structure of a telephone book. As a consequence, SGML does not describe a fixed set of tags, but a way to define an appropriate set of tags and the order these tags should appear in a document instance. Such a definition is called a *document type definition* or DTD.

Having a markup language reflecting the structure of the document — instead of the physical layout capabilities (and limitations) of the systems used to process the document — is an intuitively appealing idea. It enables one to define a simple and lean markup language, well suited to markup documents from a specific domain. Because the syntax of the language is defined in a standardized format, processing software can be build upon generic SGML tools (e.g. parsers), and can be kept simple due to the limited set of tags. Resulting document instances can be platform independent and possibly outlive the software and hardware used to process them.

Although HTML — the markup language of the World Wide Web — is an application of SGML, the scenario sketched above is the exact opposite of the "one size, fits all" approach of current Web browsers. One of the main reasons of the success of the Web was the initial simplicity of HTML. A quick look at the draft HTML 3.0 specification [2] (or at an arbitrary 'Netscape enhanced' Web page) will learn that most of that initial simplicity has gone. Many HTML documents lack structure and depend heavily on (the current version of)

Received 3 April 1996
Revised 14 June 1996

the browser used by the author. Most of today's Web browsers are large, complicated and often inflexible software products, which do not take advantage of the flexibility provided by SGML parsers.

Despite the complexity of the latest HTML specifications, the functionality of the Web can still be considered primitive in comparison with other hypermedia systems [3,4]. To allow for more complex hypermedia documents, other document formats than HTML are needed. This article advocates the employment of SGML technology in Web-aware software, and sketches the architecture of a prototype SGML Web browser developed at the Vrije Universiteit. Additionally, we give some examples to show how this browser can be used to process extensions of existing document types, as well as completely new ones.

The article is structured as follows. We start by a brief introduction to SGML. In "SGML on the Web" we discuss the software architecture underlying our approach to developing Web applications. We will illustrate how our approach accommodates (extensions of) HTML as well as arbitrary SGML documents containing multimedia data such as video and audio by giving some examples of active documents. In "Software Components" we characterize the software components used in the realization of our approach. We conclude by evaluating our approach and indicating issues for future research. The material concerning the basic SGML concepts and the active document examples has been adapted from 'Animating the Web — An SGML based approach' [5], which describes the architecture of our Web browser in more detail.

2 RELATED WORK

Advantages of SGML-aware Web browsers (and servers) are discussed in [6]. Various style sheet languages for SGML documents have been proposed. An ambitious proposal is the Document Style Semantics and Specification Language [7], which is a (draft) ISO standard. DSSSL Online [8] specifies a subset of DSSSL to implement style sheets to be used by SGML browsers. Several other methods, including a simple mechanism using mapping tables, are described in [9]. The W3C developed the Cascading Style Sheet mechanism [10] to attach style sheets to HTML documents. In contrast to our work, these proposals focus on the physical layout and are less suited for the description of dynamic and interactive hypermedia document formats.

HyTime [11] defines several syntactical conventions (known as *architectural forms*) for describing the most fundamental hypermedia concepts of SGML documents: addressing, linking and alignment. Some commercial Web browsers [12] support a subset of the functionality defined by the HyTime standard.

The expressive power of a general purpose programming language (e.g. Java) can also be deployed by including script fragments or applets directly in a hypermedia document. While the resulting documents are often hard to maintain and even harder to process by other applications (e.g. index engines), the combination of applets technology and distributed object systems [13,14] appears to be a powerful approach.

3 SGML — BASIC CONCEPTS

An SGML document essentially consists of two parts: a prolog, containing the document type declaration, followed by the document instance, containing the data interspersed with markup.

3.1 Document instance

A document instance is a hierarchical structure of (possibly empty) *elements*, where each non-empty element contains other elements or character data. Each element has a name (the *general identifier*) and the start and end of an element are indicated by tags (typically <name> content </name>). Begin or end tags may be mandatory or optional. Moreover, elements contain zero or more *attributes*. Consider the example document below.

```
<memo security=confidential>
 <to>Anne
 <cc>Anton<cc>Bastiaan
 <from>Jacco
 <subject>EP submission
 <body>
   Dear Anne,
   I'm sorry we couldn't make it before the deadline,
   but we will send you a PostScript copy of our
   EP submission before next Wednesday.
 </body>
</memo>
```

The root element *memo* has one attribute, indicating the security level of the document. The memo element contains six other elements. A *to* and two *cc* elements stating the addressees, a *from* element specifying the sender, a *subject* and a *body* field. All elements of memo contain character data and no other elements. Note that the tags emphasize the logical structure of the document rather than stating how it should be formatted.

3.2 Document type declaration

The document instance above must be preceded by a *doctype* declaration. The main part of the document type declaration is the *document type definition* or DTD. It defines the elements of a document and the required order of their sub-elements. The elements and their contents are defined by the use of *element declarations*.

```
<!doctype memo [
<!element memo          O O (to+,cc+,from,subject?,body) >
<!element (to|cc|from|subject|body)        - O (#pcdata) >
<!attlist memo security (low|confidential|topsecret) low >
]>
```

The second line declares the memo element, and defines its content as a sequence of one or more *to* elements, one ore more *cc* elements, a *from*, an optional *subject* and a *body* element. The two 'O' characters stand for "omit", indicating that the begin and end tag of *memo* may be omitted. The third line defines the elements containing character data only. Their start tags are mandatory, indicated by the '-'. The list of attributes of each element is declared by an *attlist* declaration. Attributes can be of different types, and be mandatory or optional. The DTD can specify a default value, as is shown in the case of the security attribute. The DTD declaration may be contained within the *doctype* declaration, but is typically defined by a separate file. In that case, the *doctype* declaration contains a reference to that file.

3.3 Processing instructions

Processing instructions are used to pass system dependent information to the application to tell how the document is to be processed. Processing instructions are contained within '<?' and '>' characters and can appear on arbitrary places within the document. In the following examples, we employ processing instructions to indicate the URL of a style sheet.

3.4 Entities

Fragments of markup and character data can be given a name using an *entity* declaration. The declaration of an entity is part of the document type declaration, but the entity may be used within the document instance. The contents of an entity may be defined by a string, or may be contained in an external file, in which case the entity declaration contains a reference to the file. External entities may be referenced by a *system identifier*. Support for these identifiers is system dependent and may include filenames, URLs and database queries. Consider a variant of the previous example:

```
<!doctype memo
  system "http://www.cs.vu.nl/˜hush/demo/memo.dtd" [
<!entity ps "PostScript"> ]>
<?stylesheet lang=Tcl
  src="http://www.cs.vu.nl/˜hush/demo/memo.style">
<memo>
   ...
   ... but we will send you a &ps; copy of our submission ...
  </body>
</memo>
```

The first line references an external DTD by means of a system identifier, and the second line defines an entity for later usage. Note that the entity definition is enclosed within the square brackets of the *doctype* declaration. The third line contains a processing instruction to define the location of the style sheet. In contrast to the URL of the DTD, which is resolved by the parser, the URL of the style sheet is passed to the application without further processing by the parser.

To avoid system dependent identifiers such as filenames, an extra indirection is provided by the concept of *public identifiers*. These identifiers are assumed to be publicly known, and the SGML parser of the target application is expected to be able to resolve them. Typically, a local catalog file is used to map public identifiers onto system dependent ones. *Formal* public identifiers have a standardized and meaningful inner structure, to facilitate automatic resolving without the use of catalogs. In the following examples, we refer to the HTML 3.0 DTD by means of a formal public identifier.

4 SGML ON THE WEB

As stated before, many HTML documents depend on the browser technology used to display them. The employment of SGML in Web-aware software supports a more document-centered approach, because it allows one to use a different markup language for each type of document, and adapt the processing software to the structure of these documents, instead of adapting documents to the software.

However, SGML offers only part of the solution. Applications need to attach semantics to the syntactic constructs defined by the DTD. SGML does not address the semantics the elements a document contains. Style sheets have been proposed to attach such semantics. The expressive power of the style sheet language and the ways it can address the functionality offered by the processing software (e.g. a Web browser) is extremely important, since it determines the look and feel of the resulting documents. Especially in a hypermedia environment, the interactive and dynamic nature of the documents requires the use of a very powerful style sheet language.

The example in the following section describes how to extend HTML with a *video* tag, which is used to include inline video fragments in HTML documents. While SGML is used to specify the syntactic details (e.g. in which context the video tag is allowed, which attributes are required, which ones are optional, etc.), one needs a style sheet language which is powerful enough to specify the operational semantics to be able to instruct the browser how to process the new tag. A style sheet language specially designed to handle documents containing video will probably allow a clear, declarative description of the video tag in an associated style sheet. However, such a special purpose language will be far less useful for applications not anticipated by its designer.

Therefore, we will use a general purpose and extendable procedural scripting language to attach operational semantics to the syntactic constructs defined by the DTD. Which language should be used is a matter of taste and, more often, a matter of pragmatics (we will use Tcl [15] in our examples). Many interpreters of modern scripting languages are embeddable and extendable, which make them good candidates for a style sheet language. More important is the application programmers interface (API) of the underlying software system, which should be accessible from the style sheet language. The interactive nature of hypermedia documents often requires the manipulation of the user interface. To describe the functionality of such tags, a style sheet needs to address the underlying GUI toolkit. To specify how a browser should handle multimedia data, the audio and video components of the platform should be made accessible as well. For more complex hypermedia documents, access to hyperlink databases and a multimedia scheduler will be required. Finally, the parser itself should be accessible to enable the style sheet to have full access to the document's contents and structure.

5 ACTIVE WEB DOCUMENTS – AN EXAMPLE

Developing a complete document type from scratch is a complex task. Fortunately, one can often reuse (parts of) existing ones. As an example, we will first illustrate how an existing document type (in this case HTML) may be extended with an additional *video* tag. On a syntactical level, we do this by developing a new document type definition, which uses the standard HTML DTD. On an operational level, we supply a style sheet which defines how the video components of the underlying system should be deployed to display the video data.

5.1 Extending Document Types

We will start by illustrating how to extend the default HTML document type definition. The document instance given below is specified in HTML but employs in addition two extensions for active documents. The first extension is a *video* tag, which is used to display inline video fragments. The other is an *applet* tag, used to embed small applications written in a scripting language. We use Tcl in the examples. In the example below, the *applet* displays some notes that are played when the user clicks on the image.

```
<!doctype demo
  system "http://www.cs.vu.nl/˜hush/demo/video.dtd" [
<!entity title "Hush browser inline video/applet demo"> ]>
<?stylesheet lang=Tcl
  src="http://www.cs.vu.nl/˜hush/demo/video.style">

<demo>
  <title>&title;</title>
  <h1>&title;</h1>

  <p>This page shows the ability to inline video into
  an HTML-page: <p><video
  src="file:/usr/local/public/animations/mpeg/canyon.mpg">

  <p>Inline applets are supported as well.
  Here is an interactive arrow editor:
  <p><applet class=arrow>
  <hr>
  <p>hush@cs.vu.nl, Vrije Universiteit Amsterdam, 1996.
</demo>
```

The first line of the example defines the type of the document (demo). It is specified in a separate document type definition (DTD) that we will describe below. The parser automatically retrieves the DTD from the Web if its location is specified by a URL. The next line illustrates the use of an *entity* declaration, an SGML mechanism used here as a primitive macro facility defining the title of the document. The title entity is used twice, in the *title* and *h1* tag, and will be expanded by the parser. The third line specifies the style sheet that is needed to display the contents of the document. The use of style sheets will be explained later. The *video* element requires a *src* attribute defining the location of the video file. Finally, the *applet* tag is used to inline embedded script code defining an interactive arrow editor.

Whether active document elements are to be defined by a new, specific tag or by the more general applet mechanism is a matter of taste. A new tag requires modification of the DTD and style sheet but describes the element in a more declarative way, which gives the application more freedom in displaying the contents. For example, a browser may decide to display a text alternative if the local platform does not support video.

5.1.1 DTD

Recall that a document type definition defines the structure of a document by describing the *elements* and *attributes* that can be interspersed as tags with the document content. The following DTD extends the (draft) HTML 3.0 DTD [2] with the *video* and *applet* elements required for the example above.

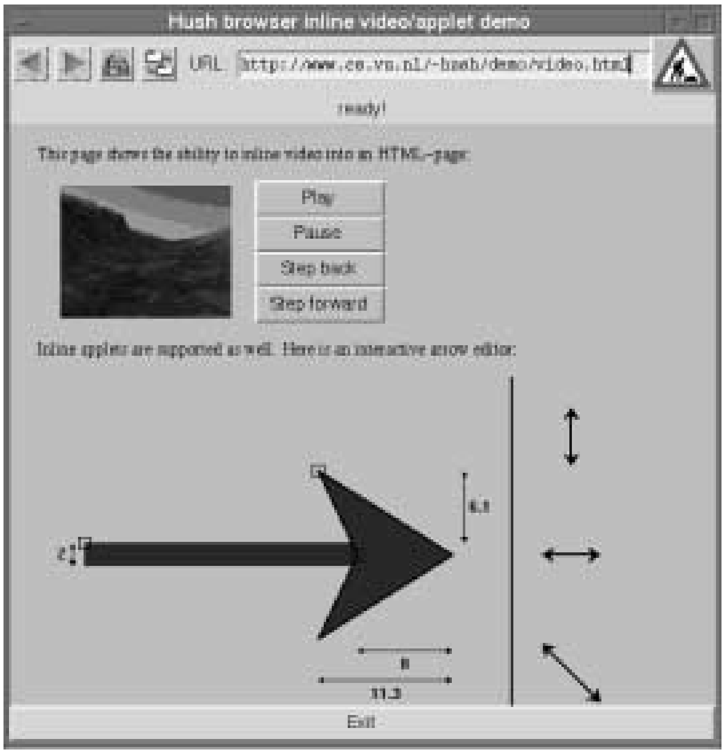
```
<!entity % special "tab|math|a|img|br|applet|video ">
<!entity % htmldtd public "-//IETF//DTD HTML 3.0//EN">
%htmldtd; <!-- include standard HTML 3.0 DTD -->

<!element demo O O (%html.content;) >

<!element (applet|video) - O empty>

<!attlist applet
  class cdata  #required   -- class name of applet        --
  lang (Tcl|Java|Python)
               Tcl         -- Default script language     --
                           -- URI is defined by HTML DTD: --
  src   %URI;  #implied    -- defaults to base URL        --
  data  %URI;  #implied    -- location of additional data --
>
<!attlist video
  src %URI;    #required   -- URI of video fragment       --
  alt cdata    #implied    -- optional text alternative   --
>
```

The first line extends the *special* parameter entity defined by the HTML DTD with *video* and *applet* elements, enabling the use of *video* and *applet* tags where image or anchor tags are allowed. The second definition of *special* (in the HTML DTD) will be ignored by the parser. In the second line, the SGML parameter entity mechanism is used to include the draft HTML 3.0 document type definition, which is referenced by a formal public identifier. The *demo* element consists of the sub-elements defined by the HTML DTD. The two *O* characters in the element declaration indicate that begin and end tag of the *demo* element may be omitted, since the begin and end of the document can be derived by the parser. In contrast, the begin tag of a *video* or *applet* element is mandatory, the end tag optional. The video and applet elements have optional (*implied*), mandatory (*required*) and default attributes (e.g. the *lang* attribute defaults to *Tcl*).

The information contained in the DTD is used by the parser to generate a complete and validated document instance. Note that this task could be performed by an HTTP-server as well, which would significantly simplify the design and implementation of Web clients. Therefore, there are strong arguments to add SGML functionality to servers as well [6].

5.1.2 Style sheet

Style sheets define how the various elements should be processed. The *hush* browser (see figure 1) defines a default style for HTML elements. However, these styles can be redefined and extended by a document instance using a special processing instruction, notated as <`?stylesheet url`>. The browser retrieves the URL specified in the processing instruction and uses it to display the contents of the document. The example specifies a URL to a style sheet that describes how to process the new *video* tag. Recall that processing instructions are application dependent, so the parser passes the text in a processing instruction directly to the application. At the moment, we use an experimental style sheet language based on Tcl. An example of a style sheet fragment that specifies how the *video* tag should be processed, is given below.


Hush browser inline video/applet demo
URL
http://www.cs.vu.nl/~hush/demo/video.html
ready!
Play
Pause
Step back
Step forward
Exit

This page shows the ability to inline video into an HTML-page:

Inline applets are supported as well. Here is an interactive arrow editor:

Figure 1. The example document as displayed by the hush browser

```
# example of Tcl style sheet

proc VIDEO {atts data} {
  HMextract_param $atts src       # nasty Tcl details
  upvar #0 HM$win var

  set top $win.videoframe         # create root frame
  pack [ frame $top ]
  $win window create $var(S_insert) -window $top

  frame $top.bf                   # create control buttons
  button $top.bf.b1 -command "$top.film start"
                    -text "Play"
  button $top.bf.b2 -command "$top.film stop"
                    -text "Pause"
  button $top.bf.b3 -command "$top.film step next"
                    -text "Back"
  button $top.bf.b4 -command "$top.film step prev"
                    -text "Forward"
```

```
    pack $top.bf.b1 -fill x         # pack buttons into frame
    pack $top.bf.b2 -fill x
    pack $top.bf.b3 -fill x
    pack $top.bf.b4 -fill x
    pack $top.bf -side left
                                    # use hush video widget
    pack [ xanim $top.film ] -side left -padx 20
    set tmpfile [urlget_tmp $src]   # use hush web interface
    $top.film file $tmpfile         # to get video form web
  }
```

The fragment shows the low level Tcl code needed to display the video fragment and some control buttons. While the style sheet mechanism clearly needs some refinement, our approach supports the extension of existing document types and allows for extensive experimenting with new document types.

5.2 Creating New Document Types

The document instance of the previous example was very similar to plain HTML documents, which made it worthwhile to reuse the original HTML DTD and style sheet. However, for some applications HTML is not suited at all and a completely new document structure is needed. The next example shows a document instance of a simple musical application.

```
<!doctype score system "music.dtd" [
<!entity G7 "<note name=keynote>g<note>b<note>f"> ]>
<?stylesheet lang=Tcl src="music.style">

<score>                         <!-- document instance -->
  <chord id=chord1>&G7;         <!-- entity usage      -->
  <chord id=chord2>&G7;
  <chord id=chord3 name=Cmajor>
    <note name=keynote>c<note>e<note>g
</score>
```

The first line defines the (filename of) a new document type definition. The next line contains an entity definition describing a G7 chord. An SGML processing instruction is used to specify the filename of the style sheet. The root of the document hierarchy is the *score* element, consisting of several chords. Chords are build of notes, which are described by single characters. The first two chords use the entity defined before, specifying the notes of a G7 chord. The third occurrence of *chord* describes a C major chord.

5.2.1 A new DTD

The DTD corresponding to the simple musical document given above defines the structural elements and their attributes. When the application processes the document, the parser will fill in the default duration for all notes, resolve the entity definition and add the missing end tags for the notes and chords.

```
<!-- Simple music DTD -->

<!element score - O (chord)+    >
<!element chord - O (note)+     >
<!element note  - O (#pcdata)   >

<!attlist (chord|note)        -- all have same attributes  --
   id       id    #implied    -- optional identifier       --
   name     cdata #implied    -- optional name attribute   --
   duration cdata "4"         -- default duration: quarter -- >
```

The DTD defines three structural elements: a *score* containing several *chords*, a *chord* containing several *notes* and a *note* consisting of data. Both chord and note have three attributes: *id*, *name* and *duration*. The first two are optional and the last has a default value of 4. Note the difference between the *id* attribute, which has to be a unique identifier, and the *name* attribute, which can be an arbitrary string.

5.2.2 Style sheet

Playing the document by a Web browser does not necessarily involve displaying the data visually. The simple style sheet shown below simply collects notes and chords and plays them by using the *play* command. Note that most of the timing relations are implicit in the document. For example, the notes within a single chord are to be played in parallel, and the chords themselves are to be played sequentially. However, this is not explicitly defined by the document instance or DTD and can only be intuitively derived from the element names. Even in the style sheet below, these timing relations remain implicit.

```
# Simple Tcl style sheet for music example

set score  ""; set chord  ""

proc SCORE { atts data }
           { global score; set score "t120 " }
proc CHORD { atts data }
           { global chord; set chord "(" }
proc NOTE  { atts data }
           { global chord; set chord "$chord$data<" }

proc /SCORE { } { global score; play $score }
proc /CHORD { } { global score chord;
                  set score "$score$chord r)" ; set chord "" }
proc /NOTE  { } { }
```

The procedures corresponding to the open and close tags build a string representation of the score. At the opening of the score element the string is initialized with a command defining the tempo at 120 beats per minute. During the parsing process the string is extended with the parsed notes. After the last chord has been parsed the resulting string is: `t120 (g<b<f< r)(g<b<f< r)(c<e<g< r)`. This string is played after the score end tag has been encountered. The Tcl command *play* used to play the notes is provided by the *hymne* extension of *hush* [16,17].

6 SOFTWARE COMPONENTS

The components constituting the hush Web browser are build upon the *hush* class library [18,19], which offers a C++ class interface to the Tcl/Tk [20] GUI toolkit. Additionally, it offers a discrete event simulation package and multimedia support, including VRML, software video and sound synthesis [16,17]. All hush components can be accessed by means of both a C++ and script interface, which allows one to use these components in C++ applications, style sheets and Web applets.

The browser can be used as a document validation tool since we use SP [21], a validating SGML parser to parse the SGML documents. The parser generates events during the parsing process when it encounters open and close tags (even when optional tags are omitted in the document instance), character data, processing instructions, entities, etc. Style sheets specify the way these events are processed. A default style sheet for HTML is provided, which may be overruled by other style sheets specified by individual documents or by the user.

Currently, our research is focused on providing higher level support for the definition of style sheets, including mechanisms to provide support for more complex hyperlinking and the scheduling of synchronized multimedia fragments by means of a distributed logic programming language [22].

7 CONCLUSIONS

Employing SGML technology in Web applications can have several advantages, on a software level, but especially on a document level. It allows one to use document types especially suited for reflecting the structure of documents of a specific domain, and document instances which are free of markup describing physical layout. However, the solutions provided by SGML are mainly addressing the definition of syntactical concepts (i.e. tags and attributes), whose semantics need to be specified as well. Common style sheet languages are not suited to describe highly interactive and dynamic hypermedia documents.

Using an extendible general purpose scripting language for style sheets has several advantages. First, the style sheet language can offer full access to the underlying system, including the graphical user interface, multimedia devices, document parser, and hyperlink base. Secondly, it allows for the introduction of new tags and document types without changing the browser. Finally, new software components can be made accessible by extending the scripting language.

A disadvantage is the low level interface of our style sheet language and the lack of standardized APIs, which make it hard to define platform independent style sheets without sacrificing functionality. Until SGML-aware Web browsers become more common, SGML encoded documents need to be converted to HTML. Still, the SGML browser described here provides a suitable test environment to experiment with more complex hypermedia document types.

REFERENCES

1. International Organization for Standardization, ‘Information Processing — Text and Office Information Systems — Standard Generalized Markup Language’, Technical Report 8879:1986, ISO, (1986).

2. D. Raggett, ‘Document Type Definition for the HyperText Markup Language (HTML DTD)’. Expired Internet Draft, Part of the HyperText Markup Language Specification Version 3.0, March 1995.
3. Gary Hill and Wendy Hall, ‘Extending the Microcosm Model to a Distributed Environment’, in *ECHT’94, The European Conference on Hypermedia Technology*, pp. 32–40, (September 1994).
4. G. van Rossum, J. Jansen, K. S. Mullender, and D.C.A. Bulterman, ‘CMIFed: A Presentation Environment for Portable Hypermedia Documents’, in *The First ACM International Conference on Multimedia*, pp. 183–188, (August 1993).
5. Anton Eliëns, Jacco van Ossenbruggen, and Bastiaan Schönhage, ‘Animating the Web — An SGML-based Approach’, in *Proceedings of the International Conference on 3D and Multimedia on the Internet, WWW and Networks, 17-18 April 1996, Bradford*. British Computer Society, (April 1996).
6. C. M. Sperberg-McQueen and Robert F. Goldstein, ‘HTML to the Max — A Manifesto for Adding SGML Intelligence to the World-Wide Web’, in *Proceedings of the Second International World Wide Web Conference ’94: Mosaic and the Web*, (October 1994).
7. International Organization for Standardization/International Electrotechnical Commission, ‘Information technology — Processing languages — Document Style Semantics and Specification Language (DSSSL)’, Technical Report DIS 10179.2, ISO/IEC, (1995).
8. (Compilation by) Jon Bosak. DSSSL Online Application Profile, December 1995. Available from http://occam.sjf.novell.com:8080/docs/dsssl-o/do951212.htm Formerly known as DSSSL Lite, available from http://www.falch.no/people/pepper/DSSSL-Lite/.
9. J. Warmer and H. van Vliet, ‘Processing SGML Documents’, *Electronic Publishing — Origination, Dissemination and Design*, **4**(1), 3–26, (March 1991).
10. Håkon W. Lie and Bert Bos. Cascading Style Sheets, level 1, Februari 1996. W3C Working Draft. Available at http://www.w3.org/pub/WWW/TR/.
11. International Organization for Standardization, ‘Information Technology — Hypermedia/Time-based Structuring Language (HyTime)’, Technical Report 10744:1992(E), ISO, (1992).
12. SoftQuad Inc. Panorama PRO. See http://www.sq.com/.
13. Xerox Corporation, ‘ILU: The Inter-Language Unification System’. Available from ftp://ftp.parc.xerox.com/pub/ilu/ilu.html, 1995.
14. The Object Management Group. CORBA 2.0 Specification, 1996. Available from http://www.omg.org/.
15. J.K. Ousterhout, ‘Tcl: An Embeddable Command Language’, in *USENIX*, (1990).
16. Jacco van Ossenbruggen and Anton Eliëns, ‘Music in Time-based Hypermedia’, in *ECHT’94, The European Conference on Hypermedia Technology*, pp. 224–270, (September 1994).
17. Jacco van Ossenbruggen and Anton Eliëns, ‘Bringing Music to the Web’, in *Proceedings of the Fourth International World Wide Web Conference*, pp. 309–314, (December 1995).
18. Anton Eliëns, ‘Hush: A C++ API for Tcl/Tk’, *The X Resource*, (14), 111–155, (April 1995).
19. Anton Eliëns, *Principles of Object-Oriented Software Development*, Addison-Wesley, 1995.
20. J.K. Ousterhout, ‘An X11 Toolkit Based on the Tcl Language’, in *USENIX*, (1991).
21. James Clark, ‘SP — An SGML parser’. Availble at http://www.jclark.com/sp.html, October 1995.
22. Anton Eliëns, *DLP — A Language for Distributed Logic Programming*, Wiley & Sons, 1992.