An extensible rendering engine for XML and HTML

Paolo Ciancarini, Alfredo Rizzi, Fabio Vitali
Dept. of Computer Science, University of Bologna
{ciancarini|rizzia|vitali}@cs.unibo.it

Abstract

*XML has been proposed in order to bring to the web a markup language free of the shortcomings of HTML, in particular the inextensibility of the set of valid elements (tags). Stylesheet languages have been proposed for XML, in order to provide precise and sophisticated typographical control over the appearance of text-based data. We have developed a rendering engine for HTML and XML documents, providing rudimentary support for typography, but allowing easy extensions (*displets*) for any kind of data, including non-textual ones, such as math, charts, graphs, etc. Some extensions have already been developed: here we present the one for supporting Z, a notation for formal specifications of software systems.*

Keywords

XML, Markup languages, XSL, Stylesheet Languages, Displets

1 Introduction

One of the most important contributions of the HTML markup language is its text-based support for interface elements such as input fields, buttons, choice lists, etc., along with structural and formatting commands for text within the data format of network documents.

Such support has allowed complex interfaces to be described in a text-based source document side-by-side with proper and traditional text content. This in turn has made it easier for authors and developers to create interactive form-based information systems: the same source code could be easily created by hand for simple applications, mock-ups, prototypes, or as tailorable templates, or automatically by simple procedures of complex applications.

Within the limits of the form support currently defined in HTML, a new generation of client-server applications has been made possible, where interface elements, context information, and on-line help can share the same environment, provide information and support to each other, and therefore enrich traditional interfaces with hypertext [BV97].

On the other hand, HTML only allows a few elements, that is, those that are explicitly defined in the standard. Whenever some authors' needs exceed the capabilities of the elements already defined in HTML, a different approach has to be used: either the existing tags are abused for a different purpose than that for which they were created, or an image is used, or a Java applet is created providing the desired functionality.

It would be useful to give markup support to many types of non-textual data, particularly specialised notations (mathematics, music, chemistry, etc.), and other kinds of data whose visualisation is essentially graphical. For instance it would be useful to be able to describe arbitrary graphs as markup. Building on the experience of HTML form elements, this would have the

immediate benefit, among many others, of allowing site managers to create in an easy or automatic way several hypertext features that were left out of the World Wide Web, and that have been listed in [BVABO97], such as global and local overviews, trails, guided tours, etc.

Recently, a new standard has been announced, XML [BSM97], that promises to be able to provide for these shortcomings. XML is a meta-markup language: it allows authors to define the markup elements of their own documents, so as to tailor them as much as possible to their needs. XML elements do not have pre-designed meanings, behaviour and visual characteristics: these concerns are external to the language, and are partially approached in two ancillary standards, XLL [BD97] and XSL [ABC97].

Given the early stage of all these standards, not all issues surrounding them have been faced and solved. In particular, the discussion on the rendering (i.e. the provision of a displayable shape) of XML elements has so far been focused on typographical concerns, mostly building on existing stylesheet languages such as CSS [LB96] and DSSSL-O [BOS96]. Unfortunately, neither language provides support for non-typographical rendering: content that is best represented with images, charts, schemata, etc. needs specialised browsers for proper rendering, as experiences with chemical data (CML [MR96]) and mathematical formulas (MathML [IM97]) are showing.

The purpose of this paper is to report on a Java rendering engine for XML and HTML data that we have been implementing. The engine allows easy integration of textual and non-textual data, and provides typographical as well as graphical support for all kinds of data can be expressed in XML or properly extending HTML.

This engine builds on a previous work on extensions to HTML called *displets* [VCB97]. Displets were built on an antiquate version of HotJava and had complete access to the underlying HTML parser and the existing display libraries of the browser. The rendering engine we are describing here, on the other hand, is a completely autonomous piece of code that can work on unmodified Java-capable browsers such as Netscape Communicator or Internet Explorer.

The paper is structured as follows: in section 2 the development and characteristics of XML and related proposals is summarised. In section 3 the case for rendering of non textual data is proposed and defended. Section 4 details the rendering engine and its inner working. Section 5 describes how new display modules, which we still call *displets*, can be created in this environment. We will then draw some conclusions and sketch our future work.

2 HTML and XML

HTML has been extremely successful in allowing unsophisticated network users to become authors of fairly complex documents, even in the absence of widespread editing tools. Nonetheless, there has been in the past two or three years a widespread awareness ([SMG94], [Sie97], etc.) that HTML has reached its potential, and that a change of paradigm was necessary.

The major drawback of HTML is that it allows only a pre-specified set of elements. Authors can only use these elements, and have to limit their authoring needs to what is available within the existing language, or to force these elements beyond their intended meaning.

HTML is an application of the Standard Generalized Markup Language [SGML], that is, a class of documents conforming to an SGML Document Type Definition (DTD) that describes "HTML documents". SGML, being a meta-language used to describe classes of documents, rather than a specific class, is free of the above mentioned limitations of HTML: by appropriately creating a custom class of documents, and defining the legal elements therein, authors can provide support for any kind of rhetorical need, however complex and arcane. Metalanguages "allow groups of people

or organisations to create their own customised markup languages for exchanging information in their domain (music, chemistry, electronics, hill-walking, finance, surfing, linguistics, knitting, history, engineering, rabbit-keeping, etc.)" [Fly97].

Unfortunately, SGML is considerably more complex to learn and use than HTML, and it has been felt that this would prevent its generalised adoption. Therefore the SGML working group of W3C was asked to develop a new markup meta-language, the eXtended Markup Language (XML), that could take the place of SGML on the Web. XML documents would have to be straightforwardly usable over the Internet, compatible with SGML, and easy to create.

Of course, XML gets flexibility by giving up specificity: contrarily to HTML, the elements used in XML documents do not have a pre-defined meaning that is understood and used by XML software, and they do not have a given graphical aspect or typographical style. That is, XML only specifies the correct abstract syntax of documents, but not how each element should be displayed as, nor what behaviour should be associated with them.

Two ancillary mechanisms were designed to take care of some of these issues: hypertext linking was considered a behaviour important and widespread enough among the document types to be directly supported in the language, and the XLL specification [BD97] is meant to take care of the linking paradigm of XML. The typographical rendering of markup elements, that is part of the definition of HTML, is outside of the concern of XML, since the element set is open and the emphasis is on structure markup rather than formatting. The XSL (Extensible Stylesheet Language, [ABC97]) specification, currently just being proposed for discussion, is meant to provide a general way to attach a set of rendering instructions to the elements of an XML document.

Although at this time XSL is in a too early stage of development to be reliably discussed, it is clear that the proposed mechanism will provide XML syntax for selecting and using graphical constructs from either one of the two major stylesheet languages currently available for markup languages: CSS and DSSSL.

It is worth noting that, in spite of the relevant differences that can be found between the two stylesheet languages, neither of them supports or consider the display of non-textual data, which clearly is an important need of many of the above mentioned communities.

3 The need for displets

While the XML markup language is well suited for specialised notations, behaviours and rendering are not well supported by the X*L language family except for links and text.

The stylesheet languages available within the XSL framework focus on typographical control: they have precise instructions on the type of font to use, the margins and alignment to provide to paragraphs, how to format tables, etc. In short, they are thought essentially for textual data.

This has not deterred many special communities from using XML for standardising their notation and markup needs, but they are planning special applications and applets for the correct display of their documents.

For instance, the HTML-Math working group has changed its charter from extending HTML to specifying a XML application [IM97]. The display problem is clear for mathematical markup: using images of equations, that have been created statically and independently of the surrounding text, is limiting in several ways: the point size and type, the background, the line alignment may match the surrounding text on the system where it was produced, but not the system where it is displayed.

Furthermore, while the resolution of images and text may be the same on the screen, print resolution usually is much improved for text, but would remain exactly the same for a GIF or a JPEG image.

Also the problem of rendering software is clear to the authors of the document: at present, only specialised Java applets and plug-ins can be foreseen to take advantage of the new notation. Further, more general and flexible solutions can be hypothesised using some ways to extend the stylesheet mechanism of XML and relying on the document model that is being proposed for Web-based data [DOM97]. Currently there are some applications and browser that do or plan to support MathML (WebEQ, IBM techexplorer, MathType, etc.).

Among the browser that can display MathML it is important to mention Amaya, which is a general-purpose extensible WWW browsers. Among the things that can be extended is the management of the HTML language itself. It is thus possible to extend both the tag set that is recognized by the HTML parser, and the display objects that constitute the display of the tags themselves. Amaya uses the Thot languages to specify the extensions: the S language is used for the specification of the new structural element, while the P language is used to specify the appearance of the new elements.

The chemistry community is also using XML to provide support for the myriad of different notation needs of their science. The CML language [MR96] is an application of XML meant for chemical data, another kind of information that cannot be easily represented by text. JUMBO is a specialised browser for CML documents: it is an autonomous application (an applet is being developed in these days), and provides several independent windows where each type of chemical data is displayed.

Displets were proposed in [VCB97] as a way to extend HTML documents using Java. The HTML language was extended on a per-document basis by defining new tags as needed, and providing Java classes to take care of their graphical display. While not providing all the functionality and flexibility of a full meta-markup language such as XML, HTML extended with displets could allow all kinds of specialised notations and graphical effects while at the same time leveraging over the existing and well-known set of elements defined by HTML.

And still we are not convinced that this is a bad idea when striving beyond HTML. Admittedly, in some instances documents simply cannot be forced into the existing elements of the HTML language, and require a complete and radical re-thinking of the elements that need to be used. In these cases a full meta-language is needed such as XML, so that one can define the markup elements from scratch and without legacy influences. In many other cases, on the other hand, there are one or two special requirements that cannot be satisfactorily covered by HTML, and that require the definition and use of a few more tags than those available in the standard language. In these cases, it would be useful to be able to extend the existing HTML tag set with those few new ones, rather than starting from scratch with all the tags both special and trivial.

The key issue in our opinion is that special notations require special software to take advantage of them: we must be able to display these notations, and to activate behaviours on them: to provide a general markup meta-language is not enough. One class of behaviours (links), and one class of display elements (text) have been considered important enough to be singled out in the development of the XML standard, and will have their own specifications.

For other types of behaviours (for instance, those required by forms elements), and for the display of non-textual data, the attitude is towards creating specialised applications that can understand and enact the behaviours and the rendering option implicit in the special notation. Thus, while applications such as JUMBO or MathEQ are commendable for their generality and usefulness, there is the need for an integrated environment that provides a general rendering and behaviour engine for arbitrary markup elements.

4 The DispletManager applet

Our first experiment with rendering arbitrary, non-text-based markup extensions [VCB97] was to modify an existing browser to allow the parsing and the visualisation of new HTML-like elements. To do so, we took an early version of the HotJava browser, whose source code was freely available and modified it so that it could accept on-the-fly extensions of the HTML DTD and load the appropriate classes (called *displets*) whenever the newly defined tags were to be displayed. That experiment was extremely limited, in that we used an old version of the Java language, and worked only on a specific version of a specific browser. Furthermore, we heavily relied on the existing rendering architecture of the browser and just provided a minimal effort implementation (basically a displet was just a sequence of drawing instructions for the visualisation of the elements).

In this paper, on the contrary, we report about the DispletManager applet, a general, extensible rendering and behaviour architecture we have been working on, which can be used for both extensions to HTML and straight XML documents. This architecture is embodied in a Java applet that can be run within any Java-enabled browser such as Netscape Communicator or MS Internet Explorer.

Fundamental design requirements for the rendering engine were as follow:

- it must be possible to create special code for rendering arbitrarily odd data types, in particular non-textual data (*displets*).
- all displets must easily integrate with each other: a chart element may have a mathematical formula as one of the labels, and some staff notation as another, where some notes may act as hypertext links.
- the rendering engine must work both for extended HTML and for straight XML, and the displet classes must be identical.

Figure 1 shows the general structure of the DispletManager applet:

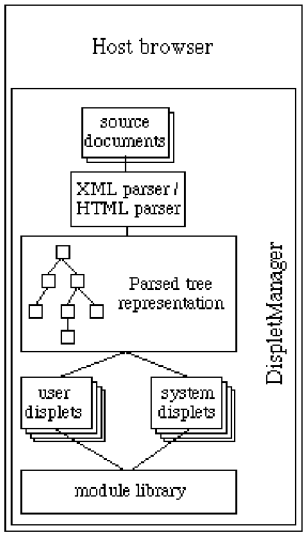

Fig. 1 - The general structure of the DispletManager applet

The document chunk to be displayed, be it HTML or XML, is loaded by the displet manager and parsed by the appropriate parser. The resulting tree is then analysed recursively (depth-first): the appropriate displet classes are activated to create the rendering of their element on the basis of the rendering of their sub-elements. No class is allowed direct access to the screen: on the contrary,

each displet creates a (set of) bitmap(s) that its ancestor can pass, ignore, modify or add to.

4.1 Extending HTML

As mentioned, we believe that extending HTML still has a function in case one needs to provide graphical support for some notation, within a document that can be easily mapped onto plain HTML.

The following is an example of a simple HTML document with a simple extension for displaying text in reverse video:

```
<html>
<head>
</head>
<body>
<applet archive="displets.zip" code="DispletManager.class" width="450" height="46
<param name="def" value="

&lt;tag name='reverse' hasEndTag
                    nonNesting
                    src='example/reverse.class'&gt;
&lt;/tag&gt;  ">

<param name="cod" value="
&lt;body bgcolor=white&gt;
        &lt;H1 ALIGN='center'&gt;&lt;reverse&gt;HTML with extensions&lt;/reverse&

        &lt;p&gt;This document contains a rendering via &lt;I&gt;displets&lt;/I&g
        selection of some &lt;B&gt;common&lt;/B&gt; HTML tags. The &lt;reverse&gt
        &lt;/reverse&gt; extension to HTML is defined for text in reverse
        colors.&lt;/p&gt;

        &lt;HR size = 2&gt;

        &lt;FORM Action='forma' METHOD=POST &gt;
                Forms are also allowed:
                &lt;BR&gt;Your name: &lt;INPUT TYPE=TEXT NAME='name' size='20'&gt
                &lt;BR&gt;Your e-mail: &lt;INPUT TYPE=TEXT NAME='e-mail' size='20
                &lt;BR&gt;Coming from:
                &lt;INPUT name ='From' TYPE=RADIO value='Africa'&gt;Africa
                &lt;INPUT name ='From' TYPE=RADIO value='America'&gt;America &lt;
                &lt;INPUT name ='From' TYPE=RADIO value='Asia'&gt;Asia
                &lt;INPUT name ='From' TYPE=RADIO value='Australia'&gt;Australia
                &lt;INPUT name ='From' TYPE=RADIO value='Europa' CHECKED&gt;Europ
                &lt;BR&gt;&lt;INPUT TYPE=SUBMIT &gt; &lt;INPUT TYPE=RESET&gt;
        &lt;/FORM&gt;

&lt;/body&gt;
">
</applet>

</body>
```

Figure 2 shows how the document results on screen.

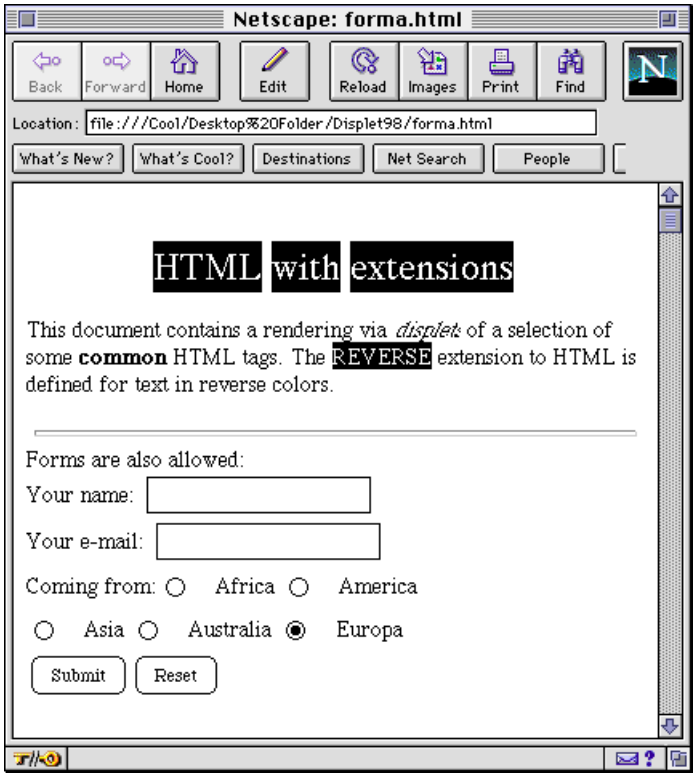

Fig. 2 - Rendering an HTML displet with an extension

The document, or just the part that contains the new element, is given as the parameter 'HTMLcode' to the DispletManager class. The standard HTML elements are all pre-defined, including links and form elements, and have their own displet classes. The 'def' parameter of the applet allows the definition of the new elements, providing some simple syntactical support and the URL of the displet code in charge of providing its rendering.

Upon loading the applet, the displet manager will start the HTML parser and patch it with the elements defined in the 'def' parameter. It will then parse the 'HTMLcode' parameter, verifying that the new elements are being used according to the given grammar. The 'def' parameter acts as a simplified DTD for the new tags: it allows to specify the names, attributes, inclusion rules and minimisation features allowed for the new elements, as well as the URL of the Java code containing the rendering applet.

The resulting tree then is examined and the proper displet for each node is activated. Each displet is required to produce a (list of) bitmaps of its content. Each displet may set parameters and wait for the displet of its sub-elements to return their bitmaps, before producing its own. For instance, the P displet sets some values (such as margins, line spacing, font and size) that may affect its sub-elements, waits for all of them to return their bitmaps (one for each word, because P combines whole words into lines) and then creates its own list of bitmaps (one for each line).

We are using a modification and update of a beta version of an old Sun HTML parser. Currently, the HTML displet package contains displets for all HTML 3.2 tags, except tables. Anchors and form elements are included and work correctly. This allows authors to use standard HTML elements within the special elements they define: for instance, it is possible to have styled text, form elements or hypertext links within a chart or a timeline.

4.2 Displaying XML documents with XSL styles

The same displet may be used for an XML document chunk. Since XML comes with no pre-defined rendering for any elements that is used, it is necessary to provide a map that connects the elements of the document to the displets that must be used for the display. We are using XSL for this.

The following is similar to the previous example of the reverse colour, but in XML:

```
<html><head>
        <title>Displet test - XML</title>
</head><body>

<applet code="DispletManager.class" width="500" height="200">
<param name="style" value="

&lt;xsl&gt;
&lt;rule&gt;
        &lt;target-element value='para'/&gt;
        &lt;css.div font-size='12'&gt;
                &lt;children/&gt;
        &lt;/css.div&gt;
&lt;/rule&gt;

&lt;rule&gt;
        &lt;target-element value='rev'/&gt;
        &lt;example.reverse&gt;
                &lt;children/&gt;
        &lt;example.reverse&gt;
&lt;/rule&gt;

&lt;/xsl&gt;  ">

<param name="XMLcode" value="

&lt;para&gt;This is an example of a text rendered in
&lt;rev&gt;reverse&lt;/rev&gt;&lt;/para&gt;

">
</applet>
</body></html>
```

The displet manager knows we are dealing with XML because of the presence of the 'XMLcode' parameter, and activates the correct parser. We are using the msxml parser by Microsoft , but plan to provide support for at least one more implementation. In order to associate the correct displet class with the relevant XML element, we are developing our own XSL processor.

According to the very early specification of XSL [ABC97], an XSL stylesheet is a XML document containing a list of style rules. Each rule is composed of a pattern selector and an associated action. For every element of the document to be displayed, the XSL processor should find the best matching pattern, and activate the associated action to provide the rendering. In our example, the selector allows to specify the actual name of the XML element to be processed. A default rule is provided that matches all patterns, so that every element can have an action to determine its display. The actions of an XSL rule are a representation of the structure of the rendering objects (in stylesheet parlance, the *flow objects*) that have to be created. The flow objects that are planned to be available in XSL rules derive either from CSS or from DSSSL. The ambiguity is due to the fact that there is a partial overlap in the meaning of CSS and DSSSL flow objects, and therefore the need may arise for using either model. Flow object elements may be combined with literal text, and elements generated procedurally by the use of *ECMAscript* scripts [ECMA97]. The `<children/>` element specifies that the sub-elements of the element being considered need to be recursively processed for determining the correct rendering of the element.

Our implementation of the XSL processor is prototypical and rudimentary, but features and functionalities are being added daily. In our interpretation of the proposed specification, the actions are names of displet classes. We plan to create both a CSS and a DSSSL package, that will properly subclass and tailor the appropriate internal classes of the rendering engine. Differently from the proposed specification, therefore, it is necessary to preface the name of the flow object with the name of the style mode: for instance, css.div or dsssl.paragraph instead of DIV in an exclusive CSS setting and paragraph in an exclusive DSSSL setting.

Furthermore, it is possible to create one's own package of classes, and use it liberally within an XSL specification, by appropriately specifying the full name of the classes. In our example, the 'reverse' action comes from neither CSS nor DSSSL, but from the 'example' package, and is thus appropriately specified. This allows special rendering classes, such as those for rendering non-textual data, to be used in an XSL specification.

To the time of this writing, only a few pattern selectors are available. As for the action part, we are currently limiting to the main CSS flow objects.

4.3 The rendering engine

The rendering engine used by the DispletManager applet is a set of Java classes that provide the rendering for the appropriate document elements. These classes are all subclasses of the DocElement class, which provides the framework of the rendering procedure.

All classes provide a `createBitmap()` method, whose purpose is to create and return the bitmap of the flow object of the considered markup element on the basis of the bitmaps of its sub-elements. The `createBitmap()` method is usually not seen by the implementer of new classes, and provides the following functionalities:

- an active drawing environment is managed. The drawing environment is a set of parameters that are used by the rendering methods of the classes in order to decide how to create the bitmaps. For instance, a paragraph-like class may set some parameters that will be used by itself, such as margins, line spacing, alignment, etc., and some that will be used by its sub-elements, such as font name, font size, font colour, etc. The `createBitmap()` method allows a displet to set its own attributes with the `setParams()` method, and restores the previous situation when the displet is finished. Since `createBitmap()` methods are activated recursively, this creates a stack that provides the proper parameters at any level of recursion.
- the rendering of sub-elements is managed. The presence/absence of the `<children/>` element in the XSL rule, or some internal decisions for the HTML displet, may cause or prevent the rendering of the sub-elements of the current element.
- the rendering of the element is managed. After the bitmaps of the sub-elements of the element have been created (if appropriate), the `createBitmap()` method calls the `render()` method, which creates the final bitmap (or set of bitmaps) that will be returned. Different classes will implement `render()` differently: for instance, the `render()` method of a block element will collect the bitmaps of its sub-elements in a vertical stack (one above the other), and provide a single bitmap of the whole element, while the `render()` method of a paragraph will collect its sub-elements side-by-side in lines of the given width, and provide a bitmap for every line it has created; this allow the element containing the paragraph to decide how much of the paragraph to display at a time (for instance, in case of scrolling).
- active elements are specified and created. Active elements are those that will need to re-act to user and system events after they have been displayed. For instance, form elements and anchors have an associated behaviour that is activated when the user selects them.

Figure 3 shows the inheritance structure of the classes of the module library:

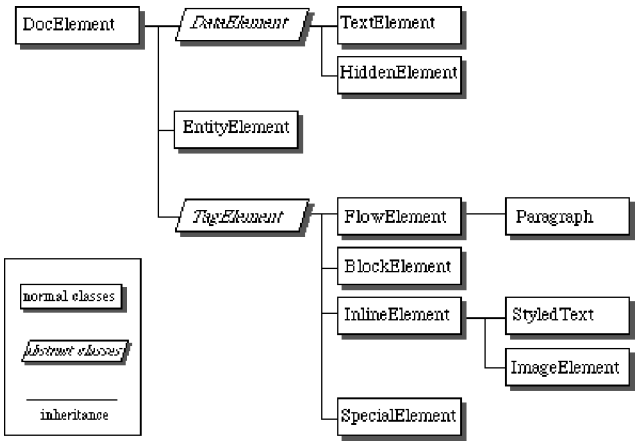

Fig. 3 - The inheritance structure of the module library

DocElements can either be data, entities or tag elements. DataElement classes are used for the content of markup elements, i.e., #PCDATA in SGML and XML DTDs. They can either be text or hidden elements. EntityElements are provided for the management of XML and HTML entities such as & or the definition of new ones. TagElements are used for the creation of the structure flow objects of the document: they are either flow objects, block objects, inline elements or special elements.

- A block element is a single object that stands alone in the vertical layout of the document. Paragraphs or tables are block elements. A flow element is a block element that is built piecemeal: while plain block elements are built from start to end before the `createBitmap()` returns, flow elements build each of their sub-element and return, and are called as many times as there are sub-elements. This allows long and complex elements to be rendered only for the possibly small section that is actually displayed. For instance, HTML and BODY are considered flow elements, so that the display of an HTML document can start as soon as the first object is completed, and be interrupted when the available display space is filled.
- Inline elements are elements that can be put side by side with their siblings. Inline elements are used within block elements and may be text-based, images or something else. The StyledText class allows the specification of text runs of arbitrary styles. Inline elements specify the places where they can be broken by creating as many bitmaps as break points. This allows the containing paragraph or block element to determine where the line should be broken.
- Special elements are completely tailorable. While in the previous classes displet programmers can only overload the `setParams()` and `render()` methods, here all methods are overloadable, and can be customised.

As an example, this is the complete source code of the 'reverse' displet:

```
package example ;
import displet.* ;

public class reverse extends StyledText {

        public void setParams(StyledTextParams p) {
                Color c = p.fgColor;
                p.fgColor = p.bgColor ;
                p.bgColor = c ;
        }
```

```
}
```

The reverse displet is a subclass of the StyledText, which is a subclass of the InlineElement class. These are classes for text-based objects that behave as in-line elements (e.g. bold, italic, etc.). As it can be seen, the programmer of such a displet only has had to specify a parameter and have the `render()` method of its superclass handle all the detail. A more complex displet is shown in the following section.

4.4 The example for Z

As a proof of concept for our rendering engine, we have developed a full set of displets for the management of Z specifications. Z [Spi92] is special notation that uses mathematical and logical elements for the specification of software systems. Z also uses a special set of symbols and graphical boxes that makes it impossible to specify using existing markup systems or even the foreseen MathML [IM97].

Software designers needing to share Z specifications have used so far special LaTeX extensions, but could not till recently use it on the Web, except by creating GIF images. The Z browser [MVHH95] is a plug-in for Netscape designed so that it is possible to show Z specifications in an HTML document, although with the clear limitations that plug-ins imply.

Since an SGML DTD for Z has existed for the last 5 years, we have decided that the proof of concept for our rendering engine would be the implementation of Z specifications both by extending HTML and by specifying a DTD chunk for XML. Z specifications are documents of mixed free text and formal specifications. Schemas are the most common graphical objects in Z formal specifications. A schema is a graphical box composed of a declaration part, where variables are declared, and an axiom part, containing predicates that must hold for those variables. An example schema is shown in figure 4.

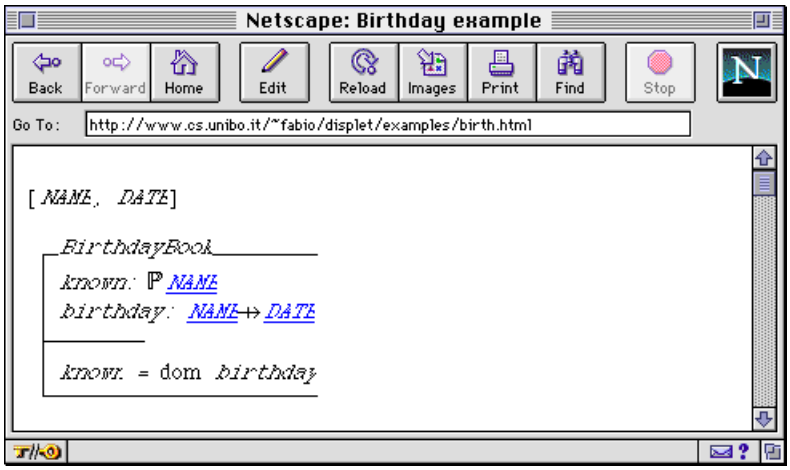

Fig. 4 - A displet for Z schemata

The preceding schema has been created with displets extending HTML. The following is the markup for the schema in figure 4:

```
<html><head>
        <title>Birthday example</title>
</head><body>

<applet code="DispletManager.class" width="300" height="700">
        <param name="def" value="
```

```
&lt;tag name='givendef' hasEndTag
                    nonNesting
                    src='zpack/givendef.class'&gt;
&lt;/tag&gt;

&lt;tag name='schemadef' hasEndTag
                     nonNesting
                     src ='zpack/schemadef.class'&gt;
        &lt;attr name='id' value=cdata &gt;
        &lt;attr name='group' value=nmtoken &gt;
        &lt;attr name='style' value='vert, horiz' &gt;
        &lt;attr name='purpose' value='state, operation, datatype' &gt;
&lt;/tag&gt;

&lt;tag name='decpart' hasEndTag
                   nonNesting
                   in ='schemadef'
                   src ='zpack/decpart.class'&gt;
&lt;/tag&gt;

&lt;tag name='axpart' hasEndTag
                  nonNesting
                  in ='schemadef'
                  src ='zpack/axpart.class'&gt;
&lt;/tag&gt;

&lt;tag name='declaration' hasEndTag
                       nonNesting
                       in ='decpart'
                       src ='zpack/declaration.class'&gt;
&lt;/tag&gt;

&lt;tag name='predicate' hasEndTag
                     nonNesting
                     in ='axpart'
                     src ='zpack/predicate.class'&gt;
        &lt;attr name='label' value=cdata &gt;
&lt;/tag&gt;

&lt;entity name='pset' data='#185' font='zpack/zfont.14.gif'&gt;
&lt;entity name='pfun' data='#193' font='zpack/zfont.14.gif'&gt;
&lt;entity name='dom' data='dom' font='TimesRoman%14%Plain'&gt;

">

<param name="cod" value="

&lt;body&gt;
&lt;givendef&gt;
        &lt;a name='name'&gt;NAME&lt;/a&gt;,
        &lt;a name='date'&gt;DATE&lt;/a&gt;
&lt;/givendef&gt;
&lt;p&gt;
&lt;schemadef&gt;
        BirthdayBook
        &lt;decpart&gt;
                &lt;declaration&gt; known: &amp;pset;
                                 &lt;a href='#name'&gt;NAME&lt;/a&gt;
                &lt;/declaration&gt;
                &lt;declaration&gt; birthday:
                                 &lt;a href='#name'&gt;NAME&lt;/a&gt; &amp;pfun;
                                 &lt;a href='#date'&gt;DATE&lt;/a&gt;
                &lt;/declaration&gt;
        &lt;/decpart&gt;
        &lt;axpart&gt;
                &lt;predicate&gt;known = &amp;dom; birthday&lt;/predicate&gt;
        &lt;/axpart&gt;
```

```
&lt;/schemadef&gt;
&lt;/body&gt;
">
</applet>
</body>
</html>
```

We have created a complete package of Z displets providing support for the whole notation set: entities and tags closely reflect those of the Z DTD as published in [Ham95]. A conversion utility for Z specifications in LaTeX to the corresponding HTML ones has also been created.

5 Conclusions

HTML developers cannot cope with all the possible extensions that are proposed or deemed useful, and, while XML does have the expressive power that is necessary for most, if not all, the special notations that are being requested, its developers are focused in providing typographical text-based control leveraging on existing stylesheet languages.

The situation for displaying non-textual data using markup languages is grim: special interest groups, besides working on a common markup language, are usually forced to work on a full browser to display correctly the special notations. Other kinds of non-textual information, not being defended by special interest groups (graphs, charts, timelines, vector graphics, etc.) clearly lag behind.

Displets provide a useful and extensible mechanism for providing rendering of non-textual data in a extensible markup language. The displet manager applet, loading the appropriate parser, can deal with per-document extensions to HTML, and with arbitrary flow objects for XML documents.

A displet site is being created at http://www.cs.unibo.it/~fabio/displet.html, which contains the code for the rendering engines, examples for both HTML and XML, and a list of all the displets we have created so far.

Work on the displet manager is continuing. Our future aims are at creating a complete XSL engine, and to extend the displet manager to handle positionable and scriptable elements. Several examples of displets are being produced as well, many of which are not typographical in nature, and are thus beyond the capabilities of current stylesheet languages.

Acknowledgements

We would like to acknowledge the help and contribution of Stefano Pancaldi, and the help and suggestions of Michael Bieber and Chao-Min Chiu, the co-authors of the first Displet paper.

This work has been carried out with the support of the Italian Center for Research (CNR) and of the Ministery for the University and the Scientific and Technologic Research (MURST)

References

[ABC97]
S. Adler, A. Berglund, J. Clark, etc. (eds.), Extensible Style Sheet Language (XSL) Version 0.2, Unofficial Committee Draft 10 oct 97, http://sunsite.unc.edu/pub/sun-info/standards/xsl/0.2/xsl-19971010.sgm

[BD97]
T. Bray, S. DeRose (eds.), Extensible Markup Language (XML): Part 2. Linking, W3C

Working draft, http://sunsite.unc.edu/pub/sun-info/standards/xsl/0.2/xsl-19971010.sgm

[BOS96]
J. Bosak (ed.), DSSSL Online Application Profile, 1996.08.16, http://sunsite.unc.edu/pub/sun-info/standards/dsssl/dssslo/do960816.htm

[BSM97]
T. Bray, C.M. Sperberg-McQueen (eds.), Extensible Markup Language Version 1.0, W3C Working draft, http://www.textuality.com/sgml-erb/WD-xml-lang.html

[BV97]
M. Bieber, F. Vitali, Toward Support for Hypermedia on the World Wide Web, in IEEE Computer, 30(1), January 1997, p. 62-70.

[BVABO97]
M. Bieber, F. Vitali, H. Ashman, V. Balasubramanian, H. Oinas-Kukkonen, "Fourth Generation Hypertext: Some Missing Links for the World Wide Web", in International Journal of Human-Computer Studies, Academic Press, 1997(41), p. 31-65

[DOM97]
Document Object Model Specification, W3C Working Draft 09-Oct-1997, http://www.w3.org/TR/WD-DOM

[ECMA97]
Standard ECMA-262 ECMAScript: A general purpose, cross-platform programming language (June 1997), http://www.ecma.ch/stand/ecma-262.htm

[Fly97]
P. Flynn (ed.), Frequently Asked Questions about the Extensible Markup Language (the XML FAQ), http://www.ucc.ie/xml/

[Ham95]
J. Hammond, The Z Interchange Format, in J. Nicholls (ed.), The Z notation, Appendix E, 1995. http://www.comlab.ox.ac.uk/archive/z.html

[IM97]
P. Ion, R. Miner (eds.), Mathematical Markup Language, W3C Working Draft, 10-Jul-1997, http://www.w3.org/pub/WWW/TR/WD-math-970710

[LB96]
H. Lie, B. Bos, Cascading Style Sheets, level 1, W3C Recommendation 17 Dec 1996, http://www.w3.org/pub/WWW/TR/REC-CSS1

[MR96]
P. Murray-Rust, Chemical Markup Language, http://www.venus.co.uk/omf/cml/

[MVHH95]
L. Mikusiak, V. Vojtek, J. Hasaralejko, J. Hanzelova, Z Browser: A Tool for Visualization of Z Specifications, In: Bowen, J. P. (ed.); Hinchey, M. G. (ed.), ZUM '95: The Z Formal Specification Notation Proceedings. Lecture Notes in Computer Science 967 p. 510-526 (1995).

[SGML]
International Standards Organisation, ISO 8879, Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML)

[Sie97]
D. Siegel, The balkanization of the Web, http://www.dsiegel.com/balkanization/

[SMG94]
C. M. Sperberg-McQueen, R. F. Goldstein, HTML to the Max: A Manifesto for Adding SGML Intelligence to the World-Wide Web, Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web, http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Autools/sperberg-mcqueen/sperberg.html

[Spi92]
J. M. Spivey, *The Z Notation, a reference manual* (2nd edition), Prentice Hall, 1992

[VCB97]
F. Vitali, C. Chiu, M. Bieber, Extending HTML in a principled way with *displets*,

Proceedings of the VI WWW International Conference, Computer Networks and ISDN Systems (29)08-13 (1997) pp. 1115-1128

URLs in the paper

- WebEQ: http://www.geom.umn.edu/software/WebEQ/,
- IBM techexplorer: http://www.ics.raleigh.ibm.com/ics/techexp.htm,
- MathType: http://www.mathtype.com/
- Amaya: http://www.w3.org/TR/NOTE-amaya-970220.html
- JUMBO: http://www.venus.co.uk/omf/cml/download.html
- Microsoft XML parser: http://www.microsoft.com/standards/xml/xmlparse.htm
- Displet site: http://www.cs.unibo.it/~fabio/displet.html

Vitae

Paolo Ciancarini is Associate Professor of Computer Science at the University of Bologna. His research interests include: coordination languages and systems, programming systems based on distributed objects, formal methods in software engineering, and markup languages.

Alfredo Rizzi has recently graduated in Computer Science at the University of Bologna and currently works as software developer in a private firm. He is interested in object oriented programming languages and WWW technologies.

Fabio Vitali is Assistant Professor of Computer Science at the University of Bologna. His research interests include: hypertext systems, versioning systems, collaborative systems, markup languages, digital typography, and coordination systems.