Erschienen in: Proceedings, 4th International Conference on Web Information Systems Engineering workshops : 1st Workshop on Multichannel and Mobile Information Systems ; 13 December 2003, Roma, Italy / Santucci, Giuseppe (Hrsg.). - Los Alamitos, Calif. [u.a.] : IEEE Computer Society, 2004. - S. 234-242. - ISBN 0-7695-2103-7
https://dx.doi.org/10.1109/WISEW.2003.1286807

Metatemplate Driven Multi-Channel Presentation

Michael Grossniklaus, Moira C. Norrie and Patrick Büchler
Institute for Information Systems
ETH Zurich
Switzerland
{*grossniklaus,norrie*}@*inf.ethz.ch*

Abstract

The separation of content, layout, structure and view that is enforced by the various emerging content management systems and web modelling tools, renders the implementation of suitable editors for content providers very difficult. As most of these systems work on the basis of XML and XSLT and support multiple languages and presentation channels, creating and editing content has become more tedious and challenging. Various attempts at creating graphical XSLT editors have been made but most of them lack the versatility to be used in conjunction with a content management system. In this paper, we present a solution based on generic metatemplates. We then describe how we exploited this approach for an editor integrated into a content management system that we have developed.

1. Introduction

Parallel to the growing spread and increasing acceptance of XML [15], a lot of work has been done to facilitate the generation and editing of XML documents. XML has become especially important in the field of content management where it is commonly used in conjunction with XSLT [20]. The combination of these two technologies provides a standardised technique to separate content from presentation and to provide multi-channel access to data. In this context, XML is used as a structured representation of the content to be delivered, whereas XSLT contains the layout as a set of template rules that are used to transform the XML document into a graphical representation such as XHTML.

Separation of the content, structure, layout and view of the data, and providing multi-channel access to it, are key requirements of any content management system. Beyond the traditional separation of content, structure and layout, we have shown in [7] that it is also valuable to introduce the concept of a view to support finer levels of personalisation. A view in our model has been defined to perform similar tasks to that of its counterpart in conventional database systems. Hence, it allows selection of certain attributes of data objects to be displayed and governs the aggregation of associated content.

Furthermore, in addition to bringing structured content from a database to the web, a content management system should also support the management of document elements such as texts, images and links in multiple formats and languages. Suppose, for example, that our research group, the Global Information Systems (GlobIS) group, would like to provide information concerning group events as well as general breaking news on the group's intranet. Of course, the main content to be made available comes from an existing database containing all data about the group schedule, its members, news and events. Nevertheless, there are document elements such as the logo of the group or a welcome note on the main page, for instance, that do not come from the database and that have to be handled and managed separately by the system. A quick examination of any state-of-the-art web site (e.g. eBay, CNN or Amazon) will reveal that the number and range of such content elements is considerable, especially when they offer different language and/or regional versions.

Finally, a content management system should also allow multiple users with different user roles to work on a web site, while at the same time separating their concerns, checking user permissions and ensuring that business workflows are followed. A detailed analysis of different user roles and the associated requirements for content management systems can also be found in [7]. We call systems that fulfill all of these requirements "second generation" content management systems to emphasise the fact that the systems currently in use often satisfy only a subset of these requirements.

In this paper, we will focus on the requirements and roles of the web designer and the web engineer. A web designer is responsible for the graphical layout and the presentation of a web site. In the case of multi-channel access, this task is further complicated as it is necessary to specify these for each channel supported. Coming from an artistic background as a designer, the knowledge of a web designer does not include technical aspects and programming. Thus, the ability to express a layout in terms of templates cannot be expected. The web engineer on the other hand has just that knowledge, but is lacking the graphical skills of the designer. Hence, a content management system has to allow both users to execute their tasks and duties, while at the same time avoiding having to mix the skills associated with their respective roles.

Clearly, XML together with XSLT is at the heart of any such modern content management system. Although very good editors exist to work with documents containing content represented in XML, few editors also offer satisfactory tools to generate a graphical representation of this content in a visual way. Therefore web designers that work with an XML-based content management system, often have to learn to program transformations with XSL templates, or even worse, in a proprietary template language offered by the system of their choice. Hence, to fulfill the tasks associated with his user role, a web designer has to acquire the skills that are typical of the role of a web engineer. The afore-mentioned requirement of separating concerns is thus violated as knowledge associated with different user roles has to be mixed to achieve a task within such a system.

It is therefore necessary to provide web designers with tools that allow them to work with a content management system without having to acquire new skills. Such an editor should build on the principles and techniques that are already known to designers from the times when web design was done purely in HTML. However, building and implementing an editor that permits users to work with a graphical layout visually is far from trivial, as it involves the generation of XSL templates that are needed to transform the content contained in the XML document to the representation envisaged by the designer. Difficulties arise with XML and XSLT as compared to pure HTML because the layout is no longer edited directly and the corresponding markup is generated behind the scenes only. Templates that otherwise would be programmed by hand, have to be generated by the editor and, since XSLT is a fully-featured functional programming language, the generation of such code is not straightforward if one plans to support a reasonable amount of the possibilities offered by XSLT.

Supporting multiple presentation channels, another requirement of a "second generation" content management systems, is also a major challenge that such an editor has to overcome. Since the days when the world wide web was purely HTML-based, a lot has evolved. New devices such as WAP capable mobile phones, iMode devices and PDAs have recently come to the market. Those devices need special support in many respects. Whereas PDAs need a different representation in terms of complexity of the data presented only, mobile phones and iMode devices require a whole different set of new markup languages, such as CHTML or WML, to describe and render the layout. Consequently, a tool to generate such layouts has to be general enough to support all of these markup languages, but it should also be possible to support features of a target language that are not common to all languages, even allowing for new markup languages that are yet to emerge.

Another problem arises from the fact that, generally, the structure and semantics of the XML document are not known to the editor a priori. Thus, it is very hard to offer the designer a sensible choice of graphical representation of the elements in the XML document. Many attempts have been made to create such all-purpose tools that will let a designer create a layout of an arbitrary document. As we will present in section 2, many of them fail due to the generality they try to achieve. We strongly believe that although it is very difficult to edit XSLT-based layouts visually in general, it is feasible in the well-defined area of a content management system, where various constraints are enforced by the system that do not hold in general. For instance, most content management systems generate the XML that is used in the transformation process at runtime and according to a well-defined mapping. Hence, the structure and semantics of the source document are known and this knowledge can be used in the implementation of an XSL template editor. In section 3, we present our own solution to the various problems described above and, in section 4, we show how this approach has been exploited to build a visual editor for our own XML-based content management system. Finally, in section 5, we give conclusions and an outlook to future work.

2. Related Work

Many attempts at building tools to support the visual generation and editing of layouts for multiple presentation channels have been made. A lot of work is still ongoing as XSLT has not yet reached the full potential of its possible application domain. Also, there are only very few "second generation" content management systems that are really built to deal with the requirements presented in section 1. We believe that, with the broader emergence of such systems, the need for editors to work with these systems will also grow. In this section, we will give a short overview of some existing approaches, both from the commercial and research sectors. For each solution, we will briefly describe the issues they are solving and point out some possible points of critique where we believe that problems might arise when coping with future developments and requirements.

Altova's **XMLSpy** [17] is probably one of the best-known editors for XML documents that is commercially available at this time. The product suite offers a wide range of tools to create, edit and maintain XML documents. It supports a large number of standards, such as Document Type Definitions (DTD), XML Schema [16], XPath [19] and, in particular, XSLT. Although XMLSpy itself is not capable of editing XSL templates graphically, the suite contains the application Stylesheet Designer to do so. When working with Stylesheet Designer, the user is presented with a tree that represents the structure of the XML document with which he is working. The tree, however, only displays the elements and not the content of the XML source. Dragging such an element to a visual preview panel, triggers the creation of appropriate templates and match clauses based on the position of the element within the XML structure. On the preview panel, the user can more or less work as was previously possible with a visual HTML editor. There are controls to align text, change the foreground and background colour and it is possible to define tables. The editor also offers different representations of the layout being created. A user may switch from the edit view described before to the textual representation of the templates or to the preview of the transformed document as it would appear later in the web browser.

Using Stylesheet Designer is very intuitive and straightforward, nevertheless the application has some shortcomings when it is to be considered as a frontend to a content management system providing the features described above. As the representation of the XML document only displays structure and leaves out content, selecting the right element to drag onto the preview pane becomes very tedious. Imagine, for instance, that there is, within the news site of the GlobIS group, a web page for every month to display the events taking place in the respective month. To define the match clause in the XSL templates to select the appropriate event elements for such a page from the XML tree, it would be important to see the dates on which the events occur. Unfortunately, these are hidden by the editor. But there are also more severe limitations. Stylesheet Designer is limited to one presentation channel, namely XHTML, only. Supporting other channels is not possible and is also not intended by Altova. Another drawback that limits the use of this tool for content management, is its limited support for the needs of today's designers. Only very simple designs can be created visually. As there is no support for creating XPath queries, i.e. defining which elements from the XML document are transformed, graphically, sophisticated layouts can only be achieved by editing the XSL templates by hand, thereby again violating the separation of concerns.

Another approach very similar to Altova's XMLSpy is **eXcelon Stylus Studio** [10] developed by the eXcelon Corporation. Both share the concept of a tree to visualize the XML source document and a preview pane to create and edit the XSL templates graphically. When XML elements are dragged to the preview pane in Stylus Studio, the user has to decide what kind of XSL instruction should be applied to the selected data. In contrast to Stylesheet Designer, Stylus Studio offers support to create XPath expressions graphically, thereby broadening the spectrum of possible designs that can be created visually. Although the generation of a few web pages with Stylus Studio is possible, it has some limitations that would aggravate creation of larger pages or even whole web sites. Stylus Studio also hides the content of the XML document from the user and displays the structure only, thereby creating the same problems as Stylesheet Designer. Also, there are no tools to generate tables from collections of elements automatically, making the application unsuitable for data intensive web sites such as CD or book stores where a lot of data is very structured and regular and thus presented in lists or tables. Even though Stylus Studio has more visual editing capabilities than Stylesheet Designer, when working with state-of-the-art layouts, designers will also have to refrain from working graphically and instead create or change the XSL templates manually.

In autumn 2002, IBM released a technology preview of a graphical XSL editor that can be plugged into its **IBM Websphere** [12] software platform. Websphere is a content management system, application server, and e-business middleware, that supports almost every XML-based standard currently in use. Given an input XML document, the XSL editor can be used to graphically create and edit a stylesheet. IBM also uses a tree to represent the structure of the source XML. In contrast to the other solutions described above, the XSL editor also displays the data in the XML document, thereby facilitating locating the element for which the user wants to create a template. Again, there is a preview pane where the layout is visually editable. IBM provides powerful features that make the XSL editor a great tool for web site development and a real candidate for integration into a content management system. The XSL expression designer, for instance, is a graphical utility to define to which elements a given template rule should be applied and this makes working with XPath very easy. But the XSL editor is not capable of hiding XSL completely from the user as well, and is therefore probably better suited for web engineers than for web designers.

Another graphical editor is **Xopus** [18], an open source browser-based editor implemented in JavaScript. To work, Xopus needs an XML document containing the content, an XSL stylesheet describing the layout and an XML Schema defining the structure of the content. Based on Microsoft's XML library MSXML 4.0 [8] that is part of Internet Explorer 5.5 and above, Xopus renders the content on the client side using the templates from the stylesheet. The

XML Schema is used to govern the editing process and to decide which type of element can be created at what point in the structure of the XML document. At the end of the editing session, the updated content and layout is sent back to the server. Since Xopus also allows content to be edited and is easily integrated into an XML-based content management system, it is a very good tool for editors with an easy-to-use interface to input data. Xopus' support to change the presentation is, however, very limited as the overall layout is fixed and only attributes such as text style, size and colour can be changed. The editor therefore is more suited for content providers than for web designers.

The candidate recommendation **XForms** [14] of the World Wide Web Consortium (W3C) is an attempt to separate the purpose of a web form from its content. To do so, XForms provides the user with a standard set of visual controls, such as input or text fields, option, choice and submit buttons. The conceptual model of XForms also supports various events and triggers to build powerful user interfaces. Although the purpose of XForms is certainly not to help with implementing graphical tools to edit XSL stylesheets, the underlying method of defining a standard set of visual controls is very interesting. As the XForms project clearly shows, it is important to specify a uniform, high-level model, that abstracts from the concrete implementation in order to gain independence of the target presentation channel.

Having discussed a set of approaches that have emerged from the industrial sector, we now consider one solution coming from the academic community. **WebML** [5, 3, 1, 4] is without any doubt the most advanced and best-known research project in the field of modelling web sites. The maturity and completeness of WebML allowed its creators to found WebModels, a company committed to the development of commercial technologies based on the conceptual basis of WebML. WebRatio [11], a CASE tool for the Web that exploits the WebML design methodology, is currently their main product. To define layouts and generate XSL templates, WebRatio uses EasyStyle, a process similar to the one we present in this paper. Although the basic concept is the same, there are some important differences between EasyStyle and the approach presented in section 3.

Many of these differences stem from the fact that WebML is primarily targeted at designing data-intensive or database centric web sites, i.e. web sites where most of the content is very structured and regular. Support for the requirements presented in section 1, however, is very limited in WebML. WebML therefore has to be classified as a modeling language for web sites rather than a content management system. Hence, WebML offers no built-in concepts for multiple presentation channels and multi-format content, such as multi-lingual texts or images. Consequently, these issues have also not been considered in EasyStyle. Coming from the database community rather than the document management community, WebML also lacks capabilities to manage document types such as images and texts that are not coming from an underlying application database. This constraint is also mirrored in EasyStyle, as design elements such as corporate logos or buttons have to be hardcoded into the templates.

```
<table>
 <tr>
  <td>
   <easystyle:landmark-area-link position="1">
    <easystyle:link image="weather.jpg" />
   <easystyle:landmark-area-link />
  </td>
 </tr>
 <tr>
  <td>
   <easystyle:landmark-area-link position="2">
    <easystyle:link image="sports.jpg" />
   <easystyle:landmark-area-link />
  </td>
 </tr>
 <!-- subsequent links omitted for conciseness -->
</table>
```

Figure 1. Example of an EasyStyle Template

Figure 1 shows an example of an EasyStyle template that renders a simple navigation consisting of a table of links. The images that represent the links as graphical buttons, e.g. an icon and a textual label, are not treated as content data as they are referenced from the stylesheet directly. Doing so prevents the content management system from handling these objects and hinders support for multi-lingual or multi-format content. Obviously, migrating such a website to a new language is not straightforward as all templates have to be rewritten to replace the references to the images. As the model of WebML uses units such as *scroller units* or *index units* that are very tightly coupled to the final presentation of the data, this coupling can also be found in EasyStyle. Our main focus, however, is to separate the layout generated in the end from the structure of the data on the web page. Thus, the approach presented in the next section is not coupled to presentation issues of the underlying model. Finally, it remains uncertain what support EasyStyle has to offer for personalisation and multiple user-roles. The proposal of WebML to work with multiple individual *site maps* seems to be unworkable in practice as this approach is far too coarse to tackle personalisation.

In the next section, we will present our approach that is designed to address the problems presented in this section and to allow flexible multi-channel presentation using the concept of metatemplates.

3. XCM Metatemplates

In this section, we will present our approach to multi-channel presentation using the concepts of metatemplates. Metatemplates are XSL templates, that instead of generating the final markup for the layout, generate other templates to do so. Using this additional step of indirection enables this approach to cope with the requirements described before.

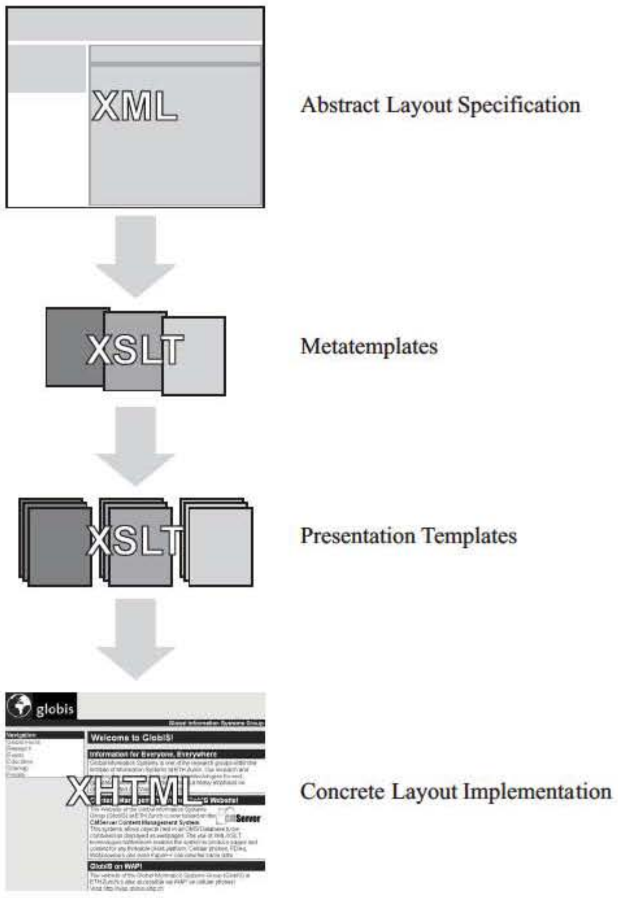


Figure 2. The Metatemplate Process

The entire process shown in figure 2 consists of four basic steps. Before explaining each of these steps in detail, we briefly give an overview of the process. At the beginning of the process is the abstract specification of the layout. This generic specification is expressed in XML in order to be able to apply the metatemplates to it. The metatemplates then transform the layout into a set of templates. Multiple metatemplates can be applied to achieve multi-channel presentations. The generation of templates is done at design-time and therefore has no negative impact on the overall performance of the system. At run-time, the templates generated in the second step are then used in the third step to render the final representation by applying them to the content of the web page that is also represented in XML. Finally, in the fourth step, this representation is sent back to the browser and displayed to the user. In the following we will explain the entire process step-by-step in more detail.

To allow designers to generate layouts that work for multiple presentation channels, it is important to represent the features they have in common in a standard way. It should, however, still be possible to extend this representation with features that are special to one presentation channel only or requirements that are yet to emerge. We have designed an XML-based markup language to express such generic layouts. All visual layouts partition a two-dimensional space into non-overlapping regions where content is placed. Hence, we have chosen the notion of a *grid* to model the basic content of a layout. A grid can contain an arbitrary number of columns and rows. The exact number of these two dimensions is determined by the concrete layout in question. *Cells* are used to store content and can be located at any point (x, y) where x denotes the column and y denotes the row in the grid. Although it is also possible for cells to span multiple columns or rows, it is not allowed for any two cells to overlap. To profit from full flexibility, cells can also contain other layout grids thereby allowing powerful reuse of previously defined layouts. This approach of modeling the layout is very similar to the one used in most toolkits to implement graphical user interfaces. For instance, in the Abstract Window Toolkit (AWT) within the Java Platform, there are layout classes such as *GridLayout* or *GridBagLayout* that are used to display graphical components on a two-dimensional plane.

To determine which types of content should be provided as standard visual controls for all presentation channels, we determined the set of document elements that can be found in all of these channels. As a result, we decided to offer *text*, *image*, *uri* and *link*. It is important to distinguish here between a Universal Resource Identifier (URI) and a link. The URI is merely a pointer to a resource on the world wide web, whereas the link is the combination of such a pointer with a resource, for example, a text or image. Also, the set of attributes that are supported on all presentation channels, and thus should be editable in these standard types, had to be determined. As presentation channels vary greatly in this respect, the set of common editable attributes is small. It comprises properties such as font size, style and colour for text or width and height for images. It becomes clear, that those common features provide very limited support for the needs of today's designers. Hence, additional concepts have been built into the layout specification to ease customisation to the needs of a specific layout. One of these concepts is extensibility, the key to provide additional, more complex types than the ones mentioned before. Imagine again the situation of the news site of the GlobIS group. In their database, they have information about high-level application objects such as news, events, talks and persons. It is unworkable to break this information down to texts, images, etc. Hence, our approach supports the definition of new additional types, that could be used in this example to represent the application concepts of the GlobIS group's database. An excerpt of the Document Type Definition (DTD) of our layout markup language, illustrating a link component, is shown in figure 3. The entire specification, detailing every attribute of all standard types, is presented in [2].

```
<!ELEMENT link (component|text|picture)>
<!ATTLIST link
   name        CDATA    #REQUIRED
   oid         CDATA    #REQUIRED
   parent_id   CDATA    #REQUIRED
   background  CDATA    #REQUIRED
   width       CDATA    #REQUIRED
   height      CDATA    #REQUIRED
   x           CDATA    #REQUIRED
   y           CDATA    #REQUIRED
>
```

Figure 3. Excerpt of the Layout DTD

The next step in the presented generation process is the application of the metatemplates. Metatemplates are an additional step of indirection, necessary to generate more than one set of templates from the described layout representation. Suppose that a designer has defined a layout for the web page of each member of the GlobIS group within their web site. To support multiple presentation channels, a set of template rules is required to transform the content into the appropriate markup for each such channel. Further, it is imaginable that there are personalisation options available on the web site, allowing a member of the GlobIS group to access the data through a more advanced interface than a standard user. This option also calls for a new set of templates. Thus, it is required to generate an arbitrary number of set of templates from the basic layout. Clearly, this could also be done using software modules to generate the various stylesheets. This approach however is far too static and cannot cope with today's requirements, as it has very poor support for extensibility. Using metatemplates to generate the sets of templates, on the other hand, is very flexible and additional channels or variants are added easily. Figure 4 illustrates a metatemplate for a link. Due to the scope of this paper, it is a very simple version that merely creates an HTML anchor tag and sets the `href` tag accordingly. Note how the template delegates the rendering of the resource contained in this link to another template by invoking `xsl:apply-templates`. Although the template shown in figure 4 seems to be quite long and complicated at first sight, it is important to bear in mind that these templates are implemented by expert programmers and do not have to be updated or replaced often. Metatemplates are not affected by mere changes of the layout of the website as they do not encode the layout itself. The only point in time that calls for new metatemplates to be written is when a new presentation channel or additional features of an existing one are to be supported.

```
<xsl:template match="link">
 <xsl:comment>
    Template rule matching a CMLink
 </xsl:comment>
 <xsl:element name="xsl:template">
  <xsl:attribute name="match">
   <xsl:text>//webobject[@oid='</xsl:text>
   <xsl:value-of select="@parent_id"/>
   <xsl:text>']//webobject[@oid='</xsl:text>
   <xsl:value-of select="@oid"/>
   <xsl:text>']</xsl:text>
  </xsl:attribute>
  <a>
   <xsl:element name="xsl:attribute">
    <xsl:attribute name="name">href</xsl:attribute>
    <xsl:element name="xsl:value-of">
     <xsl:attribute name="select">
      <xsl:text>
         property[@name='target']/webobject/
         property[@name='reference']/string
      </xsl:text>
     </xsl:attribute>
    </xsl:element>
   </xsl:element>
   <xsl:variable name="resource_id">
    <xsl:value-of select="child::*/attribute::oid"/>
   </xsl:variable>
   <xsl:element name="xsl:apply-templates">
    <xsl:attribute name="select">
     <xsl:text>
        //components/webobject[@oid='
     </xsl:text>
     <xsl:value-of select="@oid"/>
     <xsl:text>
        ']/property[@name='resource']/webobject[@oid='
     <xsl:text>
     <xsl:value-of select="$resource_id"/>
     <xsl:text>']</xsl:text>
    </xsl:attribute>
   </xsl:element>
  </a>
 </xsl:element>
</xsl:template>
```

Figure 4. Metatemplate for a Link

Working with metatemplates is more challenging than implementing transformations to a desired markup directly in XSL. A metatemplate combines three levels of markup. At the lowest level is the markup that will appear in the final document (e.g. the `<a>` tag in our example). One level above are the the tags that will make up the template generated by the metatemplate. It is important to understand, that since, in metatemplates, XSL is used to generate XSL, the tags to be generated cannot simply be included in the metatemplate. The XSL transformer would not be able to decide which directives to execute in order to generate the template and which to include in the output. Our first attempt to solve this problem involved using different namespaces, a core concept of XML. However, some of the transformers that we are using were unable to cope with such stylesheets and we found that XSLT itself provides a concept to output such nodes. The directive `xsl:element` can be used to generate nodes that otherwise would be misinterpreted by the transformer. Together with the element `xsl:attribute`, XSL provides everything needed to generate XSL. Note how these two concepts have been applied in the example given in figure 4. The last and top-most level of markup is the XSL that is really executed by the transformer. As it is very straightforward, we refrain from a detailed discussion in this paper.

```
<!--Template rule matching a CMLink-->
<xsl:template match="//webobject[@oid='OM_2134']//
   webobject[@oid='OM_2152']">
 <a>
  <xsl:attribute name="href">
   <xsl:value-of select="property[@name='target']/
      webobject/property[@name='reference']/string"/>
  </xsl:attribute>
  <xsl:apply-templates
     select="//components/webobject[@oid='OM_2152']/
     property[@name='resource']/
     webobject[@oid='OM_2158']"/>
 </a>
</xsl:template>
```

Figure 5. Template for a Link

The next and final step of the proposed generation process is the application of the generated templates to the content to generate the final markup. Figure 5 gives the template that is generated by the metatemplate given in figure 4. Looking at the template, one can see that all XPath queries, i.e. specifications of which elements match this template or are selected by the `xsl:apply-templates` directive, have been generated automatically by the metatemplate. Clearly, selection based on object ids (e.g. `OM_2134`) as done in the example is very restrictive and leads to poor reusability of the generated templates. This behaviour, however, can be changed very easily by using a slightly more sophisticated metatemplate that would produce more general match and select clauses. Hence, it is rather a demonstration of the flexibility of the metatemplate approach, than a disadvantage.

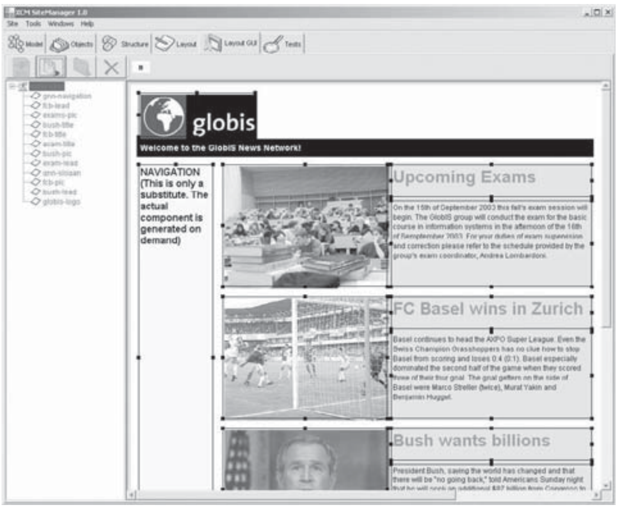
4. XCM Sitemanager

XCM Sitemanager is an editor that eases working with XCM Server [6, 9], a content management system that we have developed at the Global Information Systems (GlobIS) group of ETH Zurich. The main goal of XCM Server is to provide a research platform to test and implement the requirements of content management as presented in section 1. To implement the system, we use the Java Platform as it offers a simple way of implementing custom server components using Java Servlets. As a platform for our metadata and content databases, we have chosen OMS Pro [13], an object-oriented data management system.

As the database model for content and metadata used in XCM Server is very complex and fine-grained to allow maximum flexibility of dynamically composing content, it is can be tedious for a user of the system to work directly with the database to create and manage web sites. To simplify matters, a graphical user interface called XCM Sitemanager [21] has been developed.

Aside from making working with the content management system more user-friendly, XCM Sitemanager also serves another important purpose in that it breaks down the design process into four major steps. As these steps relate to the different user roles of persons that work with the system, the requirement of separating concerns is fulfilled.

The first task is to model the underlying application. In the example of the GlobIS news web site, a *data management* expert would decide which types of entities should be available on the web site and how they are represented. Based on this model, *content providers* would then provide the necessary data. Of course, data can also be added at a much later point in time. The *web officer* then decides how the content is to be structured. For example, it is his task to group the available information into pages and directories. To do so, XCM Server provides the powerful concept of components and containers that can be used to describe arbitrary hierarchies. The last step in designing a web site with the system is the task of the *web designer* to define how the pages are going to look and feel for all available presentation channels.



Welcome to the GlobIS News Network!

NAVIGATION (This is only a substitute. The actual component is generated on demand)

Upcoming Exams

On the 15th of September 2003 this fall's exam session will begin. The GlobIS group will conduct the exam for the basic course in information systems in the afternoon of the 16th of Sepptember 2003. For your duties of exam supervision and correction please refer to the schedule provided by the group's exam coordinator, Andrea Lombardoni.

FC Basel wins in Zurich

Basel continues to head the AXPO Super League. Even the Swiss Champion Grasshoppers has no clue how to stop Basel from scoring and loses 0:4 (0:1). Basel especially dominated the second half of the game when they scored three of their four goal. The goal getters on the side of Basel were Marco Streller (twice), Murat Yakin and Benjamin Huggel.

Bush wants billions

President Bush, saying the world has changed and that there will be "no going back," told Americans Sunday night that he will seek an additional $87 billion from Congress to

Figure 6. XCM Sitemanager 1.0

One major problem of the first version of XCM Sitemanager was its inability to visually create and edit the templates needed to generate the output. Hence, the Graphical Template Editor [2] was implemented and integrated into XCM Sitemanager. Using the process described in section 3, a web designer can now use XCM Sitemanager to create layouts graphically. An example that shows the generation of the index page within the GlobIS news web site is given in figure 6. As can be seen, the user interface is divided into two regions. On the left hand side, the structure of the content of the current web page is displayed. The designer can place content by drag-and-drop onto the preview of the web page shown on the right hand side. When placing content, the editor calculates the minimum space required to display the selected item and displays a shaded box of the appropriate size. Conflicts resulting from overlapping or lack of space are detected automatically and shown to the user by changing the colour of the placeholder to red. In addition to the standard types presented in the previous section, the Graphical Template Editor also provides support for further types that are specific to the underlying content management system XCM Server. These types include the basic concepts of component and container, as well as elements that are generated by the server at runtime, such as navigation and sitemap components. Since the type model of XCM Server is also extensible by the user, the Graphical Template Editor also provides fall-back metatemplates to cope with unknown types. It is, however, possible to also extend our editor in order to work with user-defined data types. The application further provides support to set attributes governing the appearance of the content such as text style, size and colour or image width and height. Attributes can also be set for the concepts of XCM Server. It is, for example, possible to define a background colour for a container, to visually group all of the components contained within that container. When doing so, the user can also choose to propagate settings from one level of hierarchy down to the levels below.

After having completed a layout, the work can be saved as presentation metadata in the content management system. Naturally, already defined layouts can also be loaded into the editor and changed as seen fit. When saving a layout, the editor executes the metatemplate driven generation process discussed in the previous section. To do so, it first creates the abstract representation of the layout based upon the design that has been done by the web designer. The appropriate grid is computed using a plane-sweep algorithm that determines all grid points (x, y) where content has been placed. Doing so, the editor avoids creating a too finely-grained grid as would be the case if every pixel would be transformed into a column or row, respectively. Our Graphical Template Editor supports two modes for generating a grid. The first of these two modes will generate a grid that assumes a fixed width of the layout. All columns will be exactly spaced as they were designed in the editor. The other mode works with a variable width of the layout. Hence, columns will be spaced proportionally to the maximum width. Having computed all columns and rows, the algorithm generates the layout representation in XML, while inserting references to the placed content objects at the same time. This representation is then used as an input to the metatemplates. Note that the metatemplates are not hard-coded into the application. Rather there exists a directory hierarchy where all metatemplates can be placed according to the kinds of templates they will produce. This "plug-in" way of handling metatemplates ensures utmost flexibility and extensibility. Applying the metatemplates will generate a set of templates for the components used in the web page. These templates are then stored in the content management system and metadata associations are established between the content and the generated templates. When XCM Server now receives a request for this web page on a specific presentation channel, it will retrieve the appropriate generated templates to render the page. Hence the page will be delivered to the browser as envisioned and designed by the web designer.

5. Conclusions

Providing support for multiple presentation channels in a general way is not easily done. In this paper, we have presented a number of approaches that aim at solving this problem. Although many contain interesting concepts, none of them, presents a comprehensive and complete approach.

Using an approach based on metatemplates introduces an additional level of indirection. As we have shown, this level of indirection can be employed to use the same process to generate different sets of templates for each presentation channel. The metatemplates are applied to a standard representation of the layout based on XML that unifies the similarities of the different presentation channels, while allowing for extensions to cope with the differences or features that are yet to emerge.

Further we have shown how this metatemplate driven approach to multi-channel presentation can be exploited to build an editor that allows web designers to define their layouts graphically, while still being able to implement them using different kinds of markup. The Graphical Template Editor which serves as proof-of-concept for the presented approach is part of XCM Server, an XML-based content management system that we have developed.

Although the presented approach is very powerful and flexible, a lot of work still needs to be done. Thus, we plan to enhance the editor further by allowing the designer to define generic layouts, that work independently of a concrete page and can be applied to a whole set of pages.

References

[1] A. Bongio, S. Ceri, P. Fraternali, and A. Maurino. Modeling Data Entry and Operations in WebML. *Lecture Notes in Computer Science*, 2001.

[2] P. Büchler. XSLGui - A Graphical Template Editor for Web Content Managament. Master's Thesis, Department of Computer Science, ETH Zurich, February 2003.

[3] S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): A Modeling Language For Designing Web Sites. *Computer Networks (Amsterdam, Netherlands: 1999)*, 2000.

[4] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2002.

[5] S. Ceri, P. Fraternali, and S. Paraboschi. Data-Driven, One-To-One Web Site Generation for Data-Intensive Applications. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*. Morgan Kaufmann, 1999.

[6] M. Grossniklaus. CMServer - An Object-Oriented Framework for Website Development and Content Management. Master's Thesis, Department of Computer Science, ETH Zurich, March 2001.

[7] M. Grossniklaus and M. C. Norrie. Information Concepts for Content Management. In *Proceedings of DASWIS'2002, International Workshop on Data Semantics in Web Information Systems*, December 2002.

[8] Microsoft, Inc. (msdn.microsoft.com/xml).

[9] B. Signer, M. Grossniklaus, and M. C. Norrie. Java Framework for Database-Centric Web Site Engineering. In *Proceedings of WebE'2001, 4th Workshop on Web Engineering*, May 2001.

[10] eXcelon Corporation. (www.exln.com).

[11] WebRatio. (www.webratio.com).

[12] International Business Machines (IBM). (www.ibm.com).

[13] A. P. Würgler. *OMS Development Framework: Rapid Prototyping for Object-Oriented Databases*. PhD Thesis, Department of Computer Science, ETH Zurich, 2000.

[14] XForms, Version 1.0. W3C Candidate Recommendation, November 2002. (www.w3.org/MarkUp/Forms/).

[15] Extensible Markup Language (XML), Version 1.0 (Second Edition). W3C Recommendation, October 2000. (www.w3.org/XML/).

[16] XML Schema, Version 1.0. W3C Recommendation, May 2001. (www.w3.org/XML/Schema).

[17] Altova Corporation. (www.altova.com).

[18] Q42. (www.xopus.org).

[19] XML Path Language (XPath), Version 1.0. W3C Recommendation, November 1999. (www.w3.org/TR/xpath).

[20] XSL Transformations (XSLT), Version 1.0. W3C Recommendation, November 1999. (www.w3.org/TR/xslt).

[21] S. Zweifel. Graphical Authoring Tools for Web Content Management. Master's Thesis, Department of Computer Science, ETH Zurich, February 2002.