Proceedings of WSE'99

1st International Workshop on Web Site Evolution

October 5, 1999

Atlanta, GA

Edited by Scott Tilley

1st International Workshop on Web Site Evolution

Foreword

Welcome to WSE'99! The goal of this workshop is to bring together the software engineering, reverse engineering for program understanding, and information technology communities to focus on techniques for Web site evolution. Web sites have moved from supplementary mechanisms for communication to become a primary component of most organization's infrastructure. Rather than serving simply as a passive means of disseminating information, Web sites have become the integration hub for a wide variety of activities, including electronic commerce, streaming media, and online collaboration.

With "older" Web sites now just passing five years on the Net, it seems prudent to examine how they can evolve in a more disciplined manner. As Web sites age, they suffer from some of the same afflictions as any complex software system: their structure degrades, maintenance becomes increasingly problematic, and legacy applications and interfaces hinder evolution. However, Web sites are also unique in several aspects. For example, they are often developed by people who may lack a formal computer science or software engineering background.

I am particularly pleased that the inaugural meeting of this nascent research community is being co-located with WCRE'99, the 6th Working Conference on Reverse Engineering. WCRE has a history of providing enjoyable, informative, and stimulating meetings. I hope WSE can follow a similar path, and that you enjoy the workshop.

Scott Tilley
General Chair, WSE'99
University of California, Riverside

Program Committee

Cornelia Boldyreff, Univ. of Durham, UK
Elliot Chikofsky, META Group, USA
Hausi Müller, Univ. of Victoria, Canada
Jock Rader, Raytheon Systems Co., USA
Lisa Schmeiser, schmeiser.com, USA
Chris Verhoef, Univ. of Amsterdam, The Netherlands

1st International Workshop on Web Site Evolution

Papers

WEB Sites: Files, Programs or Databases?
G. Antoniol, G. Canfora, A. Cimitile and A. De Lucia

An Architecture for Maintaining Link Structure of a Website
S. Arnold and L. Mark

WHAT: Web Hosting Analysis Tool
L. Cherkasova and M. DeSouza

Aspects to Consider for Understanding Web Site Evolution
E. Chikofsky

Server-side Tracking of New Documents
F. Douglis

Evolving an Engineered Web
D. Eichmann

Evolving the Web Into an Accessible Collaborative Learning Environment
K. Hebenstreit

Web Site Evolution: Designing and Developing for the Future
L. Schmeiser

Web Site Evolution - Towards a Flexible Integration of Data and its Representation
M.-A. Storey and J. Jahnke

On the Emergence of the Renaissance Software Engineer
S. Tilley and S. Huang

Management of Web Site Evolution
J. Verner and H. Müller

Characterising Evolution in Web Sites: Some Case Studies
P. Warren, C. Boldyreff, and M. Munro

Toward Reusable and Evolvable Web Sites
K. Wong

Enabling Technologies for Web-Based Legacy System Integration
Y. Zou and K. Kontogiannis

WEB Sites: Files, Programs or Databases?

G. Antoniol, G. Canfora, A. Cimitile and A. De Lucia
University of Sannio, Faculty of Engineering
Palazzo Bosco Lucarelli, Piazza Roma
I-82100 Benevento, Italy
antoniol@ieee.org {g.canfora,cimitile,andrea.delucia}@unina.it

Abstract

WEB sites are central to the WWW community; they are becoming the most widespread way to publish and search information, exchange opinions, communicate. WEB sites are an intriguing phenomenon: they can be easily created, visited, modified by relatively untrained users; a young kid with a browser such as Netscape or Microsoft Explorer can create his/her home pages, upload them on a server and publish his/her site. At the same time WEB sites may hide a tremendous amount of complexity.

In this position paper we claim that WEB sites are such a complex thing that the entire software maintenance and program understanding knowledge and tools are challenged by the site evolution problem.

As suggested by Philip Greenspun [2] sites may be static sites, programs, or databases. Actually, real-world WEB sites are a mixture of the three aspects. More complexity is introduced once the WEB server is taken into account, as WEB servers requires configuration and programming to control site accesses and ensure security, confidentiality and trustiness of the published information.

1. Introduction

The World-Wide WEB is a pervasive phenomenon with tremendous implications in almost all aspects of today society, ranging from the exchange of information, to economy, to the individuals' right and their ability to play a certain social role.

WEB sites, collections of (hopefully) related information accessed by means of a WEB browser, constitute the backbone of the WEB infrastructure and the essential means to publish, update and gather information.

While, it is certainly true that even a young kid with the adequate tools can easily create, upload on a WEB server, and publish his/her pages, there is no doubt that useful WEB sites are something more complex and difficult to manage. As suggested by Philip Greenspun [2], sites may be static sites, programs or databases with a WWW interface. Real-world WEB sites will encompasses these three components: static pages, programs and databases.

In this position paper, we claim that the entire software maintenance and program understanding knowledge and tools are challenged by the WEB site evolution problem. We must be able to help people evolving the static as well as the dynamic parts. The latter may encompass a variety of scripting languages (e.g., Perl, Tcl/Tk, PHP3), programming languages (e.g., C, C++, Java), databases (e.g., Oracle, Sybase, Informix, DB2, mSQL, Ingress) and tools (e.g., compiler, interpreter, WEB server, cryptographic programs).

Moreover, WEB servers requires configuration and programming so that they may control site accesses and assure security, confidentiality and trustiness of the published information. WEB publishing is an extremely serious and complex business, far more serious than it could be perceived at a first glance. WEB servers may be linked together in complex structures or may have links to other service providers (e.g., CyberCash, Paymentech, Verisign).

Consider for example an e-commerce site and the losses that could be caused by a site crash or by an unreliable WEB server: confidential information such as system or database password, or even the user credit card numbers, may be exposed to undesired accesses.

It has already been recognized [6] that WEB sites undergoes Lehman laws of continuous software evolution [4]; HTML could indeed be regarded as a kind of programming language — Caper Johnes provides figures to convert function points into HTML LOC see *http://www.spr.com/library/0langtbl.htm*. However, despite those recognized commonalities could drive site evolution, we believe that usability, security, and performances are the major aspects driving server and site evolution.

Helping people to evolve site may be difficult; even more difficult may be to predict the effect of a site tool evolution. Suppose we update our favorite WEB server and at

the same time we upgrade the databases and/or the connectors between the DataBase Management Systems (DBMS) and the suite of programs that access the database and generate HTML pages on the fly. A not careful resource allocation (e.g., number of pre-opened connections, number and size of buffers, number of pre-spanned servers or server threads), or an unexpected number of users (i.e., unexpected number of transactions per second) may slow down communications stacking users in front of a gray page.

2. Sites

Sites may be thought of as a collection of (hopefully) related pages and links to other WEB sites or service providers. Pages are files written in a markup language, known as HTML. HTML files contain free text intermixed with *tags* of the form $< element > ... < /element >$; the file is accessed via a WEB server and loaded into the browser that renders the text according to the given tag semantics.

Thus for example <I> is a tag $< /$I> will be rendered in italic, $< /$I> instruct the browser to close the <I> element and stop rendering italic. A variety of tags, interpreted by the browser, allows to format a page, include images, access other sites. Several HTML tutorial, training materials and examples are available on the net, see also [5].

2.1 Static sites

Static sites do not change the provided information over time; they are collections of HTML files, i.e. the document interpretation is completely delegated to the browser. These are the simplest sites to build, perhaps if large, not to maintain or evolve in that a simple change may require to edit a large number of files.

The greatest disadvantage of static sites is that they do not allow users to interact with the site: the user can only follows static links, he/she has no way to supply any information (e.g., user name, e-mail, comments, credit card number).

Static sites may be downloaded, mirrored, with spiders, document parsed with HTML parser. Static site analysis and evolution can be compared to program static analysis and evolution. For example, in recent papers [3], [6] traditional software metrics has been adapted and new WEB specific metrics has been proposed to evaluate WEB site evolution.

2.2 Programs

To overcame the limitations of static sites several strategies were developed: essentially, servers were given the ability to execute fragments of code interleaved with standard HTML text. This was accomplished either extending the HTML language or calling external programs communicating with them via the so called Common Gateway Interface (CGI). The CGI standard defines an interface, an abstraction which dictate what and how a program and WEB server communicate.

Being an interface, virtually any programming language could be adopted to write a CGI compliant program, also known as a *cgi-bin*. The name *cgi-bin* is derived from the name of the directory where old WEB servers expected to find CGI compliant executables. Scripting languages such as Perl, Tcl/Tk and PHP3 are currently cgi-bin de facto standard development languages. The reason relay in the reduced development cycle and the ability to port the scripts, on different hardware platforms and operating systems, without the need for recompilation, once an interpreter is available.

The other form of server-side programming, as already mentioned, is obtained by taking the markup concept to its extreme consequences: tags may be defined (e.g. <%, %>) to tell the server to execute the in-between code. Server Side Include (SSI) directive, AOLserver Dynamic Pages (ADP) and Microsoft Active Server pages (ASP) are just three examples: a *traditional* plain HTML document is a legal ADP or ASP document; embedding a fragment between <% and %> will cause the server to perform some computation.

Going back to cgi-bin, spanning a process to execute a piece of code may require a considerable amount of time, comparable with the time spent to execute the cgi-bin program. Furthermore, it may open the way to undesired attacks from intruders. WEB servers like the AOLserver or the Apache server may embed interpreters to overcame these problems. For example, the Apache server can execute both Perl and PHP3 scripts without forking external processes. Projects such as PHP and EmbPerl, are indeed equivalent to SSI, ASP or ADP pagess. The fragment of document in-between selected tags will be interpreted by the server as a Perl or PHP code fragment, executed, and the result sent to the browser.

On the contrary of static sites, an attempt to dowload with a spider a program centric site may cause the infinite reexecution of the same set of cgibin programs filling the dowloader disk space; documents cannot be parsed with standard HTML parsers.

Clearly, these sites may be thougt of as programs with an HTML user interface: the evolution of site centered on the programming paradigm requires the usual program maintenance, comprehension and evolution weapon arsenal. For example, in an ecommerce site, business rule will be encoded in the cgibin much in the same way business rules are encoded in tradition software systems.

2.3 Databases

Consider the problem of maintaining a mailing list of people visiting our site; we would give users the ability to add/remove themselves and to add/remove comments and/or new links. Perhaps, due to confidenciality reasons we want to be the only ones who can view the entire mailing list and send mail to the list. Clearly this task can be easily accomplished if we build a database backed up site: a site that is really an interface to a database.

Notice that this is somehow different than having a corporate database and publishing it: in the latter case we cannot choose the DBMS, we only can hope some means exist to connect the corporate database using some (scripting) programming language (e.g., OBDC, Jdbc, DBI/DBD).

Database backed up sites have the same characteristics of program centric sites plus underlining DBMSs. In other words, we need new weaponry to extract, represent and model the contained data and the relations among data. As an example, consider the Oracle database and the Rational Rose tool suite; Rational Rose offers an interface toward the Oracle products to extract models representing the database relations [1].

Databases centric site pose a constraint on the WEB server technology. DBMS vendors struggled to optimize transaction performances once a database connection was opened: a user session involves several transactions thus the time spent to open and setup the connection with the DBMS is negligible. Unfortunately HTTP protocol is a stateless protocol: we have no simple mean to asssociate a database connection with a given user and its transactions. As we do not like to fork processes 500000 times a day, we do not like to open and close connections to our databases 500000 times a day [2].

2.4 Server and Tools

A WEB site cannot exist without a WEB server. Even better, it cannot exist without a suite of tools that allow to build, maintain, ensure consistency, confidentiality, and trustiness of the published information. Compilers (e.g., C, Java), interpreters (Perl, PHP), cryptographic algorithms (e.g., OpelSSL, pgp), HTML editors belong to this tool constellation.

WEB servers/WEB sites may be very simple stand alone server/site or may be linked together in complex structures. They may also include links to other service providers (e.g., CyberCash, Paymentech, Verisign). Even a stand alone server must be configured to avoid attacks and piracy trying to avoid damages to the hosting computer system.

Evolving a site means also being able to evolve the access policies controlled by the server configuration files as well as the site tool constellation. In the authors experience, this is usually due to security and performance issues. Overcoming known and documented ways for intruder attacks or adding new features improving site performances can be considered the main driving factors. Upgrading a server may require reading extensive documentation and re-editing several configuration files, including user dependent configuration files (e.g., Apache .htaccess files). It may also entail to update the directory structure and/or the file system organization.

Moreover, modern servers, such as Apache, that embed scripting language interpreters (e.g., Perl) or that execute themselves (e.g., the servlet mechanism) may de facto be extended by the Webmaster adding new functionalities. Evolving a WEB server may also mean upgrading the site specific functionality to the new server API.

3. Conclusion

WEB sites may be as simple as a single file or one of the most complex sollection of cooperating software artifacts ever conceived.

In this position paper, we claimed that the entire software maintenance and program understanding body of knowledge and tools are challenged by the WEB site evolution problem. We must be able to help people evolving the static as well as the dynamic parts of a site, predicting performances, and improving security. Such a complex task will probably require the development of new paradigms and tools to face the task complexity.

References

[1] R. S. Corporation. *Rational Rose 98: Using Rational Rose*. feb 1998.

[2] P. Greenspun. *Philip and Alex's Guide to Web Publishing*. MIT Press, April 1999. ISBN: 1-55860-534-7.

[3] A. E. Hatzimanikatis, C. T. Tsalidis, and D. Christoulakis. Measuring the readability and maintainability of hyperdocuments. *Journal of Software Maintenance - Research and Practice*, (7):77–90, 1995.

[4] M. M. Lehman. Programs, life cycles and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.

[5] Musciano and Kennedy. *HTML: the definitive guide*. 1997.

[6] P. Warren, C. Boldyreff, and M. Munro. The evolution of websites. *Proc. of the International Workshop on Program Comprehension*, pages 178–185, 1999.

An Architecture for Maintaining Link Structure of a Website

Stephen C. Arnold
Drexel University
College of Information Science and Technology
Stephen.arnold@cis.drexel.edu

And

Leo Mark
Georgia Institute of Technology
College of Computing
Leomark@cc.gatech.edu

Problem:

As a website ages, multiple authors decide how to link the pages they contribute to the existing pages. The criteria authors use for creating links changes over time. This changing criteria result in inconstant links between pages. This makes it difficult for reader of the pages to find the information they want. [Garzotto at al. 95]

Solution

Force authors to use a consistent criteria for creating links between pages.

Architecture for Implementing the Solution

- Use some semantically enhanced version of HTML.

  A consistent structure of links between pages requires that both the pages and the links have some semantics associated with them. Typing of the pages and links will provide the needed semantics

- Store the webpages in a database.

  A database supporting triggers can be used to perform analysis on webpages as the are added, deleted, or updated. This analysis is to determine that the changes to the webpages do not violate the criteria of what links are allowed and/or required.

  It also can be used to provide security so that an author does not circumvent the program to submit webpages by directly accessing the pages within a file system.

- Specify the linking criteria in database triggers

  When an author submits a set of pages (as a transaction) to the website, a database procedure is triggered which analyzes the set of webpages to determine if the pages satisfy the linking criteria. The procedure will either allow the transaction to be completed, or reject the transaction. If the transaction is rejected, the procedure will give the author detailed error messages about why the transaction was rejected.

- A major issue in implementing the triggers

> The procedures triggered to enforce the link criteria are a form of database constraint. Unfortunately, database constraints have a history of failing to be practical because they increase the time it takes to complete a transaction. This is normally because enforcing the constraint requires searching a significant part of the database. This problem can be avoided if the kinds of constraints used are limited to object-centered constraints. [Delcambre at al. 91]
>
> An object-centered constraint is of a form such that all the data needed to determine if the constraint is violated can be found by tracing references from the object being changed. In terms of a link criteria, if an author changes a webpage, all the webpages which need to be analyzed to determine if the criteria would be violated can be found by traversing links from the webpage which is being changed.

Implementation of the Architecture

A website which implements this architecture can be build from existing software systems.

The semantics needed in the webpages and links can be provided in XML. The prologue section of XML can have typing attributes placed in them to give the page a semantic type. The link specification of XML also can contain a semantic type attribute.

A webserver can be modified to pull pages out of a database. This has already been done by Foo and Lim [Foo and Lim 97].

Many commercially available databases provide trigger mechanism and procedures. Oracle 8.1 provides both PL/SQL and Java as procedure languages which could be used to write the link criteria.

A simple Java applet using JDBC could be written to take the webpages sitting on the author's machine and submit them in a single transaction. This application would rely on a T3 server being present on the machine running the database. This applet could also display the error messages generated by the database if a transaction is rejected.

Example

Knowing who is responsible for procedures and who to contact if the procedures do not work is very important. Many companies distribute their procedures via websites. The following is a list some possible requirements for procedures.

- A staff member is always responsible for every procedure page.
- All procedure pages are associated with a specific department.
- If a staff member is responsible for a procedure page associated with a given department, the staff member must be in that department.
- Only a department head can change who is responsible for a procedure page.
- Only the staff member responsible for a procedure page can create, modify, or delete it.

The first three requirements can be considered a form of structure constraint. The staff persons homepage must be linked to a department page. The procedure page must be linked to both the staff persons homepage and the department page. A database procedure to detect these links could be written.

The last two requirements use the database security system to be implemented. By knowing who submitted the transaction, if the transaction attempts to modify a procedure page, the database can determine if the submitter is allowed to make the change.

Research Areas

A system to implement a website that can enforce a link criteria has not been built yet. Though the concept draws on well-established technologies, the implementation will provide a number of technical issues and make a platform for future work.

Establishing the criteria of what links are allowed, are not allowed, and how links can be changed has not been explored at all. Since this is highly domain dependent and even site dependent, developing an expression system for these criteria would be very helpful.

Automatic tools for creating the procedures that enforce the criteria are very important. Hand implementation of the database procedures will make implementation complicated and maintenance difficult. Also, considering the possibility of procedures that perform unbounded searches, it is important to make sure the constraints the procedures implement are object-oriented constraints.

Providing meaningful feedback to the authors when they submit pages is also very important for the acceptance of a system. When a database procedure determines that a transaction is to be rejected, the procedure may have a great amount of information about the author's error. It is important to get this information back to the author in a way that the information is useful.

Biblography

[Garzotto at al. 95], Franca Garzotto, Luca Mainetti, and Paolo Paolini, Hypermedia Design, Analysis, and Evaluation Issues, Communication of the ACM, 38(8): 74-86, August 1995.

[Delcambre at al. 91], Lois M. L. Delcambre, Billy B. L. Lim, and Susan Urban, Object-Centered Constraints, Proceedings of the Seventh International Conference on Data Engineering, April 8-12 Kobe, Japan, pages 368-377.

[Foo and Lim 97], Schubert Foo and Ee-Peng Lim, A hypermedia database to manage World-Wide-Web documents, Information & Management, 31: 235-249

WHAT: Web Hosting Analysis Tool

Ludmila Cherkasova, Mohan DeSouza*
Hewlett Packard Laboratories
1501 Page Mill Road,Palo Alto, CA 94303
e mail: {cherkasova,mdesouza}@hpl.hp.com

1 Introduction

As the Web increasingly becomes a core element of business strategy, the task of hosting web content has become mission critical. Few companies, however, have the resources, money and expertise to build their web site entirely in house. For this reason, many businesses choose to outsource their Web hosting to Internet Service Providers and some equipment vendors which, according to Forrester Research Inc., can slash costs by 80%.

Although the recent survey revealed over 1 million web servers on the Internet, the number of web sites exceeds this number by several times. The illusion of more web sites existing than actual web servers is created through the use of *virtual servers (hosts).*

The shared Web hosting service is based on this technique. The shared Web hosting market targets small and medium size businesses. The most common purpose of a shared hosting web site is marketing (in other words, it means that most of the documents are static). In this case, many different sites are hosted on the same hardware.

A Web hosting service uses the possibility to create a set of virtual servers on the same server. There are different alternatives to how this can be done. Unix web servers (Netscape and Apache) have the most flexibility in addressing the Web hosting problem. Multiple host (domain) names can he easily assigned to a single IP address. This creates the illusion that each host has its own web server when, in reality, multiple, "logical" hosts share one physical host.

Each virtual server is set up to write its own access log. This is a very convenient configuration for the hosted sites (customers). The site access logs allow us to analyze incoming traffic to the site both quantitatively and qualitatively. Access logs provide invaluable information on both the most oftenly requested documents and the most active, frequent visitors of the site.

However, this implementation and set up splits the "whole picture" of web server usage into mu! tiple independent pieces, making it difficult for the service provider to understand and analyze the "aggregate" traffic characteristics.

The situation gets even more complex when a Web hosting infrastructure is based on a web server farm or cluster, used to create a scalable and highly available solution.

2 WHAT's Design Approach

Our goal was to develop a tool which characterizes an overall Web hosting service profile and system resource usage in both a quantitative and qualitative way. We have chosen to report information which

* Work has been done while M.DeSouza worked at Hewlett. Packard Labs during the summer of 1999. His current address is University of California, Department of Computer Science, Riverside. CA 92521: e mail: mdesouza@cs.ucr.edu

could be used by a Web Hosting Service Provider to evaluate the current solution and to improve and optimize the relevant components using overall service profile data.

WHAT performs an analysis which is entirely based on web server access logs collected from multiple sites hosted on a server (web server farm or cluster). The tool is written in Perl for the Common Log Format, which is the most popular default for web server access logs.

WHAT is aiming to provide:

- *service characterization* a service profile, a comparative analysis of system resource usage by different customers (i.e. by hosted web sites);
- *traffic characterization* a comprehensive analysis of overall workload with extraction of a few main parameters to characterize it;
- system *requirements characterization* a related system resource usage analysis, especially mem ory requirements.

These characteristics provide an insight into the system's resource requirements and the traffic access patterns the information which is of a special interest to system administrators and service providers.

3 Service Characterization

The study [MS97] asserts that the three primary issues that characterize a site are:

- site composition and growth;
- growth in traffic;
- user access patterns.

Our Web hosting site analysis supports this statement too. The monthly growth of the requests rates for different sites differ significantly. While the typical growth for most of the sites is exponential, it takes different times for different sites to double. Some of the sites experience decrease of the traffic rates and actually demonstrate negative growth. User access patterns differ significantly too. For example, some sites have a few, very popular documents or products. •The accesses to such sites are heavily skewed: 2% of the documents account for 95% of the site's traffic. In order to design an efficient, high quality Web hosting solution, the specifics of access rates and users' access patterns should be taken into account. The traffic growth/decrease and the users' access patterns' changes should be monitored in order to provision for those changes well in time and in the most efficient way.

WHAT's design and development was driven by the case study of HP Web Hosting Service provided to internal customers. We performed the analysis which covers a four month period: from April,1999 to July, 1999. Originally, in April, the service had 71 hosted sites. By the end of July, the service had 94 hosted web sites. During this period, WHAT's analysis allowed us to monitor and analyze each particular site's traffic contribution to the overall traffic, and the evolution of the whole service by itself.

WHAT identifies all the different hosted web Sites (from the given collection of web server access logs). For each hosted web site *i*, the tool builds a site profile by evaluating the following characteristics:

- AR_i the access rates to a customer's content (in b_1*jtes* transferred during the observed period);
- WS_i the combined size of all the accessed files (in *bytes* during the observed period, so called "working set");
- FR_i the table of all accessed files with their frequency (number of times a file was accessed during the observed period) and the files sizes.

We normalize both AR_i and WS_i with respect to *AR* and *WS* combined over all the sites in order to identify the percentage contribution of each particular site.

The access rate AR_i gives an approximation of the load to a server provided by the traffic to the site *i*. The working set WS_i characterizes the memory requirements by the site *i*.

These parameters provide a high level characterization of customers (hosted web sites) and their system resource requirements. This characterization is especially useful when it is time to scale the system. It can help to identify whether additional memory is going to he enough, or whether the service provider needs to add a new server. If a new server is added, often the content is going to be partitioned as well. The site profiles help to create a balanced partition with respect to a system's resources, avoiding the "bad" partitions where the "memory hungry" sites are left on one server and the "high load" sites are moved to a new server. **WHAT** provides valuable sites analysis to be used for capacity planning and balancing tasks.

4 Traffic Characterization

WHAT provides the analysis of the combined traffic to all the sites.

It reports the number of successful requests (code 200), *conditional get* requests (code 304) and the errors (the rest of the codes). The percentage of *conditional get* requests often indicates the "reuse" factor for the documents on a server. These are the documents cached somewhere in the Internet by proxy caches. The *conditional get* request is sent to fetch the document only in case it was modified.

WHAT provides the statistics for an average response file size (averaged across all successful requests with 200 code). We also build a characterization of the file size distribution. For this purpose, we build a table of all accessed files with their sizes and access frequency information, ordered in increasing order by size. It allows us to build a file size distribution of the request in a style which is similar to SpecWeb96 [SpecWeb96] the industry standard benchmark for measuring web server performance.

Thus, for example, in our case study (for April), the average request size was 22.7 KB. The average request size (for 30/60/90% of the requests) was 0.8 KB/1.6 KB/4.6 KB respectively. From the data above, one can conclude that a workload (in terms of file size distribution) has a long tail of rarely accessed very large files.

WHAT reports a percentage of the files requested only a few times the files requested less than 2/6/10 times. In our case study, files requested less than 2/6/10 times account for 34.4%/66.0%/75.7% of all the requests. This is another important characterization of traffic which has a close connection to document reuse and gives indication of memory (RAM) efficiency for the analyzed workload. Most likely "onetimers" are the requests served from the disk. This data is helpful in understanding whether performance improvements can be achieved via optimization of the caching or replacement strategy.

Here the traffic *characterization* comes very close to *system requirements characterization*.

5 System Requirements Characterization

System requirements arc characterized by the combined access rate and working set of all the hosted sites (during the observed period of time). The tool also provides the worst hour access rate analysis.

WHAT provides the combined size of "onetimers" (in our case study this summed up to 388 MB from total of 1042.5 MB). High percentage of "onetimers" and small memory size could cause had site performance.

In order to characterize the "locality" of overall traffic to the site, we build a table of all accessed files with their sizes and access frequency information, ordered in decreasing order by frequency. **WHAT** provides working sets for 97/98/99% of all requests.

In our case study, 97/98/99% of the requests required 262.6 MB/375.7 MB/600.4 MB of memory respectively. The smaller numbers for 97/98/99% of the working set indicate higher traffic locality: it means that the most frequent requests target smaller set of documents.

6 Large Single Sites Analysis

WHAT was designed for Web hosting service analysis needs. We realized, however, that its usage can be extended to provide the analysis of large single sites in a very useful way.

Our second case study was analysis of the ***www.hp.com*** web site.

WHAT's functionality was extended to identify all the first level directories. First level directories give direct indication of web site composition. Often, the first level directories represent different business units or reflect the company products, and the traffic analysis of these directories is of interest to these units.

After that, we performed an analysis similar to the Web hosting service analysis, where the first level directories were treated as different web sites.

Such an approach to the analysis of large single web sites allows us to outline the site composition as well as determine the percentage of traffic going to the Site's different parts. **It** allows to create accurate "sub site" profiles in terms of memory usage **and load on a system. Such** analysis helps observance of the site evolution and the design of more efficient web sites.

7 Conclusion

There are several web log analysis tools freely available (Analog [Analog], Webalizer [Webalizer] to name just a few). They give detailed analysis of the most frequent accesses and the user population. This data is useful for business sites to recognize who their customers are and what documents or products get most attention.

However, these tools are lacking the information which is of interest to system administrators and service providers; the information which provides insight into the system's resource requirements and traffic access patterns.

When the site is a collection of different sites created through the use of *virtual servers (hosts)* a new analysis tool is required to understand the site's contributions to overall traffic, as well as the resource requirements imposed by each particular site. Such sites evolve in a special way: since the different sub sites "live" their different lives. **WHAT**'s analysis helps to observe site evolution and to provision for changes well in time and in the most efficient way.

8 References

[Analog] Analog: http://www.statslab.cani.ac.uk/ sretl/analog.

[MS97] S.Manley and M.Seltzer: Web Facts and Fantasy. Proceedings of USENIX Symposium on Internet Technologies and Systems, 1997, pp.125 133.

[SpecWeb96} The Workload for the SPECweb96 Benchmark.

URL http://www.specbenth.org/osg/web9fi/work1oad.html

[Webalizer] Webalizer: http://www.mrunix.net/webalizer/

Aspects to Consider for Understanding Web Site Evolution

Position statement for 1st Intl Workshop on Web Site Evolution
October 1999

Elliot J. Chikofsky
META Group Inc.
Performance Engineering and
Measurement Strategies (PEMS)

75 Lexington Street
Burlington, MA 01803 USA
ph +1-781-272-0049
fax +1 781-272-8464
e.chikofsky@computer.org

In advising and assisting Global 2000 companies, META Group has found that web site planning and evolution is rapidly becoming an essential part of any corporation's strategies for electronic commerce and business for the next decade. The issues of web site evolution are driven by global electronic commerce requirements and evolution, not just technical considerations. The technology of web site evolution must be considered in this context.

Global Business and Technology Evolution Observations

META Group recognizes these trends that are applicable to understanding the global environment in which web site evolution occurs:

Business Requirements

1. Global supply chain requirements, electronic interchange, and the externalization of business computing (80%+ by 2003) will drive the renovation and revitalization of users' legacy application investments, and necessitate ubiquitous Web/browser technologies for global business interaction (e-legacy). By 2004, more than 90% of legacy data will be accessible via I*net technologies.
2. The integration of legacy systems via increasingly robust messaging/queuing, Java, and Web-enabled middleware will facilitate heterogeneous connectivity throughout the global enterprise IT ecosystem (2003/04). While traditional subsystem components (e.g., CICS, Tuxedo, IMS) will be enhanced to handle and manage the exploding global (e-business) transaction load, they will be masked and leveraged via maturing middleware-like products (including Web front ends).
3. By 2002, 70% of the Global 2000 will have formulated portal strategies or aligned with a major Internet portal (e.g., Yahoo, AOL, Microsoft) and multiple "portlets" (e.g., Amazon, InsWeb, Travelocity). Portals will provide extended business and consumer services (e.g., bill payment, service/goods brokering), resulting in significant transactional revenue (adding to advertising and "presence leasing" revenue) by 2003/04.
4. Personalized and customized relationships, derived from integrated back-office (legacy, ERM, data marts, etc.) and front-office (call center, Web, TV/radio, paper, POS) interactions, will be standard within customer relationship management architectures by 2002. Externalization capabilities (EPI, workflow, middleware, etc.) will enable organizations to enhance relationships with both consumers and trading partners, while supporting application and intermediary interactivity.

Job Roles

5. By 2000, "Web architects" will manage dynamic content management platforms comprising XML, metatagging/indexing, document management, information retrieval, and secure publishing/access services, supporting numerous/diverse content owners and generators. By 2002/03, Web infrastructures

(driven by database, groupweb, and platform vendors) will provide these features as standard functionality.

6. Through 2002/03, Global 2000 IT groups will continue to struggle to find/afford staff and reallocate resources to more value-added roles (e.g., business liaison, component integrator, marketer, customer officer, enterprise project manager, new-age technologist). Critical IT characteristics (2003-05) will include governance principles, strategic sourcing methodology, infrastructure sales/marketing, metrics derivation, collaboration/learning, and competitive value articulation.

Web Delivery Technologies

7. By 2002, XML components and standards (e.g., XML/EDI, XSL, WebDAV, DOM) will be the nucleus of Web information management. However, XML development will proceed slowly through 2000, even as XML facilities become embedded within application server, database, document management, and other products.
8. Through 1999/2000, traditional AD environments for multi-tier client/server will expand to encompass application services, object model frameworks, and integrated development environments (IDEs); this will drive increasingly componentized architectures. Concurrently, Web development tools and application servers supporting target-agnostic implementations will mature and converge with conventional client/server IDEs. By 2002/03, evaluation criteria will shift from infrastructure (e.g., scalability, reliability, availability) to developer productivity as technology and architecture become commoditized.
9. Maturing Java will expand beyond the presentation layer to encompass flexible, user-driven application logic (2001/02); componentization of legacy systems (via Component Broker, Enterprise JavaBeans, et al.) and efficient, more portable applications will drive the blurring of platform personalities (OS/390, Unix, NT, OS/400) for both e-legacy and new applications.

Web Site Evolution Observations

As a developer, user, and consumer of web sites in commercial and non-commercial applications, including sites for IEEE groups and conferences, I have observed the following characteristics of web site evolution:

- Many, if not most, web sites do not evolve; They are replaced. Web sites are subject to wholesale change due to the pressures to "keep it interesting", "keep it fresh". Web site developers are often told that they must make wholesale changes in updating their site to "make people want to come back for something new".

- Incremental improvements are discouraged by the need to attract good search engine placement. Search engine spiders are so busy with considering the explosion of new pages that incremental improvements to existing pages do not get their attention. The issues of needing and allowing your web site to be found force a behavior of more extensive change.

- Multiple evolutions are going on at the same time. These interact in a complex pattern, where trends and requirements reinforce or cancel (make unnecessary or irrelevant) others in parallel evolving paths.

    - Evolution of the business requirements for electronic commerce and dissemination.
    - Evolution of backend technologies.
    - Evolution of the interfaces between back-end and front-end technologies.
    - Evolution of the web presentation technology, such as the inclusion of frames, HTML extensions, XML-cognizant software, security services, and so forth.
    - Evolution of the application-specific web content.

Server-side Tracking of New Documents

Fred Douglis
douglis@research.att.com
AT&T Labs–Research
Florham Park, NJ, USA

September 10, 1999

Abstract

AIDE/WN is a tool for automating the "What's New?" page found on many websites, to customize the display of new pages for each user. It uses cookies, something that is already common for other purposes, to identify a particular user across visits and to track the versions of each resource that the user sees. It uses the *Htmldiff* tool to display the specific changes on request, and at the option of the web administrator, permits users to view the version history of a page.

1 Introduction

Pages on the Web change in unpredictable ways. For this reason, many sites include pointers to a "What's New on this Site" page. (I abbreviate this as **WN** henceforth.) These are maintained by webmasters or others with direct access to the content, and are updated based on their notion of what might be interesting.

In some ways, manually maintained WN pages are desirable. They use a human's judgment to distinguish between small, uninteresting changes and things that should really grab one's attention. Furthermore, they permit a person to summarize the changes in a simple fashion, add emphasis when appropriate, age less interesting changes off the WN page faster than the most important changes, and so on.

Indeed, Chen and Koutsofios's *Website News* system [3], which tracks changes recursively from a site's home page, has demonstrated that the number of pages that change is much greater than the number highlighted in most site's WN pages. It stands to reason that if every changed page appeared in every WN list, the list itself would be relatively useless.

At the same time, these generic WN pages are quite limited. If the same page is presented to every user, then after a user sees the WN page, she is likely to see the same changes presented as "new" again in the near future. The basic problem is that what's *new* to one user is *old* to another. A second problem is that determination of what's interesting is left to the webmaster. A user who has seen a particular page on the site and would like to know when that page is updated must rely on the judgment of the webmaster to include an update to that page in the WN list. Finally, a third (and somewhat less important) problem is that the WN list might change before a user ever sees it: a new page shows up in the list and then drops off again. What's *old* to most of the visitors to the site is *new* to the infrequent visitor.

The *AT&T Internet Difference Engine* (AIDE) has been developed over the past several years as one way to address these problems [4]. It allows users to specify pages of interest, be notified when the pages change, and most importantly, see *how* the pages have changed since the user last viewed them. A page that changes several times between visits by a particular user will be highlighted to show *all* the changes during that interval.

Here I describe a derivative of AIDE, which integrates AIDE's page tracking and version management with the HTTP server, rather than relying on individual users to explicitly access AIDE to request notifications. In fact, the WN page can look like any other web page (e.g., `http://foo.bar/whatsnew.html`), except that its contents will reflect exactly what is new to the user accessing it. I refer to the derivative as **AIDE/WN**.

2 Customization

Customizing the WN page to each user requires that the browser store a *cookie*, which it passes to the server on each request. The use of cookies is somewhat controversial because of privacy considerations [6]. Therefore, the AIDE/WN system permits the content provider to present the user with an invitation to see customized WN information through the use of cookies, and requires the

```
Rewrite Exec     /tracked/whatsnew.html     /home/aide/cgi-bin/whatsnew
Rewrite Exec     /tracked/?                 /home/aide/cgi-bin/record_visit
```

Table 1: Directives to rewrite URLs to invoke CGI scripts.

user to confirm that the use of cookies for this purpose is acceptable.

2.1 CGI Redirection

Cookies are useful only if a server-side program, such as a Common Gateway Interface (CGI) script, intercepts and interprets the cookie. Sending a cookie to access a static HTML file would be useful if the HTTP server did something with the cookie directly, but thus far HTTP servers appear to use cookies just to pass data to scripts. In the future I plan to modify the *Apache* server to handle cookies directly, but for now I use David Kristol's *htd* server with path redirection that turns all URLs into invocations of a CGI script. Currently, this is done by making accesses to

`http://aide.research.att.com/tracked/`

just like accesses to

`http://aide.research.att.com/`,

but redirected through a script. That script uses a backend database to record accesses with a particular cookie.

The redirection is accomplished via the directives in Table 1. The `whatsnew` script invokes a database query to list all pages viewed by the user, with the modification timestamp as of each access. Then it uses a brute-force technique to determine what has changed: it invokes the UNIX *stat* call to check each file at the time the WN page is created. (The list could be computed on a coarser granularity and cached, if and when performance becomes a bigger issue.)

The WN list can therefore be updated in real time to show exactly those pages that have changed since the user last saw them. In addition, other pages can refer to the WN list if and only if new pages exist. When one views the root of the `tracked` hierarchy, a comment `<!-- NEW_PAGES -->` is converted into an anchor for the WN list whenever new pages exist, with the text "There are *N* new pages." When no new pages exist, the comment has no effect. Figure 1 gives an example of the AIDE/WN "What's New" page.

The `record_visit` script updates the database each time a tracked page is accessed. This has the disadvantage of turning accesses to static files into CGI invocations, but (as discussed elsewhere) one could modify the HTTP server to handle this data directly. If the overhead of individual database updates is too high, the server could log accesses and update the database periodically to amortize the costs.

3 Historical Data

Having a list of which pages have changed is useful, but experience with AIDE suggests that in many cases it would be hard to determine *a priori* what the changes are if the only information is a single bit ("look over there!"). AIDE/WN uses the Revision Control System (RCS) to access past versions of each page and to make sure that the version a user is viewing is available in the future. It does the latter by checking in a new version if the current version is more recent than the most recent checked-in version. On the other hand, if the most recent version has already been checked in, AIDE/WN uses the comment provided at the last check-in time as a hint to the user about how the page may have changed.

Given the RCS history, AIDE/WN can present the user with a view of how a particular page has changed since it was last seen, or in fact how it has changed at any two points in time. AIDE/WN does this by checking each page for a comment

`<!-- SHOW_DIFFS -->`

and replacing this text with special HTML markup that inserts an anchor to permit one to invoke *Htmldiff* to see how the page has changed. An example appears in Figure 2(a), and the source to this example appears in Figure 2(b).

The revision history is modified slightly if the current version is not new. In that case, AIDE/WN displays a message about seeing how the page has changed since the previous version, but otherwise functions the same.

4 Issues

The current prototype is functional, but has some open issues.

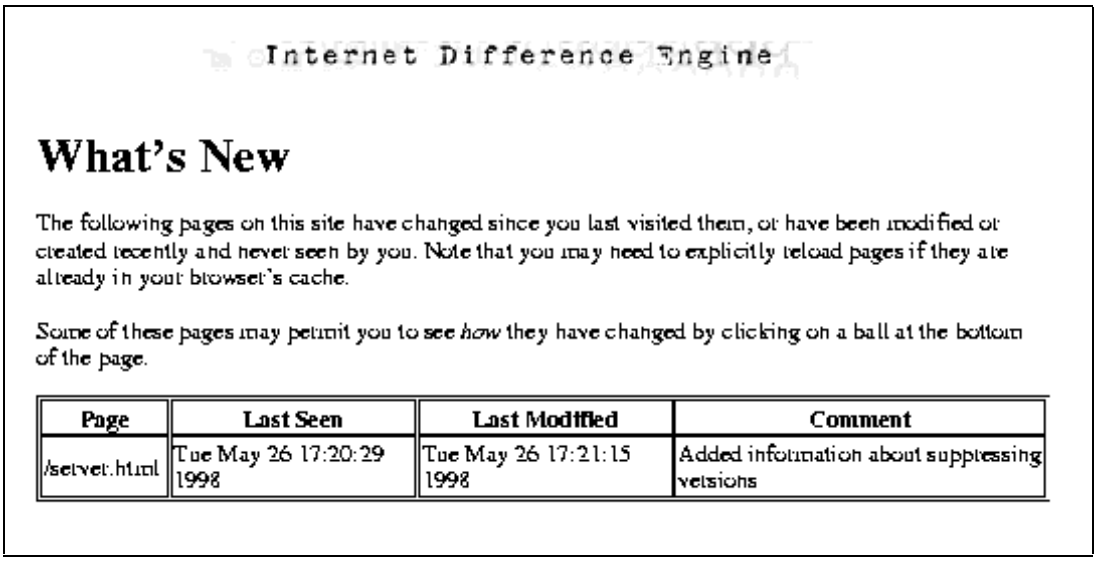


Figure 1: Customized list of what's new to a particular user, with RCS log information about the most recent change.

Contents of page....

- *You last saw this page at:* Fri, 01 May 1998 13:03:31 GMT.
  To see how this page changed since then, click on the ball.

(a) What the user sees.

```
<table> <tr> <td rowspan=2 align=center>
<FORM METHOD="POST" ACTION="/cgi-bin/rcsdiff" ENCTYPE=application/x-www-form-urlencoded>
<INPUT TYPE="hidden" NAME="diff" VALUE="diff">
<INPUT TYPE="hidden" NAME="file" VALUE="/examples/index.html">
<INPUT TYPE="hidden" NAME="current" VALUE="1">
<INPUT TYPE="hidden" NAME="1.23" VALUE="1">
<INPUT TYPE=image NAME="Submit" alt="submit!" VALUE="Submit"
src="http://www.research.att.com/icons/RedBall.gif" border=0>
</FORM> </td> <td>
<i>You last saw this page at:</i> Fri, 01 May 1998 13:03:31 GMT.
</td> </tr> <tr> <td> <i>To see how this page changed since then, click on the ball.
</td> </tr> </table>
```

(b) HTML source to the example, somewhat compressed for space.

Figure 2: An example of the information presented to the user when differences are offered.

4.1 Performance

Computation of the differences is time-consuming using the current implementation of *Htmldiff*.[1] The system caches *Htmldiff* output so that subsequent accesses are faster, but the first access can take many seconds or even minutes. Automatic computation of all possible differences (the current version compared against each old version that any user has seen most recently) is possible but not scalable.

The database operations to update the information about which user has seen which version, and to determine which pages have changed, are reasonably efficient but still not extremely fast. This means that dynamic computation of the number of new pages on each access to the root page might be undesirable, and instead that information could be calculated less frequently and cached. There are some systems (.e.g, [5]) that focus on caching dynamic data, invalidating it only when appropriate, which could apply here.

4.2 User Interface

The user interface is perhaps AIDE/WN's weakest component. Right now, the WN list consists of an HTML table with one line per changed file. Each line reports when the user last saw the page, when it was modified, and any available RCS comment. There is no notion of prioritization, where the most important recently changed pages could be listed first. In contrast, the regular AIDE system allows users to specify an arbitrary integer priority for each page, which is then used to sort the WN list, and a manually generated WN list would have arbitrary order and text as decided by a webmaster.

4.3 Caching Effects

By default, in HTTP/1.0 [2], dynamic data is generated with no `Last-Modified` header and is not cached. Since AIDE/WN is taking static files such as `index.html` and wrapping a CGI script around them to be able to track cookies, it could have the undesirable effect of making all pages on a site uncacheable. Instead, the CGI script explicitly generates a `Last-Modified` header based on the actual modification timestamp in the file system, just as the HTTP server normally does.

However, there is a catch. When a page actually changes, and appears in the WN list, visiting that page should ideally show the current version of the page, offer to display changes from the previous version, and remove the page from the WN list. In practice, since the page is considered cacheable, a browser that revalidates pages only when it first starts will display the older, cached version when one follows the anchor referring to the changed page. The user must then explicitly reload the page to see the latest version. It does not appear that HTTP/1.1 [1] is any less susceptible to these cache consistency issues.

[1] A new version of *Htmldiff*, using a much more efficient comparison algorithm, is under development.

4.4 Suppressing Versions

One catch to the automatic generation of WN lists is that they may catch exceedingly trivial changes, such as typographical errors. The regular (user-driven) version of AIDE suffers from the same problem and AIDE/WN could benefit from any techniques to ignore uninteresting changes.

5 Conclusions

AIDE/WN as it currently stands is a proof-of-concept. It demonstrates that the "What's New on this Site" page, which so many sites support, can be customized on a per-user basis in exchange for added computation and an imposition on users' privacy.

References

[1] T. Berners-Lee, R. Fielding, H. Frystyk, J. Gettys, P. Leach, L. Masinter, and J. Mogul. RFC 2616: Hypertext transfer protocol — HTTP/1.1, June 1999.

[2] T. Berners-Lee, R. Fielding, and H. Nielsen. RFC 1945: Hypertext transfer protocol — HTTP/1.0, May 1996.

[3] Yih-Farn Chen and Eleftherios Koutsofios. Website news: A website tracking and visualization service,. In *Poster Proceedings of WWW8*, Toronto, Ontario, Canada, May 1999.

[4] Fred Douglis, Thomas Ball, Yih-Farn Chen, and Eleftherios Koutsofios. The AT&T Internet Difference Engine: Tracking and viewing changes on the web. *World Wide Web*, pages 27–44, January 1998.

[5] Arun Iyengar and Jim Challenger. Improving web server performance by caching dynamic data. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 49–60, December 1997.

[6] D. Kristol and L. Montulli. RFC 2109: Http state management mechanism, February 1997.

Evolving an Engineered Web

David Eichmann

School of Library and Information Science / Dept. of Computer Science
University of Iowa
david-eichmann@uiowa.edu

1 – Introduction

Consider the World Wide Web as a single, collaboratively-generated artifact. While perhaps not a universally held assumption, for the majority of the general public there is little if any distinction between browser and server, between Internet as transport layer and the protocols existing upon that transport layer, between an animated GIF image and an applet rendering a video clip streaming in real time. There is only the Web.

2 – General Context

As software engineers interested in the evolution and adaptation of software systems over time in the face of evolving user requirements, we are presented with a system the likes of which has never been seen before – a single-instantiation, non-stop, fault-tolerant, incrementally-upgradable system with millions of users. Let's consider each of these characteristics:

- **Single-instantiation**: This is our core assumption in this paper. If there is a single Web (ignoring the possibilities of several distinct subgraphs comprising a partition of the Web), then we have 'system' that as a whole will never be supplanted with a replacement – only maintained over time.
- **Non-stop**: If the Web is the sum of all server and browsers installed on network-connected computers, then it is highly unlikely that at any given point in time there is no server in the world not serving some page to some browser for a user's edification/titillation.
- **Fault-tolerant**: The number of failure modes on the Web is awe-inspiring; from mean-time-to-failure issues on server hardware to Java bean component version mismatches, there is scarcely a way in which an automated system can fail that isn't exhibited on or by the Web. And yet, most people eventually get to what they seek.*
- **Incrementally-upgradable**: The Web is a dramatically open and modular system. There are a multitude of servers/browsers, a scant few used by the large majority of sites/users and the remainder in use by anywhere down to a single organization or individual. It is not uncommon (particularly in academic settings) for a single desktop system to have both Netscape and Internet Explorer installed, and frequently multiple versions of the same browser. Users download new browsers effectively at random, with virtually all fielded versions functioning on some machine on the network. Server/servlet configurations vary equally widely from site to site and content providers update pages sometimes on an hourly basis (consider a weather radar page).

If the Web is to evolve as a system in a controlled fashion, the scale of that control must be carefully chosen. It is out of the question to put the entire Web under configuration control or to require all users in the world to migrate to a specific version of any browser, let alone a specific browser. Instead, we will consider a number of dimensions of the Web as software system, addressing the evolutionary issues of each, with the intent of providing a basis for localized evolution. It is at this level that most of the literature regarding Web engineering exists [1, 3, 5, 7].

* Which is not to say that they get what they *want*… The Web is far too large and search engine technology just isn't that good yet.

3 – Content vs. Infrastructure

The assumption we make here is that the *infrastructure* components of the Web – servers, browsers, application development environments, etc. – have their own, more traditional life-cycle. While the domain may not yet be well understood, we still assume that established techniques for development and evolution apply. Just because a web server is a piece of web infrastructure doesn't alter the nature of C programming and test on a UNIX platform…

Instead, we focus upon the *content* of the Web – the pages, applets, servlets and other program – that the user perceives as a seamlessly connected network of textual/graphic structure. Hence, Apache and Communicator are out of scope, but an HTML page and its embedded applet are in scope.

The vocabulary of the Web is still evolving. For purposes of discussion, *site* will be used to describe a given machine/domain name, *page* will be used to describe a distinct document/artifact (usually corresponding to a file in a site's file system), and *Weblet* will be used to describe a set of interrelated pages developed and maintained by a single organization or individual. The definitional boundaries of a Weblet are fuzzy in the same way that the definitional boundary of any system are fuzzy, and are normally driven by application context.

4 – Two-Dimensional Web Evolution

We draw a distinction between the relatively incremental evolution of content *per se* and the more problematic evolution of the manner in which that content is expressed. This is a key evolutionary concern for the Web due to the extent to which expression issues are exposed to the users. Unlike more traditional applications, end users are commonly expected to adapt to changing expression formats on their own.

4.1 – Temporal Maintenance of Static Technology

Web sites have been well compared in complexity to software systems [10]. An integrated set of Web pages is as dependent on well-formed syntax as is any library of procedures or methods in a traditional programming language. However, much of the discussion of Web page creation and maintenance functions only on the appearance of a given page when rendered in a browser [7]. Little attention is paid to issues of comprehension, navigation or accessibility [8, 9, 11]. Common errors occurring in Web site development and maintenance include:

- **Grammatical errors**: simple misspecification of legal HTML tags. This problem was very common in the early days of the web, when most pages were hand-crafted.
- **HTML version mismatches**: more than just a simple grammatical error, this arises when a vendor or HTML version-specific construct is used with the presumption that all readers of the page employ a browser compatible with that of the author.
- **Link target misspecification**: few environments provide for automatic construction of a link in a document currently under maintenance. It is left to the maintainer to validate any new links.
- **Link rot**: a colorful phrase referring to the invalidation of an existing hyperlink due to the removal or relocation of the target page.
- **External content format mismatches**: In configurations where significant Weblet content resides in pages that are not renderable by the browser directly, format issues easily arise when compatible 'helper' applications such as Acrobat Reader have not been installed or are not available. This frequently is driven by technology migration issues.

4.2 – Technology Migration

Technology migration involves the systematic replacement of one manner of content expression with a (typically) newer manner of expression. While not necessarily an inherent problem, it is virtually inescapable on the Web due to the lack of infrastructure configuration management, which can only be characterized as chaotic. The whole notion of 'Web time' exemplifies the dramatic rate of change exhibited by the various components technologies on the Web. For a Weblet to maintain consistency and utility, all component pages employing a given level of technology (HTML versions, Java virtual machines and libraries, etc.) need to migrate to a new level atomically, similar in nature to a transaction commitment in a database system. Otherwise, the opportunity arises for a user navigating the Weblet to encounter (both forward and backward) version shifts that are incompatible with their infrastructure configuration.

This aspect of Web evolution is extremely sensitive to granularity. Clearly, a single page can be brought into level conformity relatively easily; all potential targets for modification / replacement are local to the page and presumably under the same set of authorization rights as is the page itself. This also holds true for small Weblets maintained by a single individual. Note, however, that the opportunity for mismatch immediately increases as we move from a single page to some number of pages. As Weblet responsibility moves from the individual to the group and to the organization as a whole, consistency issues are blocked by differing technology capabilities, organizational boundaries, even infrastructure availability. It is at this boundary between single site / small Weblet and multiple site / large Weblet that the greatest need for tool support and an engineering perspective exists.

5 – A Taxonomy of Content Generation

How content is generated can have a significant impact on the ease with which a Weblet can evolve. The current state of Web technology offers a broad spectrum of potential architectures:

- **Static content, generated and maintained with manual tools**: The earliest, and still most common means of managing content.
- **Static content dynamically wrapped with contextual material (including use of frames)**: This offers a systematic means of providing a common look-and-feel to a Weblet (shared navigation headers/footers, etc.), improving consistency and validity.
- **Static content driven from a version management system**: this supports rapid deployment / retraction of complete Weblets, avoiding most 'transactional' errors for users. This is basically orthogonal to the preceding scenarios.
- **Static content wrapping localized active content**: this includes both java script mechanisms and applets that do not communicate with their server
- **Static content wrapping interactive content**: extending applets with the ability to communicate with their server allows for Web 'structure' to adapt to changing server state, without alteration to the Web configuration itself.
- **Dynamic content**: this includes both CGI programs and servlets. Each has their own set of maintenance issues, but share with static content the potential for link rot relating to Web structure not generated within the local application. This architecture also exhibits all of the traditional problems of software system evolution, since application content is controlled completely programmatically.

Note that this list is really only a partial ordering of architectural variations, with the earlier bullets characterizing simpler systems and the later bullets characterizing more complex systems. This middle entries tend to be somewhat independent from one another.

6 – Towards a Web Process

Clearly, Web development can benefit from a recognition of process as important to the success of projects. Unfortunately, there is little in the way of engineering culture present and one of the few significant treatments of engineering issues for Web content recommends a modified waterfall process, while at the same time cautioning the reader regarding the lack of stable requirements and developer domain knowledge [7].

Workshops in Web engineering issues have begun to be held [3, 5], but the majority of papers address development and consistency issues (e.g., [6]) rather that evolution issues. Dart's work in configuration management [2] is a beginning, but does not address the granularity issues raised above.

7 – Conclusions

The Web is a fractal system – challenges apply equally well to a given Web site as they do to the Web as a whole. The issues discussed here operate at three distinct levels. At the macro level are very few tools – perhaps only the Web search engines, and they only address the Web as a whole as spider fodder. The scale of the Web matched against the rate of content change makes macro-level support effectively incomputable.

At the micro level of a Web page are a dazzling array of tools, most of which are rudimentary at best. Web authoring tools such as Netscape Communicator and Adobe PageMill provide a WYSIWYG means of generating content (a far cry indeed from the few early tools), but virtually no support for evolution beyond the mere editing of existing content.

At the meso level of Weblets are tools that basically are only capable of telling a maintainer what is broken, but are unable to effect repairs on their own. Solderitsch clearly identified this as a problem as early as 1996 in his call for CAWSE, Computer Aided Web Site Engineering, tools [10]. The opportunity for definition of a new genre of tools here is great.

8 – References

[1] Bieber, M., "Hypertext and Web Engineering," *Proc. 9th ACM Conf. on Hypertext and Hypermedia: Links, Objects, Time and Space–Structure in Hypermedia Systems*, 1998, pp. 277-278.

[2] Dart, S., "Containing the Web Crisis Using Configuration Management," *Proc. 1st ICSE Workshop on Web Engineering*, May 16-17, 1999, Los Angeles, CA.

[3] Eichmann, D., "ICSE'97 Workshop on Software Engineering (on) the World Wide Web," Proc. ICSE'97, pp. 676.

[4] McEneaney, J. E., "Visualizing and Assessing Navigation in Hypertext," *Proc. 10th ACM Conf. on Hypertext and Hypermedia: Returning to Our Diverse Roots*, 1999, pp. 61-70.

[5] Murugesan, S. and Y. Deshpande, "ICSE'99 Workshop on Web Engineering," *Proc. ICSE'99*, pp. 693-694.

[6] Olsina, L, D. Godoy, G. J. Lafuente and G. Rossi, "Specifying Quality Characteristics and Attributes for Websites," *Proc. 1st ICSE Workshop on Web Engineering*, May 16-17, 1999, Los Angeles, CA.

[7] Powell, T., *Web Site Engineering: Beyond Page Design*, Prentice-Hall, 1998.

[8] Shneiderman, B., "Reflections on Authoring, Editing, and Managing Hypertext," in *The Society of Text*, E. Barrett (ed.), MIT Press, Cambridge, MA, 1989.

[9] Shneiderman, B., "Designing information-abundant web sites: issues and recommendations," *Int. Journal of Human-Computer Studies*, v. 47, no. 1, 1997, pp. 5-29.

[10] Solderitsch, J., "Reuse and the World Wide Web," *Proc. of A NASA Focus on Software Reuse*, George Mason University, September 23-27, 1996, pp. 383-388.

[11] Tauscher, L. and S. Greenberg, "How people revisit web pages: empirical findings and implications for the design of history systems," *Int. Journal of Human-Computer Studies*, v. 47, no. 1, 1997, pp. 97-137.

Evolving the Web Into an Accessible Collaborative Learning Environment

Karl F. Hebenstreit, Jr.
US General Services Administration
Office of Governmentwide Policy
Center for Information Technology Accommodation

18th & F Streets NW, Room 1234
Washington, DC 20405
Voice: 202-501-0004
E-mail: karl.hebenstreit@gsa.gov
Website: http://www.itpolicy.gsa.gov/cita

This paper will provide a high-level overview of Project PINNACLE as a broad conceptual framework for including the needs of people with disabilities in our discussion of web site evolution, then conclude with some observations about the strategic directions needed for incorporating accessibility into the future web infrastructure, with references to web sites for mentioned activities.

Project PINNACLE [1], an initiative of the US General Services Administration's Center for Information Technology Accommodation, has evolved over the past year to address development of the technological infrastructure needed for the emerging Knowledge Age. Project PINNACLE provides the encompassing framework needed for a systems engineering approach to address the component issues:

- "Public Information Networks" refer to web sites;
- "Accessible" refers to ensuring that all people can fully participate in shaping this infrastructure;
- "Collaborative" refers to environments that enable and promote communication among people through networked computers;
- "Learning" refers to those online environments required to realize lifelong learning goals;
- "Environments" refer to the organizational development efforts to align enabling technologies with appropriate human resource policies resulting in the overall organizational culture.

Project PINNACLE has been greatly influenced by the seminal work of Douglas Engelbart, whose 1962 study provided a broad framework for viewing the computer as the most sophisticated tool mankind has developed as a means to augment a person's basic capabilities [2] . His research identified four classes of these augmentation means -- artifacts, language, methodology, and training -- which he recognized as being organized as a hierarchical system: the Human, using Artifacts, Language, and Methodology, in which they are Trained (the H/LAM-T system). Within this system, humans and artifacts (software and hardware tools) "*comprise the only physical components of the H/LAM-T system. It is upon their combined capabilities that the ultimate capability of the system will depend... Exchange across this 'interface' occurs when an explicit-human process is coupled to an explicit-artifact process*." The implications of this H/LAM-T system for accessibility stems from the realization that accessibility issues are essentially contained within the realm of user interface design. The major barrier that has arisen for people with disabilities with currently implemented technologies is that the predominant (but implicit) assumption guiding design has been that there is a uniform set of explicitly-human capabilities -- no disabilities, or that everyone has full visual, auditory, motor, and cognitive capabilities -- for which software and hardware tools can be developed. To correct this inaccurate assumption, universal design principles [3] have been developed which essentially state that no function should require the ***exclusive*** use of a particular (sensory or motor dexterity) capability. In other words, human interface should be designed to take full advantage of each person's capabilities while extending the tool's capabilities as necessary. For software developers, the overarching principle is to implement a multimodal design whereby all major features can be performed in a variety of ways.

Web Accessibility

The first two PINNACLE components are being addressed extensively through the work of the World Wide Web Consortium's Web Accessibility Initiative (WAI) [4]. This effort has developed a five-part strategy: 1) ensuring that Web technologies support accessibility; 2) developing guidelines for accessibility; 3) developing tools to evaluate & facilitate accessibility; 4) conducting education and outreach; and 5) coordinating with research and development. The guidelines activity is subsequently comprised of three-parts: Web Content,

Authoring Tools, and User Agents. The **Web Content Accessibility Guidelines** are intended for webmasters and content developers; **Authoring Tools Accessibility Guidelines** are intended for the developers of HTML editors, document conversion tools, and other tools that generate Web content from databases; and **User Agents Accessibility Guidelines** are intended for developers of software that accesses Web content, examples including graphical, text, and voice browsers, multimedia players, and some software assistive technologies used in conjunction with browsers such as screen readers, screen magnifiers, and voice recognition software. One of the best aspects of this work is the recognition that the guidelines fall into three categories --Priority 1 denotes those issues that must be met or some people will not be able to access the content; Priority 2 denotes those issues that must be met or some disability groups will be at a disadvantage (such as taking substantially longer or expending considerably more effort to complete a task); and Priority 3 provides a means for recognizing best practices or techniques with respect to accessibility.

Other activities in the accessibility area include the IEEE Internet Best Practice standards working group [5], ANSI/HFES 200 standard [6], and the NCITS Information Technology Accommodation Study Group. This study group is determining the feasibility of an Alternative Interface Interaction Protocol (AIIP) [7] that would enable communication among multiple technologies working together to provide human-computer interfaces. One of the driving factors influencing this dynamic area is the passage of the Rehablitation Act Amendents of 1998 (PL 105-220), which mandate that electronic and information technology acquired by the federal government must be accessible for people with disabilities, with standards integrated into the Federal Acquistion Regulations and enforcement by August 7, 2000 (more information is available at reference [1]).

Collaborative Environments

While the development of personal environments that can enhance each person's information-handling capabilities are important, the ability to leverage these enhanced capabilities for groups of people working together is the logical extension. The increasing importance of groupware cannot be overemphasized. While there are many definitions of groupware [8], the best groupware environments -- whether real-time (synchronous) or anytime (asynchronous) -- provide a shared virtual environment that facilitates team communication and working together.

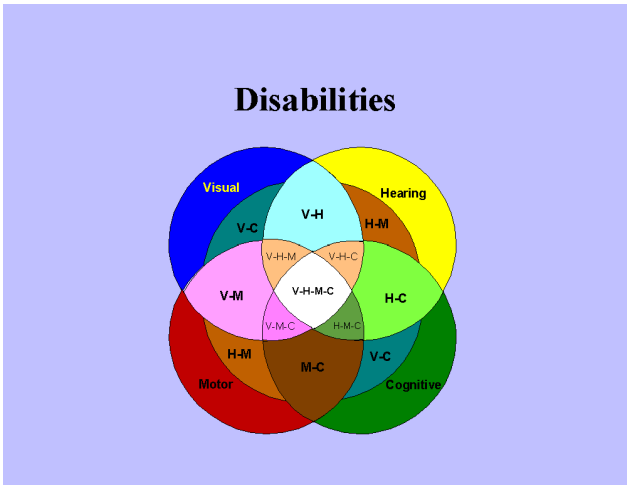
These shared environments add another level of complexity to the human-computer interface, because the groupware interface must also provide support for human-human collaboration. Each member in a group has their own personal interface with a computer, plus interface considerations for fostering interpersonal interactions. Properly implemented, groupware builds a bridge between humans by making the networked computer to networked computer interactions as transparent as possible. The goal for collaborative environments is that an organization with all members having a disability should be completely indistinguishable from one with no disabled employees from the customer perspective.

Learning Environments

Early attempts at computer-based education too often automated traditional practices such as fill-in-the-blank and multiple choice quizzes. These attempts did not take advantage of the computer as a new medium, which Roger Schank expresses as: "*[making] exciting learning environments. And you can do that very well without a computer, but computers offer the possibility for revolutionary change... Is this possible? Absolutely, but computers are part of the fix. They're part of the fix because right now it's really not possible to have individualized, one-on-one, learn-by-doing situations. The kind that we who have studied the mind know is the right way*. "[9] In addition to individual learning environments, computers also offer the capability of supporting collaborative learning. As articulated by the Lotus Institute, "*The **Learning Team Centered** approach creates an environment in which knowledge emerges and is shared through the collaboration of individuals within learning teams. An assumption of this model is that changes in mental models and*

behaviors occur most successfully through a Learning Team Centered approach. In a learning team, expertise and prior knowledge are explicitly incorporated into the knowledge transfer process with the creation of new knowledge as the result. The Learning Team Centered approach is most effective in[10]

In developing online learning environments, it is critical that the education and software engineering communities work together to not only consider requirements for pedagogically soundness, but also address the needs of people with disabilities. Too many of our current courseware environments require exclusive use of visual and/or high degrees of manual dexterity that at best leave people with these disabilities at a disadvantage, or at worst preclude their participation is these educational opportunities.



[Organizational] Environments

The key to successful implementation of these groupware and learning environments is often the alignment and coordination of human resource policies and technological infrastructures to mutually support each other. Working in these environments is quite different, leading to widespread changes in the organizational culture. While somewhat out of the original scope of focusing on web accessibility, the PINNACLE efforts (particularly Douglas Engelbart's Bootstrap Alliance and accessibility) has identified the need to explicitly focus on human-centered computing.

Too much of our discussions are focused solely on the technology, rather than on using technology to enable and improve communication among people. While the goal to enable each person to be independent to the greatest degree possible is desirable, our discussions also focus almost exclusively on individuals, although the increasing complexity we are faced with in addressing societal issues is demanding greater collaboration among people.

Another area that needs more focus is encouraging and enabling communications among people with disabilities. For example, identifying four disabilities -- visual, hearing, cognitive, and motor -- would lead to 15 (2 raised to the 4th power, minus 1) possible combinations. By the way, this would best be presented to visually-oriented people in a Venn diagram with four overlapping circles (Figure 1). These 15 combinations would include the 4 disability groups by themselves, 6 pairs, 4 groups of three, and one of all four (which is "Universal Design"). Given the predominant reductionist perspective, most of our attention has been focused on each of the four disability areas, with recent attention given to addressing all four areas simultaneously ("Universal Design"), although many advocates within the cognitive disability area would argue that their needs have not been adequately addressed to date.

Strategic Directions

WAI Guidelines Framework: Employing a systems engineering perspective would lead to a framework for depicting the relationships among the Web Content, Authoring Tools, and User Agent guidelines and how they work together to provide the overall web experience for each person.

Enabling direct communication among people with disabilities: This would be one of the most strategic ways to address accessibility issues. Assistive technology has progressed to the point where we can enable a blind person and deaf person to communicate directly (a blind person using a TTY software package with a screen reader). To what degree can the other pairs be addressed -- visual and cognitive, visual and motor, hearing and motor, hearing and cognitive, cognitive and motor?

Connectivity Models: To date, there has been a widespread (but implicit) assumption that there will be constant, always reliable connectivity with the Internet - the ***"Always Connected"*** model. While this might eventually be achieved, more (explicit) attention needs to be given to the alternative ***"Connect as Necessary"*** model which would extend the 3-tier client/server architecture developed for corporate internal LANs to the Web. The result would be that a person could be self-reliant within their local environment to the greatest extent possible, only connecting to distributed resources over the web when necessary.

Ubiquitous Distributed Learning Environments: In a keynote speech at Lotusphere '99, Nobel Laureate Gary S. Becker discussed the economic implications of extending the United States postsecondary education model to other countries, noting that this would require building 14,000 new universities in India and 20,000 new universities in China! His conclusion was that the emerging Knowledge Age would require widespread use of distributed [web-based] education. [11] It is imperative that these distributed learning environments be accessible for people with disabilities.

Organizational Development: Two ideas that warrant further attention in guiding web site evolution are **COncurrent Development, Integration and Application of Knowledge (CoDIAK)** [12], as articulated by Douglas Engelbart, will be a core strategic capability for Knowledge-based organizations, and the Appreciation, Influence and Control process model developed by William E. Smith [13].

Long-Range R&D: Two prominent government reports that are guiding the long-range research and development efforts to address the concerns raised in this paper are the National Research Council's More Than Screen Deep: Toward Every-Citizen Interfaces to the Nation's Information Infrastructure [14]and the recent President's Information Technology Advisory Committee (PITAC) report to the President, Information Technology Research: Investing in Our Future. [15]

References

[1] PINNACLE (**P**ublic **I**nformation **N**etworks' **N**eed for **A**ccessible **C**ollaborative **L**earning **E**nvironments) http://w3.gsa.gov/web/m/cita.nsf/Portals/PINNACLE

[2] Engelbart, Douglas C. "Augmenting Human Intellect: A Conceptual Framework", 1962: http://www.histech.rwth-aachen.de/www/quellen/engelbart/ahi62index.html

[3] Universal Design principles: http://trace.wisc.edu/world/gen_ud.html

[4] W3C Web Accessibility Initiative: http://www.w3.org/WAI/

[5] IEEE Internet Best Practice standards working group: http://computer.org/standard/Internet/

[6] ANSI/HFES 200 Standard: http://www.ansi.org http://www.hfes.org

[7] NCITS Information Technology Accommodation Study Group, which is determining the feasibility of an Alternative Interface Interaction Protocol (AIIP): http://www.uniac.com/

[8] Baecker, Ronald. Readings in Computer-Supported Cooperative Work: Assisting Human-Human Collaboration. San Mateo, CA: Morgan-Kaufmann Publishers, 1991, p. 1.
http://w3.gsa.gov/web/m/cita.nsf/Books/1558602410
[9] Roger Schank, Introduction to Engines for Education: http://www.ils.nwu.edu/~e_for_e/nodes/I-M-INTRO-ZOOMER-pg.html
[10] Lotus Institute white paper, "Distributed Learning: Approaches, Technologies and Solutions. "
Http://www.lotus.com/home.nsf/welcome/institute
[11] UNext web site: http //www.unext.com/
[12] CoDIAK Capability: http://www.bootstrap.org/augment-132811.htm#6
[13] AIC: The Process -- http://www.odii.com/aic-process.html
[14] National Research Council, More Than Screen Deep: Toward Every-Citizen Interfaces to the Nation's Information Infrastructure: http://www.nap.edu/readingroom/books/screen/
[15] PITAC Report: http://www.hpcc.gov/ac/report/

WEB SITE EVOLUTION: DESIGNING AND DEVELOPING FOR THE FUTURE

Lisa Schmeiser
schmeiser.com

There are three areas that website developers should focus on before they begin building, overhauling, or repairing a website. In the following pages, those areas and specific focal areas are outlined for discussion.

PROJECT MANAGEMENT

Developing a site requires canny strategy and intelligent implementation, but you need to step back a level and look at the project that's calling for that strategy and implementation. Therefore, your first focus should be on project-management, or the practice of running the website-remodeling endeavor you're about to start. The three tenets below explain how to think like a smart website maintainer.

Always have a long-term plan

One of the unfortunate side-effects of working in Net Time is that many web development teams find themselves working in reactive mode rather than proactive mode. They build sites as an effort to address the features or mission a competitor has recently debuted. As a result, the majority of planning time and effort go toward shaping the site as a response to another site instead of developing a site that strives to represent its creators and serve its audience in the best possible capacity.

Instead of playing on the defensive, website developers should adopt a different approach to their site's ongoing improvement: working on building their site at their pace. Users are rarely pitting one site against another: the Web is too big for that kind of competition. Instead, they're looking for sites that perform distinct tasks better than similar sites. Most users are intelligent to know that the first site to offer a new feature may not be the best executor of that feature, so you should remove the false pressure of competitor launches from your development cycle.

Avoid imitating others in an effort to get ahead. What your team should be doing is answering the following questions:

- What does our site already do well? What does it do better than any related sites? How does it do this?
- What do our users associate with our site? Why do they come visit us?
- What do we want our site to do better? Why?
- How can we make this improved function the best example out there?

It's all right to go to your competitor's sites and check them out; doing comparative research will often help you hone your own sense of how you want your website to work. But don't let your competition set the pace. Ultimately, it doesn't matter who's first out of the gate with a new website. What counts is who does the best job.

Weigh strategy and implementation equally

I recently excused myself from a project that anticipated 90 days for business strategy and only 30 days to build the site. My rationale? My job entails figuring out how to translate business objectives and design specs into bug-free code and clickable sites. If the realization of the project wasn't going to be given the same priority as the planning, chances were high that the implementation and maintenance of the site were going to be a pressure-packed nightmare.

This is another area where the misperception that websites spring into creation overnight really hurts the sites in the long run. A good website is more than a viable idea and a great interface; it includes a well-thought-out backend structure, a code library that anticipates future expansion, and a rigorous testing regime before going public. When you skimp on the technical development time, you skimp on the time frame for finding elegant and flexible implementations for the strategic goals you've set up.

To prevent the phenomena of spec'ing castles in the air and allowing no time for building, retool your strategy process: for every strategic goal your team comes up with, attach a timeline and resource assessment to meeting that goal. It ruins some of the fun, but it gives so-called "strategists" a clue as to what it takes to fulfill their goals. The result translates into either a scaled-down project or more realistic deadlines – either of which is good news for the implementation team.

Apply a fine filter

A fine filter is a polite term for one that catches a lot of hogwash, or other farm animal byproducts of your choosing. Because of the rapid-fire pace in website development, there's a real temptation to let software and hardware vendors forecast the trends for you and provide solutions to problems you didn't even know you had. Avoid the temptation, and subject your technology to a rigorous screening. Before incorporating any new web-based technology – be it markup, software, or hardware – ask yourself the following:

- Will this solution save me time?
- Does it fill a need I already have?
- Does it fill a need I knew I had before hearing about this product/technique?
- Will this solution save me resources?
- Will this solution make my life easier in the long run?
- Is this a technology that works well with other, already-implemented technologies?
- Will this technology be time-intensive to extract from my system if I decide to replace is?

If the answer to any of these questions is "no," reconsider the benefits you'd reap from the latest, greatest technology trend. The last question you should ask yourself is: would it be easier or more effective for me to build the solution myself? If the answer is yes, give the idea serious consideration.

STRATEGY

This is the stage in website development where you analyze what's working on your site, what's not working, and what you plan to do to build a better version of the product you already have. This stage sets the priorities and goals for the implementation stage, so it's important to keep a clear focus on the reasons you're upgrading or maintaining your site.

Don't forget why you're building the site

If you cannot clearly articulate what your site does and how it does it, you're in trouble. You're in trouble because you've forgotten why you're paying attention to the site. Finally, you're in trouble because you're probably putting a lot of time and brainpower into something that's not even remotely related to the site you have or the one you want to build.

Know what the site's goals are. Articulate them clearly, as calls to action. It will probably help your team if you separate your company's goals for the site from your goals for the user experience on the site. Your user could care less if you're hoping to make an extra $10 million per quarter; they care about buying what they want, when they want it. Conversely, the nice folks in accounting aren't losing sleep over the placement of a navigation bar; they want that extra revenue. Your job as a strategist is to figure out how one type of site goal can fulfill the other.

Reduce, reuse, recycle

Your goal in upgrading, expanding, or repairing the site should be to do as little as possible. Keeping this in mind, if you're planning to revamp or expand an existing website, always ask yourself the question, "what do we do now that already works?"

Too many sites don't take inventory of their assets and think of ways to reuse them in later versions. As a result, they throw out the qualities or functions that their users wanted. Alternately, sites don't take advantage of whatever already works, so the development teams spend valuable time reinventing the wheel with every release of the site. Or – and this is the most common scenario – the development team takes the

"reduce, reuse, recycle" adage only halfway and ends up smashing a new front end onto an existing website, thus making for a very ugly and functionally confusing user experience.

Therefore, team members involved in website strategy meetings should always ask: what do we have that we can use it again, and how can we seamlessly integrate it into the new site?

IMPLEMENTATION

This last stage reifies all the ideas and goals generated in the strategy stage. In addition to converting all the strategy goals from bullet points to a tangible website, this stage is also vulnerable to introducing functional errors or legacy issues through deadline crunches or mismatches between site goals and technology used to implement them. Therefore, it's in your best interests to build intelligently and balance short-term deadlines against long-term resource expenditures and benefits. Surprisingly, the best implementation steps you can take are not directly related to the technology, but to the people who will be using that technology.

Don't be afraid to say no

The last large-scale project I worked on called for us to patch a content server to an application server; the application server would then patch into an Oracle database, which was itself receiving periodic data dumps from a mainframe. The links in this chain of servers and data repositories were a series of Java beans meant to act as containers for data and translators for queries up and down the backend chain. The team in charge of the Java beans was given a ridiculously tight deadline; the leader assured us he'd meet the deadline.

You can guess what happened next: burdened by the lack of time and too much to do, his team turned in most of its work late *and* untested. As a result, work in the production environment crashed constantly; diagnosing the problems and fixing them took twice as long because we couldn't tell if the problem was in the Java beans or the separate parts they were supposed to link. As a result, the site launched four weeks later than planned. If the team leader had asked for an extra week – or demanded it – and backed up his argument with a likely rush scenario, the site could have been only a week over deadline.

The moral of the story: always weigh deadline demands against the steps you know you need to do in order to do the job right the first time. Be sure to factor in time for testing – both drafting the test plan and executing it – and pad the schedule for mishaps. I know this is counter-intuitive in the fast-paced, high-intensity development environment we all think is normal, but it's better to plan for the worst and produce the best possible product than it is to plan for the best and scramble like mad when the unanticipated happens.

Remember the human factor

The most critical factor in the implementation stage is a project's personnel. The greatest intangible resources and assets exist in the team members' heads – find ways to get that knowledge out of their heads and into the code.

Documentation is one good way; establishing and maintaining clear lines of communication is another. Strategy phases tend to be very well-documented, but the deadline rush and detail-oriented work of implementation often squeeze out meeting notes, email summaries, or how-tos detailing the considerations and solutions leading to a successful technical implementation.

Take the time to establish and maintain these channels, and to encourage their use among team members. The shared knowledge is a time-saver in future iterations. It prevents redundant work and alerts you to production pitfalls ahead of time.

SUMMARY

The focuses described above may seem either obvious or counter-intuitive to the working practices you think lead to successful websites, but they address a basic dilemma facing everyone in web development: how to maximize previous effort and learn from mistakes while still maintaining a competitive development pace. The solution lays in addressing *how* people work first.

Web Site Evolution - Towards a Flexible Integration of Data and its Representation

Margaret-Anne Storey and Jens H. Jahnke
Department of Computer Science, University of Victoria, Canada

Abstract

This position paper discusses the inherent difficulties caused by data-driven as well as user-interface-driven evolution of Web sites. New data requirements arise in order to accommodate new data sources, or to delete and change existing data. User interfaces are constantly evolving to suit new customer requirements or to take advantages of new technologies. Maintaining the integrity and security of the data is of utmost importance, but the aesthetics and usability of the site are equally important. This is especially true in the eCommerce arena. Users with non-technical backgrounds are doing business using the Web as a communication medium. Furthermore the Web itself has allowed the integration of data sources not previously possible. These issues are particularly pertinent for library Web sites. In our position paper, we evaluate current technologies and identify the lack of separation between user interface and data structure concerns as a major cause for evolution and maintenance problems in web site design.

1. Background

Currently, the World Wide Web is dramatically changing how information is gathered and distributed by institutions, companies, and consumers. During the past four years, the number of Web sites has increased from about 23,500 (June 1995) to 7,078,194 (August 1999) [1]. A major driving force behind this development has been the increasing popularity of Web-based services in the private sector. The Web, with its user friendly and less intimidating interface, has been broadly accepted by the public. In contrast to other Internet-based information services like *ftp* and *gopher*, the Web was specifically developed for browsing complex multi-site information networks using graphical visualization and simple point&click navigation techniques. This paradigm has enabled even inexperienced users to explore this new medium without having to learn a textual command language.

The success of a Web site is due not only to the usefulness of the data it represents but also to the accessibility of the information by the average user. Web sites evolve to accommodate *changes in the data* represented by the Web site or to accommodate modifications in how this data is accessed, i.e., *changes to the user interface*. The following section describes these two aspects of Web site evolution in more detail. Section 3 discusses several techniques for integrating data and its representation in information based Web sites. The remainder of the paper focuses on a specific application domain of Web-based information services, electronic library gateways. Section 4 gives an introduction to the central issues in this area, whereas Section 5 describes a specific project with the McPherson library of the University of Victoria.

2. Web Site Evolution - two sides of the coin

This section describes some of the challenges resulting from data driven and user interface driven evolution.

2.1 Data evolution

The necessity to modify the structure of data maintained in information systems causes severe update problems in today's software industry. Prominent examples of this phenomenon include the Y2K-Problem [2] and the Euro-conversion [3]. We can transfer many lessons learned in the development of traditional Information Systems (IS) to the domain of net-centric information systems. One important lesson to be learned from IS research is the requirement to separate data from its representation and from the *business logic* which processes the data. The failure to separate the data from the business logic in legacy software systems developed in the 60's and 70's resulted in complex reengineering problems. The reason for these problems is that the data, business logic, and external representations are highly dependent on one another which implies complicated updates in the case of a change in the data structures (meta data), business rules or user interface. The introduction of standardized file access systems (e.g., SAM, ISAM) and *database management systems* (DBMS) contributed to a decoupling of this tight integration in IS. The middleware and view mechanisms of modern DBMS allows developers to hide many details about the internal representation of a logical data structure from the rest of the application code. As a benefit, changes caused by data evolution can be performed more locally.

We expect that similar techniques will prove beneficial for maintaining complex, data-intensive Web sites. However, there are additional challenging problems which need to be

solved. For example, with net-centric IS, an important goal is to develop technologies that facilitate the integration of several distributed and heterogeneous data sources. Today, an increasing number of successful Web sites maintain only a small amount of original data content but provide categorized and annotated pointers to other data sources.

Given the predicted popularity of eCommerce, Web based information systems, such as airline-independent best-price flight reservation systems and electronic library gateways, will gain even more importance in the future. Such information systems need to be flexible as they evolve rapidly due to the integration of additional data sources and to changes in the user interfaces. Some central issues to consider are: how to harmonize different heterogeneous data structures; how to deal with overlapping data; and how to enforce consistency constraints (e.g., transaction management). Similar issues have been encountered for several years in the domain of distributed and federated DBMS [11]. However, since this research is based on assumptions of a relatively small and fixed number of participating data sources, the results have to be reevaluated with respect to the scale and dynamics of Web-based information management.

2.2 User interface evolution

The user interface of a Web site may be viewed as an external representation of the data or information contained in a site. Users need to interact with Web sites either to find specific information or to browse and explore for more knowledge about a particular subject. In addition, users often have to input data (particularly in eCommerce applications) such as their address or product ordering information. In essence, Web site user interface design is as complex a task as application user interface design. Indeed the user interface of many applications are currently slowly migrating to Web based interfaces.

User interface design is without doubt a challenging problem. A good design is usually the result of several iterations. One of the problems with a Web based information system is that the pressure to evolve is persistent as users demand updated or new information as soon as or even before it becomes available. More sophisticated users are also requesting all the bells and whistles that are provided by the latest browsers and software tools. In contrast to this, more and more non-technical users are seeking simple, easy-to-learn and intuitive interfaces. Novices do not want to download and install plug-ins and they are often not aware that there is yet another version of the default browser. These conflicting styles of usage lead to difficulties when making design decisions during site design.

In the next section, we review several existing technologies for building Web based information systems and discuss the impact of these techniques on data evolution and user interface evolution.

3. Techniques to integrate data and its presentation on the Web

In 1992, Connolly introduced the first specification of the *Hyper Text Markup language* (HTML) as a means to describe Web pages. Although several extensions of HTML have been proposed and implemented (e.g., forms, images and frames), the basic approach has not changed in principle: data is described as text enclosed by formatting directives (HTML tags). While this solution has proven sufficient for static information such as advertisements and contact information for companies and organizations, it is hardly viable to maintain complex information that is updated frequently.

The *Common Gateway Interface* (CGI) has been introduced to process and represent more dynamic information contents. It allows the developer to embed programs or scripts in HTML pages. These programs are written in various languages and permit the storage and retrieval of data from a Web server. This technique is widely used in combination with HTML forms to access data in files or databases from the Web. However, with respect to maintenance and evolution issues for complex Web information systems, the CGI approach causes significant update problems. This is because CGI scripts, which serve as mediators between the data and their HTML representation, are *"hard-coded"* at a low level of abstraction. Each change in the meta-data or in the external HTML interface requires a change to these scripts. This means a significant update and subsequent testing overhead for complex applications such as large product catalogs.

An attempt to raise the level of abstraction for the coupling between data and representations are dedicated HTML-database gateways offered by multiple DBMS and third party vendors [4]. These solutions enable developers to embed database queries directly into HTML documents. Still, the queries and the HTML form have to be updated when the meta data evolves. Another approach which provides better support for data evolution is HTML generators for databases, e.g., Ardent Software's *O2Web*. [5], which support the automatic generation of HTML representations for data objects without any additional programming. Each object has a generic HTML presentation. Programmers can customize these HTML production methods by overloading the system-supplied methods. The most important disadvantage of such solutions is that they are targeted to

a specific DBMS platform and do not support the integration of heterogeneous data sources.

Recently, the *Extensible Markup Language* (XML) [6] has been proposed as a successor to HTML. XML is in fact a superset of HTML. A key feature of XML is that the semantics of the tags used are not predefined (as in HTML). The XML document defines the (data) structure of a Web page while its external representation is defined separately by another specification written in the *Extensible Style Language* (XSL) [6]. XML browsers, if used broadly as a substitution for HTML-based systems, would enable the user interface design to be more independent from the actual data content.

Another platform independent approach which enables access to heterogeneous data sources employs *Java* applets in combination with the *Java DabataBase Connectivity* (JDBC) [7]. Currently this solution is rarely used for accessing data at Web sites. It is likely to become increasingly popular because it allows the Web server to distribute data-oriented computations among its clients. But a remaining problem is that the client software has to be updated whenever the applet changes. If multiple data sources have to be accessed, maintainability can be improved by employing object-oriented integration services like CORBA and COM [8]. Such services facilitate encapsulation of distributed data sources and enhance their robustness against modifications. These techniques are slowly being adopted.

4. Electronic libraries

The vast variety of services offered through the Web makes it hard to find an approach to tackle the problem of Web site evolution in general. Therefore, we are initially restricting our research to a specific application domain, *electronic libraries*. Today, an increasing number of libraries employ computers to maintain an electronic database of their catalogs. Traditionally, search and order services have been provided by dedicated information terminals provided in the libraries. Recently, many libraries have started to migrate these services to the Web. This migration has several benefits:

- Users can access the library 24 hours a day from their offices or homes. This improves the accessibility of catalog information and reduces the effort required to maintain numerous information terminals.
- Many new library users have previous experiences with other Web-based services. This means they can search and browse the library with little training;
- Catalogs of remote libraries and bibliographic services can be integrated within the library gateway.

This last point is probably the most important benefit of Web-based library services although significant interoperability and evolution problems arise with the integration of different library catalogs. Because of these problems most current electronic libraries provide only superficial integration: they offer similar user interfaces for different data sources but rely almost entirely on human intelligence to provide coherence for the content. Deep data integration of library data has been identified as a "*grand challenge*" research problem because of the heterogeneous and evolutionary nature of the different data sources [9]. Still, this deep data integration is of crucial importance, if we are to tackle open issues like overlapping and complementary data and the presentation of uniform and consistent query results to the users. Most current research efforts are located in between these two extremes. They aim to establish a standardized protocol for library database integration and information retrieval. Since 1995, ANSI/NISO has developed the *Information Retrieval Standard* Z39.50 which has been accepted by ISO in late 1996 [10]. Lunau and Turner of the National Library of Canada summarize experiences with this new standard [10]. They report that in principle, the uniform interface works well as a data integration platform. Still, they point out that the key issue which remains is how to create and maintain a valid mapping between the data objects and attributes of the individual databases and the uniform access layer.

5. Case study: McPherson Library Gateway

The McPherson library is a significant library used by thousands of students and researchers at the University of Victoria. The library has a well established Web presence (called the McPherson Library Gateway). The Gateway is used for accessing information about the library's physical collections and provides access to electronic indices and texts. Despite their best efforts, the McPherson library staff are overwhelmed by the amount of electronic information that is now available. Currently, the library includes over 60 different indices for articles. They have realized that the current Web site architecture is neither scalable nor flexible enough to suit these rapid changes and needs to be updated.

To solve this problem, they bought a number of dedicated Z39.50 middleware products to integrate the different data sources. We are collaborating with the library management to offer our expertise in data evolution and user interface evolution during the reengineering of their Web site architecture. Some of the issues we will focus on are described in the following paragraphs.

Scalability: The current architecture of the Gateway Web site cannot scale to handle the increasing number of

students using the service. In addition, it is becoming increasingly cumbersome to integrate electronic catalogs, journals and texts that become available almost on a daily basis. The architecture needs to support easy insertion and deletion of data resources as well as provide a consistent set of user interfaces to these new resources. Peak usage times should be considered, for example, just before midterm exams. Furthermore, the architecture should be able to support multiple site maintainers working concurrently.

Flexibility: The design needs to be flexible so that data resources and information needs, not currently envisioned, can be integrated in the future. For example, the current architecture does not allow users to concurrently search through video and book collections. Possible future changes may allow users to search through musical resources or even digital videos online. The current library card system is very homogeneous with respect to the varied nature and granularity of the resources it represents. Each card has a set number of fields and works fairly well for written material. The future architecture will need to provide access to more heterogeneous data resources and should provide access to information and their components at varying levels of granularity.

Consistency: The Web site will need to provide a consistent set of user interfaces to many different resources. In addition, many of the users will have had experience or will quickly gain experience as they use or link to other library Web sites. Consistency across multiple libraries is especially important.

Usability: The Gateway Web site will become in essence a virtual library. Although it will not be housed in a concrete building, it is still important for the patrons to achieve a sense of presence as they browse the site. Such characteristics will improve the usability of the site. The user interface needs to be suitable for many different user profiles, such as students, researchers, teachers, graduate students and members of the public. The characteristics for each of these user groups will need to be collected and continually updated as the sophistication of each of these user groups continues to evolve.

Information retrieval research shows that there are two main styles of navigation: searching and browsing,. Users frequently switch between these two activities. While browsing, many users don't always know what they are looking for, or may not know the correct label to use during a search. The site should impose a structure on the information so that the library patrons can create their own personal paths to the information they require. Users may also want to create customized views of the library for their own purposes.

Maintainability: The architecture design needs to be such that it would be relatively easy to add, update or delete heterogeneous data resources. Maintainers will have the task of deciding which catalogs and electronic journals/text library to add or remove from the site. In addition, they will also have to deal with the ambiguity in cataloging items as they become available. However, certain business processes may be simplified such as book recalls, holds and fines for overdue articles. Other concerns are training issues for the site maintainers and the library patrons

To date we have begun preliminary discussions with the library management. They are enthusiastic about our involvement, as their task is a very challenging one. We hope to be able to suggest solutions that will ensure that the new architecture for their Web site will be suitable for many years. Although we will be considering a library information system, the results should be generalizable to other applications.

References

[1] M. Gray. Growth and Usage of the Web and the Internet. Massachussets Institute of Technology, 1999. Http://www.mit.edu/people/mkgray/net.

[2] R. A. Martin. Dealing with dates: Solutions for the Year 2000. *Computer*, 30(3):44-51, 1997.

[3] K. Grotenhuis. Crossing the Euro rubicon. *IEEE Spectrum*, 35(10):20-33, 1998.

[4] G. Ehmayer and G. Kappel and S. Reich. Connecting Databases to the Web - A Taxonomy of Gateways. Proc. of the 8th Intl. Conference on Database and Expert Systems Applications (DEXA 97), Toulouse, France, 1997.

[5] O2Web User Manual. Ardent Software, Inc. 50 Wahington Street, Westboro, MA 01581-1021, USA. 1997

[6] S. Holzner. XML Complete. McGraw Hill, 1998.

[7] P. Patel and K. Moss. Java Database Programming With JDBC. Coriolis Group Books, 1996.

[8] T. Mowbray and W. A. Ruth. Inside CORBA: Distributed Object Standards and Appliations. Addison-Wesley, Reading, MA, USA, 1997.

[9] C. Lynch and H. Garcia-Molina. Interoperability, Scaling, and the Digital Libraries – Research Agenda. Report on the IITA Digital Libraries Workshop, May 1995. http://www-diglib.stanford.edu/diglib/pub/reports/iita-dlw/main.html

[10] C. Lunau and F. Turner. Summary of Issues Related to the Use of Z39.50. National Library of Canada. http://www.nlc-bnc.ca/resource/vcuc/ezarlsum.htm

[11] A.P. Sheth. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. Intl. Conf. On Very Large Data Bases (VLDB '91). Morgan Kaufmann Publishers, 1991.

On the Emergence of the Renaissance Software Engineer

Scott Tilley† Shihong Huang

Department of Computer Science
University of California, Riverside

Abstract

The nature of software engineering as currently practiced is being dramatically altered by the Web. In the past, a software engineer could prosper with knowledge of basic algorithms and data structures, being proficient at a few programming languages, and having rudimentary project management skills. In today's net-centric world, the skills required by a software engineer have increased dramatically. New types of expertise, including distributed component technology, computer security, and Internet standards like XML, are becoming core knowledge areas. To successfully perform Web site evolution, one must become a renaissance software engineer, with knowledge that is both broad and deep.

Keywords: software engineering, Web, renaissance, net-centric computing, evolution

1 Introduction

Many people say that the world is becoming so complex that no one can know everything. This is, of course, self evident. But many people take this to be true even in their chosen discipline. For example, in computer science, if someone is a "theory" person, the assumption is they can't be a "network" person too. The very nature of a Ph.D. degree is to focus on a very narrow problem space; it becomes difficult to widen one's perspective afterwards.

Yet today's software applications are in fact moving in the opposite direction. It is true that no one can know everything about a particular area. But to be successful in today's net-centric and Web-enabled world, one now needs to know a lot about a lot; ignorance of other areas is no longer an agreeable position. There are now so many different aspects to application software that there is a (re)emergence of the renaissance person, one who is comfortable operating in several disciplines, one who readily moves between computer science, software engineering, and information technology. The complexity of system construction and evolution in a Web-oriented world is driving this renaissance.

A software engineer from a previous era needed to know about computer science issues such as algorithms and data structures, operating systems, programming languages, and so on. They also needed to be aware of non-technical issues such as project management, effort estimation, and risk analysis. Today's software engineer engaged in Web site evolution activities needs to know all these things—and more. One might ask why should a competent C programmer need to be a network security expert and a database expert as well. The answer is that these are just some of the new skills required of the new renaissance software engineer.

As depicted in Figure 1, terminology related to Web site evolution activities is voluminous and suffers from acronym overload. Further complicating matters, there are several different terms are often used to refer to essentially the same thing. The following sections provide brief commentary on three specific topics that are rapidly emerging as essential areas of expertise: distributed component technology, computer security, and Internet standards.

† Contact information: Voice: +1 (909) 787-6438. Fax: +1 (909) 787-4643. Email: stilley@cs.ucr.edu. Web: www.cs.ucr.edu/~stilley

Figure 1: Acronym Soup

2 Distributed Component Technology

The emergence of distributed component technology (DCT) is one of the most important developments for the Web. DCT is essentially a combination of software components and a middleware infrastructure. However, each area (components and middleware) is itself quite complex. Components have their genesis in object-oriented programming (OOP) techniques. Middleware has evolved from work on distributed computing, software development environments, and repositories.

A component is an independently deliverable package of software services [Brown 97]. Each component has three facets:

1. A *specification* that describes what a component does
2. An *implementation design* that describes how it does it
3. An *executable* that delivers that capability on a designated platform

There are different approaches to providing components following different standards for creating each of these three facets. For example, Microsoft provides the Component Object Model (COM) and ActiveX components for Web-enabled applications. Sun provides Java Beans. Component-based development (CBD) refers to the tools and techniques that enable construction of applications from new and pre-built components.

An important aspect of any component-based approach is the distribution mechanism used to deploy the components and to facilitate their communication with one another and with commonly-provided system services. There are currently three basic distribution infrastructure approaches:

1. DCOM/COM+/DNA, as advocated by Microsoft

2. Java/RMI and Jini, as espoused by Sun Microsystems
3. CORBA, as proposed by The Object Management Group (OMG)

One of the reasons DCT is so important in today's software engineering efforts is its supporting role in Web-enabled applications. Migrating legacy systems from single-tier mainframes to a modern 3-tier architecture relies on DCT capabilities. For example, in making components available from the database back-end through the application server to the accessing client across a network.

There is little doubt that DCT is an intricate technology area. There are few areas of computer science or software engineering that are in such a state of flux as DCT. The technology is changing extremely rapidly, driven in part by demands placed on Web-enabled applications. Moreover, there remains little agreement upon basic issues from the major players in the industry. However, DCT offers significant advantages to current application engineering. It also brings to the fore one issue that has been historically under-developed by software engineers: security.

3 Computer Security

One of the most important areas in current information systems theory and practice is computer security. The growth of the Internet and heterogeneous application environments has placed an increased emphasis on security as a first-class quality attribute [Barbacci 95]. Previously treated as an add-on in most software efforts, security is now a primary concern of network-based and Web-enabled systems [West-Brown 98]. Security is an essential aspect of Web site evolution; improving security policies and implementations for large-scale Web sites is a very common and increasingly important activity in the current Internet climate.

Becoming knowledgeable about security can be a daunting task. It is an area that is rapidly changing, is very broad, and requires significant commitment. As new applications are brought onto the market, new security risks arise in tandem. In some cases, new standards are proposed that appear to be secure, but their implementations are flawed. Witness the seemingly endless series of patches to the major Web browsers that users must apply to secure their computers when connected to the Internet.

To be proficient in security requires at least a passing familiarity with topics that include encryption techniques, digital certificates, and computer viruses. Each of these areas in turn requires specialized expertise in itself. Becoming adept at any of these areas requires steadfast effort, primarily because it is a very unforgiving subject. For example, knowledge of "a few" computer viruses is not sufficient; you need to know about most (if not all) of them. Otherwise, a new virus will undoubtedly emerge that can compromise your system. Computer security is an area that requires a winner-take-all mentality; there is rarely room for "good enough" protection when your Web-based enterprise depends on it.

Given the current push to add electronic commerce (e-commerce) to existing Web sites, security has become particularly important. This type of enhancement is potentially very lucrative for an organization, but it is also fraught with peril. The cat-and-mouse game between system administrators attempting to protect their networks and hackers attempting to crash a system or gain entry to an Intranet is played out daily. Privacy issues, transaction processing, and even digital money are often new areas to many software engineers. Yet they are critical aspects of modern software engineering practice and Web site evolution.

4 Internet Standards

There are a great many new Internet-related standards currently under review by a variety of oversight organizations and user groups. One of the most important developments for the Web is the next-generation HTML technology, the Extensible Markup Language (XML), a data format for structured document interchange. The World Wide Web Consortium (W3C) adopted XML 1.0 as a standard in February 1998 [W3C 98].

Most people know that HTML is a markup language, providing a series of tags used to describe the appearance of text, data, and other elements on a Web page. XML moves beyond the somewhat limited capabilities of HTML by allowing users to specify their own tags. In other words, XML incorporates meta-data into a Web page's description. XML is a strict subset of the meta-language SGML (Standard

Generalized Markup Language), offering many of the same capabilities of SGML but without the complexity. By using XML, information contained in a Web page can be searched, indexed, and processed with more accuracy than with a regular HTML page.

XML has already been employed for special purposes, such as Microsoft's Channel Definition Format (CDF) used for pushing content in Internet Explorer 4 and 5. Users define new relations of interest in a portion of the Web document that contains the Document Type Definition (DTD), which consists of new tags that become available for use in the same manner as the tags that are predefined in HTML. It is true that the XML community will need to standardize on vocabularies for different application domains, so that users will know how to name their tags to ensure interoperability. However, this will likely occur quickly in domains that already have a standardized ontology, such as Web search engines.

There is interest in XML from both the presentation community and the middleware community. From a presentation point of view, XML can provide greater clarity of page descriptions and intended rendering of Web content. Used in conjunction with Cascading Style Sheets (CSS), XML forms the basis for Dynamic HTML (DHTML), a technique for client-side animation of Web pages. From a middleware point of view, XML offers the potential for a common exchange format. Since it is a textual language, as opposed to a binary standard like COM, users can read the XML description and gain an understanding of the intended capabilities of the new tags and entities. XML can be used as a point-to-point integration mechanism, serving as a bridge between different distributed component technologies, such as DCOM and Enterprise Java Beans (EJB). Prior to the existence of XML, the usual way to obtain similar results was to arduously code separate server-side CGI scripts. By providing a standard way to represent both data and meta-data, XML should lead to better interoperability between databases, languages, and tools.

It has taken a while for tool vendors to add XML parsers and interpreters to their products. However, XML is now supported by several Web browsers and associated tools. It is expected to enjoy considerable attention and support by major players in the coming year. Software engineers working on Web site evolution projects cannot afford important developments like XML.

5 Summary

It is neither possible nor desirable to keep abreast of absolutely every single trend in software. Some trends are really just fads, quickly fading away. Other trends, however, are truly important; the Web is one of these trends. It is having a sea-change on the nature of software engineering as currently practiced.

In the past, one could prosper with basic knowledge of algorithms and data structures, programming languages, and rudimentary project management. In today's net-centric world, the skills required by a software engineer engaged in Web site evolution activities have increased dramatically. New areas of expertise, such as distributed component technology, computer security, and Internet standards like XML, are becoming core knowledge areas. To be successful, one must become a renaissance software engineer, with knowledge that is both broad and deep. This implies keeping abreast of both research in academia, developments in industry, and watching governmental policy. Failure to do so will inevitably shorten one's career. Besides, it's just too interesting not to!

References

[Brown 97] Brown, A. "Background Information on CBD." *SIGPC*, Volume 1, Number 18. Riverside, CA: S.R. Tilley & Associates, 1997.

[Barbacci 95] Barbacci, M. R.; Klein, M. H.; Longstaff, T. H.; and Weinstock, C. B. *Quality Attributes* (CMU/SEI-95-TR-021, ADA 307888). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995.

[West-Brown 98] West-Brown, M. and Ellis, J. "Security Matters—Doesn't It?" *SEI Interactive*, Volume 1, Number 2. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998.

[W3C 98] The official XML specification. Available online at http://www.w3c.org/XML.

Management of Web Site Evolution

June Verner
College of Information Science
Drexel University
Philadelphia, USA
june.verner@cis.drexel.edu

Hausi A. Müller
Department of Computer Science
University of Victoria
Victoria, BC, V8W 3P6 Canada
hausi@csr.uvic.ca

When we consider the management aspects of web site evolution we must ask, "How is this type of project different from other types of projects, and what other kinds of project might show management similarities?" Currently there is little research that answers these questions (though hopefully WSE 99, the Workshop on Web Site Evolution, will provide some answers). Where can a project manager tasked with a web site evolution project begin? We believe that the management of web site evolution is basically no different from managing legacy software evolution projects and that, although all the questions regarding evolution of legacy systems have not been answered, we at least have considerable experience in this area. Large web based systems, particularly if written by people who lack formal programming or software engineering training will probably be as difficult to evolve as large, heavily modified, structurally decayed legacy systems. Both of these system types typically suffer from a lack of accurate documentation. Documenting business rules is as hard today as it was 20-30 years ago. Thus, it is not surprising that the extraction of business rules from large web enabled systems is as elusive as from legacy software systems.

Similar difficulties arise when we wish to migrate large business critical legacy systems from a standalone mainframe or workstation platform to the web. This is often the first step in the evolution of a web-based system. Over the past few years we have seen a tremendous expansion in the number of commercial web sites as organizations increasingly move into electronic commerce. Increasing acceleration in this direction means that software engineers will have to cope with escalating numbers of large legacy system migration projects. This trend will likely continue for some time as businesses realize the necessity and advantages that will accrue by having a web presence. Evolving a large existing information system to have web capabilities typically involves adding new functionality, adapting the user interface to web browsers, and installing firewalls to increase security, reliability and availability to serve millions of potential customers. This is a difficult undertaking as it often involves the understanding, changing and testing of a large part of the source code. While the documentation accumulated over the years for a legacy system is usually voluminous, it is often out of date. Figuring out the major differences between source code and the existing documentation is as hard as understanding the source code without documentation. We may also find that no one in the organization understands the system completely.

Because of our belief that the processes required for this type of project and those for evolving a large web based system are similar we will focus our discussion on management issues surrounding the migration of a large mission critical legacy system to the web.

Before the actual migration project can proceed a solid business case must be developed. The need for a web enabled system and a cost-effective migration path must be demonstrated. Thus, the first step is to conduct a cost-benefit analysis to justify the migration of the system. Without this justification there is no point in proceeding further. Estimating the costs and the risks of a migration project is inherently difficult. Software engineering economics books [Boeh81] typically focus on pure forward engineering rather than software reengineering. The lessons learned from designing software from scratch do not necessarily apply to the realm of software reengineering.

Web evolution projects are risky for many reasons. One risk factor is the technology, which evolves at an amazing rate. Locking in too early or too late on middleware and web standards can easily break an evolution project. Moreover, it is not clear whether the skill set of traditional software developers is sufficient to extract the necessary knowledge from a legacy system to perform effective migration to the web. Thus, much more research is necessary to provide accurate cost

estimation and risk analysis for software reengineering and web site evolution projects.

If the migration benefits are expected to outweigh the likely costs, the actual migration effort can be undertaken. A software reengineering process consists of two major phases: reverse engineering and reengineering.
The reverse engineering phase involves data gathering, knowledge organization, and information abstraction [Till95] including:

- developing a high level inventory of source code, data sets, scripts, design documents, and ancillary files;
- extracting software artifacts at various levels of detail;
- summarizing and abstracting the wealth of software artifacts by synthesizing virtual subsystem structures from software artifacts; and
- identifying user interface beacons which can serve as starting points for impact analysis to identify the user interface code.

Analysis paralysis can easily set in during this first phase. Is a coarse grained analysis of the software artifacts sufficient? How can we decide that we have enough knowledge to make informed decisions for the actual reengineering tasks? Do we need to understand all the business rules built into the system before we can embark on the actual migration phase? Do we follow the same reengineering strategy for a medium size system as for a multi-million line system? Which strategy is the most appropriate for the evolution tasks at hand?

The reengineering phase involves changing and testing of code and data including:

- risk analysis and planning for migration;
- injecting middleware technology;
- conforming to selected web standards;
- accommodating security requirements;
- assessing availability and performance requirements;
- migrating affected subsystems;
- testing target information system;
- conducting a user study with selected customers; and, finally,
- deploying new information system fully on the web.

Predicting the costs of these individual stages is not easy. At this point in time we have solid data for predicting the costs of Year 2000 conversion projects [Jone98]. Is this data readily applicable to Web site evolution projects or arbitrary reengineering projects? We believe that it is a good starting point. Here are cost estimates for software reengineering projects adapted from actual Year 2000 conversion projects:

- 20% planning, inventory analysis, and technology evaluation;
- 30% reverse engineering and reengineering;
- 50% testing and fielding.

The costs of the actual reverse engineering and reengineering processes are comparatively low, only 30% of the overall costs. Thus, a good technology solution is necessary but not sufficient and a good management approach is vital for the success of the project. This is an important finding and taking it into account can significantly contribute to the success of a web site evolution project.

The key to lowering the costs for testing is access to software engineers with extensive domain knowledge and to tools to automate the actual conversion process. Investing heavily in tools for automating the conversion process however assumes that the tools can be applied repeatedly. Alas, cost prediction for software engineering is problematic. We are all familiar with reports of cost overruns and projects that, for various reasons, do not meet their estimates and/or requirements. Cost prediction is always difficult in situations where the requirements are fuzzy—there are many fuzzy factors in a web site evolution project

Why don't we rewrite the system from scratch? For a small web based system this might be an option. But as soon as the system or web site has gone through a few evolution cycles this option is no longer viable. The crux of the matter is the business rules or requirements that are hidden in the code as opposed to be presented on a silver platter. Because we are dealing with a large legacy system, we may not know the business rules on which the system is based upon. So redeveloping the system from scratch is probably not a viable option since the costs for extracting the business rules will likely be prohibitive.

In order to discover how a textual or graphical user interface can be migrated to a web-based user interface or how to guarantee certain levels of security, we need to factor out the relevant parts of the system. To do this we must decompose the system into its subsystems according to various

functional and/or non-functional criteria. Different systems vary on a scale of decomposability into subsystems; they range from easily decomposable systems to non-decomposable systems [BrSt95]. The closer they are to the non-decomposable end of the spectrum the more difficult they are to deal with. If the whole project at this stage proves to be very difficult and the system is more or less non-decomposable the project may be terminated at this point, but this decision will depend on the actual business case that was initially made.

The staff that we employ to perform these tasks must be highly skilled, but we may not have access to software engineers of the right caliber. Worse we don't even know what the proper skill set is for evolution projects. While from experience we have a good idea what skills are necessary to develop software and web sites from scratch, we know little about what kind of background is necessary to extract and understand artifacts and business rules that make up legacy systems. All we know is that the skill sets are fundamentally different.

Once we have gathered the artifacts and decomposed the subject system into subsystems we can do a risk analysis. Included in our risk analysis we must consider the difficulties involved in changing the system, and how much and what parts of the system need to be understood and changed. Are we dealing with one third of the system or one tenth of the system? The Y2K domain provides another striking example. Two key approaches to Y2K remediation are the windowing approach (i.e., a solution to the date problem involving a hundred year window) and the expansion strategy (i.e., expand year data items from two digits to four). The former results in the modification of 0.5% of the entire source code whereas the latter affects approximately 10%. Testing 0.5% of the source code is much more practical than 10%. Having to change 10% of a system is too risky and costly compared to 0.5%. Hence, most Y2K solutions employ the windowing approach. Another great advantage of the windowing solution is the fact that it is incremental. The expansion approach requires that the entire system be changed before an executable system can be produced. In contrast windowing affords an executable after the correction of an individual subsystem. An incremental approach is often a crucial requirement for a reengineering project to mitigate the risks involved.

The actual reengineering phase needs to be carefully planned. Three key planning decision criteria are:

- available components technologies;
- desired level of automation; and
- amount of source code to be tested.

There is usually ample opportunity to replace entire subsystems with commercial middleware and commercial-of-the-shelf components. Injecting commercial components during reengineering can reduce the maintenance of the subject system considerably but it is easy to bank on the wrong technology. Many tasks of a reengineering effort can be readily automated. On the one hand investing in methods and tools to automate mundane transformation tasks is only worthwhile if the tasks are repeated many times. On the other hand the testing effort is usually greatly reduced if the changes are made using an automatic procedure. As we continue evolving our web based system many reengineering steps will be repeated over time. Thus, it is probably worthwhile to invest in the development of tools to automate those steps.

The major problems we see with managing the evolution of web based system are:

1. We don't really know what kinds of skills our staff need, and where they might get the education and training they require, to evolve large web based systems. We know we need user interface experts with program understanding expertise but what makes up the rest of the skill set for these people?

2. Is changing to keep up with the technology going to be a major financial burden? How will we find out what this burden is likely to be and how can we estimate it without spending large sums of money before finding out that we can't afford to go further?

3. How can we properly separate concerns in a web site evolution project? Issues such as security, reliability, and availability seem hopelessly intertwined. How can we factor out these aspects into separate subsystems so that, when the inevitable technology advances occur, we can easily change one and only one of these subsystems without affecting the others.

References

[Boeh81] B. Boehm. Software Engineering Economics. Prentice-Hall, 1981.

[BrSt95] M. Brodie and M. Stonebraker. Migrating Legacy Systems: Gateways, Interfaces & The Incremental Approach, Morgan Kaufmann Publishers, Inc., 1995.

[Jone98] C. Jones. The Year 2000 Software Problem: Quantifying the Costs and Assessing the Consequences, Addison-Wesley, 1998.

[Till95] S. Tilley. Domain-Retargetable Reverse Engineering, Ph.D. Thesis, Department of Computer Science, University of Victoria, January 1995.

Characterising Evolution in Web Sites: Some Case Studies

Paul Warren, Cornelia Boldyreff, Malcolm Munro
Centre for Software Maintenance, University of Durham, UK
{P.J.Warren, Cornelia.Boldyreff, Malcolm.Munro}@durham.ac.uk

Abstract

Using a set of prototype tools, this research has been analysing the evolution of a number of HTML websites of different kinds and sizes. Structure information and simple metrics have been collected, and work is progressing to formulate a predictive model of website evolution. A number of preliminary observations have been made, including some unexpected points, and they will form a basis for new metrics and ongoing work.

1. Background

World-Wide Web maintenance is of increasing importance. The WWW is growing rapidly and is being used more and more as an important medium for communicating information and as the basis for providing communication and information based services. Hence it is important that Web sites and their contents be properly maintained.

Unfortunately so far people have allocated far more importance to developing technology to add more and more novel features within Web pages rather than developing and maintaining pages of quality. There is currently very little work being done to systematically address the growing maintenance required for ensuring the Web continues to provide useful information and communication based services in a timely fashion.

Better tools are needed to analyse hyperdocuments. Currently few exist and none are provide universal solutions. These tools are usually developed to address specific problems and do not produce general maintenance information. Platform independent tools, perhaps tools developed using Java, are needed.

More research on metrics relevant to the attributes of hyperdocument maintainability, readability and quality is needed. Tools to measure these attributes are also needed. Such tools could provide useful feedback to Web developers and maintainers. Assessment tools to help evaluate hyperdocuments before installation on a Web site are especially needed by Web site managers.

It is in this last area, of the use of metrics, that the research presented in this paper is focussed. The work draws on both hypertext metrics research, and on research on the use of metrics in understanding the evolution of conventional software.

2. Approach

Work is concentrating on understanding the evolution of websites, and revising models to characterise this in terms of well-defined metrics. Case studies are being used as a basis for real-world observation, and are proving invaluable in refining methods, metrics, and tools. There is still a considerable body of data to analyse, further modelling and design of measures to undertake.

As regards the case studies, a number of websites of different sizes and kinds have been chosen for analysis. A short list of some of these websites is given in the Appendix. The sites are being analysed at intervals of one week to obtain information about structure and content, and metrics are being taken. The sites give a wide range of characteristics to examine, and at least two sites of each size and kind have been chosen, so that observations can be confirmed on a broader base. A set of tools is being developed to identify changes in these web-sites, and take metrics, and hypotheses are now being formulated and revised on the basis of these. This research will continue to evaluate metrics from software evolution and general hypertext theory,

and to develop and refine further metric systems more suitable for the particular nature of Web hypertext.

The web metrics and download tool, SiteSeer, described in [2], has now been improved to permit parsing of very complex web documents, which might include arbitrarily long comments (over 1024 characters), Javascript, and very long strings. Non-HTML hyperlinks can also in some cases be identified and extracted, which promises to be a useful feature for any future research on non-HTML websites. As a result, a clear view can be obtained of a website as a whole. With improved handling of time-outs and other real-time faults, and other features for handling incorrectly constructed HTML documents, SiteSeer is now invaluable as the core tool for this research.

A number of incidental tools have been constructed for analysis of web-site structure from the downloads obtained from SiteSeer. Among these is a simple "Delta" tool for examining the differences between successive downloads of a given web-site, and work is progressing on tools to extract, compare, and represent graphically the static website and delta information and metrics.

3. Choice of metrics

The initial measures being observed have been very simple ones, rather than the much more sophisticated indirect metrics now available for hypertext analysis. This is due to the requirement to understand how these simpler general metrics inter-relate before progressing to implement the mechanisms needed for handling calculations for specific but complicated metrics across real-world websites involving tens of thousands of interlinked documents. A variety of interesting metrics for hyperdocuments are explored in [3].

The initial measures chosen include ***per-document metrics*** such as Halstead's Software Science family of measures, fan-in, fan-out, counts of various kinds of HTML tag, lines of code (LOC), and so on, and ***per-site metrics*** such as Number of Modules (NOM) and McCabe's Cyclomatic Complexity measure; all of these have the benefit of being easy to collect.

4. Results from case studies

Early results suggest that there is a correlation between the density of links in a document, and the likelihood of new links being added to it, up to some critical level, where the document is split. The rate of change of a website appears to be dependent on the number of documents, and in the case of the smallest websites in this study, no changes whatsoever have occurred during the trial period. A fundamental hypothesis is that websites evolve according to analogous rules to software, and this is being tested with reference to Lehman's FEAST project [1] and his well-known laws of software evolution. These results and others yet to be obtained are expected to form the basis for a predictive model.

The following subsections expand on some of the empirical observations made.

4.1 Misplaced insertion

One of the websites studied [Cart] has shown a phenomenon which is familiar to programmers of conventional software: a new section of the website has been inserted at an inappropriate place. This new section is well-authored, and presents information concerning a recent major development in the website owner's product base. Nevertheless, the original vision for the site appears to have been abandoned. Where the site originally took the form of an orthogonal tree, the new sections have been inserted as tours starting from one the leaf documents.

4.2 Changed style

The new sections mentioned above in [Cart] have a style which borrows from the original, but has a definite individual character that is maintained throughout the new sections. This has been observed in another website of comparable size, [DLR], again in a situation where the product base has achieved a substantial step forward. Here, the original style was abandoned altogether, to the extent that the top-level menu of the site has not been brought up-to-date.

A hypothesis arising from the two phenomena surrounding mass insertions is that this may be due to a tendency for a new author to work on new pages away from the original web site, and only later think about where the new section will fit into the original whole.

4.3 Stable website

Another site [STA] has remained entirely unchanged throughout the time of the case studies so far. Even given its small size, only 18 documents, plus images, this seemed surprising. This may be explained in a number of ways. First, the site may be disused. Second, it may be so small that it is statistically unlikely for any change to have occurred during the cycle. Third, some commercial consideration (such as lack of time or staff or funds) may be holding back further development. Fourth, this website may (for some reason) not fall into the E-type software family, and so be exempt from Lehman's 1st Law (Continuing Change). This last possibility is being explored in more detail, because of its extreme bearing on the fundamental assumptions underlying this research. However, an intuitive assumption would be that the rate of change of a small website should increase with the number of documents, and this also is being investigated.

4.4 Identifying High-Change Documents

The relationship between a web-document's fan-in and fan-out, and the overall structure of a website is obviously a close one. Observations suggest there is a correlation between the density of navigational links in a document, and the likelihood of new links being added to it. It is interesting to note that it is the *density* of links which is important. A larger document may well contain more links, but will also contain more material other than links. However, some documents are apparently designated as "link" documents, and contain little else.

The studies will continue to monitor the insertion points of new links, to discover whether there is some critical maximum number of links in a document, at which point the document will be split, and whether this is related to any splitting of documents for other reasons.

5. Summary

The case studies discussed above have given a starting point for the formulation of metrics to describe to anatomy of a website. Undoubtedly, some of the observations made will prove to be random fluctuations, and will have little long-term bearing on the development of a website. Yet the successful work of Lehman and the FEAST project urges that analysis of even a single variable can give useful information to describe and predict the development of a software-based system.

Initial results from this study suggest that clear links can be made between simple observed variables such as number of modules (documents), number of (navigational) links, probability of change, and so forth. These variables are analogous to variables describing conventional software and its evolution, for which a large body of research is available.

Bibliography

[1] M. Lehman, J. Ramil, P. Wernick, D. Perry, and W. Turski, "Metrics and Laws of Software Evolution - The Nineties View", technical report 568, Department of Computing, Imperial College, London, 1997, [http://www-dse.doc.ic.ac.uk/~mml/feast1/]

[2] P. Warren, C. Boldyreff, and M. Munro, "The Evolution of Websites", *Proc. International Workshop on Program Comprehension 1999*, IEEE Computer Society, Pittsburgh PA, May 1999.

[3] A.E. Hatzimanikatis, C.T. Tsalidis, and D. Christodoulakis, "Measuring the Readability and Maintainability of Hyperdocuments", Software Maintenance: Research and Practice, vol.7, 77-90 (1995).

Appendix: a list of the case-study websites mentioned

[Cart] CarterCopters Aviation, USA [http://www.cartercopters.com]

[DLR] Docklands Light Railway, London [http://www.dlr.co.uk]

[STA] Swimming Teachers' Association, UK [http://www.sta.co.uk]

Toward Reusable and Evolvable Web Sites

Kenny Wong
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada *

Abstract

Web sites for an annual event like a major conference follow a regular pattern of evolution. Consequently, tools and techniques for managing this evolution can have tremendous leverage at easing duplicated effort and consolidating design issues. This position paper proposes the need for enhancing web site reuse, suggests a partial approach based on experience with current technology, and outlines avenues of research to support reusable, evolvable web sites.

1 Introduction

Major academic conferences are all about the same—in structure. In the beginning, a venue is selected to host the conference in a particular year (perhaps up to three years in advance). The venue, date, general and program chairs, and sponsors are announced. About one to two years in advance, a more formal announcement adds information about the theme, the conference committee, and other supporters. Also, the announcement requests contributions, such as tutorials, workshops, and technical papers, and outlines important dates. Several months before the conference, the accepted contributors are notified to render their work in the standard proceedings format and submit a copyright release. At about that time, an advance program of conference events is issued, along with registration and accommodation details. The conference happens and then becomes history.

Accordingly, the web site for an academic conference evolves through five major stages with different audiences. In stage one, the site promotes the candidate venue as a suitable one for hosting the conference. The main audience is the steering committee. In stage two, the site provides details about making contributions and formatting publications. The main audience are contributors such as tutorial speakers, workshop leaders, technical paper authors, and tool demonstrators. In stage three, the site describes information about attending the conference. The main audience are potential attendees. In stage four, during the conference, the site provides the latest schedule and daily announcements. The main audience are actual attendees. In stage five, the site becomes an archival repository, perhaps providing electronic versions of papers or handouts of invited talks. The main audience are researchers.

2 Problem

Even with such a regular structure, many conference web sites, especially those that change locales year to year, are developed apparently from scratch. Web maintainers, have to rediscover an appropriately evolving structure to present the right information at the right time. Referring to past sites is misleading because their content is typically structured for later stages. What is needed is a reusable conference "site-in-a-box" that provides the basic, evolving web site structure, possibly in a variety of graphic layouts, along with some boilerplate

*This work was supported in part by the Institute for Robotics and Intelligent Systems Network of Centres of Excellence (IRIS NCE), the University of Victoria, and the University of Alberta.

sample content and a usage guide. This structure could be customized for the conference as appropriate (e.g., to add special events). It then becomes a matter of providing the real content (e.g., list of committee members, list of paper topics, hotel description, corporate logos, contact addresses, etc.) for a particular year. The structure and content are partly reusable for following years. The sites for the upcoming years of a conference could be developed in a parallel, pipelined fashion, and delivered in stages (see Figure 1).

Besides saving some effort, there are other benefits in consolidating and reusing design knowledge. First, the conference web site needs to be highly accessible. Not everyone is using the latest browser with the fanciest plugins on a high bandwidth connection. Some are using an old browser, overseas, on a serial modem line, with images and JavaScript turned off, through a 14 inch screen. About 30 to 40 K bytes of data on the initial page is the maximum amount tolerable. Second, the site needs to be easily navigable. For example, irrelevant information about making a paper submission should not clutter the site when the technical program has been finalized. An evolvable structure could elide or reveal certain content at certain times. Third, the site should look professional, with sharp images, decent typography, good color, and proper composition. These graphic design issues can be partly embodied in the sample layouts and boilerplate content (and tested in a variety of environments).

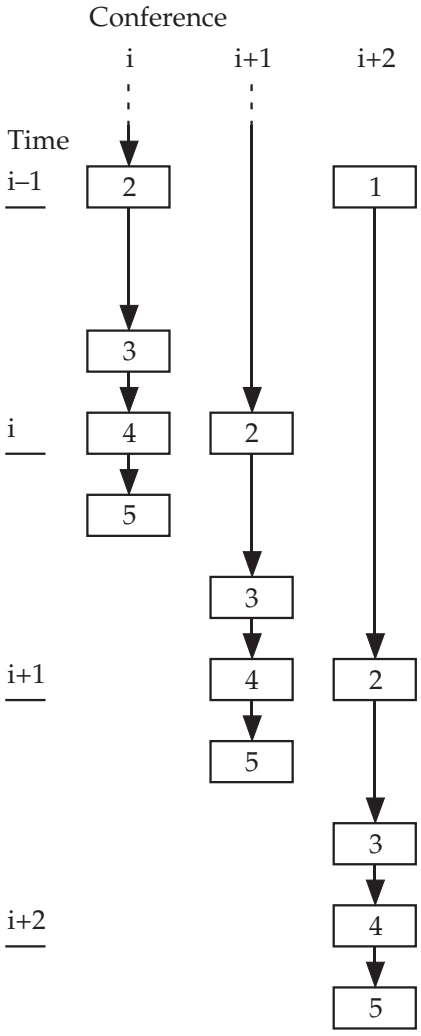


Figure 1: Staged delivery of a product line of conference web sites.

3 Related Work

There are many services that generate a personal web site for an individual, based on specifying certain preferences, interests, and other particulars. Also related are services that provide web site shells into which a company inserts its specific content. These services are aimed at individuals or organizations and not temporally-oriented entities like annual events and projects. That is, the above services do not adequately address the subsequent evolution of the site after inception (other than casual updates). Nevertheless, the techniques used to generate a web site based on various parameters or to populate a web site shell could be useful in implementing a reusable conference site-in-a-box. The evolution of a web site for an event like an annual conference follows a regular pattern. Hence, there is potentially better long-term reuse when dealing with event-oriented sites.

4 Experience

The basics of a potentially reusable, evolvable web site can be partly implemented with current technology. The following brief description is based on experience in developing the ICSE 2001 web site [1]. The tool used was Adobe GoLive, which runs on Mac OS and Windows [2].

The approach centers on separating content from structure. GoLive components or HTML fragments are used to contain pieces of complete content (e.g., a banner, a navigation bar, a block of text, etc.). Layout is represented through a common template (e.g., a 4×2 table). Structure is represented by a set of web pages that are instantiated from the template. This template has slots whereby the individual web pages have specific fragments inserted (see Figure 2). Changes to a fragment causes automated changes to every web page that uses the fragment. This is especially useful when changing the navigation bar to reveal different pages during the evolution of the site.

GoLive provides site management features to allow files to be moved, renamed, and links to be updated appropriately (see Figure 3). There are no version control features, however. Thus, manual cut-and-paste or file renaming is needed to switch among alternative navigation bar fragments. Moreover, the entire assembly of fragments and files needs to be copied for another layout (e.g., text-only) because fragments may contain links and these links must refer to a particular web page and hence a particular layout. Another level of indirection is needed. Alternative layouts should, in the future, be done with style sheets, but the limitation suggests a general need for some level of configuration management and version control for the engineering of reusable and evolvable web sites.

5 Directions

Navigation bars could change during evolution to show only pertinent content. Better support is needed for alternative or conditional content in site design tools. Version resolution can be done at site installation time instead of serving time.

Full configuration management may be overkill for maintaining simple web sites. There are, however, times when some configuration of content needs to have their links redirected. Better link management is needed.

Also, templates or stationery documents should be improved so that changes to them are propagated to all instantiations. One research avenue is to develop a language to enable the flexible construction and instantiation of web site structures and content.

6 Summary

Web sites with regular patterns of evolution benefit significantly from better reuse. Separating structure from content is a promising approach that can be partly implemented in current tools. There are, however, limitations in the tools, but these limitations suggest avenues of future research. Site maintenance capabilities should be improved to enhance reuse in the long term evolution of web sites. These capabilities may entail some level of configuration management and version control or a web site processing language.

References

[1] ICSE 2001 Web Site. Refer to <http://www.csr.uvic.ca/icse2001/>.

[2] Adobe GoLive Product. Refer to <http://www.adobe.com/prodindex/golive/main.html>.

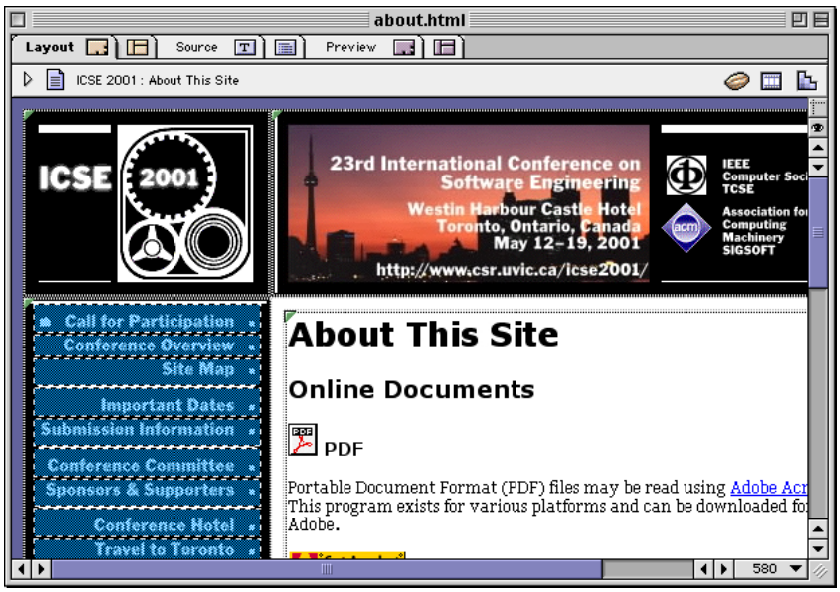


Figure 2: The web site "about" page has a tabular layout with fragments inserted for the banner, navigation bar, and basic content.

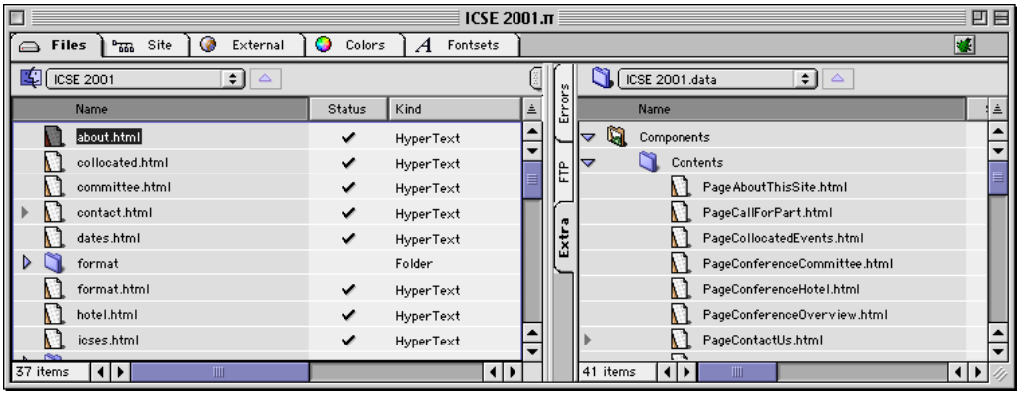


Figure 3: Site management with structural pages in the left pane and content fragments in the right.

Enabling Technologies for Web-Based Legacy System Integration

Ying Zou Kostas Kontogiannis
University of Waterloo
Dept. of Electrical & Computer Engineering
Waterloo, ON, N2L 3G1
Canada

Abstract

With the exponential growth of the Internet and the multi-tier distributed system architectures, there is a urgent demand to develop Web-based and component-based applications to reduce the time to the market and to leverage existing software. This position paper presents an approach to integrate existing legacy applications to a Web-enabled Network-Centric environment. The integration process focuses on the identification of the legacy components, their consequent wrapping using CORBA Objects, and finally the deployment of the application in the Enterprise JavaBeans platform. A scripting language that is encoded in an XML format can be used for allowing thin clients to communicate with legacy components.

1. Introduction

With the growth of the Internet, there is an urgent need to develop Web-based applications. Initially the Web was presented as a giant URL-based file server for publishing static information in a hypertext electronic form. With the incorporation of client/server architectures, CGI scripts and, Java applets the Web evolved into an interactive medium that provides dynamic information services. However, these tools are slow and mostly support stateless transactions. Morover, downloading "fat" Java applets imposes many limitations due to low bandwidth Internet access. Today, new requirements for Web-enabled applications are emerging.

Firstly, organizations would like to take advantage of the Web in its various forms (Internet, Intranet, and, Extranets), for their enterprise computing. This will allow them to leverage functionality from their existing legacy systems without having to rebuild these systems [1]. Furthermore, most organizations need to port their legacy software assets to a distributed Web-enabled environment.

Secondly, the integration of heterogeneous applications and systems is forced through market requirements due to the plethora of operating systems, languages, networking protocols and, data representations [2]. With its universal access across heterogeneous hardware and software, the Web is the best choice to address this challenge [9].

Thirdly, there is a constant shift towards thin-clients in multi-tier architectures away from the "fat-client" paradigm that is predominant in the traditional client/server topologies. The latter requires more memory and longer download time. Thin client architectures shift the focus from the client to the server, by leaving as little code as possible on the client side. This results in faster applet downloading and less client RAM requirements. Thin-clients require the servers to process the data, and to provide complete services such as transactional service, security service and, naming service.

Finally, portability of code and applications become a critical issue, as more types of devices and embedded systems (handheld, wireless) are participating in distributed applications.

It is apparent that the current HTTP/CGI paradigm cannot meet these requirements. The next generation of the Web, the so-called Object-Web, applies distributed object technologies to multi-tier architectures and, is gradually adopted as the development and deployment platform for distributed applications [5].
To integrate a standalone legacy application with other systems the Web offers unique opportunities. Technologies, such as CORBA [6], XML and, Enterprise JavaBeans [10], offer powerful integration mechanisms. This position paper discusses these technologies and sketches an architecture for legacy system integration using the Web as the medium.

2. Enabling Technologies

2.1 CORBA

The predominant architecture for applications requiring access to remote objects is OMG-CORBA. In particular, it offers several advantages over DCOM and COM+, including platform, language, and vendor independence. CORBA ORBs are available for almost all operating systems, such as Windows/NT, Unix and, AIX. The CORBA IDL is used for separating the interfaces from the implementations of the remote objects. IDL as a uniform interface, bridges different programming languages including C, C++, Smalltalk, COBOL and Java. It hides the implementation details from the client code and integrates heterogeneous languages. Moreover, the CORBA IIOP over TCP/IP is an open industry standard, which allows ORBs from different ORB vendors to seamlessly inter-operate and, provides an effective backbone for system integration.

2.2 Component-Based Enterprise JavaBeans

Due to its platform neutrality, multithreading, dynamic loading into JVM, security mechanism and, portability, Java is a widely used environment for object-oriented programming and Web-based software development.

In the software industry, the current trend is towards developing software components with specialized functionality instead of large, special purpose software applications. Enterprise JavaBeans is a Java-based, server-side component architecture for developing, deploying and, managing enterprise-level applications. It deals with the issues that allow components of a distributed application to communicate with each other. Moreover, it allows for the developers to focus on the application business logic and provides a standard component infrastructure to quickly assemble distributed components.

2.3 CORBA and Enterprise JavaBeans

CORBA and Enterprise JavaBeans have been developed independently, they are complementary to each other and, can be seamlessly integrated. JavaBeans uses CORBA/IIOP as the transport mechanism for the pure CORBA client and server. An EJB server can be built on top of an ORB by using the CORBA Naming Service and transactional services.

2.4 XML

The Extensible Markup Language (XML), is one of the key technologies for the future Web-enabled applications. It provides a machine understandable and human readable syntax and, allows for the developers to define their own tags in different domains. XML is widely used as the standard format for data representation, data integration, data storage, message exchanging, and for defining scripting languages.

3. Proposed Architecture

Businesses, agencies and, software houses have invested a lot of money on their legacy applications. At this point, no one can afford to spend the same money, effort and, risk again by re-implementing the same systems for another platform or programming language. One way to minimize risk and cost is to translate the legacy code from one language to another by using translator programs [7]. Even though this approach solves the problems of porting a legacy application form one platform to another, does not solve the integration issue. Another approach is to re-architect and to translate the legacy system in an Object Oriented way and language (i.e. C++ or Java) [2]. Each identified object along with its associated methods is supposed to implement a specific function of the original system. Wrappers can be applied to encapsulate each C++ or Java object obtained from the original system and make it available to a distributed Network-Centric environment. Even though this approach is promising, it causes a lot of network traffic due to the large number of objects involved.

The proposed approach focuses on the use of Web as a infrastructure medium, the utilization of Reverse Engineering to identify coarse granularity components and, the use of Distributed Object Technologies to address issues that stem from distribution (message passing, concurrency control, interaction semantics). In brief the proposed approach consists of the following steps:

1. Use reverse engineering techniques and restructuring to identify and generate a

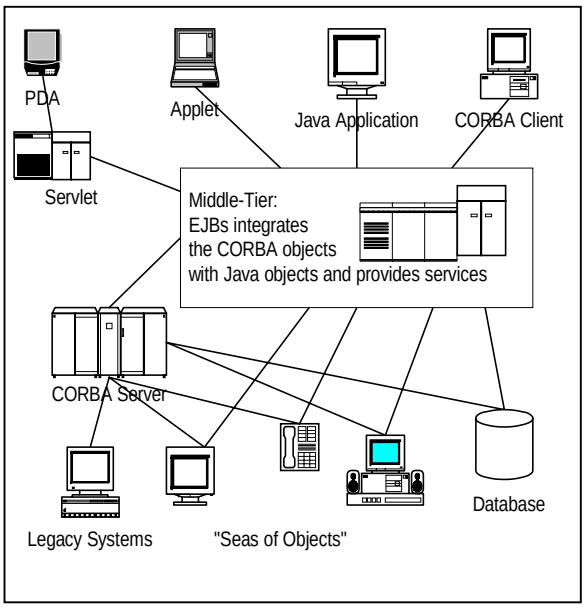


Figure 1: Overall Architecture

decomposition of the legacy system into modules

2. Analyze the interfaces of the selected legacy components and store their signatures in a component repository using XML format
3. Generate the CORBA/IDL and CORBA wrappers from the component repository
4. Use EJBs (Enterprise JavaBeans) to develop the application server, in order to integrate the CORBA wrappers and to provide the services to the Web-based applications
5. Define a scripting language using XML, to allow for the utilization of Web technologies for legacy components to be invoked and computational tasks to be specified.

3.1 Identification of Components

A component can be considered as a reusable unit, that offers specific functionality and can be easily integrated with other COTS or custom made software components. So far, most software analysis efforts have been focused on clustering and decomposition techniques based on cohesion, coupling and, other source code features. In most cases, legacy system decomposition has become a clustering problem. Nevertheless, no matter how successful clustering techniques are on decomposing a legacy system into subsystems, they do not take into account constraints related with the target architecture sought. We believe that decomposition should take into account constraints of the target architecture and, allow for these constraints to be incorporated in the clustering/decomposition process [4].
Moreover, in order to obtain the required target decomposition that allows for integration and distribution, domain knowledge has to be incorporated in the decomposition process.

For the integrity issue, a clear component interface should be specified to decoupage the internal dependencies from the clients, thereby promoting component independence and portability. The interface separates the detailed implementation from the abstract description. The specification hides the collaborations amongst the group of classes and publishes the services (functions or operations) to the consumers.

Finally, it is important to choose a standard format to describe interfaces so that, the identified legacy components can be easily shared with other applications. XML provides a light-weight data representation platform as compared to the heavy weight object database. XML markup tags can also be used for denoting the components' interfaces.

3.2 Wrapping Identified Components

In order to integrate the identified components to a heavily heterogeneous distributed environment, we must define an appropriate middle-ware. CORBA is the appropriate infrastructure mechanism due to its platform, language, and vendor independence specifications it supports
The wrapping process can proceed in three steps.

First of all, a single IDL interface is generated, by denoting the component interfaces in an XML format. Second, externally visible operation identifiers are extracted from the component specification and registered in the CORBA IDL. Finally, the IDL compiler generates the client-side stub and the server-side skeleton object.

The CORBA wrapper inherits from the server-side skeleton classes and encapsulates the legacy components. The wrapper acts as a façade: it offers clients a single, simple interface to the underlying remote objects. It glues together the CORBA distributed capabilities and the standalone components. The wrapper re-directs its public operations to the appropriate external-visible operations in the components.

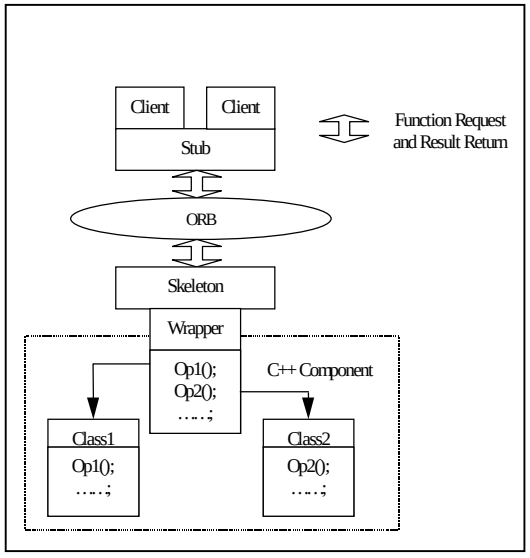


Figure 2: Wrapping a component

It is notable that the object wrappers are housed within the CORBA server infrastructure, providing the back-end services.

3.3 Integration of CORBA Objects

Application servers occupy the middle tier in multi-tier distributed applications. They have been widely adopted as the runtime environment of choice for integrating heterogeneous applications. Different application servers are implemented on different technologies. Enterprise Java Beans (EJBs) provide means to facilitate the invocation of distributed objects in a more standardized way than CORBA. CORBA is suitable for the lower level aspects of distributed applications, while EJBs handle in a more standardized way issues related to communication, security and, concurrency control. EJB is considered as a workbench for "plug and play" components and, for developing application servers. The application server assembles the individual CORBA wrappers, integrates them other remote objects and, makes services available to a front-end Web-enabled application (i.e. a Web browser).

EJBs utilize the CORBA naming services to locate the CORBA objects, and to dispatch the request to the CORBA objects over the CORBA/IIOP. They provide a link between Web-based applications and the back-end diversity software applications, databases, transactional systems.

3.4 Scripting Language for the Thin Clients

In a thin-client environment, applications are downloaded to the clients on-request. Clients can issue individual requests to specific servers using CORBA or any other message pasing mechanism. However, given the infrastructure that the IIOP, http, CORBA and mark-up languages now offer, it is much more efficient to allow thin-clients issue multiple requests to many servers. These requests can be building blocks of aggregate tasks that are requested by a thin-client. Moreover, the requests can implement a more sophisticated paradigm such as the Event-Condition-Action paradigm where, specific requests and actions are carried out only after specific events have been intercepted and specific conditions have been fulfilled. In order to allow for thin-clients to posses such functionality, we must define a scripting language that allows for the composition of services available in the application servers and, the formation of aggregate on-demand tasks. The scripting language can be implemented using XML and be sent to servers using the standard http protocol. EJBs can be used to intercept and interpret the transmitted by the thin-client scripts. In such a way, the client can take advantage of a Web browser to compose scripts (i.e. Web forms), and does not need a special compiler or, script interpreter. The script can be sent to a servlet. The servlet can resolve the script, call the corresponding components in the application server, and return the result to the front-end client. From the client's point of view, the scripting language is directly executable.
In such a scenario, XML becomes the primary candidate as the implementation vehicle for the scripting language. To interpret the scripts, the servlet should maintain the repository for the mapping between the script keywords and the available services in order to be directly runnable by the servlet.

4. On-going Work

This position paper is exploring Web technologies as an integration medium for legacy systems. Currently, we are implementing a prototype for the proposed integration architecture at the IBM Toronto Lab, Center for Advanced Studies.

We are using Refine, Rigi and PBS [3], for analysis and legacy system decomposition. We have selected the IBM Visualage Enterprise Edition to develop the Enterprise JavaBeans application server [8], the IBM Websphere as the EJBs deloyment environment and, the Visibroker as a CORBA implementation.

References

[1] Brown, A., Barn, B., “Enterprise-Scale CBD: Building Complex Computer Systems From Components”, In Proceedings of STEP’99, Pittsburgh, PA. September, 1999.

[2] G. Canfora, G., A. Cimitile, A. DeLuccia, ``Decomposing Legacy Programs:A First Step Towards Migrating to Client-Server Platforms", In Proceedings of IEEE IWPC'98, June 1998.

[3] P. Finnigan et.al. “The Software Bookshelf”, IBM Systems Journal, Vol. 36, No. 4, November 1997.

[4] Gamma, E., Helm, R., Johnson, R., Vlissides, J., “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1994.

[5] Held, J., Susch, C., Golshan, A., “What Does the Future Hold for Distributed Object Computing”, StandardView Vol. 6, No.1, March/1998.

[6] Hoque, R., “CORBA 3”, IDG Books Worldwide, Inc., 1998

[7] Kontogiannis, K., Martin, J., Wong, K., Gregory, R., Muller, H. and Mylopoulos, J., “Code Migration Through Transformations: An Experience Report”, In Proceedings of CASCON’98, Toronto ON., November 1998.

[8] Picon, J., Edwards, C., Scenini, G., “Using VisualAge for Java Enterprise Version 2 to Develop CORBA and EJB Applications”, International Technical Support Organization, http://www.redbooks.ibm.com.

[9] S. Tilley, et.al. "On Using the Web as Infrastructure for Reengineering.", In Proceedings of IWPC ’97: Dearborn, MI; May 1997.

[10] Vogel, A., Rangarao, M., “Programming with hEnterprise JavaBeans, JTS and OTS: Building Distributed Transactions with Java and C++”, John Wiley & Sons, Inc, 1999