A Software Development Metaphor for Developing Semi-dynamic Web Sites through Declarative Specifications

Diomidis Spinellis, Vassilios Karakoidas and Damianos Chatziantoniou
Department Management Science and Technology
Athens University of Economics and Business
Greece
email: {dds, bkarak, damianos}@aueb.gr

Abstract

Traditionally, the realization of Web sites involves either static content developed using web authoring tools or dynamic content delivered by a database driven front-end, where the structured content is organized in a relational schema and dynamically generated on the fly. The limitations of statically-authored web pages are easy to discern and for a number of applications, the use of a database introduces a level of additional complexity that makes the choice a part of the problem space rather than the solution space. Based on the distributed software development approach, we present a suitable for managing middle-sized semi-dynamic web sites. The technological dimensions of this approach are well-known open source technologies, such as version control systems and XML transformation tools.

Keywords

Web site implementation, semi-dynamic web sites, XML, make, BIBTEX, XSLT, X-Schema (XSD), HTML.

1 Introduction

Traditionally, the realization of Web sites involves either static content developed using web authoring tools like Microsoft's Front Page and DreamWeaver, or dynamic content delivered by a data-oriented front-end, where the structured content is stored in a relational database and dynamically generated on the fly. When our group faced a series of problems with both the above approaches, we decided to explore ideas for a radically different implementation style, based on the declarative specification of all the site's elements.

The limitations of statically-authored web pages are easy to discern. The content is authored in an unorganized manner, and, as a result, can be inconsistent in both

structure and presentation. While the use of cascading style sheets can help one obtain a consistent look, their use still requires discipline. Moreover, the authored pages remain unstructured and the resulting site can be difficult to modify and reorganize. Furthermore, the static authoring model often imposes a centralized management and maintenance style; all additions and changes have to go through a single person, creating a bottleneck, often leading to outdated content.

Adopting a database driven approach is supposed to solve the aforementioned problems. Separating the source data from its (dynamically generated) marked-up version (HTML code) leads to a consistent yet flexible generation of web pages. In addition, the database's relational model imposes its structure on the data being stored. Finally, a database back-end allows concurrent updates by different users.

However, for a number of applications, the use of a database introduces a level of additional complexity that makes the choice a part of the problem space rather than the solution space. A database-driven web site requires the implementation of a front-side interface to transform the web site's content into HTML code, and a back-end interface to allow stakeholders enter, review, and update data. The back-end client interface typically requires setting up and maintaining appropriate access permissions. These may need to be integrated into an organization wide authentication facility, or operated under a specific security policy. In the second case, procedures for setting up passwords, resetting them, and revoking them need to be established and followed. A properly running database also requires a skilled database administrator to install it, maintain it, organize backups, and perform modifications to the database schema. In addition, marked-up content is generated by a front-end program accessing the database, therefore the front-end and the database must be extremely secure and robust [1], running on a 24×7 basis. The front-end, being an executable program working on untrusted data (the web page requests) can become the target of malicious attacks, and must therefore be inspected and audited to ensure its robustness [2].

To minimize the risk of an attack against the database (that would jeopardize the organization's data) the database server has to be installed on a machine separate from the web server, behind a properly configured firewall. Finally, the extraction of content from a database often induces the web site's designers and stakeholders to adopt a query-style interface. Such an interface is typically less usable than browsable web pages, and the served content is often ignored by search engines, leading to reduced visibility of the (meticulously structured) content [3, 4].

All in all, a database-driven approach appears to be suitable only for those with ample resources to justify the full development and appropriate maintenance of a sophisticated infrastructure.

Having faced the problems we described above within our organization, we reasoned that a fresh approach was needed to tackle them. Specifically, in an internal effort to develop and maintain our group's web site we had already abandoned the ad-hoc authoring tool–based approach because it led to an inconsistent look and stale content, while the maintenance of a subsequent database-driven design approach was proving intractable for the resources that our group could afford. In the following sections we describe the methodology we developed for implementing semi-dynamic web sites, illustrate it by means of a case study covering the requirements, design, and implementation, and discuss the lessons we learned.

2 Related Work

Here goes the related work.

3 Methodology

Our proposed methodology for developing semi-dynamic web sites uses as its metaphor the distributed software development activity, as popularized by many high-visibility distributed open-source development projects, such as Eclipse [5], Mozilla, and FreeBSD. Using a metaphor as a guiding principle for a development activity is a practice we adopted from extreme programming [6]. The methodology can be summarized in a few key points.

- Structure the site's data as XML files, or another declarative formalism. Content developers create and edit these to modify the site's contents.
- Use XML schemas and corresponding tools to validate the content.
- Exploit modular XHTML [**?**] to create schemas for rich data elements. These can be used to allow the inclusion of specific XHTML tags in the site's XML files, without having to reinvent HTML from scratch.
- Adopt a version control system for distributing the content, templates, transformation, and schemas across content developers and administrators, synchronizing updates, and keeping a record of the project's history. (Thomas and Hunt [7] aptly compare a version control system with a global undo command with unlimited undo levels.)
- Create the site's look and feel in HTML format through XSLT transformations [8].
- Express verification and building dependencies among data elements and schemas through *make* [9] or *ant* [**?**] rules.
- Make the local generation of the site's content the default building option to allow developers and administrators to verify the results of their work, before putting them online.
- Use existing mechanisms and tools, such as the secure session shell, public key authentication, and group membership permissions to implement authentication and authorization policies.
- Have an instance of the project on a centralized server, kept up to date through the version control system, as the source to generate and export the definitive version web site's contents.

One can easily discern from the above points the similarities of our approach with distributed software development. A version control system is used to distribute the artefacts among developers in their current form, and a building tool, such as *make* or

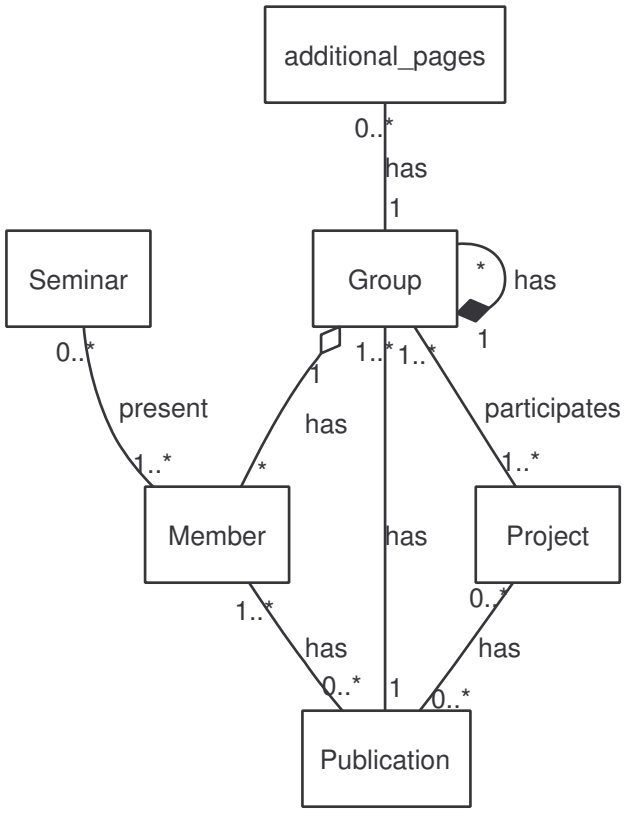

Figure 1: Overview of data element relationships.

ant guides the building process. Developers edit and build their work locally, and commit it to a central server when ready. The repository on the central server is the source for creating the end-result result of the product, through a *daily build* process. In our case however, the developers work with XML and XSLT files, instead of programming language source code files.

To illustrate our methodology in concrete terms the following sections contain as a case-study the design and implementation of our research group's web site. Through the case study we will demonstrate the key points applied in practice, discuss important technical and human issues we faced, and present the lessons we learned.

4 Requirements

The functional requirements for our center's web site were simple, but not trivial, and had already been satisfied twice in two different forms. The site should present our

research center's members, groups, publications, and projects in an organized fashion. Our center is divided into five groups. It numbers about 50 members, is a participant of 35 past and current research projects, and the originator of about 400 publications. The site's pages should represent the content and the relationships we illustrate in Figure 1. The self-referential relationship on the group entity exists because the center should an "umbrella" group of multiple subgroups. Members of our center and our research projects are associated with one or more groups. Publications are associated with one or more members, groups and projects. For the sake of simplicity, we have omitted from our description and the diagram a number of additional relationships, such as the member directing a group or managing a project. As an example of the type of content we were looking for, each member, group, or project should have a web page with a list of the corresponding publications; each group should have pages listing its projects and members. In order to have the ability add web site data that is not part of the aforementioned categories, each group can have additional pages of unstructured XHTML content, in a dictated manner. Finally, each member of our groups may be performing a presentation in our group's weekly seminar.

As we hinted in the previous section, the problem with the previous implementations was not the creation of the site satisfying the functional specifications, but the lack of a number of important non-functional requirements. Before embarking on our third attempt, we articulated those non-functional requirements we thought important, to ensure that our third attempt would produce a result with a longer life span. The following is a list of the non-functional properties we deemed important enough to guide our design.

Openness The tools used in the realization of the web site should be available as open source, or supported by multiple vendors. We wanted to avoid becoming tied with a particular proprietary tool. We reasoned that openness would mitigate two risks: (1) finding a maintainer trained to use a particular proprietary tool, and (2) obtaining resources for upgrading and maintaining the tool.

Observability The semantic distance between the specification of an element and its implementation should be minimal [10]. The site's look and content should be maintainable using standard tools and techniques. If possible, the site's maintainer should not be required to learn a scripting language like PHP, or PERL, or a framework like J2EE or .NET. An approach based on declarative specifications [11] and domain-specific languages [12, 13] would allow end-users, or members close to end-users to be involved in maintaining the site, without risking the bottleneck of going through multiple intermediaries.

Robustness The web site should not depend on any external programs other than the web server for serving its content. Users updating the data, should be able to author, validate and review their changes without requiring network connectivity. This would allow them to work productively over dial-up connections or while on the road. In addition, all the editing users should be able to work on a platform of their choice (UNIX, MICROSOFT WINDOWS and MACINTOSH) using only a text editor of their choice. Minimizing the dependencies on additional servers

(such as a database or an application server) and on the network should result in a more robust and easier to maintain system.

Parsimony The implementation effort for the system should be minimal. This would minimize errors and maintenance costs. We reasoned we could satisfy this requirement by using existing tools, if their choice satisfied the other non-functional properties.

5 Design Process

5.1 Conceptual Framework

The system's conceptual framework is illustrated in Figure 2. The main concept in our system is the *Organization*. Each *Organization* has information that interests *Content Consumers*. They access the organization's *Public Data*. To obtain information regarding the *Organization*. It has also *Content Developers* who are usually its members. *Content Developers* update the *Public Data* for the *Organization*. *Content Administrators* are supporting the *Content Developers* (mostly troubleshooting and account management) and inspecting the submitted *Public Data*.

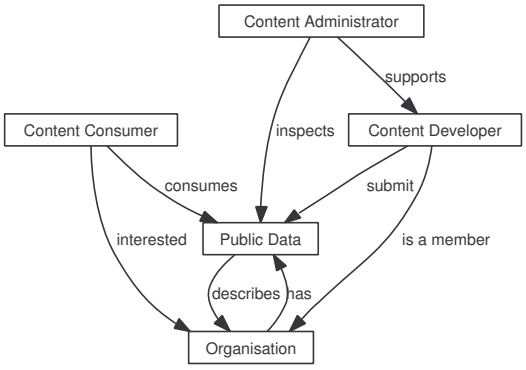

Figure 2: Conceptual design of our system

5.2 Motivation and Design Paradigm

Our main guiding principle was to create a continuous, multi-person development activity. Live web sites continuously evolve; adopting the content authoring paradigm implied by our first approach was a mistake. Empirical evidence supports this observation. Figure 3 illustrates the changes in usability rankings given to 21 Greek government department web sites between the years 2002 and 2003 [14]. The X shape in the rank changes between the one year and the next is, we believe, the result of statically authored web pages degrading to a point of irrelevance, and then being overhauled

from scratch. The change of ranking can be explained as decay of the web site when dropping, and re-engineering when rising.

A database-driven approach also hinders evolution. Changes to the content's presentation require the modification and installation of the front end page generator; not a typical lightweight operation. Changes to the data schema are even more intrusive requiring a synchronized modification of the data, the front end, and the back end.

Continuous multi-person based projects are quite common in software development. Numerous programmers and engineers contribute and coordinate their work through a version control system, like CVS [15] that maintains a master repository of the source code. Concepts like the *daily build* [16] or the *current* and *stable* branches, as practiced by numerous open source projects, allow the maintenance of a known-good product. What we needed for our approach were appropriate, declarative language-based formalisms for expressing our data, its transformation into web pages, and an efficient generation process.

5.3 Processes

Figure 4 shows a UML [17] use case diagram of our system. We have three actors (roles) interacting with the system:

ContentDeveloper A Content Developer is responsible for uploading data in the system. In our case, is typically a person for each research group. Content Developers can upload data files and import new bibliography entries.

ContentAdministrator The Content Administrator is responsible for the maintenance and the further extension of the system. A Content Administrator corrects problems with the data, develops new features or corrects existing ones in the transformation scripts.

ContentConsumer The content consumers are typically outside of our system and have access only in the generated web site.

The use cases that comprise our system are:

Maintain schemas, transformations and users This use case describes the maintenance and development procedures of the system. Each Content Developer can update the data schemas the transformation scripts in order to correct errors and add new features.

Update local repository Content Developers have a local copy of the system's files on their workstation. and must update the local repository each time they use the system, in order to synchronize their local copy of the system with the master copy. To do so, they must execute an update repository command. If there is a conflict, the users must correct the problem and rerun the update command.

Commit changes Content developers and Content Administrators must commit the changes from their local repository to the CVS repository in the web site server.

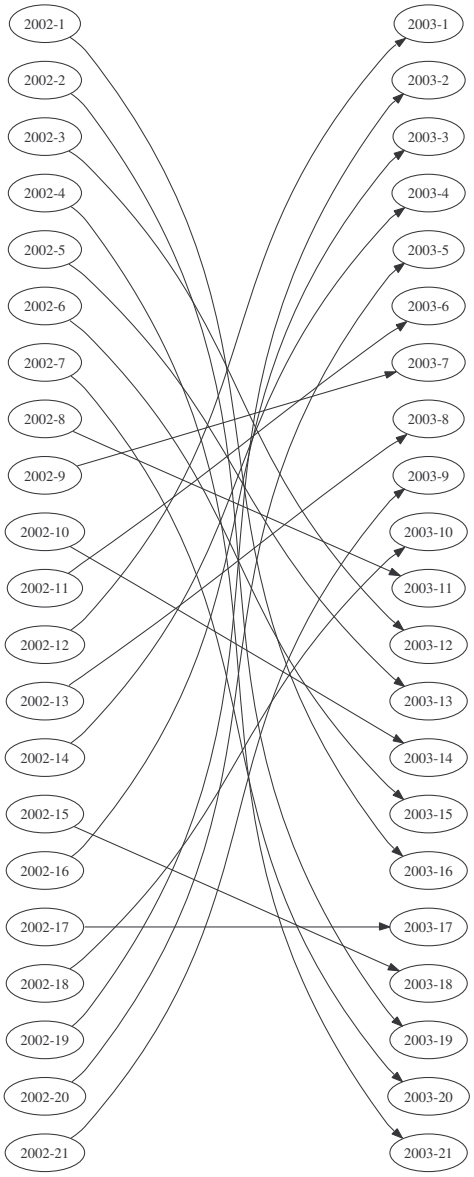

Figure 3: Yearly changes in web site rankings

Maintain content This use case allows Content Developers to maintain the content of the web site. Content consists of group, member, project and seminar data or bibliography collection entries.

Generate web site Content administrators and developers can generate the web site

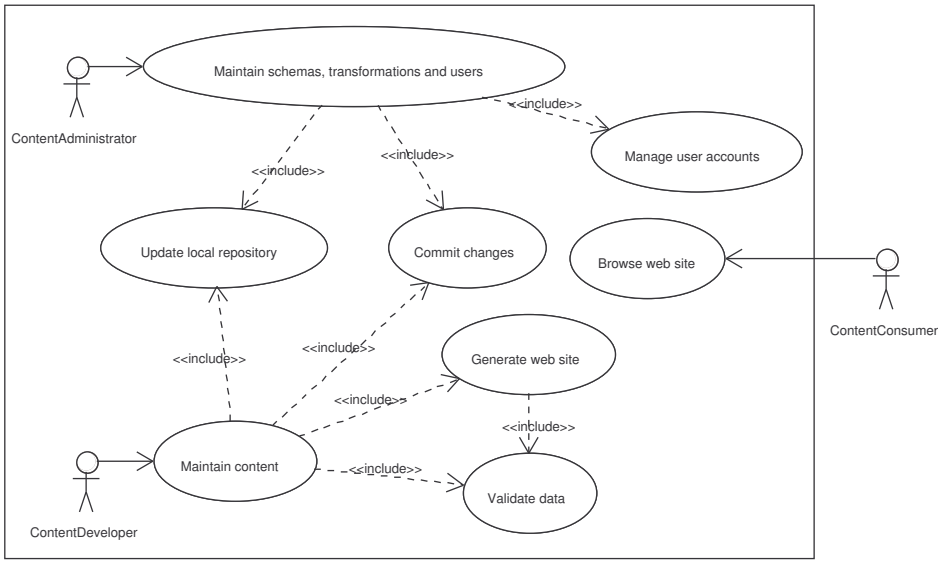

Figure 4: UML Use Case Diagram of the System

locally for preview. Once the content is generated, the user can review the newly integrated content and inspect it for possible presentation problems.

Validate data Content developers can validate the data in the local repository before the final commit. For the validation process the system use appropriate data schemas.

Manage user accounts Content Administrators perform user management in the system. User management consists of two main processes: (1) Authentication , (2) Ownership & Policy.

Browse web site This one describes the web site browsing process.

In Figure 5 we show the activity diagram of the most complex use case in the system. The Content Developer first updates the local repository and then begins data maintenance by adding, removing or modifying data files and bibliography entries. Upon completion, data validation must be performed before the commit to the data repository. If the data repository is valid, then local web site generation must be performed to validate the presentation. After that, the data is ready to be submitted to the main data repository. After the update of the data repository the process terminates.

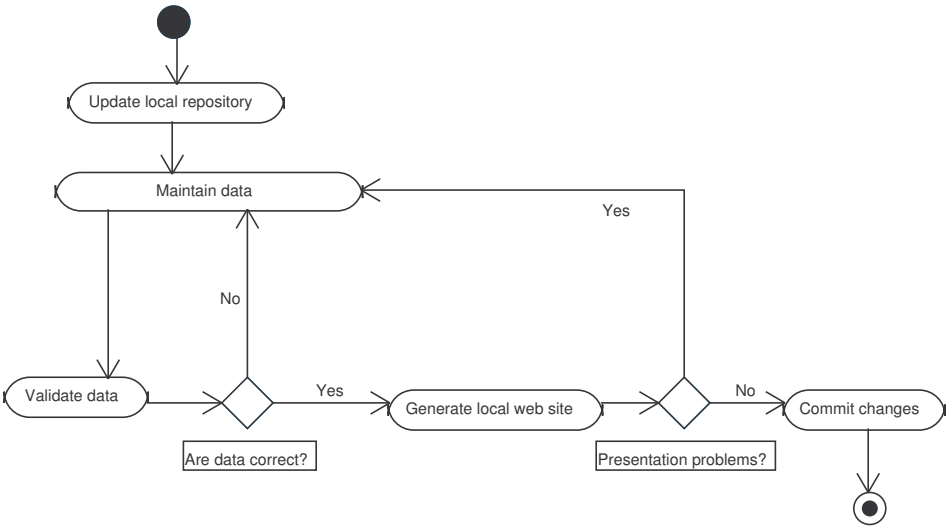

Figure 5: UML activity diagram for Maintain Content

6 Implementation

6.1 Key Technologies

Once the design was finalized, implementation proved to be an almost hollow activity, since it did not involve almost anything of what is typically described as coding.

The first step involved selecting and setting up the appropriate tools. In order to meet the requirements we set in section 4, we decided to use popular open technologies as key elements of our system. We adopted the concurrent versions system (CVS) [15, 18] to coordinate the distribution and update of all the system's components. Authentication for managing content was handled by the Unix group membership mechanism of the host where the CVS repository was installed. We also used BIBTEX [19, 20] and BIB2XHTML [21] for transforming the publications into HTML and xmlstarlet [22] for validating and transforming all other XML-based data [23]. For data transformations we implemented a system based on XSLT [24], a language for transforming XML documents. Finally, GNU make [25] and a couple of shell script constructs were used for handling the project's makefile. The complete setup including all tools proved to be portable between Unix and Microsoft Windows, with team members working on machines running different versions of MICROSOFT WINDOWS, GNU/LINUX, and Free-BSD.

6.2 System Development

The next step was a series of iterations where we modeled the data's schema on representative XML files. Concurrently, we implemented the validation DTD / XSDs (Docu-

ment Type Definition / X Schemas) and the transformation XSLTs. First we developed the DTDs for the data validation, later on we decided to use XSD schemas in order to perform further data validation to our system. The version control system was already proving its value at this point for coordinating the work between the two of the paper's authors. Because many page elements, like a project's description, could contain content more elaborate than plain text, we used W3C's modular XHTML specification for importing existing elements in our DTD / XSDs. This helped us keep our schema description simple, but the corresponding schema expressive.

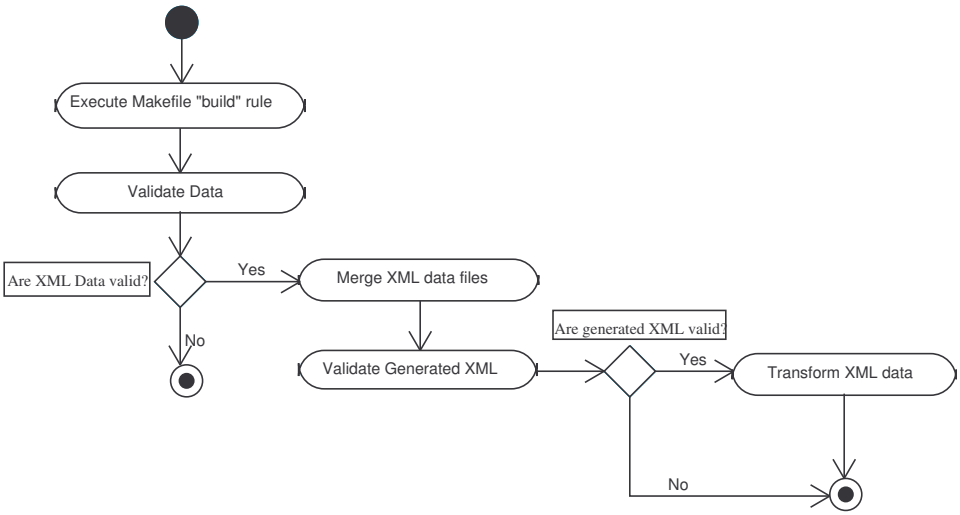

Figure 6: UML Activity diagram for generating the web site

In Figure 6 we illustrate the web site generation process. The automated validation and generation of content was expressed as makefile rules [26]. The individual XML files are merged in a single XML file for cross validating identifier reference attributes (IDREFs). The same file is also used to extract the identifiers of all projects, members, and groups into makefile variables. A simple loop then generates the XHTML files corresponding to each of the above elements. Each project, member, group etc. refers to the relevant XML IDs. Upon generation, links are created to connect relevant generated pages e.g. each member hyperlinked with the corresponding publications, which are two different independently generated XHTML pages.

The XHTML content is, by default, generated on the local machine, where its maintainer can verify it. After the new content is validated and verified, the maintainer can commit the change to the central CVS repository. A separate makefile rule can then be used, to execute an update command on the host serving the content to the web. The command retrieves the updated data from the CVS repository and regenerates the pages on the web-server's file area. All components of our system are under version control, all pages are automatically tagged with identifiers denoting their source, helping

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="seminar">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="sem_date">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:pattern value="[0-9]{8}" />
      </xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="sem_time" type="xs:string" />
    <xs:element name="sem_title" type="xs:string" />
    <xs:element name="sem_room" type="xs:string" />
    <xs:element name="sem_url" type="xs:string" minOccurs="0" />
    <xs:element name="sem_duration" type="xs:string" />
   </xs:sequence>
   <xs:attribute name="by" type="xs:string" />
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 7: Seminars X-Schema file

the traceability of changes. All exchanges between the developers' machines and the CVS and web host are performed using the secure shell (SSH) as the transport protocol guaranteeing the data's integrity and confidentiality.

You can see representative samples of a seminar's XSD schema description in Figure 7, XML data in Figure 8, XSLT transformation in Figure 9, and XHTML result in Figure 10.

The directory structure of a typical local repository is illustrated in Figure 11. Most of our users are working MICROSOFT WINDOWS environments and as a result we decided to upload a MICROSOFT WINDOWS version of the needed tools in the main repository under the directory *bin*. The directory *data* contains the XML and BIBTEX files and *schema* includes all the available DTD / XSD and the modular XHTML specifications. The *public_html* and *build* directory are used for the creation of the web site locally, while *doc* contains the available documentation for the Content Developers and Content Administrators. Finally, the directory *tools* has a small selection of utilities (PERL and shell scripts) that provide statistical information for our web site.

The installation procedure is very simple, and the bootstrap tools it requires are only CVS for the initial check out and an SSH client. The needed keys for the secure shell session are provided once for each user by the Content Administrator. A full version of the system demands 8 megabytes of space on the local machine, plus some extra for the local content store and generation.

```
<?xml version="1.0"?>
<seminar by="m_bkarak">
 <sem_date>20040314</sem_date>
 <sem_time>19:00</sem_time>
 <sem_title>Regular Expressions</sem_title>
 <sem_room>906</sem_room>
 <sem_url>http://www.eltrun.gr/seminar/presentations.ppt</sem_url>
 <sem_duration>3 hrs</sem_duration>
</seminar>
```

Figure 8: A typical seminar XML file

```
<xsl:template match="seminar" mode="full">
 <h3><xsl:value-of select="sem_title" /></h3>
  <xsl:element name="a">
  <xsl:attribute name="name">
   <xsl:value-of select="sem_date" />
  </xsl:attribute>
</xsl:element>
 Presenter: <xsl:apply-templates
             select="/eltrun/member_list/member [@id=current()/@by]"
             mode="simple-ref" /><br />
 Date: <xsl:call-template name="date">
        <xsl:with-param name="date" select="sem_date" />
       </xsl:call-template> <br />
 Time: <xsl:value-of select="sem_time" /><br />
 Duration: <xsl:value-of select="sem_duration" /><br />
 Location: <xsl:value-of select="sem_room" /><br /><br />
 <xsl:element name="a">
  <xsl:attribute name="href">
   <xsl:value-of select="sem_url"/>
  </xsl:attribute>
  Download the presentation
 </xsl:element>
</xsl:template>
```

Figure 9: The XSLT transformation file for seminars

```
<tbody valign="top">
<tr>
<td align="left" width="82%">
<h2>Eltrun Seminars</h2>
<a href="index.html#20040314">14 March 2004 - Regular Expressions</a>
<br><br><br>
<h3>Regular Expressions</h3>
<a name="20040314"></a>
Presenter:
<a href="../members/m_bkarak.html">
Mr. Vassilios Karakoidas
</a><br />
Date: 14 March 2004<br />
Time: 19:00<br />
Duration: 3 hrs<br />
Location: 906<br><br />
<a href="http://www.eltrun.gr/seminar/presentations.ppt">
Download the presentation
</a>
```

Figure 10: A generated HTML seminar file

In Figure 12 we illustrate a web graph [27] that shows references between the XHTML pages. The above Figure shows the references of a member ("m_dds" is the ID for Diomidis Spinellis), with other pages in the web site. The pages starting with "p_" are projects, with "m_" members, and "g_" groups. At the time of writing the site generated 209 XHTML pages with 1368 hyperlinks.

7 System Adoption

Our research center is multidisciplinary: under its roof are both specialized software engineers using the same tools we adopted in their everyday work, and researchers whose background is management science, marketing, or finance who are comfortable with GUI interfaces. Upon completion of the development, we were somewhat concerned by the way our group would receive the new way of work we proposed, in order to maintain the web content.

Our fears were justified. The first presentation of the system to its users ended almost in a revolt. Non-technical users expressed their inability to comprehend what an XML document was, while technical members helpfully argued for providing a GUI front end. By targeting the users with the least technical experience, promoting our system's "soft" attributes, such as the use of open source software tools, and convincing them to try to enter a few elements into the system, we were able to overcome the initial reservations and start the data migration process.

The next round of problems surfaced when users began entering malformed or

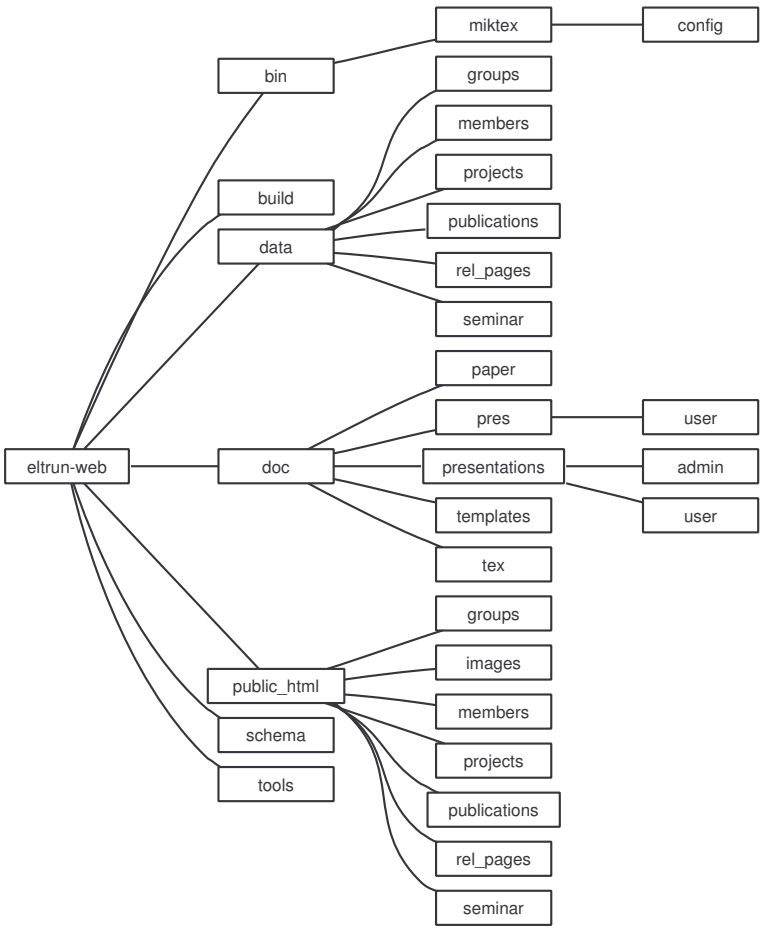

Figure 11: Directory structure of local repository

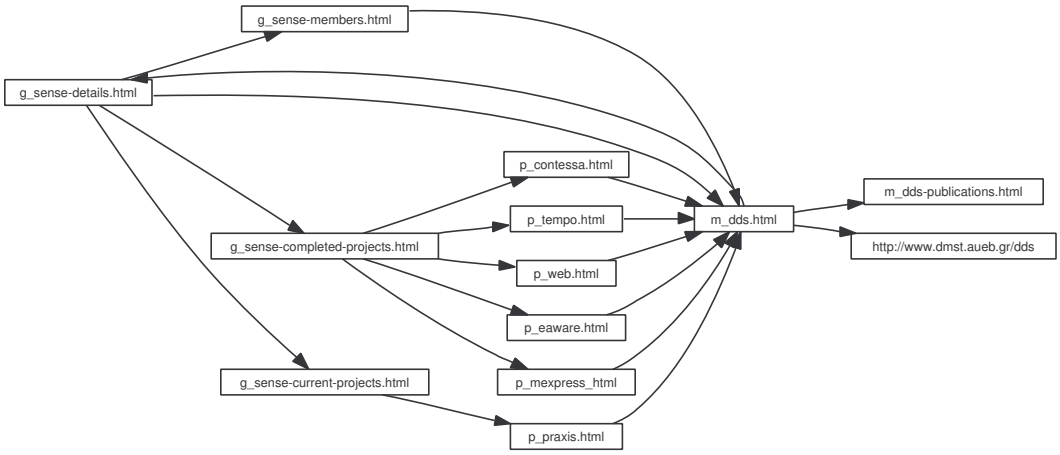

Figure 12: Web graph of the generated site

invalid data into the system. This resulted in all users acquiring the copies of the malformed XML files, and strange error messages given to unsuspecting users. As is the custom in a number of development efforts, we had expected the users to verify the changes they made before committing them into the CVS repository. Non-technical users were however not aware of this etiquette and were committing their changes with the hope they were correct. A shared mailing list had established to explain the importance of following the correct procedures when committing changes. We also instituted a "pointy hat" policy. Committing a malformed file would award its committer a (virtual) "pointy hat", which would then be passed on to the next committer to err. After a few days we got the impression that non-technical users were becoming confident in their work, even proud of sharing sophisticated tools and processes with software engineers. Our technical persons experienced more problems than the inexperienced ones and that came as a surprise for us. Some of them were already familiar with the key technologies, and they tried to change the tools proposed with others more user-friendly. These initiatives resulted in corruption of the local repositories, malformed XML data and wrong bibliography entries. After a few false attempts they decided to stay with the procedures of the system.

Two weeks after the initial system presentation all users where able to upload and maintain their data. The inexperienced users learned how to edit XML files, importing BIBTEX entries into the system and committing to the CVS repository. They just followed steps in a procedure that they see as a black box.

8 Lessons Learned

We believe that our approach and many of the lessons we learned can be applied in numerous similar situations, leading to a lightweight, structured, consistent, and maintainable web site building method. The proposed design satisfies the non-functional

properties we listed in Section 4, and that our approach stands a higher chance to succeed where the two other approaches failed. The initial user reaction was not favorable, but this can be explained by the significantly higher requirements we placed on our users. Typical users are not well acquainted with command line tools, and often see them as a threat to their productivity. Instead of giving instructions by email to an unfortunate web site maintainer, they now had to become active members of an evolving web site maintenance effort. Not all members of our research center proved ready to take this responsibility. Many groups delegated the maintenance to a single person. Others started with a centralized approach and later divided the maintenance responsibilities as they came to appreciate the efficiency benefits of the distributed site maintenance. Still, however, we succeeded in distributing the previously entirely centralized maintenance effort across our groups. Summarizing, we believe that adopting a software development metaphor and tools for developing and maintaining semi-dynamic web site is a practical and worthwhile approach.

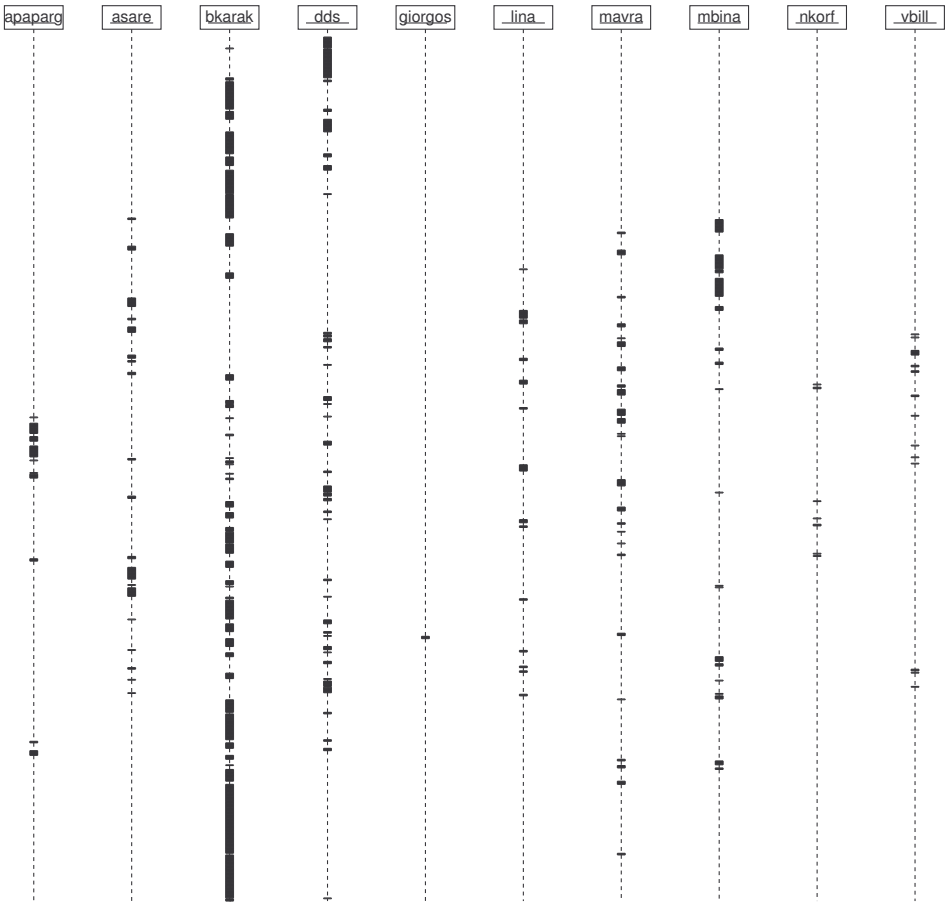

Figure 13: Commit progress time line

In Figure 13 we illustrate the CVS *commit* commands that have been performed by the Content Developers and Administrators. Each swimlane in the Figure represents a committing member of our system. Each horizontal tick represents a single commit instance.

Content Administrators are *dds* (Diomidis Spinellis) and *bkarak* (Vassilios Karakoidas). During the initial lit we can see that only the two committed changes. It was the development period of the system. After the initial development, a few pioneer content developers started to use the system and commit XML and bibliography data. In this period we also tested the system thoroughly and developed the finalized the presentation of the web site. After the end of the test period all users became active and began to commit data in a parallel manner. Figure 13 proves that we achieved one of our primary goals, converting the web site maintenance monolithic procedure into a distributed over time and multiple user development activity.

9 Acknowledgments

We would like to thank Prof. Manolis Skordalakis for his very perceptive comments during the compilation of this paper. The authors would also like to thank George Oikonomou and Marianthi Theocharidou who reviewed versions of this paper and provided many corrections and observations.

References

[1] John Viega and Gary McGraw. chapter Database Security.

[2] Huiqun Yu, Xudong He, Yi Deng, and Lian Mo. A formal approach to designing secure software architectures. In *Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE'04)*, pages 289–290, March 2004.

[3] Michael K. Bergman. The deep web:surfacing hidden value. *Journal of Electronic Publishing*, 7:3, August 2001.

[4] Jaroslav Pokorny. Web searching and information retrieval. *IEEE Computer Software*, 6(4):43–48, 2004.

[5] Erich Gamma and Kent Beck. *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*. Addison-Wesley, Boston, MA, 2004.

[6] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Boston, MA, 2000.

[7] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, Boston, MA, 2000.

[8] Elliotte Rusty Harold and W. Scott Means. XML *in a nuttshell*. O'Reilly and Associates, Sebastopol, CA, 2001.

[9] Stuart I. Feldman. Make—a program for maintaining computer programs. 9(4):255–265, 1979.

[10] Diomidis Spinellis and V. Guruprasad. Lightweight languages as software engineering tools. In J. Christopher Ramming, editor, *USENIX Conference on Domain-Specific Languages*, pages 67–76, Berkeley, CA, October 1997. Usenix Association.

[11] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. Declaratrive specification of web sites with STRUDEL. *The VLDB Journal*, 9(1):38–55, 2000.

[12] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. *SIGPLAN Notices*, 35(6):26–36, 2000.

[13] Diomidis Spinellis. Notable design patterns for domain specific languages. *Journal of Systems and Software*, 56(1):91–99, February 2001.

[14] Emmanuel J. Yannakoudakis. Evaluation of web sites for ministries and the public sector. Technical Report 94, Greek Computer Society (EPY), September 2003. Greek Computer Society Bulletin.

[15] Moshe Bar and Karl Franz Fogel. *Open Source Development with* CVS. The Coriolis Group, Scottsdale, AZ, 2001.

[16] Micheal A. Cusumano and Richard W. Selby. *Microsoft Secrets*. The Free press, 1995.

[17] Object Management Group. Introduction to OMG'S unified modelling language (UML), March 2004. Available online at http://www.omg.org/gettingstarted/what_is_uml.htm.

[18] Derek Robert. CVS – concurrent versions system v1.12.5, December 2003. Available online at http://www.cvshome.org/docs/manual/cvs-1.12.5/cvs.html.

[19] Oren Patashnik. BIBTEXING, February 1988. Available online at ftp://sunsite.unc.edu/pub/packages/TeX/biblio/bibtex/distribs/doc/btxdoc.tex.

[20] Leslie Lamport. *LaTeX: A Document Preparation System, 2nd Edition*. Addison-Wesley, Reading, MA, 1994.

[21] Diomidis Spinellis. bib2xhtml – convert bibtex files into HTML, June 2004. Available online at http://www.spinellis.gr/sw/textproc/bib2xhtml/.

[22] Mikhail Grushinskiy. Xmlstarlet command line XML toolkit, February 2004. Available online at http://xmlstar.sourceforge.net/.

[23] World Wide Web Concortium. extensible markup language (XML), August 2003. Available online at http://www.w3c.org/XML/.

[24] World Wide Web Consortium. XSL transformations (XSLT) version 1.0, November 1999. Available online at http://www.w3.org/TR/xslt.

[25] Free Software Foundation (FSF). GNU make, April 2002. Available online at http://www.gnu.org/software/make/.

[26] Andrew Oram and Steve Talbott. *Managing projects with make, 2nd Edition*. O'Reilly and Associates, Sebastopol, CA, 1991.

[27] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tompkins, and Eli Upfal. The web as a graph. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–10. ACM Press, 2000.