EXTENDING HTML IN A PRINCIPLED WAY WITH *DISPLETS*

Fabio Vitali
Dept. of Computer Science
Università di Bologna
Mura Anteo Zamboni, 7
I-40127 Bologna

Chao-Min Chiu
Graduate School of Management
Rutgers University
University Heights
Newark NJ 07102, USA

Michael Bieber
Institute for Integrated Systems Research
New Jersey Institute of Technology
University Heights
Newark NJ 07102, USA

Abstract

Displets provide authors and programmers with a way to freely extend the HTML language on a per-document basis in a principled manner. Currently, in order to be accepted, HTML elements must be approved by the official HTML review board. Non-standard extensions have appeared, and have relied on the commercial power of the proponents for acceptance.

Two major forces are driving the extension process of the HTML language: those who favor a better description of document elements, as with SGML, and those who would like better control over the final appearance of documents, as with Postscript and other display-oriented languages. Special notations (such as mathematics, music, etc.), are hardly considered - if at all - in defining the HTML standard. We designed displets to fill this frustrating gap.

Displets are Java classes that are activated while rendering an HTML document. Displets provide graphical artists a better control over the final appearance of HTML documents, librarians and indexers a better description of their content, and those in need of new notations a way to describe and use graphical objects in a manner compatible with the graphical and structural habits of the HTML community.

1 Introduction

The current debate on HTML sees two opposing positions as preeminent. One wants better control over the final appearance (the *rendering*) of a document. The other advocates better control over the description of the structure and role of the parts of the document.

The first group, lead by graphic designers, would like the standardization efforts on HTML to cease and allow *ad hoc* plentiful extensions to the language to cover all visualization needs. The second group, the SGML community, would prefer HTML to abstract from the description of the document's physical appearance, and let style sheets guide the final mapping of structural elements to their visual representations.

Furthermore, commercial software developers often find it irresistible to improve and detour from the published standard, and seek a commercial advantage by extending HTML with new,

proprietary tags ([16], [17], [18]). They hope, of course, for users to switch to their browsers to be able to read and fully appreciate Web pages using those new tags.

Yet, it is our impression that all these approaches miss a critical need of authors of distributed documents - support for special notations. Such notations include (and are not limited to) specialized graphical objects (object-oriented graphics, graphs, charts, etc.), specialized notation for specific fields (chemistry, electrical and electronic engineering, music, mathematics, software engineering, chess, etc.), specialized alphabets (hieroglyphics, cuneiform, etc.), special symbols (philosophical, religious, astrological, alchemical signs, etc.)

Generally there is one basic way to include these types of notation in an HTML document - by providing an image of the symbol's graphical depiction. For particularly complex or dynamic objects, it is now possible to write Java applets that display the object as their sole effect. Applets may include specifications of the object details inside their code, making it therefore completely hidden. Alternatively applets may receive these specifications as parameters put in the HTML code, but without a required syntax or style, possibly resulting in an arcane and opaque syntax.

We propose a cleaner approach to extensions to the HTML language: *displets*. Displets are small Java modules similar to applets. A properly extended browser would activate them while parsing an HTML document, everytime it encounters pre-declared new tags and would let them handle the display operation for the relevant objects. Declaring both the extensions that will be used in the document and the displet class needed for their display at the beginning of the document, enables any kind of customized extension to the HTML language without loss of generality, while maintaining wide-spread compatibility and stylistic elegance.

We have developed a proof-of-concept implementation of displets by modifying the 1.0 alpha3 release of the HotJava browser. A few extensions have been designed and implemented on that architecture.

The paper is structured as follows. Section 2 summarizes the current state of the HTML language. In section 3, we discuss the need for HTML to support new notations. Section 4 introduces and describes displets. Section 5 covers some implementation details. Section 6 presents several extensions to HTML we have developed using displets, and describes others that we would like to create. Section 7 describes a series of suggestions on how to implement displets within existing and future standards.

2 The HTML language

The HTML language was designed as part of the World Wide Web effort at CERN at the beginning of the nineties. Its goal [1] was to provide an easy page description language suitable for:

- hypertext news, mail, on-line documentation, and collaborative hypermedia;
- menus of options;
- database query results;
- simple structured documents with inline graphics; and
- hypertext views of existing bodies of information

HTML was designed to be an application of the Standard Generalized Markup Language (SGML, [14]), that is, a class of documents conforming to an SGML Document Type Definition (DTD) defining "HTML documents." SGML is an international standard for describing marked-up text in an electronic format. Its wide acceptance and influence are owed to several characteristics, among which we would like to highlight the following ([23], [10]):

Descriptive markup
: An SGML document has its content enriched by embedded tags that describe its parts in terms of their role and meaning, rather than the kind of processing necessary to display them. This allows one to define a generic markup that can be used for any purpose: from display on a wide range of devices (from dumb terminals to highly graphical workstations, from simple line printers to sophisticated typographical systems) to content-based analysis and categorization (useful for large document systems, indexers, search engines, etc.), to processing for information extraction, restructuring, and update. SGML is only interested in the meaningful structuring of the documents, and leaves the task of assigning graphical attributes to document elements to rendering software.

Open set of document types
: SGML documents may respect constraints defined by their type (as proscribed in their Document Type Definition, or DTD). This means that only some elements may appear in a SGML document, and that some elements may have constraints as to where and how to appear.

Human-readable representation
: SGML markup appears as text tags surrounding the described elements. Both the character set used and suggested practice emphasize readability in the mark-up, since no special software may be available for marking the text, and using meaningful words as tag names improves understandability of the text in all situations.

HTML is a specific document type described using an SGML DTD. As such, it inherits some of SGML's qualities: it shows embedded markup with meaningful names, some constraints in the nesting of elements, and some structuring support (the header tags). On the other hand, being a single DTD, HTML has the drawbacks of being a closed document type: only the existing elements can be used and no extension can be created unless approved by the appropriate committee.

This fact has caused several problems in the development of the language itself. Since HTML+ [20], on through HTML 2.0 [2] to HTML 3.2 [21], every proposed enhancement to the language has had to deal with an approval process, with the attacks of competing proposals, and with the need to keep the additional features simple, usable and orthogonal to each other. Several extensions were first implemented in commercial software and then proposed to the language committee with the force of an already wide-spread usage.

Further needs could have been met with further language extensions (e.g., math markup [8]), but had little support from influential user bases or commercial software, as they were deemed too complex for fast implementation or no consensus was reached among the competing proposals. The needs remain, and are met in the meantime with different, more complex and less elegant solutions. In-line images have long been the standard way to include content elements the graphical depiction of which was outside HTML capabilities, but this usage stresses rendering, not content. Server-side extensions (e.g., [12]) make it possible to specify new tags that are automatically substituted with appropriate content before being delivered to the browser. They rely on the fact that even though a document needs to be shown on several types of browsers, it still resides on only one server, so that it is safe to include new tags of which that server is aware.

In the current discussion, graphic designers compete with SGML enthusiasts to provide new extensions. Extremists appear on both sides. David Siegel [22], for instance, proposes all standardization in the HTML language be stopped at the current state and that the market decide successful extensions via natural selection of the fittest. C. M. Sperberg-McQueen and R. F. Goldstein [24] suggest using pure SGML documents and activating SGML viewers as plug-ins or external viewers.

Others have more moderate views. The April 1996 W3C Workshop on High Quality Printing from the Web had several proposals for consistent font rendering over the Web ([25], [27]) and new markup elements for managing the display [7]. At the November 1994 First International Workshop on WWW Design Issues, some researchers discussed ways to include SGML features in the Web by improving the structure control ([19], [26]) or by developing simplified SGML-like dialects to replace HTML ([11]).

Important proposals are those receiving support from the W3C in one form or another. Stylesheets [5] are a way to attach rendering and formatting instructions to HTML tags. They allow authors to specify the presentation in a precise way without cluttering the content of documents. Style rules can be connected to HTML elements through implicit association, specification of linked stylesheet documents (with the LINK element), or the use of a new proposed header tag called STYLE, in which formatting instructions can be specified according to one of several stylesheet specification languages ([13], [15]).

The Extensible Markup Language (XML, [6]) is an important proposal by the W3C Working Group on SGML. XML is a simplified SGML with many of the arcane features removed in order to produce a more usable and understandable language. XML documents thus are still valid SGML documents, which leverages existing software.

3 The need for HTML extensions

The previous discussion shows that any proposed new extension is bound to annoy or be ignored by an important authoring community. If the extension is related to rendering, many in the SGML community will criticize it. If related to content or structure, many graphic designers will ignore it, continuing to do HTML hacks or creating images until obtaining the desired specific effect for some specific (high-end) browser, with disregard for compatibility and content analysis. The need for an approval process by the HTML editorial review board and an implementation on commercial applications contributes to the shared discomfort.

Suppose an author needs to display the following chart in a Web browser:

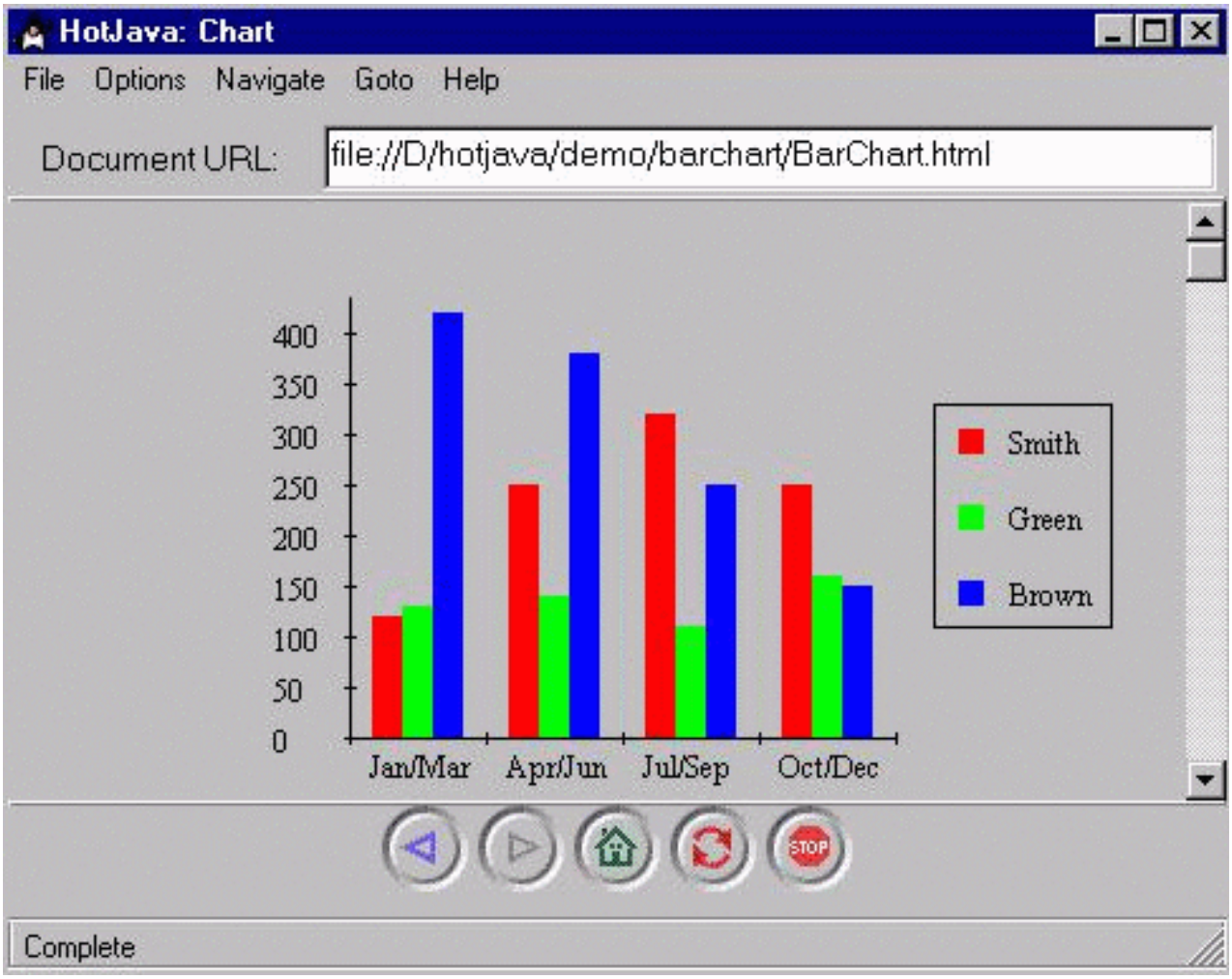

Figure 1: A simple chart in an HTML page

Currently a few solutions are possible:

- creating a static GIF or JPEG image while writing the document;
- having a CGI application create a GIF or JPEG image on-the-fly; or
- having a Java applet draw it during the display of the page.

In the first case, the chart is static and fixed; when changing the data, the author would have to redraw the picture using whichever application he or she used the first time. In the second case, the author would have to mess with CGI scripting and with the dynamic generation of graphic data. In the third case, the author would have to write his or her own Java applet or use an existing one and learn how to pass the parameters to it.

Obviously, the latter solution is probably the most elegant and common one given the current availability of tools. Unfortunately, this solution shares a big drawback with the other ones: the data that form the charts are not in an easily understandable or reusable format, as is dictated by the syntax of the applet parameters. In short, the applet programmer decides how the chart data should be specified, without any necessary regard for matching the markup of the rest of the page, and with several restrictions on the available character set (we cannot, for instance, use angle brackets or double quotes in the text to be drawn).

Furthermore, the applet content is completely separated from the content of the page. This leads to a dangerous loss of hypertext in Web documents [3]. Applet authors cannot use hypertext features

(such as links on the objects); display HTML text in the chart labels; enable human readers to understand the chart data by reading the HTML source code; or enable bots, indexers and research engines understand its content - unless the applet programmer has gone far out of his or her way to duplicate these characteristics from the browser in the code of the applet itself.

Therefore, while applets have the procedural power to support all needed notations, they strongly discourage content markup, both in syntax and in structure, and are not an elegant choice for new data objects.

The ideal solution would be to specify the chart data as markup included in the HTML source code, instead of as parameters of the applet. Unfortunately this is currently not possible. Yet the whole point of structured markup *à la* SGML is to be able to name and describe all the parts of a document in a significant manner. The whole point of typographic languages *à la* Post Script is to be able to specify exactly how and where to display all the parts of a document. HTML forces authors instead to fit all the parts of a document in a limited set of available markup tags, and removes most rendering control from the authors.

4 Displets

Displets are our proposal for creating HTML extensions in a principled, general way that will satisfy everyday HTML authors, as well as both disputing communities.

- New tags can be created on a per-document basis, without approval or explicit support from commercial browsers.
- All new tags are explicitly defined before being used, using a simple syntax.
- Specification of syntactical and structural constraints can be provided.
- Procedural support is given for creating any kind of graphical object with styles, font, images, and graphical primitives.
- Complete control on rendering of new and nested elements is provided.
- Support for interactive elements (e.g., link anchors) is provided.

Displets are Java modules that are automatically activated whenever some pre-declared tags are parsed by the HTML parser during the display of the HTML document. The author will be able to include any kind of tag provided some displet exists to handle the data thus specified.

4.1 Displets within HTML documents

Displets are specified within HTML documents as new markup tags. They are preceded by their declaration, and then used within the document as if they were legal HTML markup.

An example of an HTML document enhanced with displets may help in understanding their use:

```
<HTML><HEAD>
<TITLE>Test for chart</TITLE>
<TAG NAME="CHART" HasEndTag NonNesting SRC="http://hertz.njit.edu/chart/chart.cla
        <ATTR NAME="type" VALUE="BAR, PIE, LINE" REQUIRED>
        <ATTR NAME="width" VALUE=number>
        <ATTR NAME="height" VALUE=number>
</TAG>
<TAG NAME="TABLE" HasEndTag IN="Chart"SRC="http://hertz.njit.edu/chart/table.clas
        <ATTR NAME="HasLegend" VALUE=boolean>
</TAG>
<TAG NAME="TR" HasEndTag IN="Chart"SRC="http://hertz.njit.edu/chart/tr.class">
        <ATTR NAME="Type" VALUE="LABEL, DATA" REQUIRED>
        <ATTR NAME="Color" VALUE=RGB>
```

```
</TAG>
<TAG NAME="TH" IN="Chart"SRC="http://hertz.njit.edu/chart/th.class"></TAG>
<TAG NAME="TD" IN="Chart"SRC="http://hertz.njit.edu/chart/td.class"></TAG>
</HEAD><BODY>

<P>Some normal HTML text followed by the chart:</P>
<CHART TYPE=BAR> <TABLE HasLegend>
    <TR TYPE=LABEL>
        <TH> <TH><EM>Jan/Mar</EM> <TH>Apr/Jun <TH>Jul/Sep <TH>Oct/Dec
    </TR>
    <TR TYPE=DATA COLOR=RED>
        <TH>Smith <TD>125 <TD>257 <TD>327 <TD>250
    </TR>
    <TR TYPE=DATA COLOR=GREEN>
        <TH>Green <TD>137 <TD>140 <TD>110 <TD>160
    </TR>
    <TR TYPE=DATA COLOR=BLUE>
        <TH>Brown <TD>421 <TD>380 <TD>250 <TD>150
    </TR>
</TABLE> </CHART>
</BODY></HTML>
```

Table 1: An HTML document defining and using a chart displet

A displet-enhanced HTML document has two characteristics: new tags are introduced (with a simple syntax) at the beginning of the document, and, in the body, the newly defined tags are mixed with standard HTML ones to create the needed document.

According to this syntax (others can be defined; a discussion can be found in section 7), for each tag the main characteristics are described in a simplified syntax that closely resembles SGML DTDs: the name of the tag, the syntax constraints, and the available attributes. There are two major difference from DTDs:

- the rendering function (i.e., the displet class) is mentioned explicitly; and
- the valid containers for each element are specified, rather than the valid content as we would have with SGML. This avoids the use of finite state grammars in the specification of an element's content, and in our opinion, greatly simplifies understanding of the syntax for non-SGML experts.

As the example shows, some tags have not been defined from scratch, but are a customization of existing ones. This allows non-compliant browsers to still make use of and display the document content as best as it can.

For instance, leveraging the fact that unknown tags and attributes are usually ignored by browsers, we are re-using TABLE, TR, TH and TD tags for specifying the charting data. Therefore an *unaware* browser would display the previous document as containing an appropriately formatted table:

	Jan/Mar	**Apr/Jun**	**Jul/Sep**	**Oct/Dec**
Smith	125	257	327	250
Green	137	140	110	160
Brown	421	380	250	150

Table 2: The same chart displayed by an unaware browser

By specifying the IN="Chart" attribute within the table-related tags, we signal that the code of the displets should be called only when these tags are found within a CHART tag, so that tables described outside of charts are still handled normally - as tables.

This solution should appeal to SGML enthusiasts because the HTML language can be extended with new tags. Structuring the documents is more flexible and complete, the new tags are explicitly defined, and some syntactical prescriptions (albeit simpler than in SGML) are reinforced.

It also should appeal to graphical designers. Page designers are not constrained to forcing their graphical ideas into existing tags. Also, the final rendering of the elements can be finely controlled by the displet code.

Furthermore, it should appeal to those with special notation needs. Whole new notations can be defined (mathematics, music, etc.), as well as simple elements (special characters, dingbats, etc.), or complex structured data formats (object-oriented graphics, charts, etc.). Authors can easily include all in HTML documents, having the same ease of use and comprehension as with the other HTML tags.

Also, it may appeal to the HTML review board, because it is the only method so far to extend HTML without deferring to endless committee discussions or to the market force of commercial software, or employing complex specification mechanisms such as SGML and XML.

Finally, it may appeal Web authors, who will be able to mix and match pre-defined displets creating sophisticated complex without recurring to programming.

4.2 Displets as Java modules

A displet is a Java class whose methods are called during the parse of an HTML document. Displets have control over the display of themselves and their immediate surroundings, such as the HTML code contained between their start and end tag, or immediately following the end tag. They are not usually called while the document is on screen, unless the displet explicitly requests this service.

The Java code for a very simple displet looks like this:

```
import awt.Graphics;

class HelloWorldDisplet extends browser.Displet {

    public void startElement() {
        graphic.drawString("Hello world!", 10, 25);
    }
}
```

Table 3: The "Hello world" displet

Displets require a modified HTML parser that allows external classes to be declared and invoked, and that allows external classes to modify the rendering parameters (such as text font, size or style, margins, etc.)

The modified HTML parser must accept the definition of the new tags upon encountering the TAG element, and invoke the appropriate methods when encountering the start and end tag of the newly defined element.

Whenever encountering an unknown tag, the HTML parser should verify whether a TAG tag with that name has been defined and is permitted to occur in the current context (for instance, if it follows the specifications of the IN and NonNested attributes). If so, the parser will create a new instance of the displet with an array of the valid attributes (as specified by the ATTR elements contained in the TAG element). The parser also verifies and performs appropriate housekeeping tasks when encountering the end tag of the newly defined element.

Upon displaying the page, the `startElement()` method of the displet should be called when encountering the start tag of the new object. Within this method it is possible to create graphical objects and to change the settings of the current rendering (font, margins, etc.). The `endElement()` method is called in conjunction with the new element's end tag. This method usually should be used to restore the original settings that were modified in the `startElement()` method.

Furthermore, the browser should be capable of calling the appropriate methods whenever a user event happens within the boundaries of the displayed object. This need is clearly shown for the A tags, where the browser must react when the user clicks on or moves the mouse over the displayed object.

5 Implementation

We have created a proof-of-concept implementation of displets using Java 1.0 alpha3 and the 1.0 alpha3 release of the HotJava browser. Although this forced us to use a non-standard and outdated version of the language, and a limited set of HTML options to start off with, it allowed us to use the source code of an existing HTML parser and browser that we could modify and extend.

HotJava 1.0 alpha3 parses and displays HTML documents through the interaction of the appropriate subclasses of three different classes: Tag, TagRef and DisplayItem. Tag instances (one for each tag type) are passive objects that simply notify the HTML parser of their existence. TagRef instances are created by the HTML parser every time it encounters the proper tag, and are used as an internal representation of the HTML document. When displaying the document, each TagRef is associated to a relevant DisplayItem, which decides how to display itself.

The current implementation defines two new tags: TAG and ATTR:

```
<!ELEMENT tag - o attr*>
<!ATTLIST tag
          name        CDATA                          #REQUIRED
          hasEndTag   (hasEndTag)                    #IMPLIED
          nesting     (nesting)                      #IMPLIED
          interactive (interactive)                  #IMPLIED
          in          CDATA                          #IMPLIED
          src         CDATA                          #REQUIRED>

<!ELEMENT attr - o (#PCDATA)?>
<!ATTLIST attr
          name        CDATA                          #REQUIRED
          value       CDATA                          #REQUIRED
          required    (required)                     #IMPLIED>
```

Table 4: The definition of the TAG and ATTR elements

Three classes are defined, namely the DispletTagRef, DispletDisplayItem and Displet. A DispletTagRef is created whenever a declared new tag is encountered during the parse of the

document. The DispletDisplayItem is then activated by the browser during the layout of the document page. This class creates an instance of the appropriate Displet class (the only class supplied by the user), which can create a graphic object or modify the layout settings as needed.

If the INTERACTIVE attribute is specified, the displet class will also receive all the users' events in the area, in which case the appropriate methods (`mouseDown()`, `mouseUp()`, etc.) will be called. An interactive displet can do exactly the same things as an applet, although the calling method is different (displets should prepare and paint themselves in the `startElement()` method).

This structure is parallel and similar to the one used for managing applets:

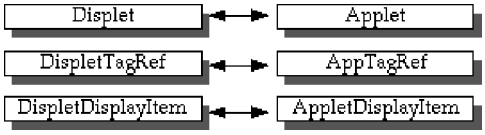

Figure 2: The analogy between applets and *displets*

The user implementing a new displet simply has to subclass Displet. Displets are built around two basic methods: `startElement()` and `endElement()`. In the first, a displet should prepare and draw its graphical representation and/or modify the current setting for the next elements. In the second, a displet should restore the original settings. Two objects are available to all methods in a displet class: the **`graphic`** object, providing the context for the drawing routines, and the **`document`** object, providing the current values for variables such as the current margins, the current font, etc.

6 Displet examples

In our proof-of-concept implementation, we have created a small set of working extensions to HTML. A few more are in the works, and should be ready at about the conference time.

6.1 Charts

A chart is a graphical depiction of a table of numerical values. In accordance to this definition, and in order to maintain compatibility with other browsers, charts use table tags to define the data they need to show. A complete example of the grammar and use of charts is given in sections 3 and 4.

6.2 Graphs

A graph is a set of circular and rectangular shapes called nodes connected by arrowed lines called arcs. The following is the declaration of such a displet in the HTML document:

```
<HTML><HEAD>
<tag name="GRAPH" hasEndTag nonNesting src="http://hertz.njit.edu/displets/graph/
    <attr name="width" value=number>
    <attr name="height" value=number>
</tag>

<tag name="NODE" hasEndTag nonNesting in="GRAPH"
src="http://hertz.njit.edu/displets/graph/node.class">
    <attr name="name" value=string required>
    <attr name="pos" value=integerPair>
    <attr name="size" value=integer>
    <attr name="shape" value="rect , circle">rect</attr>
</tag>
```

```
<tag name="ARC" hasEndTag nonNesting in="GRAPH"
src="http://hertz.njit.edu/displets/graph/arc.class">
    <attr name="name" value=string>
    <attr name="from" value=string required>
    <attr name="to" value=string required>
    <attr name="arrow" value="to, from">
    <attr name="thickness" value=integer>
</tag>
</HEAD><BODY>
<graph width=400 height=300>
    <node name="first" pos=290,120 size=160,40 shape=rect>
        <A HREF="http://hertz.njit.edu/first.html">This is the first label</A>
    </node>
    <node name="second" pos=100,20 size=160,40 shape=rect>
        <B>This is the second label</B>
    </node>
    <node name="third" pos=100,120 size=120,50 shape=rect>

    <arc from="first" to="second" arrow=from>This has a label, too</arc>
    <arc from="first" to="third" arrow=from>
</graph>
</BODY></HTML>
```

Table 5: Defining and using a graph

This is how the graph looks on the screen:

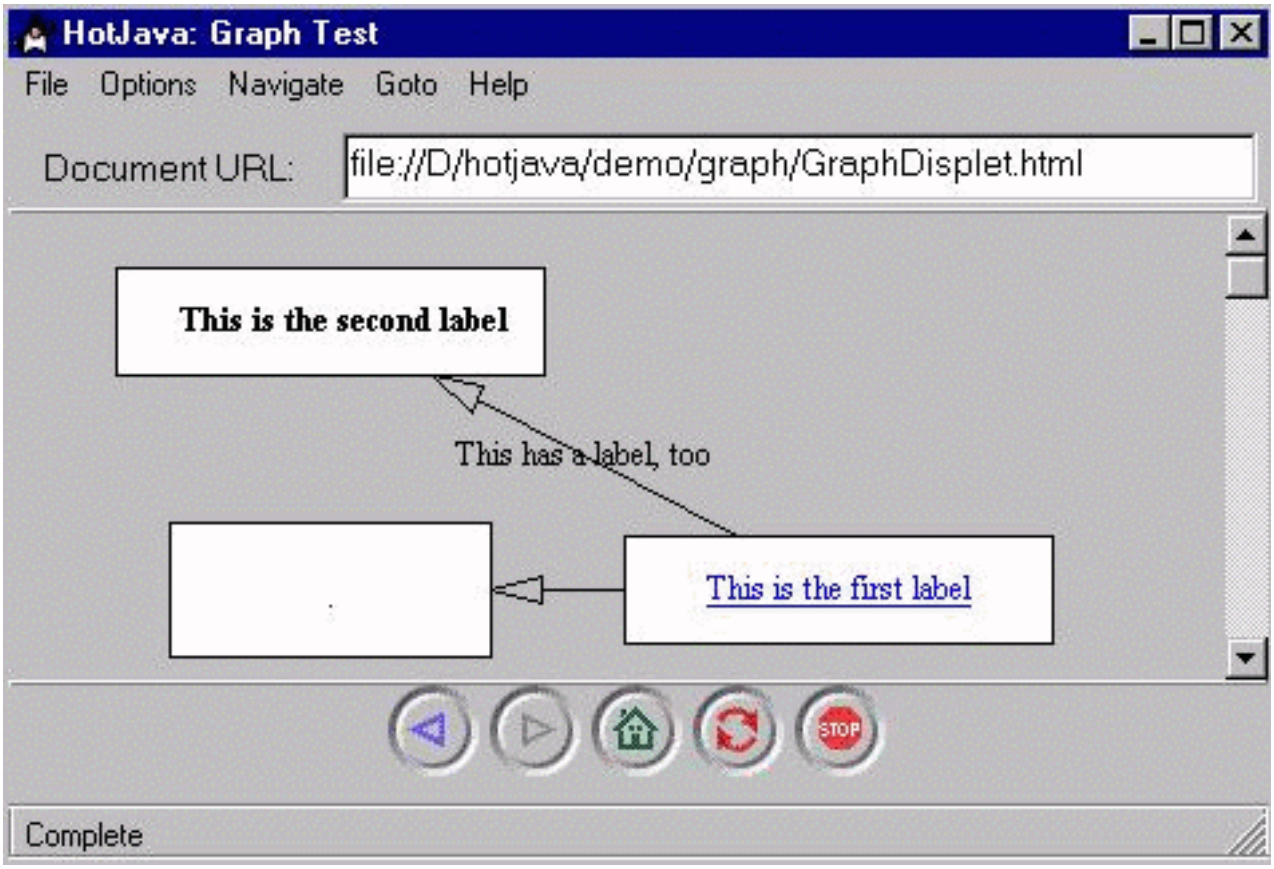

Figure 3: A graph element in an HTML page

6.3 Anchor groups

Anchor groups are a way to implement multi-endpoint links [4]. the user finds a clickable area that, instead of leading to a single destination, pops up a list of choices each leading to a different destination. As a displet, this corresponds as an anchorGroup tag surrounding a set of plain HTML A anchors.

The following is a declaration for AnchorGroup:

```
<HTML><HEAD>
<tag name="agroup" hasEndTag interactive src="http://hertz.njit.edu/displets/agrc
    <attr name="name" value=string>
    <attr name="align" value="top, middle, bottom">
</tag>

<tag name="a" hasEndTag src="http://hertz.njit.edu/displets/agroup/a.class">
    <attr name="HREF" value=URL required>
    <attr name="ALT" value=string>
    <attr name="selected" value=boolean>
</tag>
</HEAD><BODY>
<agroup name="agroup1" align="top">
    This is an example of multi-endpoint link:
    <a href="http://www.njit.edu" selected>Go to NJIT</a>,
    <a href="http://info.rutgers.edu">Go to Rutgers</a>
</agroup>
</BODY></HTML>
```

Table 6: Defining and using an anchor group

By defining anchor groups as collections of anchors, we can guarantee some compatibility with existing browsers, since the links, rather than being grouped in a single pop-up menu, will be displayed side by side without problems.