Managing Complex Documents Over the WWW: A Case Study for XML

Paolo Ciancarini, *Member*, *IEEE Computer Society*, Fabio Vitali, and Cecilia Mascolo

Abstract—The use of the World Wide Web as a communication medium for knowledge engineers and software designers is limited by the lack of tools for writing, sharing, and verifying documents written with design notations. For instance, the Z language has a rich set of mathematical characters, and requires graphic-rich boxes and schemas for structuring a specification document. It is difficult to integrate Z specifications and text on WWW pages written with HTML, and traditional tools are not suited for the task. On the other hand, a newly proposed standard for markup languages, namely XML, allows one to define any set of markup elements; hence, it is suitable for describing any kind of notation. Unfortunately, the proposed standard for rendering XML documents, namely XSL, provides for text-only (although sophisticated) rendering of XML documents, and thus it cannot be used for more complex notations. We present a Java-based tool for applying any notation to elements of XML documents. These XML documents can thus be shown on current-generation WWW browsers with Java capabilities. A complete package for displaying Z specifications has been implemented and integrated with standard text parts. Being a complete rendering engine, text parts and Z specifications can be freely intermixed, and all the standard features of XML (including HTML links and form elements) are available outside and inside Z specifications. Furthermore, the extensibility of our engine allows any additional notations to be supported and integrated with the ones we describe here.

Index Terms—Document management systems, hypertext, active documents, XML, Java, specification documents, Z notation.

1 Introduction

In recent years, we have seen the World Wide Web being slowly transformed from an environment for sharing documents and data among members of specialized communities (be they scientific, research, artistic, social ones, etc.) to a general-purpose new medium for advertising and marketing commercial enterprises to the public at large. Since commercial use has a larger impact and therefore power on the advances of the medium, the specific needs of specialized communities have been overlooked in the further development and advances of this new medium.

For instance, the use of the WWW as an environment for software design introduces new problems and challenges: The use of the WWW to support software process workflows, sharing specification documents, allowing to read and write them, and providing hypertextual links among documents is felt as a hot topic [18], [27], but little specific aid to software designers is available on the WWW at large.

A very important need that many communities of engineers have is the support for special notations that are current or even absolutely necessary within that community. Currently the Web is very poor in supporting special notations. The typographical rendering of WWW documents is usually defined using the HTML markup language; currently, it is the basis of most intranet document management systems [2], [26]. In its many versions, HTML provides textual support for elements such as input fields, buttons, choice lists, etc. along with structural and formatting commands for text within the data format of network documents, and, of course, the hyperlinking capabilities that gave it its name.

It has been extremely important that HTML allowed both complex interfaces and proper and traditional text content to be described in ASCII-based source documents. HTML has shown the way that text-based support for nontextual content eases understanding, tool creation and debugging of applications that deal with it. Furthermore, they allow a complete intermix of different concerns, such as interface elements and text characteristics, thereby fostering the creation of complex interfaces that are at the same time rich in content and sophisticated in their interaction with the user.

On the other hand, HTML is limited in that it has only a small set of allowable elements, that is, only 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 needs to be used: Either the existing tags are abused for a different purpose than that for which they were designed, or an image is used, or a Java applet is created providing the desired functionality.

These kludges have obvious and well known drawbacks, that have lead to the development of many alternative (and partial) solutions. For instance, Cascading Style Sheets (CSS) [21], [3] allow authors to separate the efforts to specify special graphic effects and the structure and determination of the actual content of the

• *P. Ciancarini, F. Vitali, and C. Mascolo are with the Department of Computer Science, University of Bologna, Mura Anteo Zamboni, 7–40127 Bologna, Italy. E-mail: {ciancarini, vitali}@cs.unibo.it.*

Manuscript received 15 May 1998; revised 28 Sept. 1998.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 109084.

1041-4347/99/$10.00 © 1999 IEEE

document, allowing complex typographical rendering to be built on top of still readable plain HTML documents. XML [6] is another tentative in that direction: Instead of forcing authors to the limited and closed set of predefined elements, XML is a metamarkup language that allow authors to define their own sets of markup elements that are most appropriate to the specific class of documents they are dealing with. Adjunct languages (XSL, XPointer, and XLink) are used by authors to associate these elements to some rendering or linking semantics for their display on paper or screen. This allows a definitive separation between the description of structures and roles of the documents and the description of their graphical rendering on a computer terminal or on a high resolution printer.

Neither solution is currently completely satisfying for supporting specialized notations because both are only concerned with supporting text-oriented content only. Many notations have sophisticated need that go well beyond texts. For instance, specification languages like Z [29] are often based on specialized notations (mathematics and logic symbols); it would be useful to be able to give a visual interpretation of these symbols and to allow them to be displayed on WWW pages.

The purpose of this paper is to report on a Java rendering engine for XML data that we have implemented. The engine allows standard typographical support for text-oriented XML documents, as well as extensible graphical support for additional needs, in particular for specialized notations. We have created a complete graphical and typographical support for formal specification documents written in Z. The rendering engine we are describing works as a completely autonomous applet inside unmodified Java-enabled browsers such as Netscape Communicator or Microsoft Internet Explorer.

The paper is structured as follows: In Section 2, we summarize the state of the art of the rendering of Z documents as hypertexts. In Section 3, we describe the idea of extending HTML with Java using the *displets* concept. Section 4 explains the implementation details of our Z browser, whereas Section 5 describes the browser from the point of view of the author. In Section 6, we draw some conclusions and sketch our future work.

2 Creating Z Specifications

2.1 Writing, Printing, and Visualizing Z Specifications

Several tools exist to this date to help software designers to write, test, and share documents containing their Z specifications. A complete guide to all the existing tools for Z can be found in the site http://www.comlab.ox.ac.uk/archive/z.html.

We can divide the available tools into four main categories: Fonts, browsers, editors, and type checkers.

True Type fonts for Z are available to use with common word processors on many platforms including Windows and Macintosh, but fonts of course only give access to the special mathematical characters of the Z language, forcing users to use nonspecific features of available tools to create the graphic boxes of schemata and other Z elements.

Customizable formatters such as LaTeX [20] are the most common tools to write Z specifications. General style files for LaTeX, such as oz.sty, fuzz.sty, ztc.sty, have been published to precisely render Z specifications.

Logica has created a syntax-driven WYSIWYG editor for Z on MS Windows platforms. Such an editor also integrates a type checker and forces the production of well-formed Z specifications by providing facilities for building, editing, checking, and viewing Z specification documents. Being WYSIWYG, the editor can display the Z constructs and symbols as they would appear on a printed page.

This paper [25] describes the Z Browser, an application for displaying Z specifications running on MS Windows. Such a tool is aimed at Z novices, and is integrated with a complete help system for Z grammar and notation, thus it supports the construction, syntactical check, and visual layout of Z documents.

Several analysis tools also exist for Z specifications. For instance, CadiZ [16] is an integrated suite of tools for creating Z documents. It understands source files in LaTeX and Word for Windows, and can visualize implicit Z expressions (i.e. schema calculi) by showing their expansions.

Finally, the ZTC [15] type checker accepts LaTeX-formatted Z specifications as well as text-based ones. ZTC also suggests using a special syntax based on concatenation of ASCII characters for mathematical symbols.

In summary, it is clear that Z is a highly structured notation both graphically and semantically complex, and that writing, checking, and displaying Z specification documents is yet an unsolved issue.

2.2 Hypertext and Z Specifications

There are several good reasons to provide hypertext functionalities to Z specifications. A complex specification is intrinsically composed of many connected chunks (schemas, etc.) that refer to each other in a peculiar, often unpredictable way. Furthermore, the idea of literate programming [19] requires that schemas and texts interleave freely, so that the reader is provided with a narrative explanation of the most complex schemas, and a formalized and exact specification of vaguer descriptions. These remarks naturally call for a hypertext solution.

Moreover, collaboration and sharing are even better reasons for providing hypertext support to Z specifications: Formal specifications are but one step in the complex process of system design, verification, and implementation [11]. Modern development processes are enacted by teams of people that cooperate, interact, and discuss. Being able to create, access, and verify formal specifications within the usual tools of our everyday work, publish them, connect them to the other deliverables of the design and implementation processes would allow a tighter integration between formal design and actual implementation [8].

Until recently, Z specifications could only be visualized on the WWW by creating images in one of the supported inline formats, such as GIF. This leads to a very cumbersome and unnatural creation process, since the Z specifications have to be created in a different environment than the text, and furthermore nonspecialized graphic editors have to be used and restrained in order to produce graphically acceptable schemas. It is also a very unnatural and clumsy way of accessing to the information: An image of a schema is a completely opaque object, where the subparts, the texts, the formulas are completely inaccessible; it is a bitmap that cannot be further processed because the content and meaning have been lost. The content of a schema cannot be searched, the specifications cannot be indexed, analyzed or verified.

A first attempt to show Z specifications on the WWW was described in [24], designing a plug-in for Netscape and Internet Explorer that accepts Z specifications written using one of the existing LaTeX styles.

Although this approach is very original it has two main limitations: First, visualizing Z documents requires the availability of the plug-in, which is architecture-dependent (it only exists for MS Windows). Secondly, the LaTeX format is alien to the available SGML-based formats suggested for the WWW: In fact, writing Z schemas in LaTeX requires a different syntax and approach than writing the surrounding free-flow text in HTML, and the specifications live independently of the host document. The first problem has been addressed: The Z browser is becoming a Java applet, which is architecture-independent and can be run on most computers of the current generation.

We know that also Bowen and others are working on a Java applet to visualize Z schemas [4]. Our approach, detailed in Section 5, is related but with relevant differences.

2.3 Advantages of Markup Languages

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 ([28]) 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 prespecified 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 [28], that is, a class of documents conforming to the SGML Document Type Definition (DTD) that describes "HTML documents." SGML, being a metalanguage describing classes of documents rather than one 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 rhetoric need, however complex and arcane.

Unfortunately, SGML is considerably more complex to learn and design documents with than HTML, and it has been felt that this would prevent its generalized adoption. Therefore the SGML working group of the Word Wide Web Consortium was asked to develop a new mark-up metalanguage, namely the Extensible Markup Language (XML) [6], to 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.

There are several standards being developed within the XML framework: The most important is XML itself, a metamarkup language that allows user to create their own set of elements for their class of documents. XPointer [23] and and Xlink [22] extend HTML linking mechanism by providing external specifications of locations, multiple links, external links, etc. XSL [10] associates rendering behavior (e.g., character and paragraph settings) to XML elements through a mapping and rewriting language. XML-Namespace [5] allows elements coming from different namespaces (document types, for instance) to live together in the same document. Very important is MathML [14], a markup language for mathematics, formerly part of the unborn HTML 3.0 and subsequently detached in an autonomous standard finally converted into XML. MathML covers most needs for mathematical rendering, and is capable of showing most of the strange glyphs that are part of the Z language, but is not thought for Z and does not provide support for other, more specific needs of the Z notation.

Interestingly, in the Z community an SGML-based language for Z specifications already exists: The Z Interchange Format (ZIF for short) [7] defines a portable representation of Z, that can be used by all tools supporting SGML. The ZIF is basically a Document Type Definition (DTD), namely an SGML specification defining the syntax of documents that contain Z specifications. In [12], a study of the usage of the ZIF was presented, according to which ZIF can be fruitfully used to create editors for Z documents using standard SGML tools, and that Z specifications encoded using ZIF could easily be included in other SGML documents.

XML documents are valid SGML documents. Most existing SGML DTDs can be used with no modifications in an XML environment. Notably, the Z Interchange Format is one of such DTDs.

It is, therefore, possible to use the definitions specified in the ZIF within XML tools, in order to create web-friendly visualizations of Z specifications. Alternatively, XML tools allow the HTML tag set to be described and extended as needed. By joining the HTML DTD with the ZIF DTD, and producing a capable browser, it is possible to write HTML documents that contain Z specifications as markup items, instead of images, thereby keeping all the useful properties that markup has over bitmaps.

In this paper, we report about one such tool, that allow the display of text-based XML documents enriched with Z specifications. This mechanism can obviously be extended to handle the display of any kind of notation within a XML document.

3 Displets and Markup Languages

Displets were proposed in [30] 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 metamarkup language such as XML (see Section 2), HTML extended with displets could allow all kinds of specialized notations and graphical effects while at the same time leveraging over the existing and well-known set of elements defined by HTML.

Our first experiment with rendering arbitrary, nontext-based markup extensions [30] was to modify an existing browser to allow the parsing and the visualization 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 visualization of the elements).

In [9], on the other hand, we reported about the DispletManager applet, a general, extensible rendering and 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 have been:

- That it must be possible to create special code for rendering arbitrarily odd data types, in particular nontextual data (*displets*).
- That 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.
- That the rendering engine must work both for extended HTML and for straight XML, and the displet classes must be identical.

Fig. 1 shows 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 recursively (depth-first) analyzed: The appropriate displet classes are activated to create the rendering (i.e., the display object) of their element on the basis of the rendering of their subelements. No class is allowed direct access to the screen: On the contrary, each displet creates a (set of) off-screen bitmap(s) that its ancestor can pass, ignore, modify, or add to.

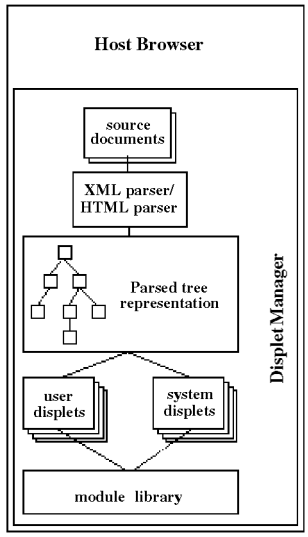

Fig. 1. The general structure of the DispletManager applet.

Several specialized browsers exist for XML-based special notations. For instance, WebEQ for MathML [31] and Jumbo for CML, a XML-based notation for chemical data [17]. Although a specialized browser would have probably been more efficient and sophisticated for Z elements, too, we felt that a general rendering engines for all kinds of notation was preferred, leading us to a more general and extensible architecture for Z and other needs.

3.1 Applying Displets to XML Documents

The XML language allows authors to define their own set of elements (tags) to structure and organize their documents. Of course these elements do not have a predefined meaning, nor even a predefined visualization. For instance, while it is known that the "H1" element in HTML has both a structural role (the heading of a first level section) and a graphical rendering (use a large font and align it on the left), a corresponding "major-heading" element in a XML document would have no machine-understandable structural role (but this is not a problem), nor a known graphical rendering (we cannot even determine whether the element is a block, a paragraph or an inline element).

The XSL [10] language is used for associating rendering information to an XML document. Each XML document that needs to be displayed on screen or on a printer would have a XSL document associated. The XSL document contains a series of "rules" mapping the XML elements of the document to one or more *flow objects* (i.e., graphical objects such as blocks, paragraphs and inline texts).

There are two competing syntax for XSL stylesheets: The first is based on a very early proposal for XML stylesheets [1], and is the one implemented on most applications and discussed in most books about XML (for instance, [13]). The second one, a W3C Recommendation, came out recently

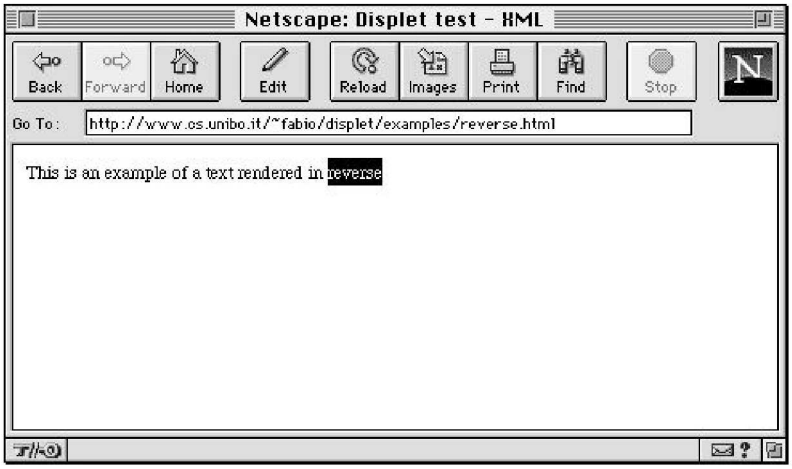

Fig. 2. Rendering a simple displet.

(August 1998, [10]) and still very little support can be found for it. Our terminology, discussion and implementation relies still on the first version.

Although in XSL the set of available flow objects is fixed, we allow the specification of new flow objects, that can be specified in the rules just like the standard ones. Each flow object corresponds directly to a displet class.

What follows is an example of a simple XML document contained in the DispletManager applet and its associated XSL style rules. The style sheet refers to two flow objects: A standard paragraph object (belonging to the CSS family of flow objects available within the standard XSL proposal), and a special "reverse" flow object that is prepared as a displet by the author of the document:

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

<xsl>
<rule>
      <target-element value='para'/>
      <css.div font-size='12'>
           <children/>
      </css.div>
</rule>

<rule>
      <target-element value='rev'/>
      <example.reverse>
           <children/>
      </example.reverse>
 </rule>

 </xsl> ">

 <param name = "XMLcode" value="

 <para>This is an example of a text
      rendered in
 <rev>reverse</rev></para>

 ">
 </applet>
```

The DispletManager applet for XML has two arguments: The first contains the style sheet document according to the XSL rules, while the second one contains the XML document that has to be displayed, using the elements that are described in the XSL style sheet associated.

Upon loading the applet, the displet manager will start the XSL engine and read in the 'style' parameter. This is parsed (by a XML parser, because it is itself a XML document) and organized. Then the XMLcode parameter is read and parsed by the same XML parser, creating a tree of elements and data.

The XSL engine will then match each element in the XML document with the *pattern* contained in each XSL rule. When the most suitable match has been found, the rest of the rule (the *action* part) is executed, creating the flow objects listed and feeding them their content (usually the rendering of their subelements, as specified by the $< children/ >$ tag).

In this example, the 'para' element of the XML document matches the first XSL rule, triggering the creation of a 'div' object of the standard CSS package (a paragraph) with a specific parameter (font-size = 12), fed with the children of the element (i.e., the words and the elements contained within the para tags). Then the 'rev' element is considered, and matched to the second rule of the stylesheet, triggering the creation of a 'reverse' object belonging to the 'example' package, fed with its content. As soon as the rendering of its content has been readied (by creating the necessary bitmaps), the displet class corresponding to the flow object is activated.

Each displet will then produce a (list of) bitmaps of its content. For instance, the 'div' displet of the CSS package will set a few parameters (such as margins, line spacing, font, and size) that may affect its subelements, wait for the XSL engine to return control after its content has been readied, and build its own content by combining the bitmaps of each word into lines according to the given constraints. Fig. 2 shows the above mentioned document results on screen:

4 The Rendering Engine

The rendering engine used by the DispletManager applet consists of 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 subelements. 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 subelements, such as font name, font size, font color, 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 recursively activated, this creates a stack that provides the proper parameters at any level of recursion.
- The rendering of subelements is managed. The presence/absence of the element in the XSL rule may cause or prevent the rendering of the subelements of the current element.
- The rendering of the element is managed. After the bitmaps of the subelements of the element have been created (if appropriate), the `createBitmap()` method calls the `render()` method, which in turn 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 subelements 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 subelements 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 react to user and system events after they have been displayed. For instance, form elements and anchors have an associated behavior that is activated when the user selects them.

Fig. 3 shows the inheritance structure of the classes 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 subelement and return, and are called as many times as there are subelements. 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 customized.

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 (eg. bold, italic, etc.). As it can be seen, the programmer of such a

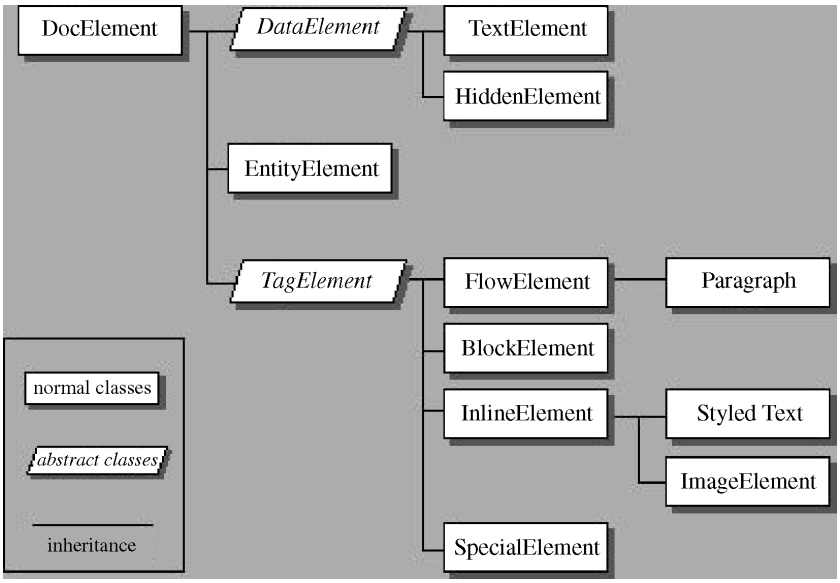

Fig. 3. The inheritance structure of the module library.

displet only has had to specify a parameter and have the `render()` method of its superclass handle all the details. The displets for showing Z specifications are shown in the following section.

5 The Z Browser

The main extension to HTML we have considered using displets is the implementation of the complete ZIF DTD. Authors writing Z specifications can create documents containing their Z specifications in a markup language similar to HTML and completely intermixable with plain text and other HTML features such as links, tables, etc.

The ZIF format defines several elements (tags) for the building blocks of the language, such as schemas, definitions, etc., and several entities (literal macros) for the special characters inherited from mathematics and logics. Each element is implemented by a displet that creates a bitmap where the content of the element is appropriately composed and the graphical elements such as boxes, lines, etc. are then added. Entities on the other hand are elements of a graphical alphabet that is contained in a single GIF image and is loaded with the displets.

The following is an example of a Z schema using the Z Interchange format:

```
<givendef>
   NAME, DATE
</givendef>
<schemadef>
   BirthdayBook
   <decpart>
      <declaration> known: &amp;
         NAME</declaration>
      <declaration> birthday: NAME &pfun;
         DATE </declaration>
   </decpart>
   <axpart>
      <predicate>known = &dom;
            birthday</predicate>
   </axpart>
</schemadef>
```

A schema is defined by a tag called schemadef, which contains three elements: the name of the schema, a declaration part, and an axiom part. The declaration part contains one or more declarations, and the axiom part contains zero or more predicates. Appropriate ordering and nesting of elements is enforced by the DTD, and is checked when parsing the document. The notations "&pset;", "&pfun;", and "&dom;" are three entities (respectively, the partial set symbol, the partial function symbol, and the domain symbol) that will be substituted by the corresponding element in the graphical alphabet containing all the relevant Z symbols. The displet manager can appropriately show document bits as the previous one in a WWW browser.

Since many Z specifiers use LaTeX to produce their Z documents, we have developed an off-line translator called "Zed2XML" that transforms Z specifications written in LaTeX using style `oz.tex` into a corresponding HTML document with the appropriate extension.

For instance, given the following Z specification (the basic birthday book example from [29]):

$[NAME, DATE]$

$$\begin{array}{|l} BirthdayBook \\ \hline known : \mathbb{P}\, NAME \\ birthday : NAME \pfun DATE \\ \hline known = \mathrm{dom}\, birthday \\ \hline \end{array}$$

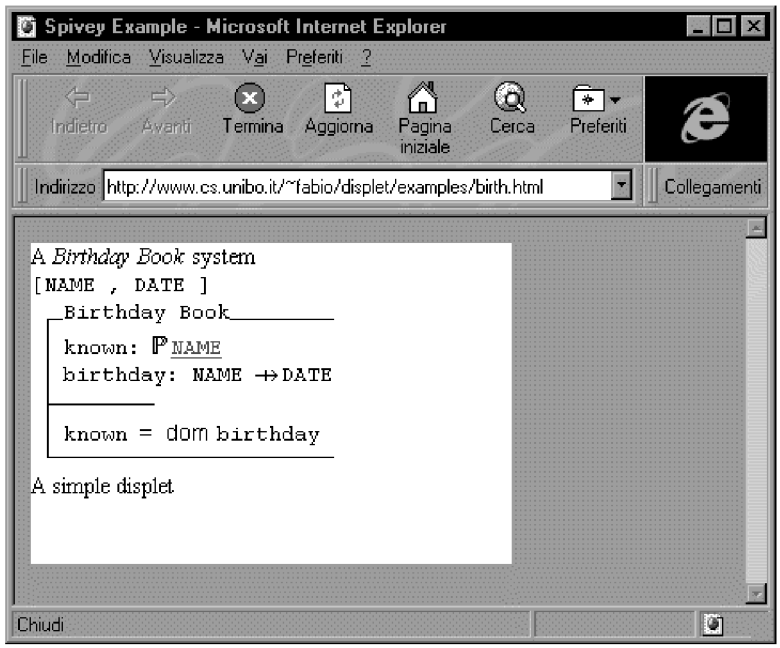

Fig. 4. Visualization on the WWW of a Z schema.

corresponding to the following LaTeX source document:

```
\documentclass[Italian, 12 pt, twoside,
   openright]{report}
\usepackage{amsfonts}
\usepackage{oz}

\begin{document}

\begin{zed}
[NAME, DATE]
\end{zed}

\begin{schema}{BirthdayBook}
known: \power NAME\\
birthday: NAME \pfun DATE
\where
known = \dom birthday
\end{schema}
\end{document}
```

The Zed2XML application transforms the previous LaTeX example in the corresponding applet specification:

```
applet archive="displet.zip"
      code="XMLManager.class"
      width=450 height=200>
<param name = "XMLcode" value="

<givendef>
   <a name="name">NAME</a>,
   <a name="date">DATE</a>
</givendef>
<schemadef>
   BirthdayBook
   <decpart>
      <declaration>
         known: pset;
            <a href="#name">NAME</a>
      </declaration>
      <declaration>
         birthday:
               <a href="#name">NAME</a>
               pfun;<a href="#date"> DATE</a>
      </declaration>
   </decpart>
   <axpart>
      <predicate>
         known = dom; birthday
      </predicate>
   </axpart>
</schemadef>

"> <param name = "style" value="
<xsl>
   <import name="htmlcss.stl"/>
   <import name="Zpackage.stl"/>
</xsl>

"></applet>
```

The output of Zed2XML is the HTML specification of the *DispletManager* applet. As it can be seen, we are following the ZIF format quite strictly. For the sake of brevity and reusability, standard stylesheets are used and invoked by a simple import command in the specification of the applet.

The 'htmlcss.stl' document contains the XSL rules to use HTML elements within XML documents. For instance, we are using here HTML links with the A tag. This is the relevant excerpt from the 'htmlcss.stl' document:

```
<rule>
   <target-element type="a">
      <attribute name="href" value="%2"/>
   </target-element>
   <css.a href="%2">
      <children/>
   </css.a>
</rule>
```

The 'Zpackage.xsl' document contains the XSL rules to use the Z displets within XML documents. This is an excerpt from this stylesheet:

```
<rule>
   <target-element type="givendef"/>
   <zpack.givendef>
      <children/>
   </zpack.givendef>
</rule>

<rule>
   <target-element type="schemadef">
   <zpack.schemadef>
      <children/>
   </zpack.schemadef>
</rule>
```

When run on a WWW browser, the previous documents is shown as in Fig. 4.

We remark that all displets integrate with each other and can refer to each other freely. In our case, the Z schema contains a hypertext link described in a different package. Z elements and plain text based XML elements freely intermix: It is possible to put standard HTML tags within Z schemas; for instance an author may require that some declarations of a schema are written in bold. The Zed2XML translator automatically connects types used in declarations to their definitions using plain HTML links. The author may freely add or modify the available links and HTML features, and include additional HTML elements, as well as native XML elements or elements belonging from other packages of displets.

6 Conclusions

We have presented a tool for visualizing Z specifications on the WWW: It fits every browser and every platform. The tool is based on XML displayed through "displets."

The advantage of having a Z browser running on all platforms is essentially that sharing of Z documents is encouraged by the diffusion of WWW on the Internet.

A possible application can be a groupware tool for editing and versioning formal documents; such a tool could be integrated with other software tools in order to improve the specification phase of the software process.

The reuse of parts of documents obviously benefits from having these hypertextual Z documents. The tools will also improve the search of pieces of specifications in complex documents: Every element in the Z specification can be labeled or linked to other pieces of documents or to URL on the Internet.

XML can be further extended in order to include new symbols and integrate Z specification with other notations: New Java classes have to be written for the new symbols.

Our most ambitious goal consists of defining all XML displets necessary in an organization to support the intranet management system of formal documents typical of such an organization. For instance, we are currently working on displets for managing UML documents.

A displet site is available at http://www.cs.unibo.it/~fabio/displet.html.

This site 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.

Acknowledgments

We would like to acknowledge the help and contributions of our students, Alfredo Rizzi and Stefano Pancaldi, and the help and suggestions of Michael Bieber and Chao-Min Chiu. We also would like to thank the anonymous reviewers for their helpful comments and suggestions on both the paper and the project itself.

References

[1] S. Adler, A. Berglund, J. Clark, I. Cseri, P. Grosso, J. Marsh, G. Nicol, J. Paoli, D. Schach, H. Thompson, and C. Wilson, "A Proposal for XSL," submitted to *W3C;* http://www.w3.org/TR/NOTE-XSL.html, Aug. 1997.

[2] J. Bannan, *Intranet Document Management,* Addison-Wesley, 1997.

[3] B. Bos, H. Lie, C. Lilley, and I. Jacobs, "Cascading Style Sheets, Level 2 CSS2 Specification," W3C recommendation; http://www.w3.org/TR/REC-CSS2, May 1998.

[4] J. Bowen and D. Chippington,, "Z on the Web Using Java," J. Bowen, A. Fett, and M. Hinchey, eds., *Proc. 11th Int'l Conf. Z Formal Method (ZUM),* Lecture Notes in Computer Science, vol. 1,493, pp. 66–80, Springer-Verlag, Berlin, Sept. 1998.

[5] T. Bray, D. Hollander, and A. Layman, "Namespaces in XML," World Wide Web Consortium Working Draft, http://www.w3.org/TR/WD-xml-names, Sept. 1998.

[6] T. Bray, J. Paoli, and C. Sperberg-McQueen, "Extensible Markup Language (XML)," *World Wide Web J.,* vol. 2, no. 4, 1997.

[7] S. Brien and J. Nicholls, "Z Base Standard," Programming Research Group, Nov. 1992.

[8] P. Ciancarini, A. Fantini, and D. Rossi,, "A Multi-Agent Process Centered Environment Integrated with the WWW," *Proc. Sixth IEEE Workshops Enablings Technologies: Infrastructure for Collaborative Enterprises (WETICE),* pp. 113–120, Boston, IEEE CS Press, Los Alamitos, Calif., June 1997.

[9] P. Ciancarini, A. Rizzi, and F. Vitali, "An Extensible Rendering Engine for XML and HTML," *Computer Networks and ISDN Systems,* vol. 30, nos. 1-7, pp. 225–238, 1998.

[10] J. Clark and S. Deach, "Extensible Stylesheet Language (XSL)," Version 1.0, World Wide Web Consortium Working Draft, http://www.w3.org/TR/WD-xsl, Aug. 1998.

[11] M. Fraser, K. Kumar, and V. Vaishnavi, "Strategies for Incorporating Formal Specifications in Software Development," *Comm. ACM,* vol. 37, no. 10, pp. 74–86, Oct. 1994.

[12] D. German and D. Cowan, "Experiments with the Z Interchange Format and SGML," J. Bowen and M. Hinchey, eds., *Proc. Ninth Int'l Conf. Z Formal Specification Notation (ZUM),* Lecture Notes in Computer Science, vol. 967, pp. 224–233, Limerick, Ireland, Springer-Verlag, Berlin, Sept. 1995.

[13] S. Holzner, *XML Complete,* McGraw-Hill, 1998.

[14] P. Ion and R. Miner, "Mathematical Markup Language (MathML) 1.0 Specification," W3C recommendation, http://www.w3.org/TR/REC-MathML, Apr. 1998.

[15] X. Jia, "ZTC: A Type Checker for Z—User's Guide," Inst. for Software Eng., Dept. of Computer Science and Information Systems, DePaul Univ., Chicago, 1994.

[16] D. Jordan, "CADiZ—Computer Aided Design in Z," S. Prehn and W. Toetenel, eds., *Proc. VDM: Formal Software Development Methods,* Lecture Notes in Computer Science, vol. 551, pp. 685–690, Springer-Verlag, Berlin, Oct. 1991.

[17] "Jumbo browser for CML,"http://www.venus.co.uk/omf/cml/.

[18] G. Kaiser, S. Dossick, W. Jiang, and J. Yang, "An Architecture for WWW-Based Hypercode Environments," *Proc. 19th Int'l Conf. Software Eng. (ICSE),* pp. 3–13, Boston, May 1997.

[19] D. Knuth, "Literate Programming," *The Computer J.,* vol. 27, no. 2, pp. 97–111, May 1984.

[20] L. Lamport, *LaTeX: User's Guide and Reference Manual,* Addison-Wesley, 1986.

[21] H. Lie and B. Bos, *Cascading Style Sheets: Designing for the Web,* Addison-Wesley, 1997.

[22] E. Maler and S. DeRose, "XML Linking Language (XLink)," World Wide Web Consortium Working Draft, http://www.w3.org/TR/WD-xlink, Mar. 1998.

[23] E. Maler and S. DeRose, "XML Pointer Language (XPointer)," World Wide Web Consortium Working Draft, http://www.w3.org/TR/WD-xptr, Mar. 1998.
[24] L. Mikusiak, M. Adamy, and T. Seidmann, "Publishing Formal Specifications in Z Notation on the WWW," M. Bidoit and M. Dauchet, eds., *Proc. Conf. Theory and Practice of SW Development (TAPSOFT 97),* pp. 871–874, Lecture Notes in Computer Science, vol. 1214, Lille, France, Springer-Verlag, Berlin, 1997.
[25] L. Mikusiak et al., "Z Browser: A Tool for Visualization of Z Specifications," J. Bowen and M. Hinchey, eds., *Proc. Ninth Int'l Conf. Z Formal Specification Notation (ZUM),* Lecture Notes in Computer Science, vol. 967, pp. 510–525, Limerick, Ireland. Springer-Verlag, Berlin, Sept. 1995.
[26] S. Ressler, *The Art of Electronic Publishing,* Prentice Hall, 1997.
[27] W. Scacchi and J. Noll, "Process-Driven Intranets—Life Cycle Support for Process Reengineering," *IEEE Internet Computing,* vol. 1, no. 5, pp. 42–51, Sept./Oct. 1997.
[28] C. Sperberg-McQueen and R. Goldstein, "HTML to the Max: A Manifesto for Adding SGML Intelligence to the World Wide Web," *Electronic Proc. Second Int'l WWW Conf.: Mosaic and the Web,* 1994.
[29] J. Spivey, *The Z Notation: A Reference Manual,* second ed., Prentice Hall, 1992.
[30] F. Vitali, C. Chiu, and M. Bieber, "Extending HTML in a Principled Way with Displets," *Computer Networks and ISDN Systems,* vol. 29, nos. 8-13, pp. 1,115–1,128, 1997.
[31] "WebEQ,"http://www.webeq.com/.

Paolo Ciancarini received the PhD degree in computer science from the University of Pisa, Italy, in 1988. He is now an associate professor of computer science at the University of Bologna, Italy. His research interests include coordination models and agent-oriented languages, Web-based document management systems, and environments, methods, and tools for software engineering.

Fabio Vitali received the PhD degree in computer science and law from the University of Bologna, Italy, in 1994. He is now a research associate of computer science at the University of Bologna. His research interests include user interface design, hypertext models, markup languages, versioning systems, and coordination languages.

Cecilia Mascolo is a PhD degree student in the Department of Computer Science of the University of Bologna, Italy, where she received her Laurea degree in 1995. Her research interests include specification, analysis, and prototyping of software systems containing mobile components.