

Better than HTML Web: XML for Programming-Free Dynamically Generated Web sites

Ronald P. Vullo, Ph.D.
Assistant Professor
Rochester Institute of Technology
102 Lomb Memorial Drive
Rochester, New York 14623
(585)475-7281
rpv@it.rit.edu

Daniel S. Bogaard
Assistant Professor
Rochester Institute of Technology
102 Lomb Memorial Drive
Rochester, New York 14623
(585)475-5231
dsb@it.rit.edu

Abstract

Traditionally the World Wide Web has been about empowering individuals to communicate. HTML made it possible for anyone to publish to a worldwide audience. But as the web grew, so did our expectations of it. Web sites are now powerful applications in and of themselves with thousands of lines of code in real programming languages making them work. But not everyone is a programmer, and so the web is not as empowering as it was in the beginning. XHTML and XML are poised to re-empower the individual. Server-side XML processing has the ability to let authors replace hundreds of lines of code with a single XML tag.

At the Rochester Institute of Technology we are building and using an open source development system, architecture, and code library for building web communities, content management systems, and portals. Named "Molly," this system provides several features commonly used to facilitate the development of an active community of users. These include live conversations and conferences ("chat rooms"), asynchronous structured conversations where participants need not all be online at the same time ("forums" or "Bulletin Boards"), calendars, news, comments, and many others. At the heart of Molly is an XML parser which processes Molly Active Markup Language (MAML) tags embedded in XHTML pages. This is what sets Molly apart and offers a glimpse into what next generation web site development will look like as XML and XHTML become ubiquitous and web application architectures begin to take advantage of the power and extensibility long promised by XML.

Introduction

One of the features of the World Wide Web that made it so compelling early on was the simplicity of HTML and the ease with which we could build web sites and share information. With just a few simple tags we could build cross-platform, world-accessible, decent looking documents. And these documents could be linked in true hypertext fashion. The dream envisioned by pioneers such as Vannevar Bush¹ and Ted Nelson² was suddenly not only possible, but easy. Is it any wonder that academics, and soon thereafter everyone fell in love with the web? The more we built the web, the more we wanted to use it, and the more we wanted to

be able to do with it. Browser developers added “helper applications,” forms, JavaScript, embedded plugins, and other ways to add interactivity. Meanwhile developers pushed the limits of Server-Side Includes and the Common Gateway Interface until “CGI” became synonymous with “real web site.” How many of us taught ourselves Linux and Perl, or bought products such as Lasso³ because we wanted our web sites to do more? A few of us were even crazy enough to build CGI applications in HyperCard.⁴ We wanted interactivity, virtual communities, e-commerce... we *wanted* web applications! And we built them. Users came, and they wanted more. We built more. And more tools for building: Cold Fusion⁵, ASP⁶, Java⁷, JSP⁸, PHP⁹, DreamWeaver¹⁰, FrontPage¹¹, etc.

Because of this, the web has become a much more dynamic and useful place. But what we lost was the simplicity of the early web. We have found ourselves reinventing functionality over and over, and spending more and more time programming.

Molly and MAML

The Molly project is an attempt at a different approach to developing web interactivity. We begin by identifying functionality we need for a site, then developing a general design pattern for that functionality. Based on this generalized functionality we design XML tags and attributes for those tags with which web developers could incorporate that functionality into their sites. Finally we build the code, usually as a module, into Molly to implement the functionality. A good example of this is Molly’s Picklist. Say you’re building an e-commerce catalog, and you want to show a list of products. Or if you’re building a virtual community you might want to list the registered users of the site. The general functionality we want is to be able to fetch a list of records from a database and display them on screen for the user. Thinking about it a little more, we decide that the user should be able to click on the list’s column headings to re-sort the list by that column, and click a link for any record to see a page with the details of that record. This is the sort of design pattern that is found on myriad sites, and is usually programmed from scratch, or by copy-paste-edit, over and over.

Our next step is to design a set of XML tags to make it easy for web page authors to incorporate this functionality. Because the result is tabular data, and will be rendered as an HTML table, we create a `<maml:picklist>` tag with several attributes familiar to HTML authors from the `<table>` tag. Added are a few attributes to specify the database table(s), search criteria, and the detail page each record should link to. We also design a `<maml:column>` tag to be nested inside the `<maml:picklist>` tag for specifying the fields to be rendered and their column headings. Figure 1 shows an example of a Picklist’s code.

```
<maml:picklist border="0" cellspacing="2" cellpadding="3" table="test"
key="name" criteria="age > 5" detail="form.maml" mode="edit">
  <maml:column field="name">Name</maml:column>
  <maml:column field="eyecolor">Eye Color</maml:column>
  <maml:column field="age">Age</maml:column>
</maml:picklist>
```

Figure 1. Code example of a Picklist with three columns to be fetched from the “test” table.

The result of incorporating this code (nested inside a <maml:windoid> </maml:windoid> tag pair to create the frame around the list) can be seen in Figure 2.

| Picklist Example | | |
|------------------|-----------|-----|
| Name | Eye Color | Age |
| Michelle | brown | 7 |
| Ron | Brown | 44 |
| Tim | brown | 25 |

Figure 2. A Picklist as rendered by Molly. (Live example at: <http://molly.rit.edu/local/examples/picklist.maml>)

Consider that by changing a single .ini file setting Molly can use any of MySQL, Oracle, PostgreSQL, or ODBC for database access, and the <maml:picklist> tag works the same on all of them with no programming changes. This is the level of simplicity which we are striving for.

Another example is in building a web site’s navigation. This is, of course, a common feature of web sites. Molly currently supports two styles of menus; “stacked” or vertical, and “tabs” or horizontal. Both are implemented with the <maml:menu> and <maml:menuitem> tags. Figure 3 shows code examples of both types. Note that the only difference is in the <maml:menu> tag’s appearance attribute (which defaults to “stacked”).

```

<maml:menu>
  <maml:menuitem href="http://cnn.com/" target="_blank">CNN</maml:menuitem>
  <maml:menuitem href="http://msnbc.com/" selected = "selected">MSNBC
    </maml:menuitem>
  <maml:menuitem href="http://spiketv.com/">Spike</maml:menuitem>
</maml:menu>

<maml:menu appearance="tabs">
  <maml:menuitem href="http://cnn.com/" target="_blank">CNN</maml:menuitem>
  <maml:menuitem href="http://msnbc.com/" selected = "selected">MSNBC
    </maml:menuitem>
  <maml:menuitem href="http://spiketv.com/">Spike</maml:menuitem>
</maml:menu>

```

Figure 3. MAML code for creating menus.

Again we have intentionally selected attribute names, such as “href” and “target” familiar to HTML authors.

Figure 4 shows the results of these two examples as rendered in a browser. Like many tags implemented in Molly, these are rendered based on template files, which are the XHTML fragments used to construct the results seen in the browser. All dynamic roll-over feedback is accomplished with CSS, not JavaScript. This allows for a deeper level of customization, still without programming.

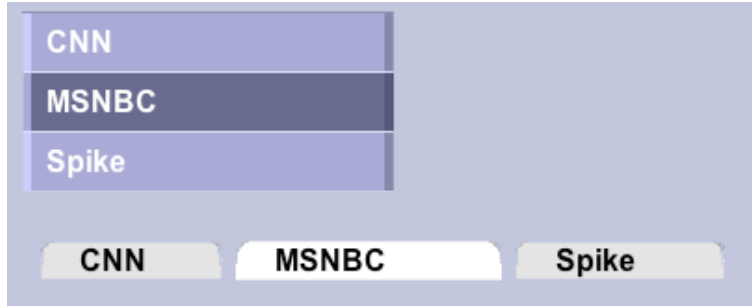


Figure 4. Menus as rendered by Molly. (Live example at: <http://molly.rit.edu/local/examples/menu.maml>)

Molly currently has over 50 functional MAML tags, so this paper will not attempt to describe them all. (Documentation of all Molly tags is available at <http://molly.rit.edu/>) What follows are brief descriptions of a few of the more powerful tags:

<maml:form>

```
<maml:form table="test" response="picklist.maml" keyfield="name" schema="on">
  Name: <maml:input type="text" name="name"/><br/>
  Eye Color: <maml:input type="text" name="eyecolor"/><br/>
  Age: <maml:input type="text" name="age"/><br/>
  <maml:input type="submit" value="I'm a submit Button"/>
</maml:form>
```

Figure 5. Molly Forms

Designed to mimic HTML forms as much as possible, MAML forms automatically connect to database tables for inserts and replacements. When the attribute “schema” is set to “on” as in Figure 5 Molly will write the SQL necessary to support the form and embed it in the page as an HTML comment.

<maml:select>

```
<maml:select table="test" key="name" criteria="age > 0" detail="form.maml"
format="off">
  <maml:record maxentries="3">
    <p>
      <span style='background-color:lightgreen;'>
        Name: <maml:field name="name"/>
      </span>
      <span style='background-color:lightblue;'>
        <maml:detail href='advancedLists.maml'>
          Eye Color: <maml:field name="eyecolor"/>
        </maml:detail>
      </span>
      <span style='background-color:tan;'>
        <maml:detail href='advancedForm.maml'>
          Age: <maml:field name="age"/>
        </maml:detail>
      </span>
    </p>
  </maml:record>
  Total Records: <maml:numrecs/><br/>
  <maml:previous>Previous <maml:numrecs/> Records </maml:previous>
  | <maml:pages delimiter="-" /> |
  <maml:next>Next <maml:numrecs/> Records </maml:next><br/>
</maml:select>
```

Figure 6. Molly's Select tags

The Select tags give web page authors the ability to incorporate information from their database into pages in a very flexible manner allowing them to mix in whatever XHTML formatting tags they like, display multiple records, automatically paginate the results, etc.

<maml:comments>

Simply placing a `<maml:comments />` tag on any page allows users to add comments to that page. Trusted users' comments appear immediately, while untrusted users' comments must be approved by an administrator. Molly has a complete groups-based permissions system whereby any page or portion thereof can be made available only to a particular group. Webmasters can create whatever groups they deem necessary using a simple web interface.

<maml:block>

The block tag allows webmasters to make portions of pages editable by any permissions group they establish. Blocks are edited in place on the page. Users permitted to edit the block see an "edit" button which displays a form via DHTML to modify the block. All other users see the page as an ordinary web page, unaware that it is editable. Because Molly processes this all at the server, even viewing the source code in the browser gives no evidence to users other than those with edit permission that a page is editable.

Open Source and Open Architecture

Molly is open source, and free to all users and developers. Molly also has a documented modular development architecture that allows programmers to easily extend Molly. Molly's XML parser requires and produces valid XHTML 1.1 code. We recognize that Molly is likely to be installed and used by web authors and webmasters with a wide range of skills, so we have tried to keep installation and configuration as simple as possible. Virtually all system level configuration is done in a simple .ini file. Many aspects of Molly are configurable via a web interface to a configuration table in the database which overrides many .ini defaults.

Molly is also designed for simple integration into existing sites. Using the molly.ini file Molly can use any existing user login database, for example. An example of this is our own departmental web site (<http://www.it.rit.edu/>) where portions of the site are Molly-based with logins connected to a pre-existing web application. Also, Molly offers four levels of parsing based on file extension. Molly parses all embedded MAML tags in .maml files. When .html files are encountered they are sent to the PHP parser with all of Molly's globals, session variables, database tables, functions, and classes available. .php files are sent to the PHP parser, but with none of Molly's environment loaded to interfere with existing scripts. All other files (.txt, .cgi, .htm, etc.) are untouched by Molly.

Future Development

While we have built some real web sites with Molly, we are always working on improvements and additions. Developing documentation is also always a high priority. We welcome any developers who wish to join the effort. Some development projects currently in progress:

Globalization

Molly will support multiple languages, rendering pages in the user's language(s) of choice. Translations will be facilitated by storing all system generated text in .ini files.

E-Commerce

Molly will have payments, catalog, and shopping cart modules. These will each be reconfigurable via MAML tags.

Distance Learning

Modules are being developed to support university distance courses for delivering courses in both synchronous and asynchronous modes.

SVG and Visualization

We have a module under development that is designed to generate Scalable Vector Graphics (SVG). SVG is an XML dialect for creating graphics. In addition to MAML, these modules parse MAGE (Molly Automated Graphics Engine) tags to simplify such tasks as automatically word-wrapping blocks of text (not supported in SVG 1.1). In addition we are building visualization tools for rendering data from databases in a graphical manner. These will range from a simple <mage:piechart> tag to sophisticated spacial data representations and simulations supporting RIT faculty research and teaching in the field of wireless networking.¹²

Dental Informatics

Molly modules are under development to parse DCML (Dental Charting Markup Language)¹³ and facilitate the development of electronic patient records and the sharing of relevant patient information for consultations with specialists.

Conclusion

We believe that the approach embodied in Molly and MAML of developing simple XML solutions to common problems holds the promise of bringing back at least some of the authoring ease of the early web. By facilitating additions by open source developers, and providing an architecture which is easy to adopt, enhance, and incorporate into existing web projects we hope to break the cycle of reinvention which has plagued web application development.

References

¹Bush, Vannevar; As We May Think, Atlantic Monthly (July, 1945)
<http://www.ps.uni-sb.de/~duchier/pub/vbush/vbush-all.shtml>

²http://en.wikipedia.org/wiki/Project_Xanadu

³<http://www.mactech.com/news/archivedisplay.mgi?id=000001907113>

⁴<http://aaa-proteins.uni-graz.at/HyperCGI.html>

⁵<http://www.macromedia.com/software/coldfusion/>

⁶<http://msdn.microsoft.com/library/default.asp?url=/nhp/default.asp?contentid=28000522>

⁷<http://www.sun.com/software/learnabout/java/>

⁸<http://java.sun.com/products/jsp/>

⁹<http://www.php.net/>

¹⁰<http://www.macromedia.com/software/dreamweaver/>

¹¹<http://www.microsoft.com/frontpage/>

^{12, 13}<http://www.lac.rit.edu/civiprojects.html>, <http://civi.rit.edu/>