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

Diomidis Spinellis,* Vassilios Karakoidas, Damianos Chatziantoniou
Department of Management Science and Technology
Athens University of Economics and Business

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. 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 methodology suitable for managing middle-sized semi-dynamic web sites. The technological dimensions of this approach are well-known technologies and open source tools, such as XML transformations and version control systems. The prototype framework we developed for the testing of the proposed methodology is also demonstrated, along with an overview of our framework acceptance.

Keywords

Web site development, content management systems, configuration management, XML, XSLT, XSD, HTML.

*Contact author. Address: Patision 76, GR-106 74, Athens, Greece. Email:dds@aueb.gr; phone: +302108203981; fax:+302108203370

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 a loosely organized manner with manually updated web pages, and, as a result, it can be inconsistent in both structure and presentation. While the use of cascading style sheets (CSS) can help one obtain a consistent look, their use still requires discipline. Moreover, the authored pages remain loosely structured and the resulting site can be difficult to modify and reorganize. In addition, the information is often duplicated among hard coded web pages. Furthermore, the static authoring model often imposes a centralized management and maintenance style; all additions and changes have to go through a single person or group, creating a bottleneck, often leading to outdated content.

Adopting a database driven approach solves 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 structure on the data being stored. Finally, a database back-end allows concurrent updates by different users. Note that the adoption of a database-driven approach me be explicit through the development of a bespoke web application, or implict through the installation of a general-purpose content management system.

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 data into HTML, 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 [VM01], 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 [YHDM04]. 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 de-

signers 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 [Ber01, Pok04].

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. Elmasri and Navathe have identified the implications that are created when an organization adopts such an approach [EN00]. They clarify that an organization should avoid using a database system, if they cannot afford the overhead cost for:

- the high initial investment in hardware, software and training,
- the generality of the DBMS for the definition and processing of data, and
- the provision of security, concurrency control, recovery, integrity functions.

In addition, they note that it is desirable to use a straight-forward regular file solution when the following criteria are met:

- The database and the related applications are simple, well defined and not expected to change
- Multiple user access to data is not required

Having witnessed the problems we described above in a number of organizations, we reasoned that a different approach was needed to tackle them. As an example, 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.

The main contribution of our work is the identification of a class of web sites where the application of lightweight development tools and techniques, such as the ones we describe, will efficiently yield structured and maintainable content.

2 Related Work

During the last decade, the World Wide Web became a popular platform for many IT applications. Researchers and companies, in their attempt to make robust web applications, developed many frameworks and tools to formalize and make efficient the creation and maintenance process.

Fraternali [Fra99] analyzed and studied the perspectives of web development. He pointed out, that the development process has five distinct perspectives: (1) *process*, (2) *models, languages and notation*, (3) *reuse*, (4) *architecture* and (5) *usability*. These

include elements like *requirements*, *prototyping and validation*, *evolution and maintenance*, that apply in software engineering scientific areas, and other aesthetic elements like *visual quality* and *degree of proactivity*. As we will see in section 5.1, in our work we took into consideration all the above perspectives and we introduced a few more, like *openness* and *observability*. As a plus, in our methodology we implement the system taking into account the organizational aspect of maintainability of content and the underlying application.

In addition, Fraternali categorizes relevant software tools into six individual categories: (1) *visual editors and site managers*, (2) *web-enabled hypermedia authoring tools*, (3) *web-*DBPL *(database programming languages) integrators*, (4) *web form editors, report writers, and database publishing wizards*, (5) *multiparadigm tools* and (6) *model-driven application generators* [Fra99]. These tools exist in both commercial and open-source incarnations, and they are widely used. In section 6.1, we will see that in our case we chose open source tools to satisfy the *openness* non-functional requirement.

Many research groups also designed and implemented frameworks to organize the web development process. These frameworks addressed issues such as *lifecycle coverage*, *process automation*, *modeling abstractions*, *reuse and components*. The most popular implementations are Araneus [MAM03], Autoweb [FP00] and Strudel [FFLS00]. Many are extended to enhance their abilities. For example, the Strudel framework introduced a declarative query language, called StruQL that was later extended by FunStruQL [FST99]. Another interesting framework is WebJinn [KL03]. This framework tries to resolve the crosscutting concerns in web development.

All the above architectures are introducing development methodologies and languages/scriptlets to achieve their goals. For example, Strudel with StruQL and Araneus with homer. Our goal was to create a system that would depend on well-known technologies like XML and CVS.

For a successful high-level design of a web application, a visual notation is often introduced to specify composition and navigation features in hypertext applications. As an example, WebML [CFB00] is a language based on standards like the entity-relationship model [Che76] and UML [Obj04]. Araneus also defined a logical model known as the Araneus Logical Model [MAM03]. These higher level abstractions are well defined and very useful for the overall design of a web site, but are based on custom, and sometimes proprietary, technologies. In our case we based our semantics on XML to provide a common data hierarhical organization layer.

Some of the characteristics of our approach have also been implemented by Jenkins [Jen04] in what he terms offline programmatic generation of web page, and in the design of the system S [SCK02], which uses a declarative approach for managing data intensive web sites. Our approach however differs from that of Jenkins in that we use a declarative specification rather than imperative code for creating the static content. Our approach also differs from that of the system S in that we use standardized and open data formats and tools for the declarative specification. On a different front, while Nguyen and his collegues [NMT04] suggested the use of software engineering configuration management technologies in the implementation of web projects, they stopped shy of adopting the corresponding software development toolbench approach we use, proposing instead a—common in web development—integrated development

environment solution.

Similarities also exist in the context of content management systems (CMS) [SET02, MT04] and enterprise content management (ECM) [NP04]. CMSs support the creation, management, distribution, publishing, and discovery of corporate information, where an ECM involves a broader range of organizational information. Leading ECM providers include Interwoven, Vignette, Documentum and others. In those systems, a motivation similar to ours leads the research and analogous goals are set [NW05, Doc05]. For example, the unsuitability of static HTML pages for modern web sites and the problems of database-driven approach are also mentioned in [NW05]. CMSs based solely on open standards architectures and XML have also been developed, utilizing methodologies similar to those described here [XYW02, OS005]. However, in our case, we do not propose a content management system but an approach for managing content based on established, widely deployed and tested software configuration management tools. Our approach can be readily adopted by teams already using CMS. Our use of industry-standard data formats and tools provides us with proven scalability (the same tools have been used to manage multi-million line software projects) and flexibility (the tools are used as building blocks and the freely environment is freely configurable). The downside of our approach is that many features that come out of the box in CMSs need to be explicitly tailored in our case.

3 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. The resultant content of our web site was often stale and inconsistent, due to the bottleneck of a single person updating the content.

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 schema, the data, the front end, and the back end. The tension between content and data management is lucidly detailed in an article by Somani *et al* [SCK02].

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 [BF01] that maintains a master repository of the source code. Concepts like the *daily build* [KAL00] 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.

4 Methodology

Our 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 [GB04], Mozilla, and FreeBSD [Jør01]. Using a metaphor as a guiding principle for a development activity is a practice we adopted from extreme programming [Bec00]. The methodology can be summarized in a few key points.

- Structure the site's data as XML files, or another declarative formalism.
- Use XML schemas and corresponding tools to validate the content.
- Exploit modular XHTML [Wor01] 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 [HT00] aptly compare a version control system with a global undo command with unlimited undo levels.)
- Create the site's look and feel in XHTML by transforming the data in a declarative way, using languages such as XSLT and XQuery [HM01, Nov03].
- Express verification and building dependencies among data elements and schemas through *make* [Fel79] or *ant* [Apa04] 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 of the 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 artifacts among developers in their current form, and a building tool, such as *make* or *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 of the product, through a—usually—*daily build* process. As we shall later see, in our case the developers work with XML and XSLT files, instead of

System Requirements		Security	
Application Server	None	Audit Trail	Yes
Database	None	Content Approval	Yes
License	Free/Open Source	Login History	Yes
Operating System	Unix, Win32	Versioning	Yes
Prog/ming Language	XML, XSLT, make	Problem notification	Yes
Web Server	Any	SSL Pages	Yes
Ease of use		Management	
Drag-N-Drop Content	No	Web Statistics	Yes
Email To Discussion	Yes	Content Scheduling	No
Friendly URLs	Yes	Sub-Sites / Roots	Yes
Template Language	Yes	Themes / Skins	No
Undo	Yes	Workflow Engine	No
WYSIWYG editor	No		
Macro Language	No		
Server Page Language	No		
Performance		Support	
Advanced Caching	No	Online Help	No
Database Replication	No	Public Mailing List	Yes
Load balancing	No	Pluggable API	No
Page Caching	Yes		
Static Content Export	Yes		
Interoperability		Flexibility	
RSS Feeds	No	CGI-Mode Support	No
FTP support	No	Content Reuse	Yes
UTF-8 support	Yes	Multi-lingual Content	No
XHTML compliant	Yes	Wiki Aware	No
WAI compliant	Yes	URL Rewriting	No

Figure 1: Methodology features at a glance

programming language source code files, and the build process is usd extensively as a part of the self-reviewing procedures.

Our proposal also covers the issue of *efficient replication management*, for almost all the available types of Content Delivery Networks (CDN). Our version control system centric approach, simplifies the integration of Server Triggered Replication Strategies [SSPvS04], where the content can be generated in each replica. On the other hand, an organization also can apply a static content HTML replication strategy, since all the web site is generated as plain HTML pages.

In order to provide a more detailed comparison between our proposed approach and other commercial or open source CMSs we created a comparison table based on criteria found in CMSMatrix web site [Cor05]. The feature matrix is illustrated in Figure 1.

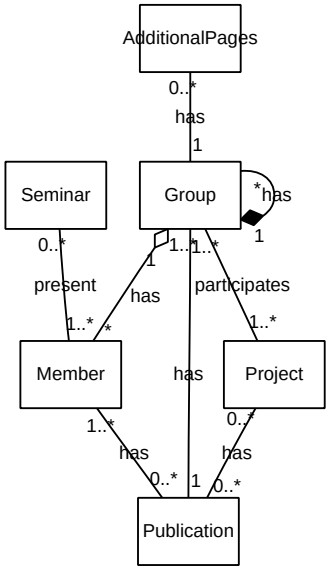

Figure 2: Overview of data element relationships.

5 Designing a Prototype

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.

5.1 Requirements

The functional requirements for our center's web site were simple, but not trivial. 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 2. The self-referential relationship on the group entity exists, because the center should be 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 to 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.

Before embarking on our implementation, we specified a number of non-functional requirements, to ensure that our third go would produce a result with a longer life span. 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. 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 [SG97]. 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 [FFLS00] and Domain-Specific Languages (DSL) [DKV00, Spi01] 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, or OS-X) using 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.

5.2 Conceptual Framework

The system's conceptual framework is illustrated in Figure 3. The main concept in our system is the *organization*. Each organization has information that interests *content consumers*, who access the organization's *public data*. The organization houses *content developers*, who update its public data. Finally, *content administrators* support the content developers (through training, troubleshooting, and account management) and also inspect the submitted public data.

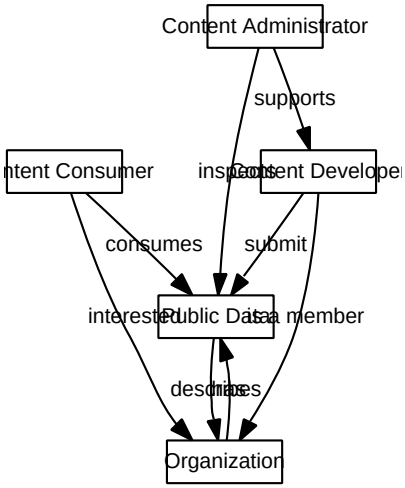

Figure 3: Conceptual design of our system

5.3 Processes

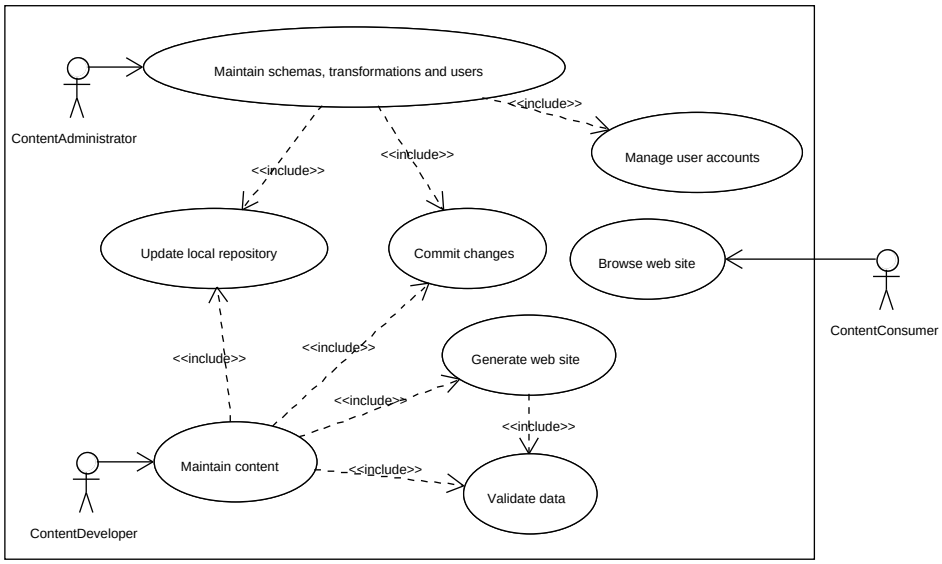

Figure 4: UML Use Case Diagram of the System

Figure 4 shows a UML [Obj04] use case diagram of our system. We have already introduced the actors associated with the system. The use cases that comprise our

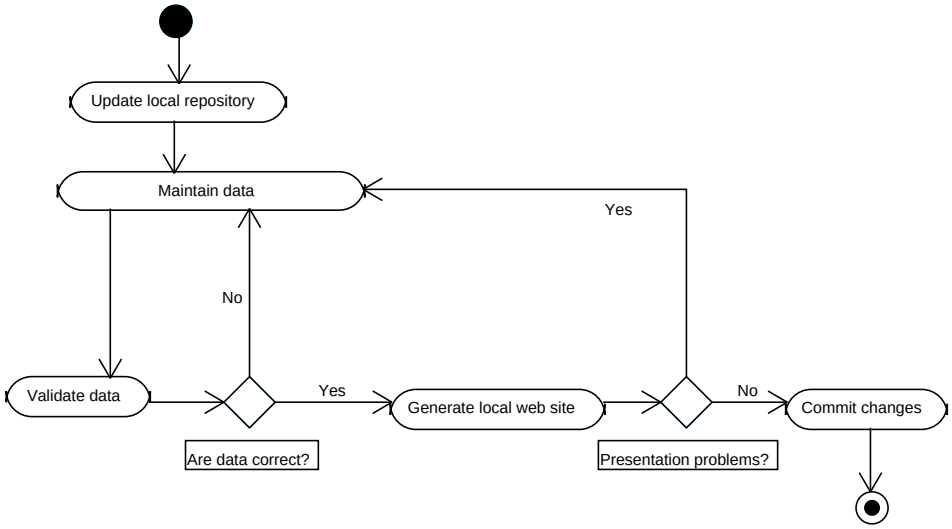

Figure 5: UML activity diagram for Maintain Content

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 and the transformation scripts in order to correct errors and add new features.

Update the 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.

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 the public web site Content administrators and developers can generate the web site 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 they use appropriate data schemas.

Manage user accounts Content administrators also perform user management in the system.

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.

6 Prototype 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 5.1, we decided to use popular open technologies as key elements of our system. We adopted the concurrent versions system (CVS) [BF01] to coordinate the distribution and updating 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 [Pat88, Lam94] and BIB2XHTML [Spi04] for transforming the publications into XHTML and *xmlstarlet* [Gru04] for validating and transforming all other XML-based data [Con03]. For data transformations we implemented a system based on XSLT [Wor99], a language for transforming XML documents. We chose XSLT as the transformation language, because it is based on XML (XSLT documents are 100% valid XML), so the developers can easily learn XSLT, if they understand the basics of XML technology. Finally, GNU *make* [Fre02] 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 FreeBSD.

6.2 System Development

The next step in our development involved a series of iterations where we modeled the data's schema on representative XML files. Concurrently, we implemented the validation DTD / XSDs (Document 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.

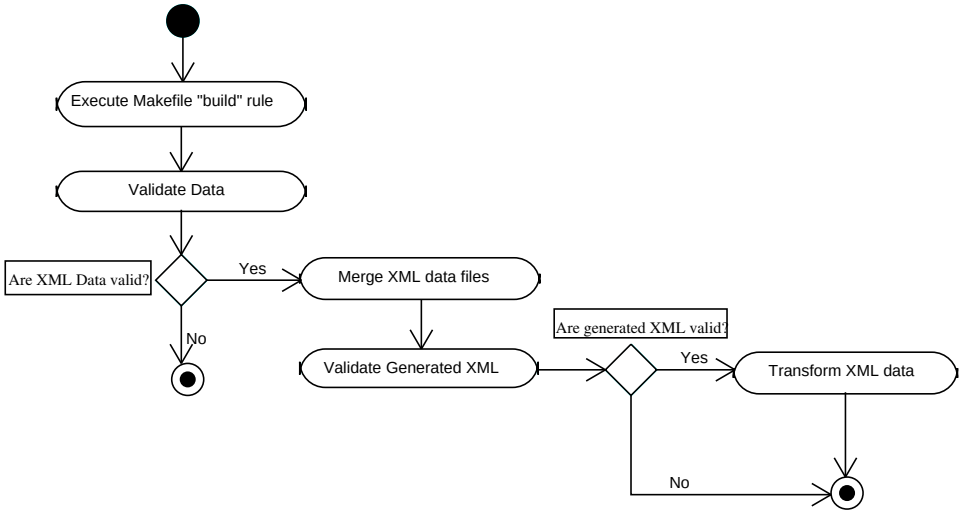

Figure 6: UML Activity diagram for generating the web site

In Figure 6 we illustrate the web site generation process. The automated validation and generation of content is expressed as makefile rules [OT91]. 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. The web pages are assembled through an iteration that applies different perspectives to the same data. It starts, by generating the group XHTML pages, then proceeds with the projects, members, groups, seminars, and publications. Each project, member, group etc. refers to the relevant XML IDs. Upon generation, links are created dynamically each time, to connect relevant pages. For example, a project is hyperlinked with its 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 filesystem. All components of our system are placed under revision control, and all pages are automatically tagged with identifiers denoting their source, helping the traceability of changes. Furthermore, all exchanges between the developers' machines and the CVS and web host are performed using the secure shell

```
<seminar>
 <sem_date>20050112</sem_date>
 <sem_time>12:00</sem_time>
 <sem_room>901</sem_room>
 <sem_duration>90 mins</sem_duration>
 <presentation by="m_bkarak">
  <pres_title>Introduction to Regular Expressions</pres_title>
 </presentation>
</seminar>
```

Figure 7: An example of the XML file describing a seminar

```
<xsl:template match="seminar" mode="full">
 <a name="{current()/sem_date}" />
 <div class="title">
  <xsl:call-template name="date">
   <xsl:with-param name="date" select="sem_date" />
  </xsl:call-template>
 </div>
 <div class="content">
  <h3>Location: <xsl:value-of select="sem_room" /></h3>
  <h3>Time: <xsl:value-of select="sem_time" /></h3>
  <h3>Presentations</h3>
  <xsl:apply-templates select="current()/presentation" mode="full"/>
 </div>
</xsl:template>
```

Figure 8: The XSLT transformation file for the seminar data

(SSH) as the transport protocol guaranteeing the data's integrity and confidentiality.

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

The directory structure of a typical local repository is illustrated in Figure 10. Many of our users are working on the Microsoft Windows environment, and therefore the system repository contains a WIN32 version of the required command-line tools 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 documentation available to 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 requires 8 MB of space on the local machine, plus some extra temporary space for the local content generation.

```
<a name="20050112"></a>
<div class="title">12 January 2005</div>
<div class="content">
<h3>Location: 901</h3>
<h3>Time: 12:00</h3>
<h3>Presentations</h3>
<table class="content">
<tbody><tr>
<td valign="top"><b>Title:</b></td>
<td><a href="">Introduction to Regular Expressions</a></td>
</tr>
<tr>
<td valign="top"><b>Presented by:</b></td>
<td>
<a href="../members/m_bkarak.html">Mr. Vassilios Karakoidas</a><br />
</td>
</tr>
</tbody></table>
```

Figure 9: The generated HTML for the seminar page

In Figure 11 we illustrate a web graph [KRR+00] that shows a subset of 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_" concern projects, with "m_" members, and "g_" groups. At the time of writing the site contained 249 XHTML pages with 3126 hyperlinks.

7 System Adoption

Our research center is multidisciplinary: under its roof are both hard-core 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 graphical user interfaces (GUI). Upon completion of the development, we were somewhat concerned by the way our group would receive the new way of work we proposed 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 positive "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 invalid data into the system. This resulted in all users acquiring the copies of the malformed XML files, and obscure error messages given to unsuspecting users. As is the

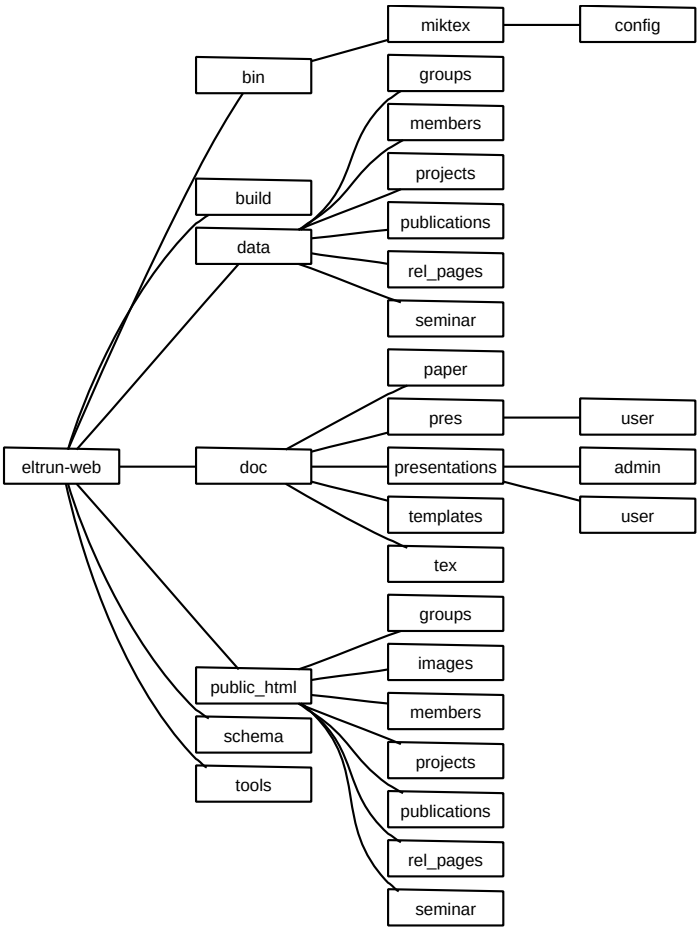

Figure 10: Directory structure of local repository

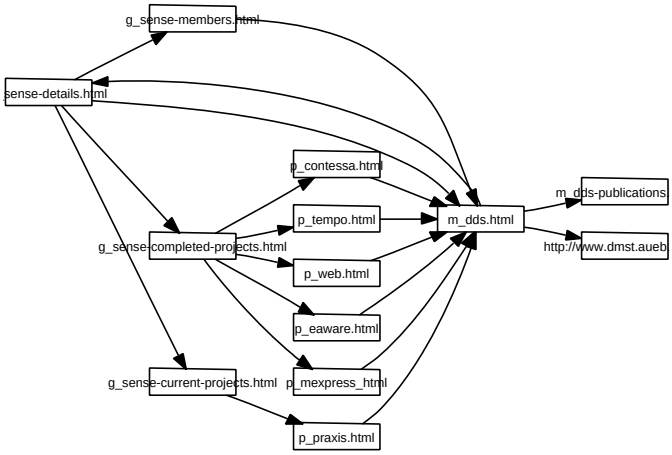

Figure 11: Web graph of the generated site

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. We used the 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.

Two weeks after the first 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 a couple of clearly defined step-by-step procedures.

A few months later our department's technical staff, decided to also adopt our methodology, and develop a similar framework to provide technical on-line documentation for our undergraduate students. This development effort started without our involvement, and we think that the independent adoption of our methodology validated the practicality of our approach.

8 Conclusions

In Figure 12 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* and *bkarak*. During the initial development period we can see that these were the only two contributors. After the initial develop-

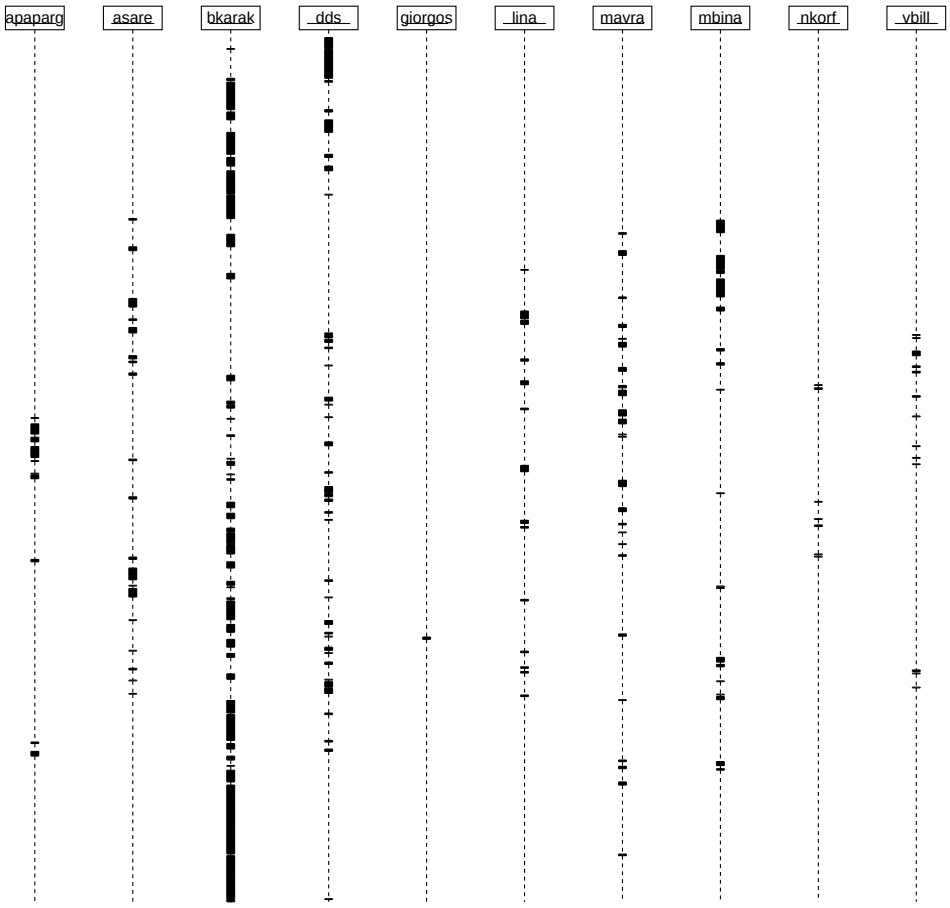

Figure 12: Commit progress time line

ment, 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 and 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 12 thus demonstrates 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.

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 5.1, and that our approach stands a higher chance to succeed where the two other approaches failed. The initial user reaction was not entirely 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 modifying the site by giving email instructions 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. In short, we believe that adopting a software development metaphor and corresponding tools for developing and maintaining semi-dynamic web site is a practical and worthwhile approach in a number of cases.

9 Acknowledgments

We would like to thank 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

[Apa04] Apache Software Foundation. The Apache Ant project, September 2004. Available online at http://ant.apache.org/.

[Bec00] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Boston, MA, 2000.

[Ber01] Michael K. Bergman. The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, 7:3, August 2001.

[BF01] Moshe Bar and Karl Franz Fogel. *Open Source Development with* CVS. The Coriolis Group, Scottsdale, AZ, 2001.

[CFB00] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web modeling language (WebML): a modeling language for designing web sites. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, pages 137–157. North-Holland Publishing Co., 2000.

[Che76] Peter Pin-Shan Chen. The entity-relationship model — toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[Con03] World Wide Web Concortium. extensible markup language (XML), August 2003. Available online at http://www.w3c.org/XML/.

[Cor05] Plain Black Corporation. Cmsmatrix — the content management comparison tool, January 2005. Available online at http://www.cmsmatrix.org/.

[DKV00] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. *SIGPLAN Notices*, 35(6):26–36, 2000.

[Doc05] Documentum. A 15 minute guide to enterprise content management, documentum white paper, January 2005. Available online at http://www.documentum.com/.

[EN00] Ramez Elmasri and Shakmant B. Navathe. *Fundamentals of Database Systems*. Addison-Wessley, Reading, Massachusetts, 2000.

[Fel79] Stuart I. Feldman. Make—a program for maintaining computer programs. *Software: Practice and Experience*, 9(4):255–265, 1979.

[FFLS00] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. Declaratrive specification of web sites with Strudel. *The VLDB Journal*, 9(1):38–55, 2000.

[FP00] Piero Fraternali and Paolo Paolini. Model-driven development of web applications: the AutoWeb system. *ACM Trans. Inf. Syst.*, 18(4):323–382, 2000.

[Fra99] Piero Fraternali. Tools and approaches for developing data-intensive web applications: a survey. *ACM Comput. Surv.*, 31(3):227–263, 1999.

[Fre02] Free Software Foundation. GNU make, April 2002. Available online at http://www.gnu.org/software/make/.

[FST99] Mary Fernandez, Dan Suciu, and Igor Tatarinov. Declarative specification of data-intensive Web sites. In *Proceedings of the 2nd conference on Domain-specific languages*, pages 135–148. ACM Press, 1999.

[GB04] Erich Gamma and Kent Beck. *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*. Addison-Wesley, Boston, MA, 2004.

[Gru04] Mikhail Grushinskiy. Xmlstarlet command line XML toolkit, February 2004. Available online at http://xmlstar.sourceforge.net/.

[HM01] Elliotte Rusty Harold and W. Scott Means. XML *in a Nutshell*. O'Reilly and Associates, Sebastopol, CA, 2001.

[HT00] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, Boston, MA, 2000.

[Jen04] Stephen B. Jenkins. Offline programmatic generation of web pages. *;login:*, 29(5):6–11, 2004.

[Jør01] Niels Jørgensen. Putting it all in the trunk: Incremental software development in the FreeBSD open source project. *Information Systems Journal*, 11(4):321–336, October 2001.

[KAL00] Even-André Karlsson, Lars-Göran Andersson, and Per Leion. Daily build and feature development in large distributed projects. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 649–658. ACM Press, 2000.

[KL03] Sergei Kojarski and David H. Lorenz. Domain driven web development with WebJinn. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 53–65. ACM Press, 2003.

[KRR+00] 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.

[Lam94] Leslie Lamport. *LaTeX: A Document Preparation System, 2nd Edition*. Addison-Wesley, Reading, MA, 1994.

[MAM03] Paolo Merialdo, Paolo Atzeni, and Giansalvatore Mecca. Design and development of data-intensive web sites: The Araneus approach. *ACM Trans. Inter. Tech.*, 3(1):49–92, 2003.

[MT04] Andreas Ulrich Mauthe and Peter Thomas. *Professional Content Management Systems: Handling Digital Media Assets*. John Wiley & Sons, 2004.

[NMT04] Tien Nhut Nguyen, Ethan Vincent Munson, and Cheng Thao. Fine-grained, structured configuration management for web projects. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 433–442. ACM Press, 2004.

[Nov03] Dimitre Novatchev. Functional programming in XSLT using the FXSL library. In *In Proceedings of Extreme Markup Languages 2003 Conference*, August 2003.

[NP04] Stig Nordheim and Tero Päivärinta. Customization of enterprise content management systems: An exploratory case study. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4*, page 40093.2. IEEE Computer Society, 2004.

[NW05] Kathy Jo Nelson and Dave Wilkinson. Anatomy of an enterprise web application , vignette white paper, January 2005. Available online at http://www.vignette.com/.

[Obj04] 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.

[OS005] The international association for open source content management, January 2005. Available online at http://www.oscom.org/.

[OT91] Andrew Oram and Steve Talbott. *Managing projects with make, 2nd Edition*. O'Reilly and Associates, Sebastopol, CA, 1991.

[Pat88] Oren Patashnik. BIBTEXING, February 1988. Available online at ftp://sunsite.unc.edu/.

[Pok04] Jaroslav Pokorny. Web searching and information retrieval. *IEEE Computer Software*, 6(4):43–48, 2004.

[SCK02] A. Somani, D. Choy, and J. C. Kleewein. Bringing together content and data management systems: Challenges and opportunities. *IBM Systems Journal*, 41(4):686–696, 2002.

[SET02] Phil Suh, James Ellis, and David Thiemecke. *Content Management Systems*. Peer Information, 2002.

[SG97] 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.

[Spi01] Diomidis Spinellis. Notable design patterns for domain specific languages. *Journal of Systems and Software*, 56(1):91–99, February 2001.

[Spi04] Diomidis Spinellis. bib2xhtml — convert bibtex files into HTML, June 2004. Available online at http://www.spinellis.gr/sw/textproc/bib2xhtml/.

[SSPvS04] Swaminathan Sivasubramanian, Michal Szymaniak, Guillaume Pierre, and Maarten van Steen. Replication for web hosting systems. *ACM Comput. Surv.*, 36(3):291–334, 2004.

[VM01] John Viega and Gary McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*, chapter Database Security, pages 381–396. Addison-Wesley, 2001.

[Wor99] World Wide Web Consortium. XSL transformations (XSLT) version 1.0, November 1999. Available online at http://www.w3.org/TR/xslt.

[Wor01] World Wide Web Consortium. XHTML 1.1 - module-based XHTML, May 2001. Available online at http://www.w3.org/TR/xhtml11/.

[XYW02] Peng Xu, Wenjun Yang, and Kehong Wang. Xml-based data rendering engine for content management system. In *WAIM '02: Proceedings of the Third International Conference on Advances in Web-Age Information Management*, pages 170–180. Springer-Verlag, 2002. Lecture Notes In Computer Science; Vol. 2419.

[YHDM04] 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.