XML Design Patterns Used in the EnterTheGrid Portal

Ad Emmen

Genias Benelux bv, James Stewartstraat 248, NL-1325 JN Almere,
The Netherlands,
emmen@genias.nl

Abstract. EnterTheGrid is the largest independent Grid portal in existence. It has been build entirely on XML-technology, from the database up to the user interface. In this paper we describe several XML design patterns, that were developed for the site. A hybrid XML data model is used as the basis to describe the items of the site. A search engine is developed using XSLT/XPath and it is described how information from different HPCN sources can be combined.

1 Introduction

EnterTheGrid [1] is a portal on the Internet that provides a single access point to anyone interested in The Grid. Currently, the three main parts of EnterTheGrid are the catalogue, the EnterTheGrid/Primeur HPCN and Grid news services - on the web and by e-mail, and XML based resources (web services) like the TOP500 in XML. The server currently attracts some 30.000 different visitors each month.

Everybody is using XML these days at least so it seems, but when one looks a little bit deeper, it is often just a report that is produced in XML out of a relational database, or some configuration file written in XML instead of property/value pairs.

Important as these applications are, they only scratch the surface of what is possible with current XML-technology. Two paradigms prohibit the view over the landscape of new possibilities:

1. A database is a relational database with an SQL interface
2. The only good programming model is an object oriented one, except when you do math then it is ok to use Fortran, or C.

A popular public domain based approach is for instance a combination of PHP [2] and MySQL [2]. Also the main commercial solutions implement this approach, with proprietary relational database systems. Also emerging e-market places [4] often follow this approach.

During the past years, relational database management systems are dominant. The difference between the relational database itself and the database management system has been blurred, and people tend to see it as one thing. Hence they associate a relational database with the possibility to handle large quantities of data in a secure and easy to use way.

Fact is that XML-documents are basically hierarchical and do not lend themselves in general to be stored efficiently in a relational database. Most of the time what people need, however, is a reliable XML-database *management system*. Once you make the paradigm shift, one sees that relational databases are much too restricted to store general XML-documents. Unfortunately, although there is a lot of development in the area, native XML-database management systems they cannot match relational database management systems yet.

The other paradigm is that the only good programming language is an object oriented one. As a result, most people do not notice that XSLT (Extended Stylesheet Language - Transformations) [5], is in fact a specialized functional programming language that can convert any set of XML-documents in a new different set. With the Xpath [5] standard for addressing any item in an XML-document, it can be used to build powerful search engines that can match SQL based ones.

Do not throw away your relational database, because there are application fields where it is superior to anything else, do not throw away you Java or C++, because there are application fields where you cannot do without, but when you start a new XML application, do not take for granted it has to be build on top of a relational database and has to be programmed in an object oriented programming language.

2 Basic Design

EnterTheGrid is a community management system [6]. There is a news section, there is a calendar, organisation profiles, etc. A typical information item, such as a news article, is somewhere in the middle between a data and document oriented type. It does not have the regular structure of a data oriented type, because, for instance, the body part of the article can exhibit a complex structure with paragraphs, tables and lists. On the other hand, they do not have the rich complexity of a real document oriented type like a manual or full report.

Hence we did design a document type definition that has a limited number of elements in a fixed structure. For parts, like the body of the article or a description of a company, we allow the use of XHTML type of structure constructs. This proves to be very powerful, as it allows to develop generic tools around this definition.

Each article, organisation profile, etc, is stored as a separate entity (an XML external parsed entity). An index (an XML document) is created that indeed includes all relevant items as references to these XML entities, into an XML document, creating one logical view of all items, the same as if they were stored in one physical XML-document. This base index can then be used to create sub indices to speed up processing. This separation between physical and logical structure of an XML document,

inherited from SGML, is seldom used, but provides in fact a powerful storage mechanism.

Because each article, organisation profile, etc, is a physical different entity, they can also be updated easily, independent from each other. For this purpose we did choose to create a web based editor that only needs an HTML browser to operate. There is no Java or other programming executable needed on the client side. There is some Javascript to ease editing, but one can do without. There are other approaches possible. For instance, creating an editor as a Java applet, using a proprietary language from one of the browser vendors or using Javascript. One even create a stand-alone editor that has to be downloaded. However, all of these approaches have the disadvantage that they do not work all off the time. Because we want authors having access from any place and any type of computer, they could even be using it only once, the HTML-only client approach is the best.

The advantage of this approach is that everything can be done on the server. The disadvantage is that everything has to be done on the server. In our case, the bulk of the editor is written in XSLT, with a little Java to get it started. It then runs in a standard Servlet engine. The advantage of doing all processing on the server is the control on what is happening. There are never surprises because of the client software not working as expected.

On the down side the problem is that the user who edits a contribution has to wait for the server to respond, which in practice is not a big problem. The main problem that arises is that choices and sub-choice dependencies (for instance subsections that differ for each section), can only be updated after contact with the server. Another disadvantage of this approach is, although the user is guided as much as possible, it still is possible in some fields to enter text that is not well-formed XML. This then has to be reported back by the server.

The editor itself is fully configurable, because of the use of XML. There are several XML configuration files that steer the process. Each time the user gets a new editor page, or editor page update, it is automatically created from this configuration files. There are two important ones.

The first configuration file describes which part of a document may be edited by a user, and how they should be called for him. So for instance when a news article in a specific publication has a title, called "Kop" and a lead part, called "Tekst", but no large body part or reference section, then the user gets only two fields with these names. He does not see how they are actually coded in XML, which is `vmp:title` and `"vmp:lead"` respectively. In the field "Tekst" a complex content structure can be entered. When some time later the user wants to edit for instance an organisation profile, he gets the fields an labels belonging to an organisation profile.

The community manager maintainer can change he configuration file, and hence what a user is allowed to change or enter in an item. Different groups of users, for instance, editors and authors, can have different editors. The layout of the editor, i.e. where the fields and labels, size, colours, and images, are determined by an XSL-FO [5] file. XSL-FO, the formatting part of the Extended Stylesheet Language is an excellent format to describe the layout of 2-D objects, be it for paper or for screens. Most people use it as an intermediate language to specify printed material in for in-

stance PDF format. The idea is that we have to "wait" for browsers that are XSL-FO aware before using them for display on a screen. We, however, use it as a specification language for defining the automatically generated editor's layout. Hence, the page that is send to the user consists of XHTML and CSS. It is automatically composed from the item to be edited (if it exists) in XML, the configuration file for what has to be displayed and the XSL-FO definition of the lay-out. In fact there are a number of other general applicable XML files that may be used. For instance for formatting dates, or translations to other languages.

The publishing system contains further a publishing module, a search module and an administration module. The search module is the most novel and interesting one to describe.

3 Implementing an XSLT/XPath Search Engine

Because all the content is stored as XML, we cannot use SQL as a search language. We could use a standard text based search-engine approach that is used on a lot of web sites. Although they can give very good results in general cases, they cannot use the structure that is inherent in the XML-content. To express this we need an XML aware search approach. Although there is work done on specifying an Xquery language[5], the XSLT/Xpath standards, already provide enough searching and location possibilities for a good implementation. Hence, our search module is build using XSLT/Xpath technology. Take for instance the XML fragment:

```
<vmp:article>
<vmp:title>Jansen en Jansen</vmp:title>
<vmp:author>Piet</vmp:author>
</vmp:article>

<vmp:article>
<vmp:title>Pietersen en Pietersen</vmp:title>
<vmp:author>Jan</vmp:author>
</vmp:article>
```

Then we can display the article title of the article that has "Jan" as author with:

```
<xsl:template match="vmp:article">
<xsl:if select="contains(vmp:author, 'Jan')">
<xsl:value-of select="vmp:title" />
</xsl:if>
</xsl:template>
```

To generalize this, we have to use a variable containing the search string and pass this to the template. If we want to make it case insensitive we can use the XSLT `translate()` function to translate both the element content and the search string to the

same case. To search for more texts in different parts of the article, we can use for instance:

```
<xsl:if select="contains(vmp:author, 'Jan')
           and contains(vmp:title, 'Piet')">
```

When we use variables as search strings, for instance called for instance `$author` and `$title`, this would be:

```
<xsl:if select="contains(vmp:author, $author)
           and contains(vmp:title, $title)">
```

When a search string is empty, the result of the `contains()` function is “true” which is what we need for the complete if-statement to work correctly. When a user inadvertently enters a space in the search-field, or adds a space or return character at the end, the `contains()` function will return “false”. To get the desired result, one can strip these characters with the `normalize-space()` function available in XSLT.

Because the general XML structure of each item is the same, extending this to a full search engine is not too difficult. For the displaying the search query to the user, we, again use the XML configuration file that maps each element name to the label too be used.

4 Combining Information from Different Sources - The TOP500 and “Overview of Recent Supercomputers”

Currently, most of the World Wide Web consists of HTML web pages. In these pages, the original content description is lost. No longer is an address an address, but it is marked up as a paragraph or paragraph, perhaps with some bold typeface added. One of the promises of the web services hype is that these new services will solve the problem for you. In fact, that is an easy promise: once you put your information in an XML-document and make that accessible through on a URI, you can access the information in it using XSLT/XPath and combine it with other sources into a new XML-document, as we will show next.

There are many sources of information on supercomputers available. One of the most important ones is the TOP500 list of most powerful supercomputers in the world. It contains data on the position in the TOP500, the performance and where the machine is located. Another source is the *Overview of recent supercomputers* [7] that contains a description of the supercomputer systems available, with system parameters, and benchmark information. With XML-technology we have combined information from the two sources to generate a new document, containing a description of each architecture, extended with a sub list of the TOP500 on that machine. The new document is generated with XSL-Transformation-stylesheets, that leave the original sources untouched. When there is a new version of for instance the TOP500, running the stylesheet is enough to generate a new, updated version of the assembled information document.

We produced an XML-version of the TOP500 [8] and showed that, by using stylesheets, this can be used to select and combine information from several lists easily. Although this is already very useful, combining information with information from other sources can make the TOP500 even more valuable. The before mentioned report *Overview of Recent Supercomputers* by Aad van der Steen e.a is such an information source. The report describes all the supercomputer systems currently available in detail. For each system it lists, for instance, the operating systems, compilers, and parameters like clock cycle, peak performance, memory size and interconnect speed. Also it contains information on benchmarks.

The previous editions of the overview report were available on paper as the primary source and html version on the web derived from the LaTeX input source.

Looking up and comparing information in the Overview report was simply a way of scanning through the document with the traditional table-of-contents as guidance. Combing information from the report with that from the TOP500 was a tedious process. Let us take as an example that you want to have a list of all the TOP500 sites that belong to a certain system architecture from the Overview report. You go to the TOP500 web site. Create a sublist for system A, print it out and stick it into the Overview report. Than do the same for system B, C and so one. I guess only a handful of people went through this process.

With XML technology, that becomes a simple task. But before we describe how to do it, let us recall what XML is. With XML you can not only write information in a document, but also describe what the information is about. When you are writing a book, you can tell what a chapter is, what a section, etc. When you write about the TOP500 list, you can tell which item is the TOP500 position and which is the address of the site that has the supercomputer. Each information element is contained by XML-tags that look like HTML tags. The difference is that you can define your own tags to exactly describe the information in your document.

When you have XML documents, you can use XSLT (Extensible Stylesheet Language Transformations) to extract information from one or more documents and combine that into a new one. You can specify in detail what information you need. The new document can be another XML document, an HTML document, or a text. When you apply a formatter to it, you can generate RTF (word), PDF, LaTeX, a database record or whatever format you need.

XSLT is a powerful tool. You can for instance extract information like: "Give me the country name of the last system in the TOP500 list, that according to the Overview report could have an HPF compiler on it." I do not know what you can do with this information, but you can ask it.

Although this looks a bit like a data base query, it is important to realise that we are locating information in documents, not in a relation database.

With the TOP500 already available in XML, the first task was to make the Overview report available in XML. The source text of the report is written in LaTeX. Although LaTeX gives a good control over the page-lay-out, it does not provide us with much clues as what type of information is described. Most of the important data on machines is hidden in tables. Hence we decide to first turn that information into

XML designing a new Document Type Definition suited for description of supercomputing systems as applied in the report.

This is an excerpt showing how a description of a supercomputer model looks like.

```
<ors:model>
<ors:name>VPP5000</ors:name>
<ors:clock-cycle unit="nsec">3.3
</ors:clock-cycle>
<ors:processor-performance
unit="gflops">9.6</ors:processor-performance>
<ors:peak-performance unit="tflops">1.22</ors:peak-
performance>
<ors:memory-node unit="gbyte">16</ors:memory-node>
<ors:memory-maximal unit="tbyte">2</ors:memory-maximal>
<ors:number-of-processors>
  <ors:min>4</ors:min>
  <ors:max>128</ors:max>
</ors:number-of-processors>
<ors:communication-bandwidth>
  <ors:point-to-point unit="gbitpsec">1.6</ors:point-
to-point>
  <ors:processor-memory
unit="gbitpsec">38.4</ors:processor-memory>
</ors:communication-bandwidth>
</ors:model>
```

Note that from the names of the tags, one can easily deduce what the information means. We defined an attribute "unit" to indicate what a number is representing.

When you define your own tags, you can choose to make them as short as possible, or as descriptive as possible. Although in the latter case documents tend to become more lengthy, I choose nevertheless for that one: it makes the documents much more self explanatory.

Next we create an XSLT stylesheet. It takes the XML document of the systems from the Overview report, and for each system matches it with manufacturer information in the TOP500 list. The lines that do this are:

```
<xsl:template match="ors:system">
...
<xsl:for-each
  select="document('top500-199911.xml')
    /top500:list/top500:site/top500:manufacturer">
...

```

The output document, also XML, has a traditional report type of structure, with chapters, sections, and tables. For this we have another stylesheet that we apply to reports and that can turn it for instance into an HTML document. We have also made this available over the Web [9].

5 Conclusion

We have showed that with standard XML and XSLT, it is possible to build an EnterTheGrid portal with the same functionality as with a traditional relational database system approach. However, in addition, it is possible to exploit the XML structure of the stored information in an XML-aware search engine that delivers better results.

We have combined the information from several HPC-resources translated into XML into a “web service” where a report is enriched with information from the TOP500. This would have been very difficult without an XML-based approach.

The methods described form basic XML design patterns that can also be deployed in other applications.

6 References

1. <http://enterthegrid.com> and <http://www.hoise.com/primeur>, last visited, January 2002.
2. PHP website, <http://www.php.net>, last accessed, January 2002.
3. MySQL website, <http://www.mysql.org>, last accessed, January 2002.
4. Kaplan, J.O, Sawhney, M, E-hubs: The new B2B Market places. Harvard Business review, May-June 2000, 97-103, 2000.
5. The World Wide Web Consortium style initiative (XSL, XSLT, Xpath), <http://www.w3c.org/style>, last visited January 2002.
6. Netgain, expanding markets through virtual communities, John Hagel II, Arthur G. Armstrong, Harvard Business School Press, ISBN 0-87584-759-5, 1997.
7. Van der Steen, A.J.: Overview of recent supercomputers, tenth revised edition, Stichting NCF, ISBN 90-72910-03-6 also available: <http://www.phys.uu.nl/~steen/web00/overview00.html>,
8. Meuer, H.W., Strohmaier, E., Dongarra, J.J, and Simon, H.D. TOP500 Supercomputer sites, University of Mannheim, 2001, XML version available on <http://www.hoise.com/primeur/analysis/top500>, last accessed, November 2001,
9. Van der Steen, A.J.: Overview of recent supercomputers, tenth revised edition, machine descriptions in XML, enriched with TOP500 data, <http://www.hoise.com/vmp/examples/ors/index.html>, last accessed, November 2001.