

Customized Web-Based Data Presentation

Francisco Saiz

Departamento de Ingeniería Infomática
Universidad Autónoma de Madrid
Cantoblanco, Madrid, 28049 Spain
+34 1 397-5014
Email: francisco.saiz@ii.uam.es

Pedro Szekely

Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292, USA
+1 (310) 822-1511
Email: szekely@isi.edu

Patel Devang

Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292, USA
+1 (310) 822-1511
Email: devang@isi.edu

ABSTRACT

This paper presents a language for specifying the presentation of data in Web pages. The language is an extension of HTML that includes constructs for specifying how to present one or more instances of a given class of data, and constructs for tailoring the presentation to the features of the data, to information in user profiles and to the capabilities of the users platform. We describe the architecture of the system, the features of the page specification language, and present examples of generated pages.

Keywords

Authoring and Publishing Tools, Computer Human Interface Issues, Customizing and Personalizing the Web, Data Bases and the Web.

INTRODUCTION

In this paper we address issues related to the dynamic generation of HTML pages. Dynamically generated HTML pages are pages whose contents are generated by software, typically by CGI scripts, or by dynamically loaded libraries that run directly in a Web server. Dynamically generated pages are often used to present data stored in databases or data produced by application programs.

Some efforts have already been undertaken in this direction. For instance, Active Server Pages (ASP), [1], mediate as a bridge between data bases and web pages, where it is possible to specify some of the required features by programming. Similarly, within the consortium W3, various guidelines have been outlined towards such a goal, like the XML language, [5]. This language is envisaged to meet the challenges involved in large scale publications, by extending the HTML language, and allowing declarative definitions of new tags easier to follow by the user.

We concentrate on the following issues:

How to conveniently specify a page that presents any instance or list of instances of a given class data? For example, suppose that we want to construct a page to show the TV schedule for a certain period, as shown in Figure 1 (this page shows a list of

instances of class "TV-Program"). One alternative would be to have someone type in the appropriate HTML using a text editor or a Web authoring tool such as FrontPage. This would work, but would be impractical. A similar page would have to be typed in every day. The Web site featuring these pages would be inflexible. It would not allow users to request the schedule for only certain stations or time periods. It would not be able to keep track of user preferences, and show users a schedule customized to their viewing preferences. An alternative that overcomes these problems would be to store the schedule information in a database, and write software that generates the HTML pages from the database information. In this paper we present a system that enables authors to specify such pages in a language that resembles HTML.

Station	8:00pm	8:30pm	9:00pm	9:30pm	10:00pm	10:30pm	11pm
ABC	Family Matters		Dangerous Minds		Spy Game		Eyewitness News
CBS	Dr. Quinn, Medicine Woman		Early Edition		Walker, Texas Ranger		CBS 2 News Nightcast
NBC	The Pretender		Gridlock				Chanel 4 News
PBS	On Tour		Country Connection With Bob Harvey		Austin City Limits		Peple Near Here
UPN	Cocodrilo Dundee				News 13		Mad About You
WB	Ritmo Latino		Tejano Contry		Puro Tejano	Variety	La Espada De Dios

Figure 1. Presentation of a TV schedule.

How to integrate static HTML with dynamically generated HTML? Most pages that display dynamic information also contain a significant amount of static HTML such as headings, explanations and pointers to related information. In traditional CGI-based schemes, even the static parts of the pages are generated by software. This is most inconvenient, because to change the static parts of the page it is necessary to change the software. An alternative approach, used in ASP, is to embed in the static HTML calls to software that generates the dynamic parts of the page. In this paper we present a system where the static and dynamic parts of the page are seamlessly integrated.

How to tailor data presentation to the needs and preferences of each user? Many Web-sites and Web-casting stations offer users the ability to define a personalized site home-page that shows users the information that they want to see. The page authoring language we present in this paper features rules that help page authors set up user-tailored pages, and pages tailored to classes of users.

How to tailor data presentation to the capabilities of the users platforms? Most Web pages are designed for a "typical user platform" which is usually a Wintel machine with a 256 color 14" monitor and a 28.8kb modem. The effectiveness of the page designs often degrades to the extent that the users platform deviates from the typical platform. For example, if the screen is much smaller, users need to scroll both vertically and horizontally. We expect this problem to become much more severe in the near future with the proliferation of Web-TV class machines, different sized portable machines, and powerful Pentium workstations. The rule-based component of our system can be used to specify platform-specific tailoring of Web pages.

ARCHITECTURE

Figure 2 shows the architecture of the system. Our system, labeled Presentation Agent (PA), is invoked by the Web-server when it receives a request for a file with extension DPML (Data Presentation Markup Language). The system is written in Java, and we use the Java Server Development Kit for the server part. Upon receiving a request, the PA performs the following actions:

1. Read the user profile, if one exists for the user identified in the HTTP request.
2. Read the DPML file and build a tree of presentation objects that represents the structure of the HTML document to be generated. For each presentation object perform the following steps (the steps will be explained in more detail in the sections below):
 - a. Read the meta-model files for the classes of data referenced in the DPML file.
 - b. Fetch the data needed to construct the presentation. The meta-model files tell the PA where to fetch the instances of each class. Both relational databases and ASCII files are supported. In the case of relational databases the meta-data file contains the SQL queries needed to fetch the attributes of an object. Our example uses ASCII files.
 - c. If the data to be presented is a list, replicate the corresponding node in the tree as many times as there are elements in the list. For example, this occurs for lines 18 and 24 in Figure 4. The `tr` presentation object (line 18) is replicated as many times as there are TV stations. The replication makes a deep copy of the tree, so the embedded `td` objects (line 19 and 27) are also replicated for each TV station. For line 24 each `td` replica is again replicated, this time for each program that the station offers on the given day.
3. Once the complete tree of presentation objects is instantiated, the PA applies the style rules. The style rules can set the properties of each object such as the color, font, etc. The rules can test attributes from the HTTP request, properties of the data, and attributes from the user profile.
4. Finally, the PA generates HTML and sends it to the Web-server, which sends it back to the Web-browser.

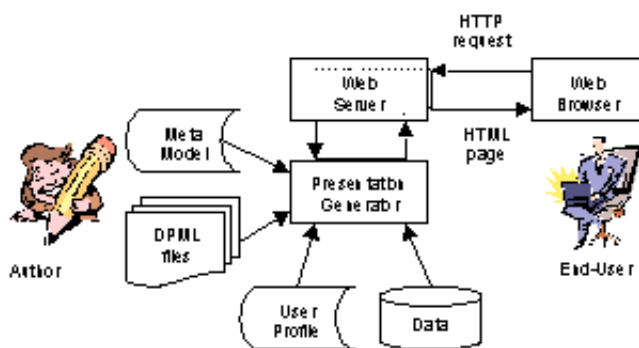


Figure 2. Architecture

META-MODEL

Our meta-modeling language is a very simple data description language designed to support two PA needs. First, meta-models, define the format for shipping data from a database or application program to the PA. In addition to relational database connectivity, we support a simple ASCII format that is easy for applications to generate. Second, meta-models give authors the means to associate arbitrary meta-information with the object definitions. Authors are expected to develop the meta-model and a collection of presentations (DPML files) together. If they need some piece of information about a class of objects in a DPML file they can define a property in the meta-model to represent it.

1. LongMetaData Duration { units="half hour" }
2. ObjectMetaData Program {}
3. [name(Name) {}
4. duration(Duration) {}
5. kind(Kind) {}]
6. ObjectMetaData Station {}
7. [stationName(Name) { attributeName="Station" }
8. programs(Program) { multiValued="yes" }]
9. ObjectMetaData Schedule {}
10. [day(String) {}
11. times(String) { multiValued="yes" }
12. programming(Station) { multiValued="yes" }]

Figure 3. Meta-model specification.

Figure 3 shows the meta-model specification for our TV schedule example. Line 1 shows the specification of a primitive meta-object. Duration is a number that represents the number of half/hours that a program lasts. We encourage authors to define their own meta-objects for the primitive types of their application domain. This is useful because authors can associate style rules with meta-objects to specify properties of their display.

Lines 2, 6 and 9 show the specification of some structured objects. Line 2 defines the meta-object Program as having three attributes (name, duration and kind). Each attribute has a name, a type (specified in parenthesis). Single inheritance between structured meta-objects is supported. The meta-object Station has the name of the station, and a list of the programs that a station will show in a given day. Finally, a Schedule specifies the day and time covered by the station, and a list of stations for which the schedule object has information.

The meta-modeling language allows authors to associate any properties they want with meta-objects or with attributes. These properties are often referred to from within DPML files. For example, line 11 in Figure 4 refers to the AttributeName property of the Station meta-object. This property specifies the label that appears in the top-left corner of the table shown in Figure 1.

DPML FILES

DPML files (Data Presentation Markup Language) specify the HTML code that the PA generates to present a collection of data objects. DPML is a superset of HTML. This has several benefits. First, authors do not need to learn a new specification language from scratch. They must only learn our additions to HTML. Second, static and dynamically generated HTML are completely integrated: authors can embed presentations of data within static HTML, and can also embed HTML within the presentations of data.

Data Presentation Specifications

The basic idea in DPML is to allow authors to extend HTML specifications with expressions that fetch values from an object or its meta-data. When the PA processes the DPML file, it substitutes these expressions by the values that they return. Also, as mentioned in step 2a) of the PA presentation algorithm in the previous page, when an expression returns a list of objects, the corresponding HTML tags are replicated. The rest of this section describes the DPML extensions to HTML in more detail.

1. <html>
2. <head>
3. <link type=text/css rel=stylesheet
4. href="http://erastus.isi.edu/MIPGenerator/CSS/TvStyle.css">
5. </head>
6. <div object='?schedule in tv:Schedule.instances()>
7. <h1>TV Programming for <value ?schedule.day></h1>
8. <table border=1 cellpadding=2>

```

9. <tr>
10. <th>
11. <data tv:Station.stationName.getProperty(attributeName)>
12. </th>
13. <th object=?time in ?schedule.times"
14.     bgcolor=PowderBlue>
15. <data ?time>
16. </th>
17. </tr>
18. <tr object='?s in ?schedule.programmings'>
19. <td
20.     style-rule= color alternates ("SaddleBrown", "Indigo")
21.         when ?s.stationName changes>
22. <data ?s.stationName>
23. </td>
24. <td object=?p in ?s.programs'
25.     style-rule='colspan=?p.duration'
26.     class=?p.kind>
27. <data ?p.name>
28. </td>
29. </tr>
30. </table></div></html>

```

Figure 4. DPML specification for the TV schedule presentation shown in Figure 1.

Variables and Expressions

DPML allows authors to declare variables to hold the data to be presented, and expressions that specify how to compute the values for them. DPML supports three types of variables: variables of type *object* hold instances of objects to be presented, variables of type *meta-data* hold the meta-data of an object, and variables of type *attribute* hold the names of attributes of structured objects. Variables can hold either a single value or a list of values.

For example, line 6 in Figure 4 defines an object variable called `?schedule`, whose value is the list of all instances of the meta-object called `Schedule` in the name-space called `tv`. There is a structured representation of all the meta-data into various name-spaces, each one holding the data referred to the same semantic context. Thus, each time the PA needs to fetch the data, it retrieves the necessary values from the corresponding name-space.

The expression language also supports filtering and sorting qualifiers. These are not used in our simple example, but these qualifiers can be used to select subsets of data objects and attributes to be presented, and to specify the order in which elements of a list should be displayed.

When filtering data objects, a specific language can be used to represent predicates on them. For instance, in the example shown in this paper, an expression such as `"object=?p in ?s.programs where ?p.duration>2"` could be considered. Likewise, sorting declarations allow the selection of a given sequence of objects to be displayed.

The <data> Tag

The preceding section described how authors can define variables to hold the data to be presented. The `data` tag is used to specify that the value of a variable or an expression should appear in the generated HTML. The syntax of the value tag is `<data expression>`. The PA will evaluate the expression, convert its value to text, and output it in the stream of generated HTML. For example, in line 27 the specification `<data p.name>` produces the name of a program as the contents of a cell in the table.

In case the evaluation of an expression returns a non primitive object, it is not clear what the conversion to text should be. In this case, *substitution rules* can be applied to determine a detailed DPML expansion of such an object. Suppose the expression "<data ? schedule>" appears within a DPML file. In this, the DPML language allows the specification of rules that define the way this kind of object is to be converted to DPML, and eventually, HTML objects.

A generic substitution rule consists of a condition on an object data, and an action that is a piece of DPML code. Such code is to replace an expression like the above one. Thus, programming is additionally supported in DPML files, being possible to take into consideration the specific properties of the data, the user profile, and the platform, as appropriate.

Example

This section explains how these DPML constructs are used to specify the display shown in Figure 1. The complete DPML specification is shown in Figure 4. We will postpone the explanation of the style rules until the next section.

Lines 1 through 5 are standard HTML. Line 6 is the first HTML tag that uses a DPML extension. The `div` tag is an HTML tag that defines a section of a document. It does not produce any output, but serves as a placeholder to include style information. In our example we use it to specify the root object of our presentation. Line 6 declares variable `schedule` and binds it to the list of instances of meta-object `Schedule`. In our example we only have one instance. Should there be more than one, the output would contain a heading and a table for each instance of `Schedule`.

Line 7 defines the heading that appears above the tables. It illustrates how HTML and DPML can be freely mixed within a specification. The heading consists of the words "TV Programming for" followed by the value of the expression `schedule.day`, which is "Sat, June 28".

Line 8 introduces the table. Lines 9-17 define the table headings. Line 11 produces the text "Station" as explained in the previous section. Lines 13-15 produce the headings "8:00pm" through "11pm".

Lines 18-29 specify the rows of the table. Line 18 specifies that there should be a row for each value of the programming attribute of the schedule. Recall that this produces the list of all Station objects that we wish to present. The first column of the table (lines 19-23) shows the name of the station. The other columns show the programs that will be aired. Note how the `colspan` attribute (line 25) is defined with an expression so that programs that last more than half an hour occupy more than one column.

Presentation Tailoring

Our system supports three different mechanisms to tailor the format of presentations: style-rules, conditional tags and cascading style sheets (apart from the substitution rules described above).

Style Rules

Style rules are condition/action pairs. The condition part specifies the situations when the rule should be applied, and the action part specifies values for HTML attributes. Our example does not contain an instance of a style rule with a condition part. Such rules are used to express something like "if `price>200` then `color=red`".

The rule in lines 20 and 21 illustrates a more complex action. This rule specifies that the color of the station label shown in the first column of the table should alternate between `SaddleBrown` and `Indigo`.

The difference between style and substitution rules lies in the action to be performed. The latter has a more complex action and, in fact, a style rule is a particular case of a substitution rule where the action appears close to the tag where it is to be applied, and it contains HTML attributes to be assigned to a certain tag.

Conditional Tags

Conditional tags allow authors to specify a boolean expression to decide whether a whole fragment of HTML should be included in the output. The syntax is as follows:

<tag condition=boolean-expression>contents</tag>

If the boolean-expression evaluates to true, the contents of the tag are processed, and the HTML it produces is sent to the output stream. If the condition is false, the contents are skipped, as if the whole tag had not been present.

Cascading Style Sheets

Cascading Style Sheets (CSS) are an HTML extension under consideration by the W3 consortium. CSS allow authors to declaratively specify in a separate file the appearance of every HTML tag. In our example we use style sheets to specify the color of each cell according to the category of program (line 26). The css file specified in line 3 specifies the colors to be used.

The PA allows authors to make extensive use of cascading style sheets. The PA automatically defines a CSS class for every meta-model object and attribute defined in a meta-model file. The generated HTML includes the appropriate style-sheet invocations. For example, the HTML generated for line 18, where the rows of the table are defined is <tr class=Station>. The style sheet file can then contain style sheet definitions of the form .Station {background=yellow}, which cause every tag that includes the Station class to have a yellow background.

RELATED WORK

CGI scripts are the most common technique to dynamically generate HTML pages. CGI scripts offer developers complete flexibility and the ability to program arbitrary mapping from data to HTML. However, writing CGI scripts is beyond the skill of many Web publishers. The DPML language provides a much more accessible alternative, and provides sufficient power to generate a wide range of pages.

DPML was inspired by Active Server Pages (ASP), [1]. ASP, like DPML, allows specifications for data presentation to be embedded within HTML text. The difference is that, in ASP, the specifications must be written in a programming language (VBScript or JavaScript). The DPML language is higher level, leading to more succinct and easy to write specifications.

DPML also borrows ideas from model-based user interfaces, [2, 3]. Specifically, the replication and conditional constructs are based on similar constructs found in Humanoid and Mastermind. The notion of style rules comes from ITS, [4], whose major feature is the support of multiple, rule-based user interface styles through a declarative statement of such goals. The main difference with these systems is that DPML provides a much more accessible language for specifying the models.

Within this stream of tools that support the representation of data in WWW, there have also been remarkable trends over the last years. Specifically, WDB, [5], provides a software tool-set that simplifies the integration of SQL databases into WWW providing access to the contents without writing any code. CATALOG, [6], is another system that creates WWW pages from databases by using a declarative specification which does not require any programming skill for such a task. However, DPML provides much more flexible capabilities when integrating data into WWW pages.

SUMMARY AND FUTURE WORK

The work reported in this paper represents the first step towards a system that meets the goals outlined in the introduction. We have written meta-models and DPML for three small domains, and are about to embark on the creation of larger examples. We expect to refine the system and release it to the public by the end of the summer. In the future we want to enhance the rule language to allow rules to transform the HTML tree being generated depending on presentation conditions for each particular case. The goal is to implement heuristics that improve the presentation of data depending on user preferences, platform characteristics, or dynamic aspects of presentations.

Another goal is to enhance DPML to support the specification of user interfaces implemented in applets embedded in WWW pages. This is yet another inspiration of the techniques used in model-based user interfaces applied to the WWW framework, which is actually a user interface with special requirements.

REFERENCES

1. Microsoft Corporation: *Active Server Pages*. <http://www.microsoft.com/iis/default.asp>

2. Szekely, P., Luo, P., Neches, R.: *Beyond Interface Builders: Model-Based Interface Tools*. In Ashlund S., Mullet K., Henderson A., Hollnagel E., White T. (eds.): Proceedings of INTERCHI'93. New York: ACM Press 1993 (pp. 383-390).
3. Vanderdonckt, J (ed) : *Computer-Aided Design of User Interfaces*. Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces (CADUI'96, Namur, Belgium, 5-7 June 1996).
4. Wiecha, C., Bennett, W., Boies, S., Gould, J., Green, S.: *ITS: A Tool for Rapidly Developing Interactive Applications*. ACM Transactions on Information Systems, Vol. 8, No. 3, 204-236 (July 1990).
5. W3 consortium: *XML Activity*. <http://www.w3.org/XML/Activity.html>
6. Rasmussen, B. F.: *WDB - A Web interface to SQL Databases*, <http://www.dtv.dk/~bfr/wdb/>
7. Martinez, J.M., Morán F.: *Catalog: A WWW gateway for DBMSs*. Proceedings of WebNet96, San Francisco, <http://aace.virginia.edu/aace/conf/webnet/proc96.html/>.

