

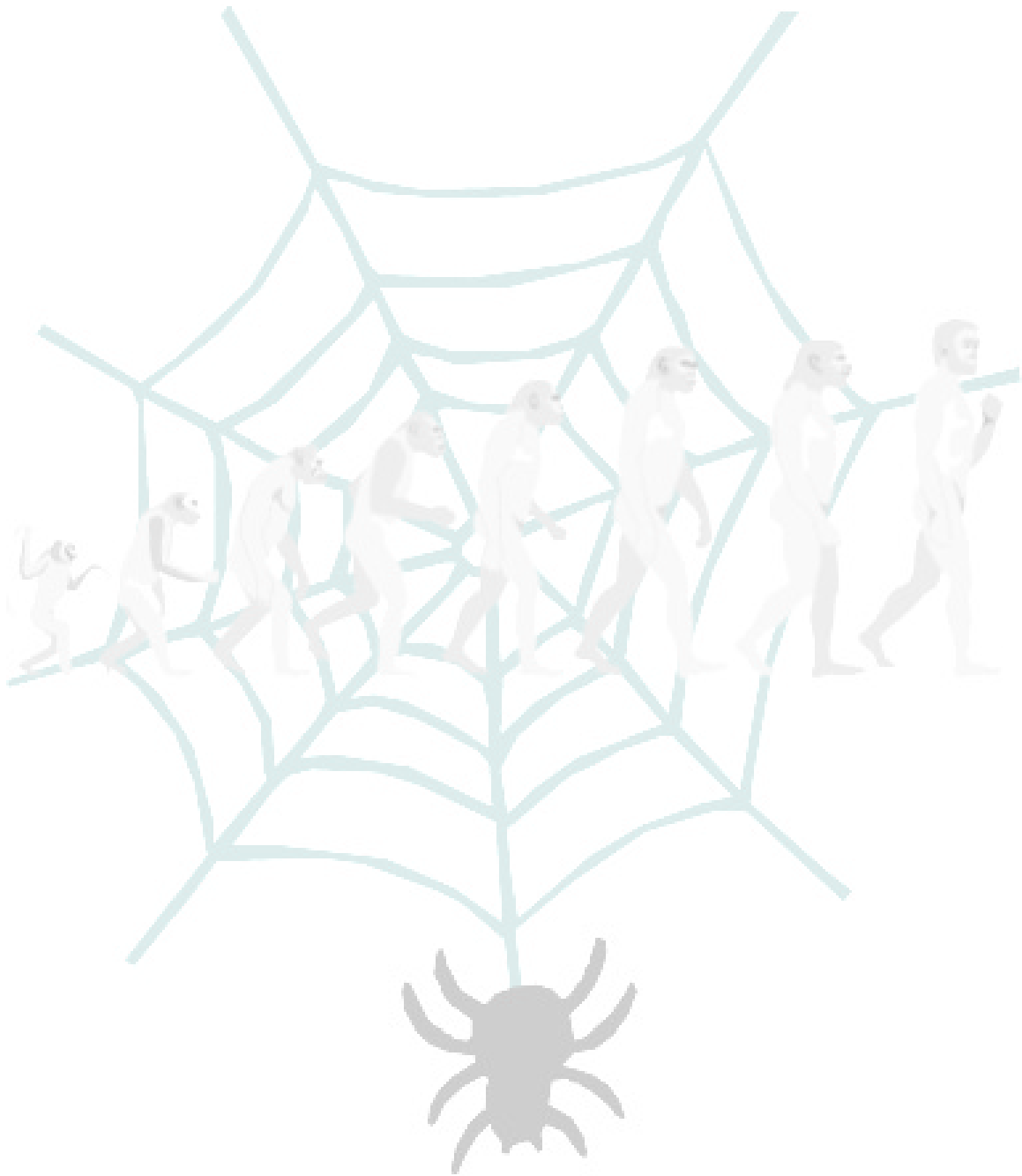


Proceedings of **WSE'2000**

2nd International Workshop on Web Site Evolution

March 1, 2000
Zürich, Switzerland

Edited by Scott Tilley



2nd International Workshop on Web Site Evolution

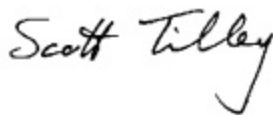
Foreword

Welcome to WSE'2000! The goal of this one-day workshop is to bring together members of the software maintenance, reengineering, and information technology communities to discuss techniques for Web-enabling legacy systems. One way of extending the lifespan of existing applications is to make them available over the Web. While there are numerous tools available to aid in this process, significant challenges remain. In addition, once migrated to the Web, legacy systems are often augmented with new capabilities, such as electronic commerce and streaming media. Such additions add to the complexity of the Web-enabled system, making its subsequent evolution even more difficult.

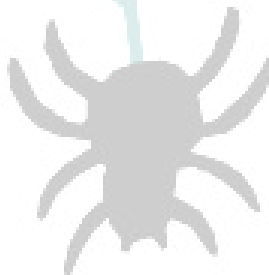
WSE'2000 provides a opportunity for the exchange on information related to exciting new research and empirical results in areas including (but not limited to):

- Extracting and reusing components from legacy systems
- Using distributed component technology in migrating legacy systems
- Managing an evolving n -tier information system
- Case studies of large-scale evolution
- Business issues in migrating to the Web

I am very pleased that the second meeting of this nascent research community is part of the 2nd Reengineering Forum Europe (EuroREF), which is meeting jointly with the 4th European Conference on Software Maintenance and Reengineering (CSMR'2000). As part of *Reengineering Week 2000 Zürich*, I hope WSE can follow the example set by the first WSE workshop in Atlanta to provide a stimulating and enjoyable workshop.



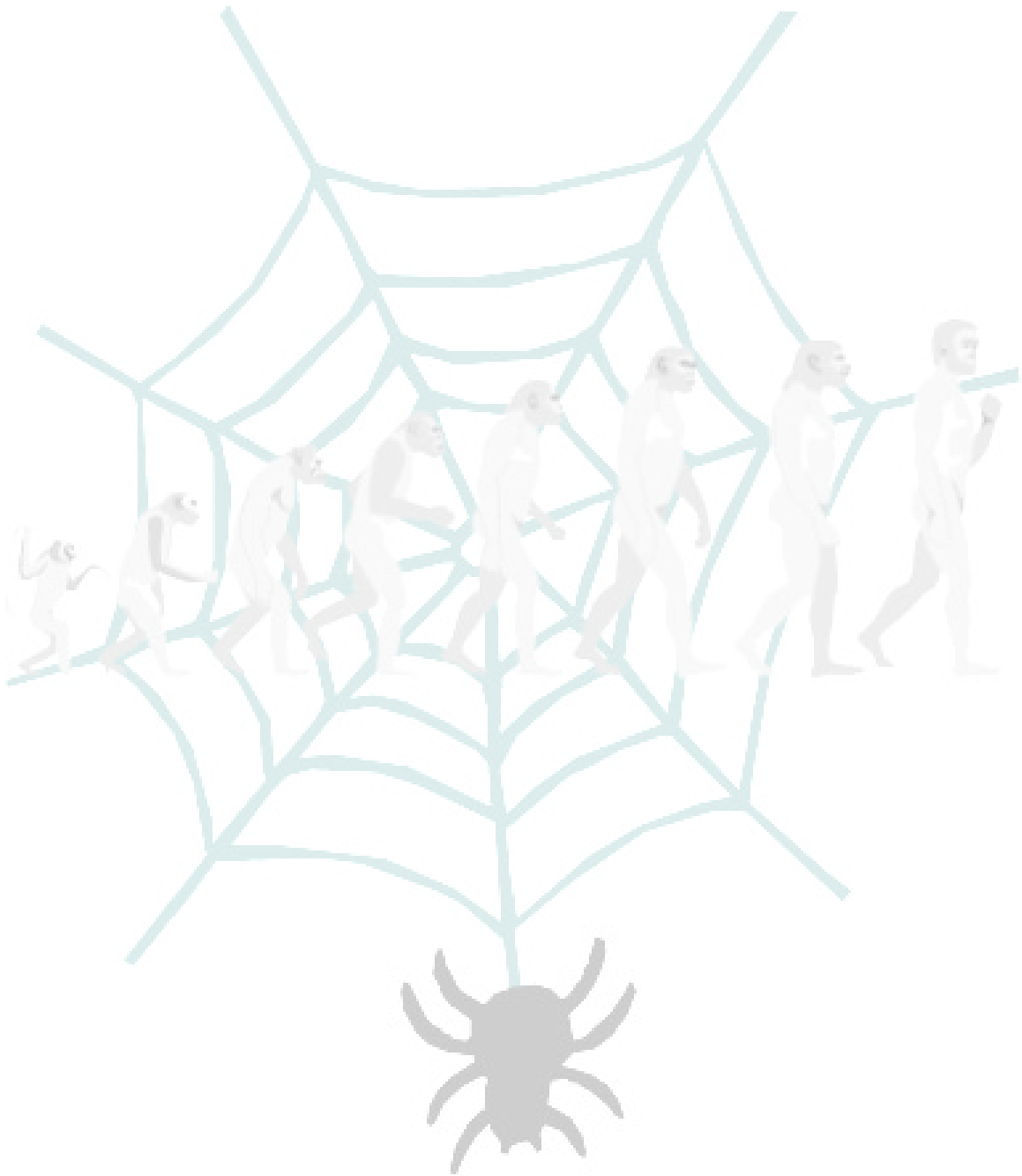
Scott Tilley
General Chair, WSE'2000
University of California, Riverside



Program Committee

Cornelia Boldyreff, Univ. of Durham (UK)
Gerardo Canfora, Univ. of Sannio (Italy)
Elliot Chikofsky, META Group (USA)

Kostas Kontogiannis, Univ. of Waterloo (Canada)
Harry Sneed, Software Engineering Services (Germany)
Chris Verhoef, Univ. of Amsterdam (The Netherlands)



2nd International Workshop on Web Site Evolution

Papers

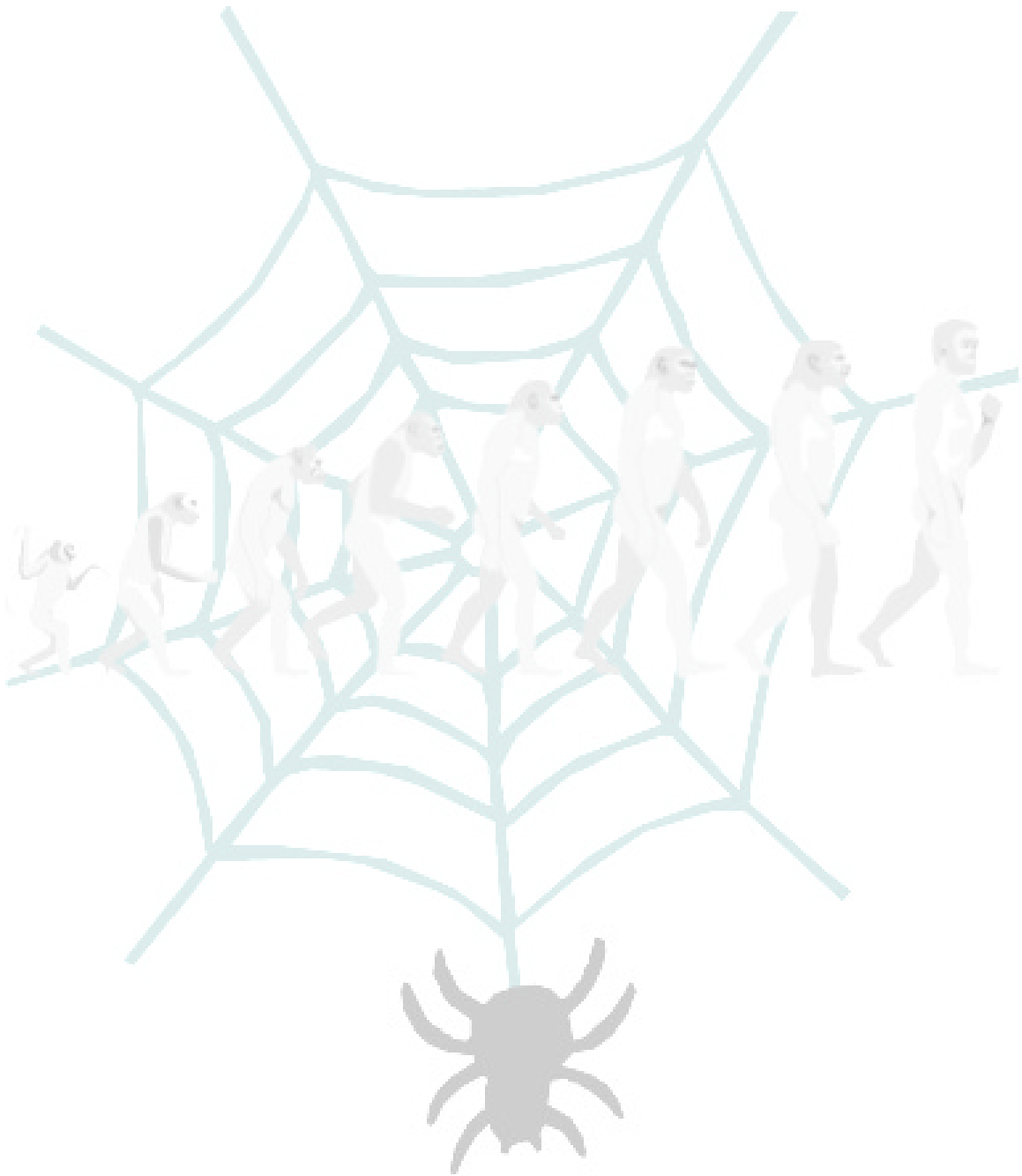
Web Site Reengineering Using RMM	WSE-9
G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia	
Web-SEM Project: Establishing Effective Web Site Evaluation Metrics	WSE-17
C. Boldyreff, P. Warren, C. Gaskell, and A. Marshall	
Software Migration Strategies	WSE-21
H. Müller	
Visualization of Web Site History	WSE-30
F. Ricca and P. Tonella	
Accessing Legacy Mainframe Applications via the Internet	WSE-34
H. Sneed	
Experiences Migrating to a Bilingual Web Site	WSE-47
S. Tilley and S. Huang	
Lexical Scanners for 4GL-Source Maintenance of a Corporate Web Site	WSE-51
B. Toeter	
Localizing and Using Services in Web-Enabled Environments	WSE-57
Y. Zou and K. Kontogiannis	



2nd International Workshop on Web Site Evolution

Agenda

- 8:30a –
10:00a **Plenary**
- 10:00a –
10:30a *Break*
- 10:30a –
10:45a **Welcome**
Scott Tilley, University of California at Riverside, USA
- 10:45a –
12:30p **Session 1**
- **Keynote:** “Accessing Legacy Mainframe Applications via the Internet,” Harry Sneed, Software Engineering Services (Germany)
 - “Web-SEM Project: Establishing Effective Web Site Evaluation Metrics,” Cornelia Boldyreff, University of Durham (UK)
 - “Visualization of Web Site History,” Paolo Tonella, ITC-irst (Italy)
 - Discussion
- 12:30p –
1:00p **Plenary**
- 1:00p –
2:30p *Lunch*
- 2:30p –
4:00p **Session 2**
- “Localizing and Using Services in Web-Enabled Environments,” Kostas Kontogiannis, University of Waterloo (Canada)
 - **Mini-Tutorial:** “Software Migration Strategies,” Hausi Müller, University of Victoria (Canada)
 - Discussion
- 4:00p –
4:30p *Break*
- 4:30p –
6:00p **Session 3**
- “Web Site Reengineering Using RMM,” Andrea De Lucia, University of Sannio (Italy)
 - “Lexical Scanners for 4GL-Source Maintenance of a Corporate Web Site,” Bas Toeter, University of Amsterdam (The Netherlands)
 - “Experiences Migrating to a Bilingual Web Site,” Scott Tilley, University of California at Riverside (USA)
 - Discussion
- 6:00p –
6:15p **Open Discussion; Next Steps; Wrap-Up**



Web site reengineering using RMM

G. Antoniol^{*}, G. Canfora^{*}, G. Casazza⁺, A. De Lucia^{*}

antoniol@ieee.org, gerardo.canfora@unisannio.it, delucia@unisannio.it, gec@unisannio.it

^{*} University of Sannio, Faculty of Engineering at Benevento

⁺ University of Naples, Department of "Informatica e Sistemistica"

Abstract

Most existing web sites have grown from the work of individuals to more general web based software systems serving large communities. Generally, the original sites have not been designed with a particular methodology in mind; even when they have a good structure, this is not documented in any form. The lack of a systematic design approach may cause the degradation of the structure when evolving the site.

This position statement discusses a case study of reverse engineering an existing web site for the management of the services of one university course. We recovered an architectural view according to the Relationship Management Data Model (RMDM) primitives of the Relationship Management Methodology (RMM). The recovered design is now being used to reengineer the web site from a file to a RDBMS based version and to generalize its functionality to serve a larger community (the students of all the university courses).

1. Introduction

During the last decades of our information centric age we observed a continuous and tremendous evolution of technologies, with a sensible improvement of the capabilities of hardware and software, while new paradigms and methodologies emerged. Moore's law states that hardware cost decreases while providing increasing performances, and Lehman's laws indicates that useful programs evolve, actually they must evolve, to meet user ever changing needs.

We claim that the laws of Moore and Lehman govern the evolution of web sites too. Web sites evolve under the market pressure [Chi99]: while the site infrastructure costs continuously decrease, the functionality required from the application software incessantly evolves. New cheaper hardware, new tools, new software packages, and new network backbones are a few examples of factors that induce the reduction of the infrastructure costs. Customers, new market opportunities, and therefore the need for new functionalities, and the need to conform to new infrastructures drive the evolution of the web site application software.

Sites can be categorized into three categories: static sites, sites that are programs, sites that are databases and, most probably, sites comprising databases, static pages, and a dynamic part [Gre98, Ant99]. Static sites are easy to create, as the information they provide do not change over time. They are collections of HTML files, and the task of interpreting and rendering the documents is completely delegated to the browser. Even if they are conceptually simple, static sites may well contain valuable information. The static nature of these sites often leads to a cloning process spreading the same structure, with minor changes, and different contents across an intranet, or even

widely. An example is a site describing a University course, which will almost certainly contain syllabus, exercises, assignments, class scheduling and additional materials. Provided the contained information and goals, course sites are almost always static sites, and they tend to seem alike for any given University: busy people try to avoid reinventing the wheel. However, static sites are not easy to evolve and maintain: a simple change may require to edit a large number of files. Even worse, static sites resemble more programs than industry-strength software systems: documentation such as requirements, high level design, navigation design, and user interface design are almost never available, nor produced. Quite often they were created under the equivalent of a SEI-CMM entry level: a, perhaps brilliant, programmer wrote his/her site, no precise methodology supported him/her and no documentation was produced.

In this position statement we claim that static sites need to undergo a sequence of preliminary activities to evolve toward more sophisticated, more maintainable, more reliable, and ultimately more useful sites. These activities may be conceived as the cascade of two phases: an abstraction phase centered around a preliminary reverse engineering activity, followed by a re-implementation phase, i.e. a sequence of forward engineering steps leading to the new site (see figure 1). In fact, as in almost all human being activities, before implementing/re-implementing an artifact (the site), we need a preliminary conceptualization phase; during this phase we build an abstract conceptual model, a high level site representation encompassing the existing site abstraction. This phase is essential, and entails the recovery of both the detailed design and the high level design of the existing site. The recovered design of the as-it-is site may or may not match with the new site goals: new activities of analysis, the restructuring of recovered information, the implementation of new capabilities, or a paradigm shift from static to dynamic/database centric site, will necessarily lead to a new site architecture; implementing this architecture is the task of the forward engineering phase.

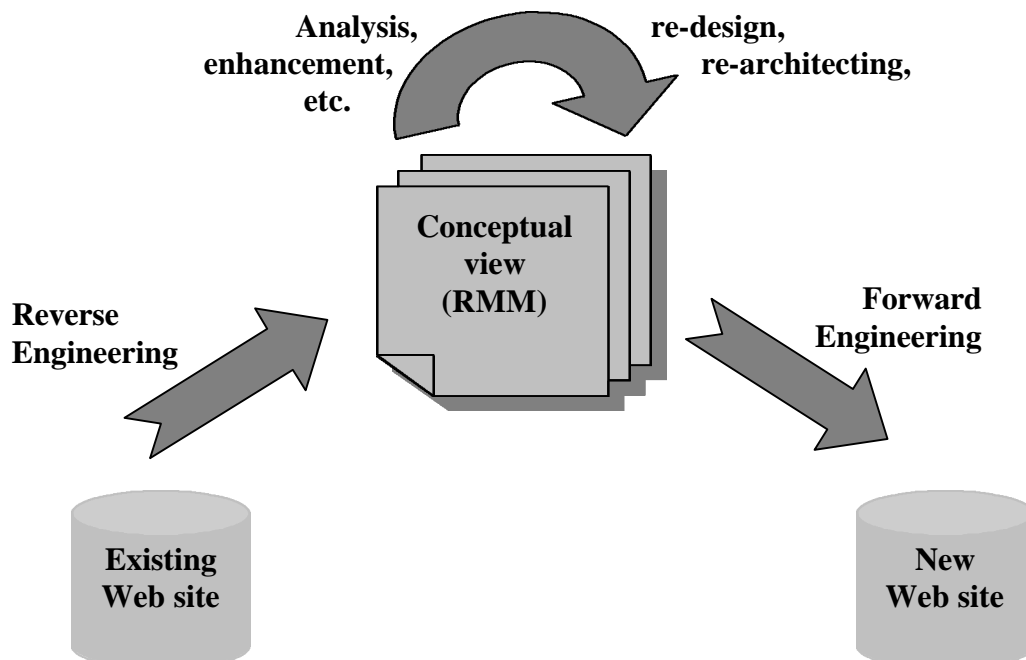


Figure 1: Web site reengineering phases

This sequence of activities must be supported by an adequate methodology, and since the reengineered site will most probably have a database-backed architecture we claim that the RMM methodology [Isa95, Isa97] be adequate. To support our claim we present our experience at the

University of Sannio, where during the past few months the outlined process was adopted to reengineer the Faculty of Engineering course sites. The remainder of the paper is organized as follows: first we briefly introduce the RMM methodology, then we describe the former structure of the static sites, and the abstraction and redesign phases. Finally, we discuss the undergoing activity of sites re-implementation.

2. RMM: a methodology to develop hypermedia applications

Several methodologies have been proposed in the literature for the design of hypermedia applications, including web sites (see [Bie97] for a survey). Among these, the Relationship Management Methodology is particularly suited to support the design and the implementation of web sites based on relational databases. The core part of the methodology is represented by a data model, namely the Relationship Management Data Model (RMDM), which allows the description and the modeling of the application domain in an abstract way. RMDM is based on the Entity-Relationship (ER) model; in particular, it describes the elements of the application domain in terms of entity types, attributes and relationships. Each entity type can have several attributes, RMDM allows the partition of the attributes of an entity in *slices* [Isa95] and the grouping of attributes of different entities in *M-slices* [Isa97]; in the following we will refer to them with the generic term slice. RMM introduces the concept of slice to bridge the gap between the logical and the presentation level of a web site: each slice contains the information to be shown in a web page. The ER model augmented with the slice design is named ER⁺ model.

RMDM offers a number of primitives to define the navigational structure of the information contained in the application domain model (see figure 2). The uni-directional and bi-directional links allow the navigation of slices within the same entity. Grouping is a mechanism that allows to access different web pages through a menu like structure (an example is the home page of a site). The navigation across different entities is possible using indices, guided tours, and groupings. An index, typically, contains a list of entity instances. A guided tour is a predefined path through a collection of entity instances and the user can move backward or forward on the path. Indices and guided tours are qualified by conditions that determine the instances that will be accessed. For example, going from a lecturer's page to the pages of the courses he/she teaches requires to condition the index with the lecturer identity. Indices and guided tours are directly derived from the relationships in the ER⁺ model. There are a number of useful combinations of index and guided tour mechanisms; an example is the conditional indexed guided tour in figure 2.

The design and the implementation of a web site is organized in seven steps, the first three steps are concerned with the design activities while the remaining steps are focused on the implementation and testing activities.

- 1.E-R Design.** This step requires the analysis of the application domain to identify the characterizing objects, properties and interactions. On the basis of the result of such analysis the ER model of the application domain can be defined.
- 2.Slice Design.** This step requires the analysis of the attributes of the entities of the ER model to organize them in slices and to define the navigational links. In general, for each entity there will be a head slice, corresponding to the main page of the entity. The result of this step is the ER⁺ model.
- 3.Navigational Design.** The goal of this step is to design the navigational paths which allow the complete browsing of the information belonging to the application domain information in the ER⁺ model. This requires the definition of the links which relate different entities as specified by the ER⁺ model. Each hypertextual link defined is derived from the relationship of the ER⁺ model and will have as target a head slice.

4. **Conversion Protocol Design.** During this step each element of the navigational design is mapped onto the target platform.
5. **User Interface Design.** This step requires the graphical design of each element belonging to the navigational model. For example, we have to define the font, the color and the size of the text that will be enclosed in each slice.
6. **Run-time Behavior.** In this step the designers decide how the navigational mechanisms have to be implemented.
7. **Implementation and Testing.** During this step the design previously specified will be implemented. Special care is required for testing the consistency of the navigational paths.

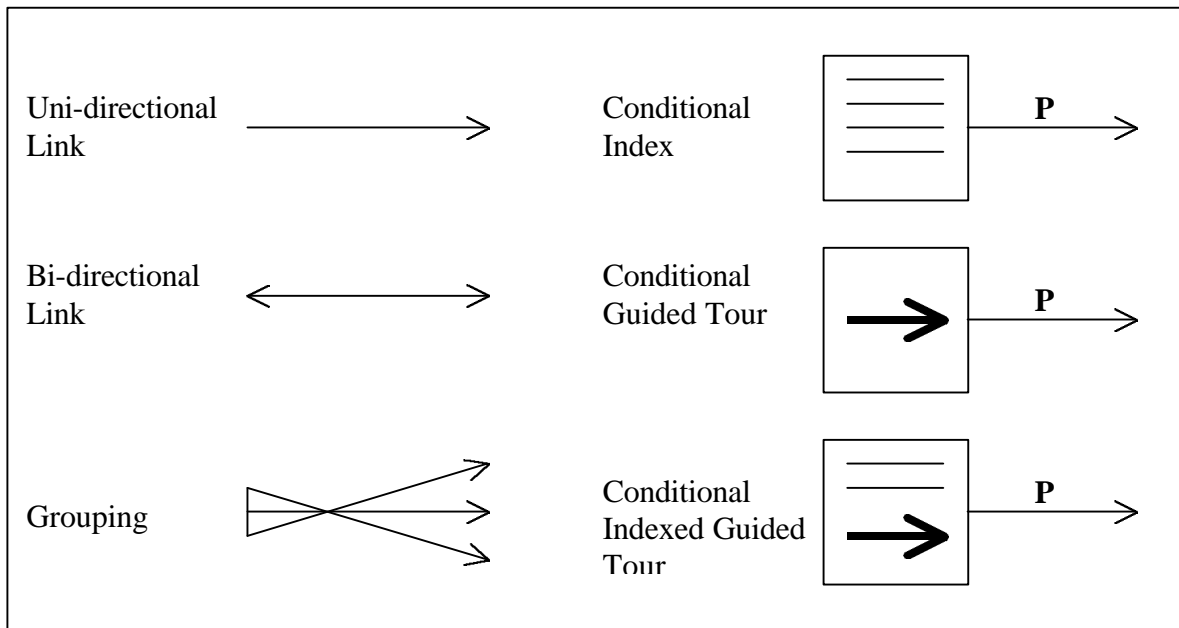


Figure 2: RMDM navigational structures

3. Redesigning the web site

The first step of our reengineering process consisted of reverse engineering the current version of the course web site, to abstract its navigational structure and to identify the entities of the application domain and the relationships among them. This required both a structural analysis of the web pages (to identify the way they are interconnected) and the information contained in the different pages.

A typical course web site is organized as a frameset containing three frames (see figure 3): when loading the index page, the frame on the left contains the menu (always in this frame), while the two frames on the right contain the presentation page (top frame) and information to contact the lecturer. Most of the other web pages, such as the page containing the program of the course, the frequent asked question and the news pages appear in the top frame too. The slides of the lectures and the booking service (exams and tutoring) pages also appear in this frame. Only booking instructions appear in the lower frame. The two services, tutoring booking and exam booking, allow students to book a meeting with the lecturer and to register for an exam date, respectively. These service are implemented through html forms and a CGI application written in C and wrapped with PHP; a student is required to input his login name and password to book the meeting or the exam; pages are also provided to check and cancel booked dates. All data (including dates) are maintained in the html pages; only student information are stored in text files.



Figure 3: A typical course site at the University of Sannio

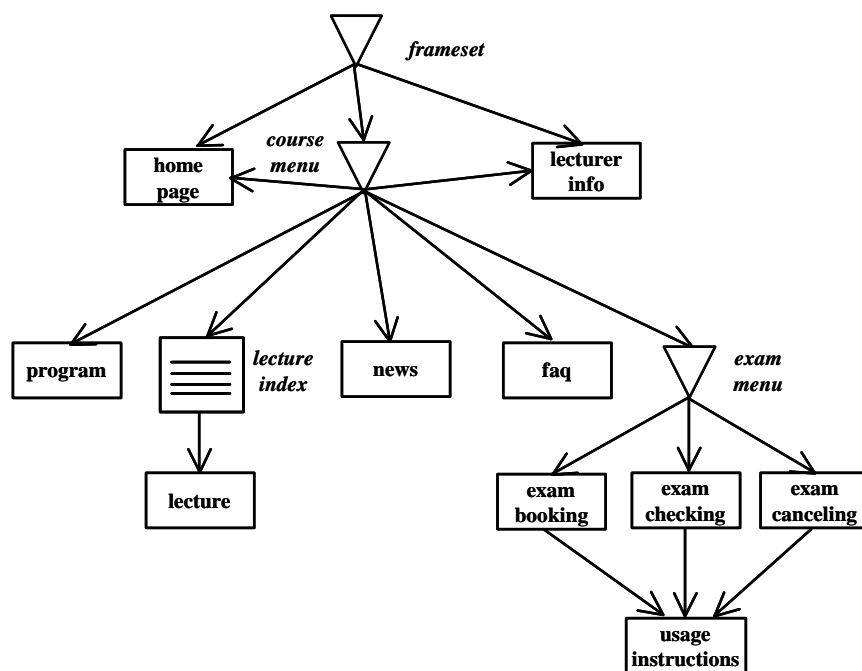


Figure 4: The recovered RMM design

Once the links between the web pages had been extracted, we abstracted the RMDM navigational structure of the web site, shown in figure 4 (for sake of simplicity the pages concerning the tutoring service are not shown, as their structure is similar to structure of the exam booking service). However, abstracting the navigational structure of a web site is not enough. It just gives a representation of the web site organization at the presentation level. The integration of the different

web sites into a common, database centered architecture requires the analysis of the relevant data contained in the web pages. The application domain information contained in the web pages concerns the logical level of the web site. In most cases there is no one-to-one correspondence between the pages of a web site and the entities of the application domain. Often attributes of one entity are shown in different pages and attributes of different entities are grouped in the same page. Each page in the navigational structure shown in figure 4 represents a slice, with attributes of one or more entities. For example, the attributes of the entity *course* are splitted across the slices represented by the home page (that contains the course name), the program page, and the lecturer info page (that contains information about the course's lecturer). The exam booking page contains attributes of the exam entity (the date) and requires the user to input login and password (attributes of the student entity).

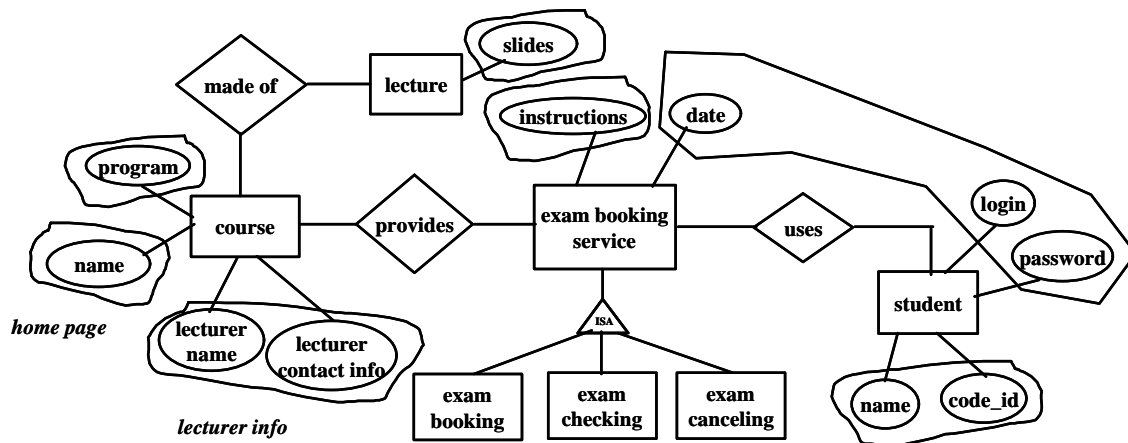


Figure 5: The recovered ER⁺ model

Figure 5 shows the ER⁺ diagram abstracted from the analysis of the information contained in the web pages and the navigational structure of figure 4. The entities are obtained by grouping attributes of different slices (web pages) while relationships derive from the navigational primitives in figure 4. For example, the entity *course* is composed of attributes from three slice (*home page*, *program*, and *lecturer info*), while the relationship *made of* between the entities *course* and the *lecture* derives from the *lecture index* in figure 4. For sake of simplicity the attributes concerning *faq* and *news* are not shown. The information of a course program have a standard and regular structure; then the attribute *program* can be decomposed in sub-attributes, but these are not shown for simplicity. Also, the compound attribute *lecture contact info* has not been expanded. Note the slice comprising the exam date and the attributes login and password of the entity student: a student input his login and password and select a date through a HTML form to access the service. Finally, information about name and identification code (*code_id*) of students is extracted from the exam booking service data files; this information is also displayed in a dynamically built page to confirm the results of the operation.

The first reengineering step consisted of restructuring the ER⁺ diagram; this entailed adding/deleting attributes of an entity, adding/deleting entities, adding/deleting relationships and redesigning the slices. In our cases, the goal was the integration of the course site with the Faculty site: figure 6 shows an extract of the new ER⁺ diagram. The entity *course* was redesigned: attributes concerning the course lecturer have been moved to the entity *lecturer* (deriving from the analysis of the Faculty web site). However, the *course home page* slice now include the names of both course and lecturer. Of course, the lecturer name is also contained in the slice *lecturer home page*.

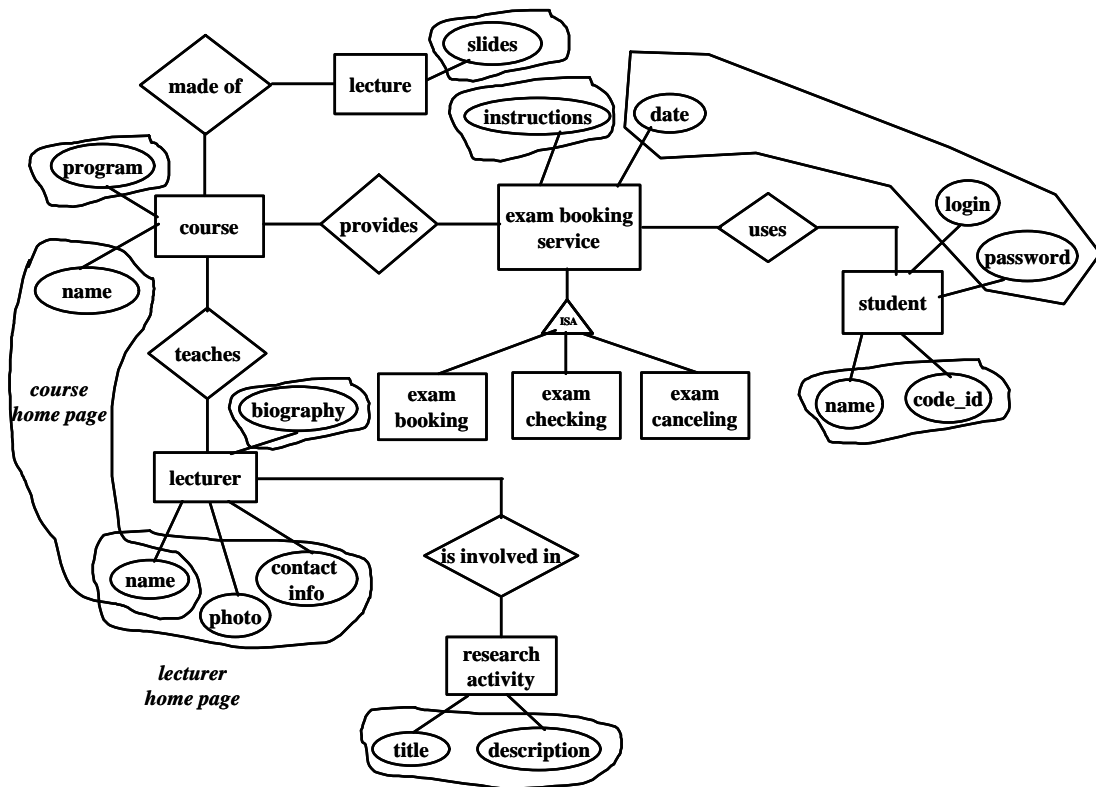


Figure 6: A fragment of the redesigned ER⁺ model

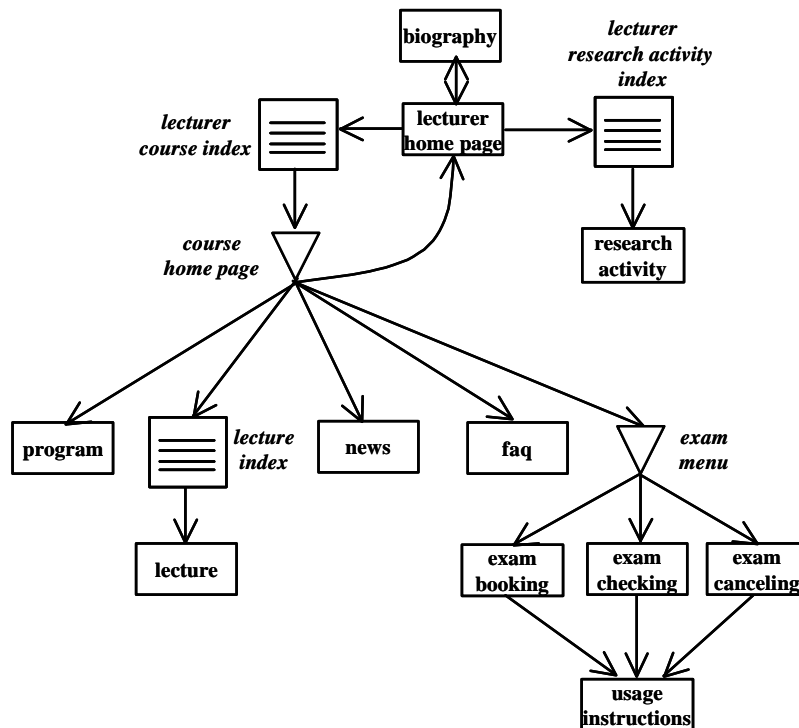


Figure 7: A fragment of the redesigned navigational structure

Once the ER⁺ diagram had been reengineered the new navigational structure of the web site was derived by applying the RMM methodology. Figure 7 shows an extract of the new navigational model. It is worth noting that the relationships teaches and is involved in have been translated into

conditional index. This means that whenever a link *courses* is followed from the home page of a lecturer, the list of all and only the courses taught by this lecturer will be displayed in the target page. For example, this can be implemented by a script that queries the database and arranges the answer in a template page.

4. Conclusions

The forward engineering phase to re-implement the site based on the conceptual model reverse engineered and re-engineered in the previous section is currently in progress. In particular, we have opted for a database-backed architecture with the dynamic generation of data-intensive pages based on the user's requests. The entire application is being developed using technologies and tools available in the public domain. The HTTP server is Apache version 1.3.9-4 [Apa], running on Linux. The database has been completely designed based on the model in figure 6 and is now being implemented using the PostgreSQL DBMS version 6.5.3 with connection pooling [Post]. PHP [Php] is being used to implement the server-side scripts that generate the HTML pages from the data in the database. PHP has been preferred to the more traditional CGI approach because of its better performances (we agree with Greenspun's statement "my Unix box does not like to fork 500,000 times a day" [Gre98]) and superior security (especially when the interpreter is loaded into the server address space).

The user interface of the site is being enhanced too. In particular, we have reduced at a minimum the use of frames as many users reported to find them distracting. We are also revising the navigation structure with the aim of preventing the phenomenon of users (and in particular occasional users) getting lost in the information space.

References

- [Ant99] G. Antoniol, G. Canfora, A. Cimitile, A. De Lucia, "Web sites: files, programs or databases", Proc. of 1st International Workshop on Web Site Evolution, Atlanta, GA, 1999.
- [Apa] <http://www.apache.org>
- [Bie97] M. Bieber, and F. Vitali, "Toward support for hypermedia on the World Wide Web", IEEE Computer, January 1997.
- [Chi99] E. Chikofsky, "Aspects to consider for understanding web site evolution", Proc. of 1st International Workshop on Web Site Evolution, Atlanta, GA, 1999.
- [Gre98] P. Greenspun, Philip and Alex's Guide to Web Publishing, <http://www.photo.net/wtr/thebook/index.html>
- [Isa95] T. Isakowitz, E. A. Stohr, and P. Balasubramanian, "RMM: a methodology for structured hypermedia design", Communications of the ACM, August 1995.
- [Isa97] T. Isakowitz, A. Kamis, M. Koufaris, "Extending the capabilities of RMM: Russian dolls and hypertext", Proc. of 30th Hawaii International Conference on System Science, Maui, HI, January 1997.
- [Php] <http://www.php.net>
- [Post] <http://www.postgresql.org>

Web-SEM¹ Project: Establishing Effective Web Site Evaluation Metrics

Cornelia Boldyreff and Paul Warren
University of Durham
Research Institute for Software Evolution

Craig Gaskell and Angus Marshall
University of Hull
Centre For Internet Computing

Abstract

Many of the problems associated with World-Wide Web sites are already well known to Software Engineering, and so are amenable to techniques and remedies associated with conventional software. A collaborative research venture between the Universities of Durham and Hull is undertaking to develop Software Engineering methods for designing and implementing "better" websites. In particular both static and dynamic metrics to evaluate the attributes that lead to good site design, evolution, and management are being considered. The project builds upon previous experience and work undertaken by the collaborators and will be supported and evaluated by use of case studies.

1. Introduction

Over the past five years Internet technology has been embraced by many educational, business and leisure users. Wide spread adoption of the Internet as a global communication medium has been largely driven by the emergence of the World-Wide Web as an easily used front-end for an ever-increasing variety of applications. Many organisations have been quick to adopt the technology and develop new web sites, although the primary motivation for this has often been to keep up with competitors who have done the same.

Despite recognition that the web is an important new medium for advertising, sales and publicity, many of the current web sites are poorly designed and are lacking in terms of underlying functionality. The key reasons for these problems can be summarised as follows:

- The underlying technology on which web communication is built was not designed to provide the real-time response that many web developers assume. This problem is becoming

more significant as Internet usage increases.

- Developers often produce their web sites using state of the art technology. However, many Internet users do not have the equipment that these developers assume.
- Often web sites are developed by enthusiastic amateurs. Sometimes the developers are technical people, who understand the technology but are lacking design and artistic flair. In other cases the sites are developed by those with more visual and artistic flair, but are constrained by a lack of technical understanding. Ignorance of fundamental HCI and Information Systems management approaches frequently leads to sites which are difficult to navigate and which fail to meet the needs of their perceived target audience.
- There is a lack of engineering standards associated with the production of high quality web sites. Although there are underlying hardware and communication protocols at the technology end, which provide a basis for delivery of information, there are no presentation or structural standards.

The fundamental problem is that most web sites are not well engineered. While individual pages are well presented, their context within the site and the web as a whole (i.e. their structural context) is often not considered. There is also a lack of traceability from the initial requirements of the web site, through design of a site structure, to implementation of an effective web site artefact, which can be evaluated to establish whether the initial requirements have been met. If an engineering approach is to be taken to web site development, the first task is to establish a consistent way in which to evaluate web sites.

This short position paper outlines a collaborative research effort between the Research Institute for Software Evolution at the University of Durham and the University of Hull's Centre for Internet

¹ Web-SEM : Web Site Evaluation Metrics

Computing. It summarises related work in this area and indicates initial work carried out by the project team. Our thesis is set in the context of previous work and the phases of our research programme are detailed. We conclude with a short summary.

2. The need for web site evaluation metrics

Reports of recent commercial research [VNU98A, VNU98B, VNU99A, VNU99B] highlight that 90% of corporate web sites (both commercial and promotional/information oriented) fail to live up to expectations. Forrester Research [Manning99] noted that key tests such as: "Is a site organised by user goals?" and "Does a search list hits in order of relevance?" were failed by at least 51% of sites investigated. Upon interviewing participants in the survey, 56% reported that their primary goal was fast performance and not usability.

Nielsen [VNU98A] highlights the lack of design consideration associated with web sites stating that "...design principles didn't come into the equation". He indicates that the companies are "poking blindly to the design space" [VNU98A]. Zona Research have found that 62% of Web shoppers give up trying to buy online because the process takes too long, or is too complicated [VNU98A]. Although a considerable amount of research related to HCI and usability in general has been undertaken, it tends to relate to individual page contents, giving little consideration to the underlying structure of the information content of a site as a whole [Morville99, Shum96]. Thus an approach is required that combines both static and dynamic analysis.

The Yale/CAIM style guide [Lynch&Horton97] suggests that all sites can be categorised according to two primary parameters: narrative structure and desired contact time. For example, a reference site typically would have a low contact time, but a highly non-linear narrative structure, whereas a training site would have a slightly longer contact time, but more linear structure. This concept is based on experience of design and implementation of multimedia applications [Mok96]. The Yale style guide focuses primarily on educational sites, considering only Training, Teaching, Education and Research and gives no consideration of measures applicable to e-commerce, Intranet or Extranet applications. It also omits to consider the vast range of entertainment and interactive applications that have appeared in the last two years.

Peter Morville suggests that lessons from library

design, usability engineering and architecture can be learned and applied to web site design [Morville99]. This theory is further expanded in his book with Louis Rosenfeld [Rosenfeld & Morville98]. However, as with most techniques in this area, their measures are highly subjective relying on perceptions of usability and success. Nevertheless, they can be used as a starting point for the extension of our current web metrics.

Within the Durham Research Institute for Software Evolution (RISE) there has been active research addressing the development and evolution of web documents and web sites for several years. Earlier work by Boldyreff and Dalton in collaboration with web developers and managers at the CCTA identified the following problems: lack of web development and maintenance methods and tools; lack of standards and guidelines to support the development of maintainable web documents; and the failure to recognise that the development of a web document relies on an engineering approach. The principal result of this earlier research has been a workbench to support development and maintenance of web documents [Dalton96]. Boldyreff and Kyaw have also investigated hypermedia design methods, in the context of web design [Kyaw&Boldyreff98, Kyaw99] and an experimental Web development CASE tool has been developed.

The most recent results from studies of website evolution are described in [Warren99a, Warren99b]; one goal of this research is to demonstrate that Lehman's laws of software evolution [Lehman80] apply to web development as well as to conventional software systems.

Previous and current research at Durham concentrated on what makes a good web site, focusing on design structure and content, and the subsequent management of changes to both structure and content. Several sites have been evaluated. Initial metrics concentrated on characterisation of changes to structure and various other static aspects such as size and complexity measurements: these are derived by analogy with conventional software, including concepts such as Number of Modules (documents), or Lines of Code. Current research plans are to extend these measurements to cover more dynamic aspects of the web such as usability. Our present static metrics have largely been developed from classical software metrics with extensions such as those proposed by Hatzimanikatis et al. [1995]. The dynamic extensions will be developed using existing evaluation approaches (e.g. YALE/CAIM [Lynch & Horton97] and Nielsen's Web Usability [Nielsen99]) and relevant standards, (e.g. ISO/IEC 9126: IT - Software

Product Evaluation - Quality characteristics and guidelines for their use). Based on experimental work that will be carried out in the proposed project, a comprehensive set of metrics, both static and dynamic, will be established for web site evaluation, along with heuristics for good web site design, evolution and management.

3. Project Objectives and Plan

The aim of the Web-SEM project is to establish a set of general principles of good web site design, based on the development of a related web site evaluation model (i.e. set of evaluation metrics). In order to satisfy these aims the project has a number of clear objectives which involve: critique of existing web site evaluation approaches; provision of a classification scheme for web sites; development of a new web site evaluation model based on the GQM approach [Solingen99] and establishment of a number of principles for good web site design.

The project involves the following 4 phases:

- Phase 1: Consolidation of existing work and background research
- Phase 2: Development and enhancement of a model for web site evolution
- Phase 3: Evaluation of the model
- Phase 4: Conclusions and planning for future work

In phase 1 the project is employing the GQM method used in our development of static web metrics to develop an initial set of quality characteristics and associated metrics to identify extensions to current web metrics that enable web site quality to be assessed. In addition, existing evaluation models and appropriate metrics from the area of the web and software engineering in general are being researched. In order to allow the project to focus its initial development, a classification scheme for different types of website encompassed by the project will be established.

Phase 2 of the project will focus on development of metrics and a model for site evaluation. Due to the dynamic nature of the web, the metrics must acknowledge that the evaluation process will take place over time. In this respect the evaluation model will require an element of dynamic analysis. This phase will identify the target web sites to take part in the research. At the end of the phase initial site evaluations will be carried out, and a number of tentative principles of good web site design will be drafted.

Dynamic evaluation of identified web sites using the metrics that have been developed will be the focus of phase 3 of the project. This is the longest phase and will also involve tuning of the evaluation model. Our evaluation results will be compared with those of existing measures. Based on evidence gathered during this phase, the draft principles of good web site design will be refined.

The final phase of the project will be one of reflection, allowing conclusions from the work to be drawn and disseminated. A further programme of research will also be planned at this stage, based on the findings of the initial project.

The project team consists of the authors of this position. Phase 1 of the project is currently being undertaken. Our initial background research has confirmed the overall soundness of the proposed project and our initial site classification scheme has been established although further refinement is anticipated.

4. Preliminary work carried out

Warren has been exploring the character of evolution of websites [Warren99a, Warren99b]. A number of websites have been used as case studies, and these were chosen to give a broad range of websites in terms of size and function.

These case studies have been instrumental in exploring and developing static product metrics for characterising the evolution of websites. Starting with simple metrics like Number of Modules (or documents), Lines of Code (lines of text or Javascript), and count of Function Points (HTML tags), more complicated derived metrics are being developed. The first of these is a Link Density metric, ρ_l . This describes the number of hyper-links per line of text in a web-document, and is strongly related to P_l , the probability of additional links being added to the document per unit time. The importance of this metric is that it connects structure-in-the-large with structure-in-the-small, making it possible to predict the future shape of a website based on measurements of its current shape and the content of individual documents.

Another aspect of this work has been to use metrics to characterise documents by kind, such as index, text pages, picture galleries, forms, and embedded applications, and to observe the behaviour differences between those document types.

To adapt metrics from conventional software analysis for use with HTML, a mapping has been developed, so that concepts can be readily

interchanged. Several features of HTML appear to have no familiar analogue in conventional software, which presents an interesting challenge for future work.

A collection of small but robust tools have been developed to assist this analysis, including an HTML parser, a web-spider, a difference-detecting system, and many statistical components. These have been designed to permit modular extension.

5. Conclusions

It is not enough to recognise the need for an engineering approach in web development and evolution: a sound basis for such an approach is needed. This is the key issue being addressed by our research.

Our thesis is that before existing web engineering methods can be further developed and refined to aid the process of designing a sound underlying information content structure and associated navigation strategies for any site, we must first identify consistent ways of measuring the success of a site as a whole. By doing this, we can identify the key principles of good web site design and evolution over time. We are proposing a scientific approach for establishing such metrics, which will be based in part on current software engineering (and our static large-scale web evolution) metrics research. Only when the metrics are in place will there be a sound basis for further development and refinement of web site design methods.

References

- [Dalton96a] Dalton, Susannah and Boldyreff, Cornelia, "Web Maintenance - The New Frontier", Proceedings of Durham Maintenance Workshop, Sept.1996.
- [Dalton96b] Dalton, Susannah, "A Workbench to Support Development and Maintenance of World-Wide Web Documents", MSc Thesis, Supervisor: Dr. Cornelia Boldyreff, Department of Computer Science, University of Durham, 1996.
- [Hatzimanikatis95] Hatzimanikatis, A. E., C. T. Tsalidis, and D. Christodoulakis, Measuring the Readability and Maintainability of Hyperdocuments. Software Maintenance: Research and Practice, 1995. 7: p. 77--90.
- [Kyaw&Boldyreff98] Kyaw, P and Boldyreff, C, "A Survey of Hypermedia Design Methods in the Context of World Wide Web Design", Computer Science Technical Report 3/98, Department of Computer Science, University of Durham, 1998.
- [Kyaw99] Kyaw, Phyto, "An Investigation of Web-Based Hypermedia Design Support: Methods and Tools", MSc Thesis, Supervisor: Dr. Cornelia Boldyreff, Department of Computer Science, University of Durham, 1999.
- [Lehman80] Lehman, MM "On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle", Journal of Systems & Software, 1, pp.213-221, 1980.
- [Lynch&Horton97] Lynch, P.J. & Horton, S "Yale C/AIM Style Guide" Yale University, USA 1997.
- [Manning99] Manning, H, "The Right Way to Test Ease of Use", Forrester Research, January 1999
- [Mok96] Mok. C, "Designing Business: multiple media, multiple disciplines", Adobe Press, San Jose, 1996.
- [Morville99] Morville, P, "Information, Architecture and Usability", Webreview, March 1999, <http://webreview.com/pub/1999/03/12/arch/index.html> (3rd November 1999)
- [Nielsen99] Nielsen, Jakob: Designing web usability: the practice of simplicity, New Riders, 1999
- [Rosenfeld&Morville98] Rosenfeld, L & Morville, P "Information Architecture for the World Wide Web", O'Reilly & Associates, 1998
- [Shum96] Buckingham Shum, S, "The Missing Link: Hypermedia Usability Research & The Web", Interfaces, British HCI Group Magazine, Summer 1996
- [Solingen99] Solingen, Rini van, and Berghout, Egon: The goal/question/metric method : a practical guide for quality improvement of software development; McGraw-Hill, c1999
- [VNU98A] VNU, "Ninety percent of corporate Web Site fail to Deliver" 21/Oct/1998 <http://www.vnunet.com/News/66500/> (3rd November 1999)
- [VNU98B] VNU, "Website Designs hamper companies" 29/Oct/1998 <http://www.vnunet.com/News/67187/> (3rd November 1999)
- [VNU99A] VNU, "Corporate Marketeers guilty of 'Web wallpaper' 07/Aug/1999 <http://www.vnunet.com/News/88301/> (3rd November 1999)
- [VNU99B] VNU, "Top UK corporates fail to capitalise on the net" 24/Jul/1999, <http://www.vnunet.com/Analysis/87319/> (3rd November 1999)
- [Warren99a] Warren, Paul, Boldyreff, Cornelia and Munro, Malcolm, "The Evolution of Websites", Proceedings of International Workshop on Program Comprehension, IWPC99, Pittsburgh, PA, IEEE Computer Press, pp. 178-185, 1999.
- [Warren99b] Warren, Paul, Boldyreff, Cornelia and Munro, Malcolm, "Characterising Evolution of Web Sites: Some Case Studies", Proceedings of the First International Workshop on Web Site Evolution, WSE'99, Atlanta, GA, October 1999.

Software Migration Strategies

Hausi A. Müller
University of Victoria

WSE-2000
Reengineering Week 2000
Zürich, Switzerland

1

Outline

- The Horseshoe Model
- Reengineering categories
- Reverse engineering strategies
- Reverse engineering theses
- Recent migration theses
- Migration strategies
- Chicken Little strategy
- Language migration
- Conclusions

© H.A. Müller, UVic 2

Research Support

© H.A. Müller, UVic 3

The Horseshoe Model of Software Migration

© H.A. Müller, UVic 4

Reengineering Categories

- Automatic restructuring
- Automatic transformation
- Semi-automatic transformation
- Design recovery and reimplementation
- Code reverse engineering and forward engineering
- Data reverse engineering and schema migration
- Migration of legacy systems to modern platforms

© H.A. Müller, UVic 5

The Horseshoe Model of Software Migration

© H.A. Müller, UVic 6

Reengineering Categories...

- **Automatic restructuring**
 - to obtain more readable source code
 - enforce coding standards
- **Automatic transformation**
 - to obtain better source code
 - HTML'izing of source code
 - simplify control flow (e.g., dead code, goto's)
 - refactoring and modularizing
 - Y2K remediation

© H.A. Müller, UVic

7

Reengineering Categories...

- **Semi-automatic transformation**
 - to obtain better engineered system (e.g., re-architect code and data)
 - semi-automatic construction of structural, functional, and behavioral abstractions
 - re-architecting or re-implementing the subject system from these abstractions

© H.A. Müller, UVic

8

Code Reverse Engineering Phases

- **Information extraction**
 - Artifact gathering from many sources, knowledge organization, analysis
- **Information abstraction**
 - Synthesize components, relationships
 - Build abstraction hierarchies
 - Subject system is not altered; additional knowledge about the system is produced

© H.A. Müller, UVic

9

Design Recovery Levels of Abstractions

- **Application**
 - Application concepts, business rules, policies
- **Function**
 - Logical and functional specifications, non-functional requirements
- **Structure**
 - Data and control flow, dependency graphs
 - Structure and subsystem charts
 - Architectures
- **Implementation**
 - AST's, symbol tables, source text

© H.A. Müller, UVic

10

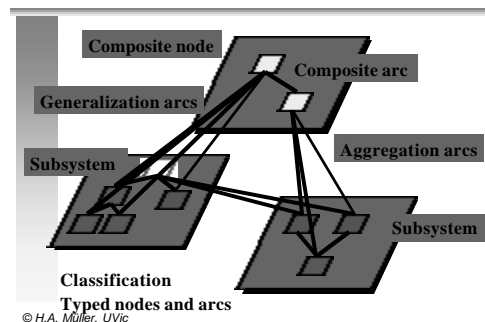
Synthesizing Concepts

- **Build multiple hierarchical mental models**
 - Subsystems based on software engineering principles (e.g., classes, modules, directories, cohesion, data & control flows, slices)
 - Design and change patterns
 - Business and technology models
 - Function, system and application architecture
 - Common services and infrastructure

© H.A. Müller, UVic

11

Graph Model



© H.A. Müller, UVic

12

Reverse Engineering Technology

- Program understanding technology
 - Cognitive models
 - Levels of abstraction
 - Synthesizing concepts
 - Filtering information
 - Slicing and dicing
- Reverse engineering environment
 - Parsers and lightweight extractors
 - Repository and conceptual modeling
 - Visualization engines (graph and web based)

© H.A. Müller, UVic

13

Software Exploration & Graph Visualization Engines

- Visualizing software structure and software architecture through graphs
 - Software Bookshelf, PBS
 - Rigi
 - Datrix
 - SHriMP
 - CIA
 - Imagix
 - GUPRO
 - Many others

© H.A. Müller, UVic

14

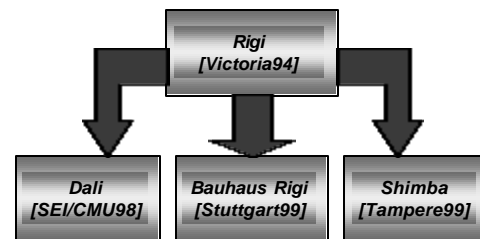
Reverse Engineering and Understanding Theses

- Domain retargetable reverse engineering [PhD Uvic, Tilley95]
- Cognitive design elements for software exploration tools [PhD SFU, Storey98]
- Continuous understanding—Reverse Engineering Notebook [PhD Uvic, Wong99]
- Integrating static and dynamic reverse engineering models [PhD Tampere, Systs2000]
- Architectural Component Detection for Prog. Understanding [PhD Stuttgart, Koschke2000]

© H.A. Müller, UVic

15

Domain-retargetable Rigi



© H.A. Müller, UVic

16

Outline

- The Horseshoe Model
- Reengineering categories
- Reverse engineering strategies
- Reverse engineering theses
- Recent migration theses
- Migration strategies
- Chicken Little strategy
- Language migration
- Conclusions

© H.A. Müller, UVic

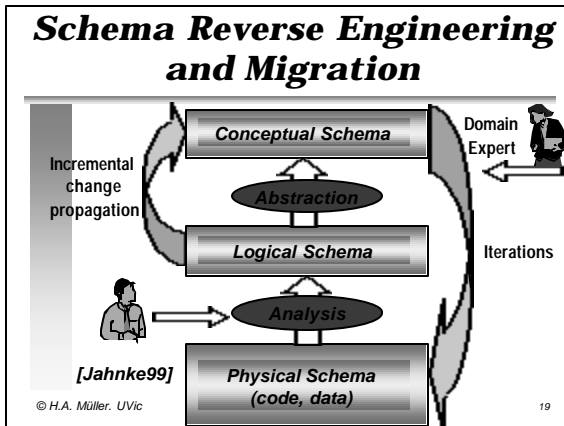
17

Migration Theses

- Varlet data and schema reengineering environment [PhD Paderborn, Jahnke99]
- In place migration of LIS to o-o platforms [PhD Karlsruhe, Koelsch99]
- Migrating C++ to Java [MSc Uvic, Agrawal99, Wen2000]
- An Environment for Migrating C to Java [PhD Uvic, Martin2000]

© H.A. Müller, UVic

18



Migration Objectives

- *Evolving business requirements*
 - *Adapt to e-commerce platform*
 - *Adapt to Web technology*
 - *Reduce time to market*
 - *Support new business rules*
 - *Allow customizable billing*
 - *Adapt to evolving tax laws*
 - *Reengineer business processes*

© H.A. Müller. UVic 20

Migration Objectives ...

- *Software evolution requirements*
 - *Higher productivity*
 - *Lower maintenance costs*
 - *Move to object-oriented platforms*
 - *Inject component technology*
 - *Adapt to modern data exchange technology*
 - *Leverage modern methods and tools*

© H.A. Müller. UVic 21

Migration Objectives ...

- *Software architecture requirements*
 - *Move to network-centric platforms*
 - *Integrate cooperative information systems*
 - *Leverage centralized repositories*
 - *Move from hierarchical to relational db*
 - *Take advantage of web user interfaces*
 - *Provide interoperability via buses and gateways among applications*
 - *Move to client-server architectures*

© H.A. Müller. UVic 22

Common Migration Requirements

- *Ensure continuous, safe, reliable, robust, ready access to mission-critical functions and information*
 - *Migrate in place*
- *Minimize migration risk*
 - *Reduce migration complexity*
 - *Make as few changes as possible in both code and data*
 - *Alter the legacy code to facilitate and ease migration*
 - *Concentrate on the most important current and future requirements*

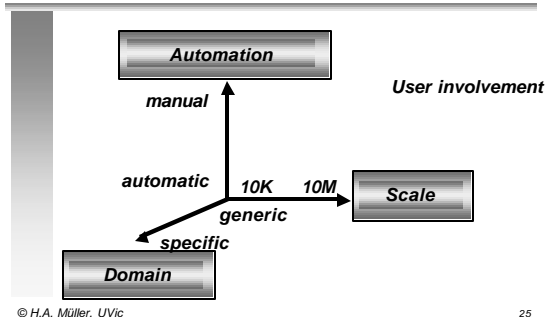
© H.A. Müller. UVic 23

Common Migration Requirements ...

- *Minimize impact on*
 - *users*
 - *applications*
 - *databases*
 - *operation*
- *Maximize benefits of modern technology*
 - *user interfaces, dbs, middleware, COTS*
 - *automation, tools*

© H.A. Müller. UVic 24

Dimensions of Migration Methods and Tools



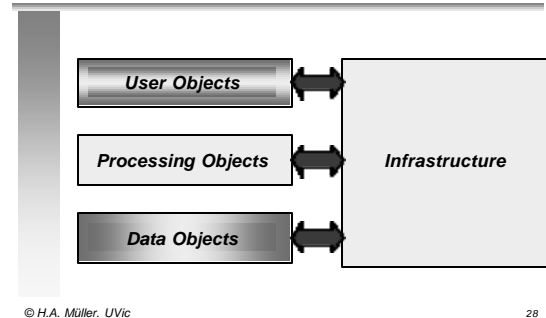
Resistance to Change

- Are some systems more difficult to change, evolve, reengineer than others?
 - Can we define a measure resistance based on business value, existing technology, new technology, evolution pace?
- © H.A. Müller. UVic 26

Separable Tiers

- Decompose legacy system into three layers or application tiers
 - Presentation (interfaces: user and APIs)
 - Processing (application code, functions, business rules, policies)
 - Data services (database)
 - Promotes interoperability, reuse, flexibility, distribution, separate evolution paths
- © H.A. Müller. UVic 27

Application Layers



Classification of LIS Architectures

- Decomposable
 - Separation of concerns
 - Interfaces, applications, db services are distinct components
 - Functional decomposition
 - Ideal for migration

There is nothing more difficult to arrange, more doubtful of success, and more dangerous to carry through than initiating changes.

—N. Machiavelli

Classification of IS Architectures ...

- Semidecomposable
 - Applications and db services are not readily separable
 - System is not easily decomposable
 - Nondecomposable
 - No functional components are separable
 - Users directly interact with individual modules
 - [BS95]
- © H.A. Müller. UVic

Classification of LIS Architectures ...

- **Hostile**
 - no access; no interfaces to internal resources; no APIs
- **Neighborly**
 - minimal APIs, access to a few internal resources (e.g., code, dbs)
- **Friendly**
 - full APIs, access to most internal resources
- [Umar97]

© H.A. Müller. UVic

31

Migration Strategies

- **Ignore**
 - retire, phase out, let fail
- **Replace with COTS applications**
- **Cold turkey**
 - rewrite from scratch
 - high risk
- **Integrate and access in place**
 - integrate future apps into legacy apps without modifying legacy apps
 - IS-GTP [Koelsch99]

© H.A. Müller. UVic

32

Data Warehousing

- **Data is needed for several distinct purposes**
 - on-line transaction processing (access in place)
 - data analysis for decision support applications (extraction of data into an application specific repository)
- **Creates duplicate data**
- **Popular approach**

© H.A. Müller. UVic

33

Gradual migration or “Chicken Little”

- **Rearchitect and transition the applications incrementally**
- **Replace LIS with target application**
- **Language migration**
- **Schema and data migration**
- **User interface migration**
- **GTE [BrSt95]**

© H.A. Müller. UVic

34

Chicken Little ...

- **The intent is to phase out legacy applications over time**
- **In place access is not economical in the long run**
- **More effective, less risky than cold turkey**
- **Allows for independent user interface and database evolution**
- **Incremental**

© H.A. Müller. UVic

35

Chicken Little ...

- **Legacy and target applications must coexist during migration**
- **A gateway to isolate the migration steps so that the end users do not know if the info needed is being retrieved from the legacy or target system**
- **Development of gateways is difficult and costly**

© H.A. Müller. UVic

36

Migration Technology Gateways

- *Mediates between operational components*
- *Insulates components from changes*
- *Translates calls and data between mediated components*
- *Guarantees data consistency between legacy and target system*
- *High reliability is critical*
- *Some commercial gateways available (e.g., SQL-to-SQL translators)*

© H.A. Müller, UVic

37

Migration Technology ...

- *Wrappers*
 - *Wrap existing subsystems*
 - *Wrap dynamically loadable modules*
 - *Wrap communication subsystem*
 - *Wrap existing data*
 - *Wrap end-user and APIs*
- *Transformation tools*
 - *Refine*
 - *TXL*

© H.A. Müller, UVic

38

Opportunistic Migration Method

- *Combination of forward and reverse migration strategies*
- *Forward or reverse migration path per*
 - *operation*
 - *application*
 - *interface*
 - *database*
 - *site*
 - *user*
- *More complex gateways are needed*

© H.A. Müller, UVic

39

Language Migration A Case Study

- *Subject system is a 300 KLOC legacy software system of highly optimized code written in PL/I*
- *Can the system incrementally be translated to C++?*
 - *Transliteration versus object-oriented design*
- *Develop tools which semi-automate the translation process to C++*
- *The translated code must perform as well as the original code*

© H.A. Müller, UVic

40

Research Method

- *Perform a concrete case study with an industrial software system*
- *Investigate methods and tools to automate the process adopted in the case study*
- *Conduct user experiments to improve the effectiveness of the developed methods and tools*
- *Investigate tool adoption problems*

© H.A. Müller, UVic

41

Manual Migration

- *First migration and integration effort was completed by hand by an expert [Uhl97]*
- *10 person-weeks to migrate 7.8 KLOC*
- *Successfully passed all regression tests*
- *Built C++ and Fortran compilers with it*
- *It works ...*
 - *... but migrated C++ code was 50% slower than original PL/I code*

© H.A. Müller, UVic

42

Performance Evaluation

- Expert identified performance bottlenecks
- Hand-optimized migrated code
- Optimized version performed better than the original version [Martin98]
 - Up to 20% better than the original code
 - Now IBM was interested ...
- Results
 - Correct, efficient
 - Translation, integration, optimization heuristics
 - Incremental process

© H.A. Müller, UVic

43

Automation

- Can the translation, integration, and optimization heuristics discovered by experts be integrated into an automated tool?
- How would it affect the performance?
- What existing tools could be leveraged to build such a tool?
- Solution
 - Use Software Refinery, Reasoning Systems

© H.A. Müller, UVic

44

Transformation Process

- Transform PLI/IX artifacts to their corresponding C++ artifacts
- Generate support C++ libraries (macros for reference components; class definitions for major data structures)
- Generate C++ source code that is structurally and behaviorally similar to the legacy source code
- CASCON98 Best Paper [Kontogiannis98]

© H.A. Müller, UVic

45

Results and Lessons Learned

- Semi-automatic transformation of large volume of code is feasible
- Migrated code suffers no deterioration in performance
- Incremental migration process feasible
- Technique readily applicable to other imperative languages
- Tool reduces migration effort by a factor of 10 over manual migration

© H.A. Müller, UVic

46

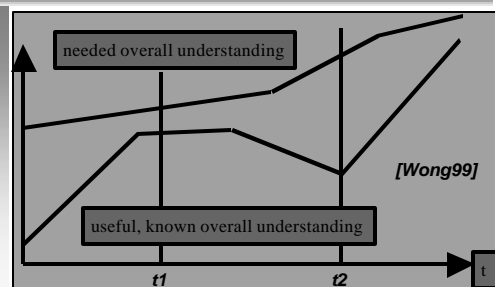
Strategy Selection Criteria

- Business value of legacy application
- Flexibility and growth requirements
- Technical status of legacy application
- Data query requirements
- Data currency requirements
- Integration needs with other applications
- Cost/benefit analysis

© H.A. Müller, UVic

47

The Understanding Gap



© H.A. Müller, UVic

48

Problem

- *What can we do during evolution to ease future understanding and migration of information systems and Web sites?*
- *We know the kind of knowledge we need but it is difficult to obtain from scratch*
- *“Big-bang” reverse engineering when the system becomes “critical” is high-risk*

© H.A. Müller, UVic

49

Solution **Reverse Engineering Notebook**

- *Apply program understanding continuously and incrementally during evolution of the info system or Web site*
- *Use software reverse engineering to redocument existing software*
- *Insert reverse engineering techniques into development*
- *Form an updated base of understanding*
- *[Wong99]*

© H.A. Müller, UVic

50

Web Site Evolution Project

- *Where can a project manager tasked with a web site evolution project begin?*
- *Web site evolution is the same as managing a legacy evolution project*
- *All the methods and technology presented in this talk apply readily to Web site evolution*

© H.A. Müller, UVic

51

Open Questions

- *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 to cope with the inevitable future technology advances?*

© H.A. Müller, UVic

52

Summary

- *Horseshoe model*
- *Data and code reverse engineering*
- *Recent theses*
- *Migration strategies*
- *Incremental strategy*
- *Transliteration*
- *In place migration*
- *Continuous evolution tools*
- *Exciting research!!!*



© H.A. Müller, UVic

An Invitation



© H.A. Müller, UVic

54

Visualization of Web Site History

Filippo Ricca and Paolo Tonella
ITC-irst
Centro per la Ricerca Scientifica e Tecnologica
38050 Povo (Trento), Italy
{ricca, tonella}@itc.it

Abstract

The economic relevance of web sites has increased for several companies, which incorporate sophisticated technologies into complex and large web based systems. As a consequence methodologies and tools are required for their design, implementation and maintenance.

In this paper the evolution of web sites is analyzed, with the purpose of supporting maintenance activities. Colors are exploited to visualize modifications that take place on the site over time. Such analysis may reveal changes not corresponding to the original design or producing undesirable effects.

A tool was developed to implement the analysis of web site evolution. Its application to some examples downloaded from the Web highlights several areas where the extracted information can improve the control on the maintenance phase and provide valuable support.

1 Introduction

Maintenance of web sites is becoming a crucial issue for several companies, whose business more and more depend on their presence on the net. While tools and techniques for the development of web sites with increasingly advanced features and with appealing interfaces obtained a great attention, the problem of their evolution and modification was somewhat neglected.

Similarly to the development of software systems, the production of high quality and reliable web sites can be achieved only if proper methodologies and techniques are adopted. Moving from the development of small, personal pages to big and complex corporate sites cannot be approached without considering a phased, incremental process with well defined activities. Moreover, the maintenance phase should be anticipated, designing the site for change and evolution, and then faced with the support of all available technologies. In fact, it is likely that maintenance

is going to absorb a very relevant effort, as it occurs with software systems.

In this paper, history analysis is presented as a means to be employed when the web site enters maintenance and needs to evolve retaining and possibly improving its quality. The evolution of a web site can be represented by using colors as time indicators. In this way the history of individual pages and links can be shown to the user in a compact and expressive form, thus indicating the origin of the different regions of the site, and highlighting updates that may degrade the original site structure.

Web site history can be visualized by **ReWeb**, a tool developed to download and analyze web sites. Its graphical interface provides search and navigation facilities, and popup windows with information on individual pages. It employs colors for the display of the site history. The results obtained by using the tool with some real world web sites are presented to illustrate its potentialities.

The existence of problems, in web site development, similar to those encountered in software before the advent of software engineering was recognized in [6], where the evolution of web sites is characterized by means of metrics, with the purpose of discovering potentially troublesome patterns.

The adoption of visualization techniques aimed at a compact and meaningful representation of the history of a system was mainly investigated in the framework of traditional software systems [1, 3, 4, 5]. Using different colors to display different versions of a software artifact allows high level and fine grain views of the evolution of the system.

2 History analysis

Knowledge about the history of a web site is useful to document the events leading to its current organization and to identify the reasons for potential structural problems. The analysis of a web site evolution requires the ability to compare successive versions of its pages and to graphically display the differences.

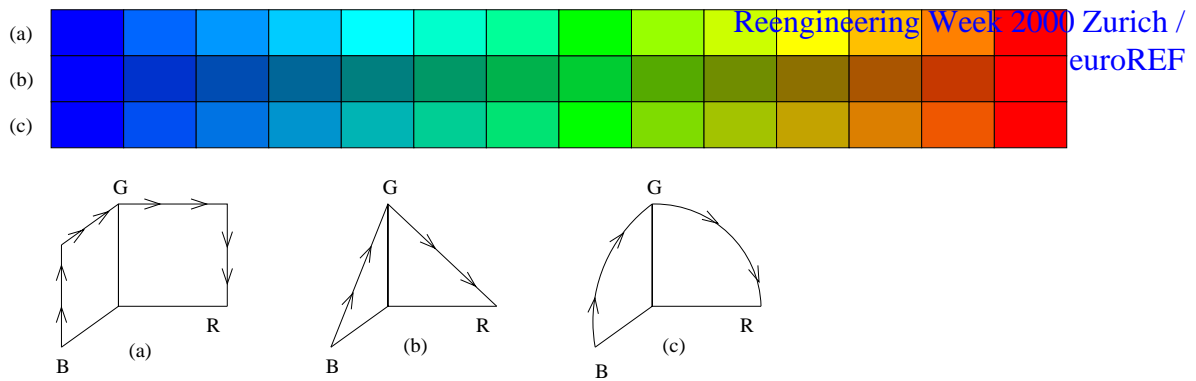


Figure 1. Paths in the RGB color cube, associated to the page introduction/modification date, going from the past to the present time. Colors related to 14 samples taken on the three paths are displayed at the top.

2.1 Difference computation

Given two versions of a web site, downloaded at different dates, their comparison aims at determining which pages were added, modified, deleted or left unchanged. Tracing the evolution of an entity over time requires that the notion of identity is defined, so that the same object can be recognized at different times. In our case the problem is highly simplified if the assumption is made that the page name is preserved, when moving from a version to the next one. In a graph representation of web sites, nodes correspond to pages, and an edge between two nodes exists if a hyperlink connects the pages associated to the nodes. A traceability map can be defined between nodes in the graph representation of the old site and of the new one, which simply pairs nodes with the same page name. All nodes in the old graph with no corresponding node in the new graph represent pages that were deleted from the site, while all nodes in the new graph without counterpart in the old graph are added nodes. When a corresponding node can be found in the old graph for a node in the new graph, it is classified either as a modified node or a node left unchanged according to the output of a character-by-character comparison of the associated HTML pages.

The choice to compare pages with preserved name among successive versions simplifies history traceability, since the mapping between old and new pages is given, and has not to be reconstructed. The drawback is that an unchanged page, whose name is modified, is considered as a deleted page, in the old graph, and an added page, in the new graph. Since the name of a page is its unique identifier, used by the other pages to access it and possibly referenced by external pages from other sites, it is likely to be preserved among successive site versions. In fact, changing the name of a page has an impact on all referencing pages, which must update their links, with the risk of missing some up-

date. Therefore the assumption of name preservation seems a reasonable one.

2.2 Visualization

The graph representation of a web site can be enriched with information about its history, by coloring the nodes and associating different colors to different time points. In particular, a scale of colors ranging from the blue (B), going through the green (G) and reaching the red (R) can be employed to represent nodes added/modified in the far past, in the medium past or more recently.

Figure 1 gives three paths, in the RGB representation of colors, that can be uniformly divided into segments to be associated to different time intervals. 14 samples taken on the three paths are displayed at the top of the Figure.

Our choice of the "best" color sequence was in favour of path (a), since it provides a high number of intermediate colors, that can be visually distinguished, and at the same time it ensures a continuous scale from B to R, so that the distance in the past and the time proximity can be easily assessed by visual inspection.

History visualization can also be applied to the links. When a link from a page to another page is first introduced, the edge in the graph representation of the site assumes the color of the source node, i.e., of the enclosing page. Then, such a color is maintained even when the page evolves, changing its color, if modifications do not affect the link.

3 The ReWeb tool

The **ReWeb** tool consists of three modules: a Web Spider, an Analyzer and a Viewer.

The Web Spider downloads all pages of a target web site starting from a given URL. Each page found within the site is downloaded and added to a directory, named with the

same name of the web site, but differentiated by date of downloading. The HTML documents outside the web site host are not considered.

The Analyzer uses different versions of the downloaded web site, the Web Spider's output, to calculate the difference between each two successive versions of the site, as explained in Section 2. The results produced in this phase are stored in files.

The Viewer provides a Graphical User Interface (GUI) to display the output of the history analysis. The selection of a site version is done through a colored button menu on the left of the interface (see Figure 2). When a button is selected, a colored graph showing the history view appears in the right part of the GUI. The graphical interface supports a rich set of navigation and query facilities including zoom, search, focus and HTML code display. The facilities for focusing on and searching a node are useful when the visualized graphs are very large. With the focusing facility it is possible to focus the view on a selected node, and to specify the number of upward and downward levels to be displayed, i.e., the depth of the focus. The searching facility is used when a specific node has to be found within the graph. Another interesting opportunity is the display of a report containing simple metrics at site level, like number of HTML pages, number of links, Lines Of Code (LOC) and *error links*, i.e., links leading to "ghost", non existing pages in the same host.

Web Spider and Analyzer are written in Java, while the Viewer is based on Dotty¹. **ReWeb** is not complete yet. In fact, future work will be devoted to improving the robustness of the tool, widening the spectrum of analyzable sites and enriching the set of facilities. The current version of **ReWeb** downloads and analyzes sites that are implemented entirely in the HTML language (including frames). In the future, "adventurous" presentation mechanisms present in the web technology, such as scripting languages, applets and dynamic objects, will be handled.

4 Experimental data

Several web sites of different type – educational, commercial, institutional – were chosen for analysis. To study their history, the sites have been downloaded every day by **ReWeb** for over three months. Among all versions, only those *significant* were preserved, i.e., those that exhibit some change. A summary of their features is shown in Table 1. The size of the sites is between 17 and 400 HTML pages, the LOC number is between 1081 and 26073, and the maximum number of significant versions is 8. 9 sites use frames, while 3 use features beyond pure HTML (HTML+). For such sites it is possible that the collected information is

¹Dotty is a customizable graph Editor developed at AT&T Bell Laboratories by Eleftherios Koutsosofos and Stephen C. North.

a subset of the complete set of pages, since some links could have been missed.

The site www.cartercopters.com (an aircraft manufacturer), used also in [6], was chosen as an example because it had the highest number of changed pages during the three months of monitoring. New pages and links were added, while others were removed. The significant versions of this site are four, at the dates: 21-10-1999, 24-11-1999, 23-12-1999, 25-12-1999. The tool **ReWeb** associates these dates to colors blue, light blue, yellow and red, thus permitting an immediate comparison between different versions. Colors are obtained by uniform sampling of path (a) in Figure 1. At the initial date the graph representation of the web site is all blue. When visualizing the second version, it is possible to note that there are nodes and edges light blue, i.e., the web site evolved. Specifically, the pages `pressrel14.html`, `pressrel.html`, `contents.html`, `index.html` are light blue. In fact, the first one was added to the site, while the others changed, with respect to the previous versions. It can also be noted that all new edges are links toward the new page and that two edges were deleted. The third version introduces eight yellow nodes plus several yellow edges. There are five new nodes, while three changed with respect to the version dated 24-11-1999. In the last graph it is possible to see five red nodes and several red edges. It is not practical to display the entire site in a window, because it is too large, but it is possible to see a portion of it by exploiting the **ReWeb** focus. If `pressrel13.html` is chosen as focus, with upward and downward depth equal to 1, and date 25-12-1999, the graph in Figure 2 is obtained. Node `pressrel14.html` and its links were introduced on 24-11-1999. In fact their color is light blue. Page `pressrel.html` changed on 23-12-1999 and is consequently yellow. Pages `contents.html` and `index.html` changed on 25-12-1999, and are red, while the other nodes never changed and are blue, the color associated with the first date.

A phenomenon of complete restructuring, or better replacement, happened for the site www.itc.it (our cultural Institute). This fact could be observed, with the help of **ReWeb**, at the date 17-12-1999, when all nodes in the history graph changed color. It agrees with the considerations on web site evolution in [2], where replacement is conjectured to often substitute incremental update. A view of **ReWeb**, particularly convenient in these cases, is the history representation with percentage bars [4] describing, in compact way, the percentages of nodes with the same color, i.e., the pages that were added or modified at the same time point. This view is interesting because main changes in the evolution of a site can be easily detected, being represented by "large" changes in color. An example of this view is provided in Figure 3.

Web sites	Pages	Links	LOC	Frames	HTML+	Last download date	Total versions	Significant versions
www.sta.co.uk	17	68	1081	no	no	14-1-2000	96	1
www.anasi.it	22	64	6110	no	yes	15-1-2000	89	3
www.storea.com	44	81	2340	yes	no	12-1-2000	28	2
www.ubicum.it	57	232	3132	yes	no	12-1-2000	27	2
www.alsi.it	68	140	9512	yes	no	15-1-2000	29	3
www.cartercopters.com	76	243	9123	no	no	14-1-2000	95	4
www.itc.it	79	616	6543	yes	no	23-12-1999	89	8
www.automatica-casali.com	95	130	5118	yes	yes	12-1-2000	11	1
www.psy.it	128	216	24972	no	no	14-1-2000	82	8
www.dei.unipd.it	238	583	7486	no	yes	14-1-2000	91	6
www.iqa.org	245	1290	28690	yes	no	11-1-2000	10	1
www.sund.ac.uk	291	3121	19095	no	no	14-1-2000	91	6
www.psy.unipd.it	317	433	14685	yes	no	14-1-2000	87	4
www.artifer.com	358	2741	13758	yes	no	11-1-2000	11	1
www.accademiadibrera.milano.it	400	291	26073	yes	no	14-1-2000	9	1

Table 1. Analyzed web sites. Frames is “yes” if the corresponding site uses frames. HTML+ is ‘yes’ if the site uses “adventurous” presentation techniques, like Java, Java-script, dynamic objects, etc.

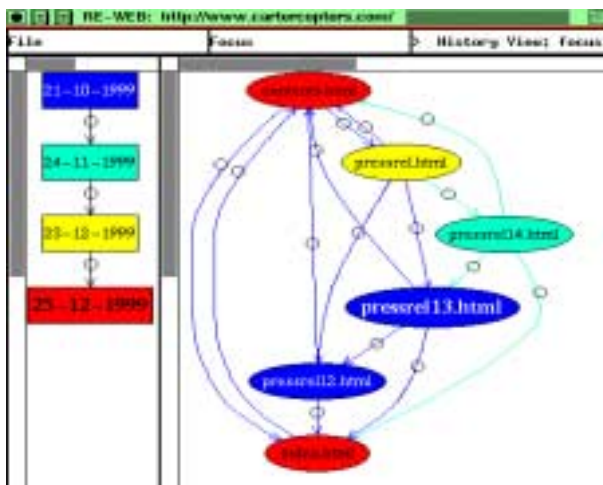


Figure 2. Example of ReWeb history view depicting the site www.cartercopters.com at date 25-12-1999, focused on node pressrel13.html.

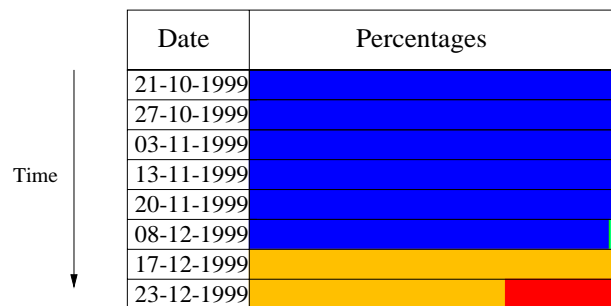


Figure 3. Example of history representation with percentage bars, depicting the site www.itc.it.

References

- [1] M. J. Baker and S. G. Eick. Visualizing software systems. In *Proceedings of the International Conference on Software Engineering*, pages 59–67, Sorrento, Italy, May 1994. IEEE Computer Society Press.
- [2] E. J. Chikofsky. Aspects to consider for understanding web site evolution. In *Proc. of the International Workshop on Web Site Evolution*, Atlanta, GA, USA, October 1999.
- [3] S. G. Eick, J. L. Steffen, and E. E. Sumner. Seesoft – a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, November 1992.
- [4] H. Gall, M. Jazayeri, and C. Riva. Visualizing software release histories: the use of color and third dimension. In *Proceedings of the International Conference on Software Maintenance*, pages 99–108, Oxford, England, August-September 1999. IEEE Computer Society press.
- [5] R. Holt and J. Y. Pak. Gase: Visualizing software evolution-in-the-large. In *Proceedings of the Working Conference on Reverse Engineering*, pages 163–166, Monterey, 1996.
- [6] P. Warren, C. Boldyreff, and M. Munro. The evolution of websites. In *Proc. of the International Workshop on Program Comprehension*, pages 178–185, Pittsburgh, PA, USA, May 1999.

Accessing
Legacy Mainframe Applications
via the Internet

by

Harry M. Sneed
Prellerweg 5
D-82054 Arget, Bavaria
Germany
Fax: #49-8104-669920
Email: Harry.Sneed@T-Online.de

Keynote for

Web Site Evolution (WSE-2000)

Zürich, March, 2000

Abstract

This paper discusses the possibilities and problems involved in accessing legacy applications and data on a mainframe via the internet. It points out the fact that the programs can be reused as they are, but that this no long term solution due to the poor quality of most mainframe programs. If the legacy systems are to have any long term value as web applications, they have to be reengineered. To this end, the paper proposes a five layer architecture with a web browser, a CGI Gateway, a control module, n processing modules and a data access shell. With this reengineering solution the original processing logic is retained, but within a new internet compatible architecture. The positive side of this solution is it's viability and compatability with the internet mode. The negative side is it's costs which are close to those of reimplementing the program from scratch. If the processing logic of a program does not make up a significant portion of the code, it may be better to redevelop existing programs in an internet language.

Keywords : Legacy Software, Mainframes, Cobol, CICS, IMS, Internet, Web, HTML, XML, CGI, CGICOBOL.

Accessing Mainframe Applications via the Internet

The key to successful internet exploitation in a typical mainframe-oriented user organisation is a combination of the internet user interaction facilities with the functionality of existing programs and the information available in existing databases. Business organisations have invested millions of dollars to implement their functionality in mainframe programs -partly online and partly in batch modus. Simultaneously, they have invested almost as much in building up complex, integrated databases, where most of the company data is stored and, in any case, the critical data. In recent years they have made an additional tremendous investment in making their system Y2K compliant and, in Europe, converting their local currencies to the Euro. Each mainframe application represents a monetary value of somewhere between 10 and 100 million dollars. (1) Large corporate organizations such as banks, insurance companies and trading companies may have hundreds of such applications ranging from back office operations like accounting and inventory management to front end operations like stock trading and sales support. With only 50 applications this already amounts to more than 1 billion dollars. The Gardner Group estimates that the average mainframe application portfolio of the Fortune 500 companies is more than 5 billion. (2)

In light of these economic realities and the fact that it took more than 20 years to reach this level of automation, it is absurd to even contemplate replacing these accumulated assets within a short term period. Even the installation of standard ERP systems requires a significant investment in effort and more than a year to accomplish. (3) Besides the fact that not all applications can be standardized, this solution also entails a break with the company past and a problem of integration with existing applications. Thus, not even standard systems appear to be a real short term solution. Companies must continue to utilize their current legacy systems for accomplishing core activities in the years to come. The problem is how to reconcile these systems with modern technology such as the internet. (4)

The strength of current mainframe applications lies in their data storage and processing capability. Huge volumes of data may be stored in hierarchical, relational and networked databases where they are relatively easily accessed. The mainframe database systems are reliable and performant. Most of them are nonrelational. Mainframe programs, in Cobol or PL1, may not be object-oriented but they still perform their task in a satisfactory manner. They have suffered from years of quick and dirty maintenance patches and certainly could stand some renovation, but this does not distract from their inherent business value. Their functional ripeness is the result of years of trial and error, something that would have to be repeated with new applications. By and large their results are correct, and this is really all that matters.

The weakness of the current legacy systems has always been their interface to the user. This started with the primitive listings they produced on mass laser printers and continued with the primitive maps they used to collect data. The 3270 terminal can certainly only be viewed as the beginning of a long technological development in user/computer interaction. For some applications it was adequate, for most not. When distinguishing between the three architectural layers of legacy information systems, it is evident that the data access layer is good, the processing layer satisfactory and the presentation layer poor. Therefore, it is obvious that the greatest short term benefit can be gained by upgrading the presentation layer. The question is how?

CGI-1

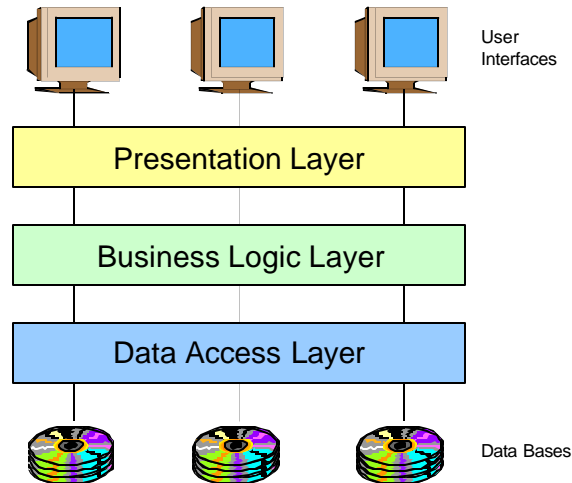


Figure 1: Ideal 3-Layer Architecture

Alternatives to accessing Legacy Systems

There are three general alternatives to improving the access to legacy applications and data bases

- screen scratching,
- client/server and
- internet connections.

Screen scratching entails emulating the 3270 terminal on a PC-workstation with a graphical user interface. Data is submitted in window type text boxes with menus, icons and selection buttons, then transformed into the original map format and transmitted to the program. All of the logic is still contained within the host program, there is no content checking on the workstation and no interaction with the user. The PC is used as a data collection device to create and display the original 3270 masks. The displays are subject to the limitations imposed by the masks, i. e. only that data can be displayed which is contained in a map. Thus the scrolling and lookup features of the output display are no better than what they were on the 3270, only graphical. The biggest advantage of this scratching technique is the fact that the mainframe program need not to be altered. It may remain exactly as it was, since the data structures it processes - the CICS-BMS or IMS-MFS maps - have not changed. This alternative has been widely used in practice exactly for that reason. (5)

The client server solution implies separating the program into a front-end and back-end. Here a program exists on the workstation which interacts with the user in collecting and displaying the data. A good part of the original program - the map formatting, data extraction, data editing, etc. is moved to the client side, leaving only a rump program on the mainframe. This of course greatly improves the data submission and display possibilities, but it also creates a problem. Not only does the existing mainframe program have to be changed, i.e. stripped of much of its original functionality, but the user has to also decide what remains on the server - in this case the host - and what is transferred to the client.

A significant portion of mainframe programs is devoted to data content checking and editing. Much of this is of a formal nature such as type and range checking. However a significant part is checking against existing databases. Since the databases are only available on the mainframe, this part has to be kept in the original program. This means that even the data checking is divided between client and server.

If the mainframe programs had been designed from the start to separate the user interface interactions from the data processing and the data access, there would be no great problem in implementing the client server solution. One need only cut out the user interface interactions which on a typical CICS program amount to about 40% of the code and reimplement them on the client side leaving the rest of the program as it was. However,

unfortunately, this is seldom the case. Most mainframe online programs have been implemented in such a manner that the user interface interactions - the data collection, editing, checking, formatting and display functions - are deeply intertwined with the data processing and data access functions making it necessary to first restructure or even to rewrite the original program in order to separate it into a client and a server side. This is the main reason for the reluctance to move to a client/server solution. In the end one winds up with a totally different set of legacy programs.

CGI-2

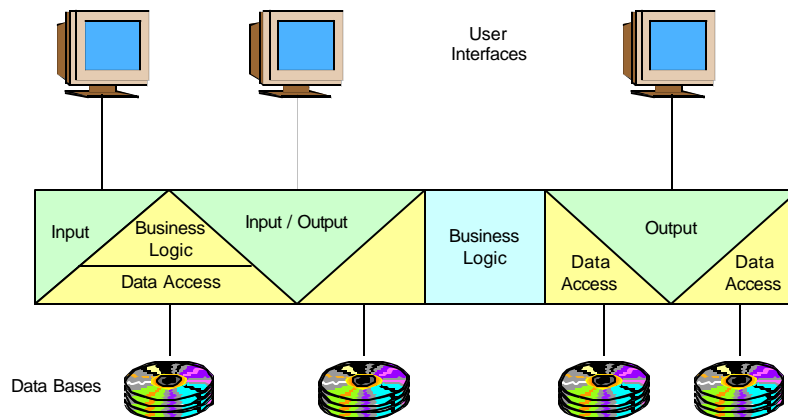


Figure 2: Real Intertwined Architecture

The implications of these problems have unfortunately not been adequately considered by those who advocate the client/server solution, i.e. encapsulating the existing mainframe transactions in an object-oriented architecture. Much of the literature on this subject does not address these detail issues which prohibit implementing the solution. (6) Alone the fact that many online data checks are made against the database means that the client side must access the data. So not only is there an interface to the application program, but also to the application data base as well. If these interfaces are not properly designed, the client/server solution will have performance problems because of the many interactions between the workstation and the host. On the other hand, if these interactions are optimized it entails redesigning the host program. This causes a dilemma since the purpose of the project is to reuse the original code to the greatest extent possible. However, if this is done, then one has to limit the capabilities of the client program to that of a screen simulator, thereby reverting to the screen scratching alternative.

Experience has shown that almost all successful client/server implementations involving mainframe programs have led to a reimplementaion of those programs to fit the requirements of the client side. Very few of the mainframe programs could be reused as such. This is a bitter consequence of architectural mismatch. (7)

The internet solution promises to be a somewhat better solution than the client/server one was, but it still inherits many of the problems mentioned above. With the internet solution the user is presented a web page in which the functions of the existing programs are offered. Having selected a function, the use must than supply the input to that function which is first formally checked and then passed to that function. The function invoked accesses the database and produces a result which is presented within a window in the web page. The data is passed in a HTML format which is interpreted by the target program. The receiving program must have a parsing routine to scan the HTML document and to extract the required data from it. The program parameter list is created from the HTML fields. These technical aspects of data interchange are all very interesting and will be discussed in more detail later on, but they do not detract from the main problem of getting the legacy program to fit in the first place.

In an article in the American Programmer J, Tibbetts and B. Bernstein discussed the possibilities of accessing legacy application on the web as early as Dec. 1996 . These authors suggested three alternative solutions to accessing the systems

- via the presentation layer,
- via the business logic layer and
- via the data access layer. (8)

Presentation Access

Presentation access is another form of screen scratching. The original CICS map is replaced by a HTML, or XML, form which is processed by the target program instead of the map. This appears to be a simple and straight forward solution on the surface, but in fact it is not. It entails a significant amount of recoding since much of the code in a typical CICS or IMS

program is devoted to interpreting and setting attribute bytes, responding to function keys and handling exception conditions, all of which are directly related to the 3270 communication techniques. Since this code is often intertwined with the business logic, it may result in a major reengineering of the program to take advantage of the new features offered by the HTML or XML data exchange form rather than a 3270 map. The advocates of this solution tend to underestimate the impact of the communication media on the structure of the program.

CGI-3

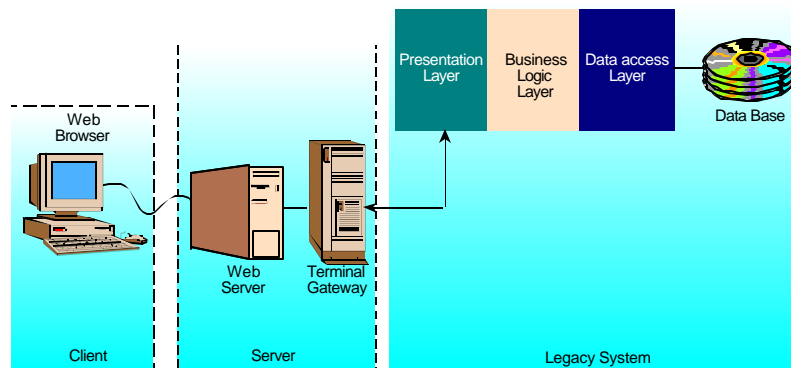


Figure 3: Presentation Access

Business Logic Access

The alternative of accessing the business logic directly presupposes that the business logic is implemented separately from the presentation logic, an assumption that seldom holds in a typical legacy program. Theoretically, it is possible to insert a thin control layer which accepts the data extracted from the HTML page and invokes the appropriate subroutines either by a „CALL“ or by a „PERFORM“. The business logic subroutines process the request, give back a result and this is forwarded back to the client. The problem here is again the fact that the presentation logic is entangled with the business logic. There are seldom cleanly separated business subroutines such as COBOL sections or PL/1 procedures without any CICS or IMS operations. The processing of business rules is interspersed with the map processing operations which have to be taken out. However, removing them can cause errors and alter the logic of the program. Thus, there has to be an automated tool to take care of this.

CGI-4

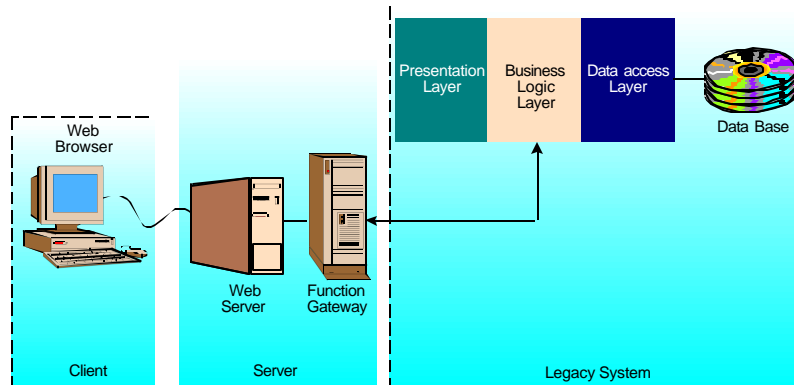


Figure 4: Business LogicAccess

Data Access Access

The third alternative is to bypass the presentation and business logic layers and to enter at the data access level. Here one assumes that the client, for example a JAVA applet only needs access to the mainframe data. The data processing can be done by the applet itself. Since much of the business logic of legacy online programs is rather trivial, this assumption is often true. However, many legacy programs, if not most, do not have a separate data access layer. The DL/1 or SQL access operations are intertwined with the CICS or IMS presentation operations and the COBOL or PL/1 business logic. There is no way to invoke them independently without executing the other operations as well. Not only that, but the operations are often dependent on one another, making it impossible to leave some out. Direct access to the data access layer is only possible where a separate data access shell already exists or where it is created. Then the modules of this shell can be called directly with the data exchange format as a linkage area as if they were named subroutines.

CGI-5

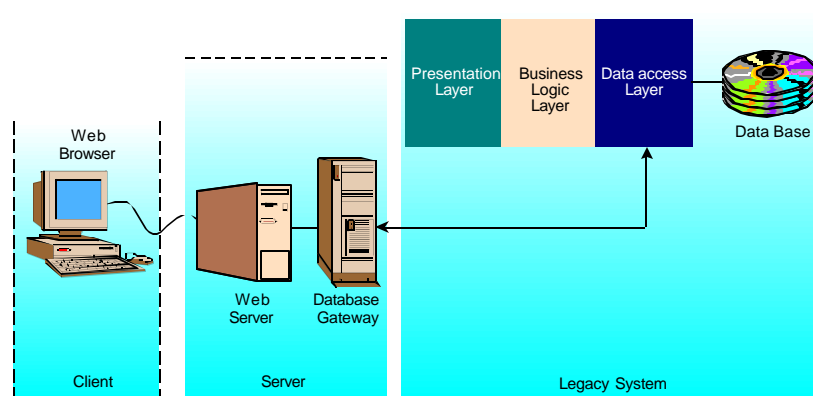


Figure 5: Database Access

Internet Gateways

IBM has made a significant effort to technically integrate the existing mainframe systems into the internet world. To this end the traditional teleprocessing monitor CICS has been extended to include web pages from which the user can access legacy programs and data bases. Enterprise JAVA Beans - EJB - is the main environment in which this takes place. Via CICS, JAVA applets can access the mainframe data bases in VSAM, IMS or DB2 or they can invoke CICS/COBOL programs which, in turn, may call other COBOL programs. To achieve this CICS programs on the IBM mainframe must first be extended with JAVA, HTML or XML interfaces. Then they can be called by the CICS transaction server which receives its requests via the internet message queue. Later the CICS programs can be rewritten in JAVA. Thus, the reuse of existing COBOL programs is only intended as a transition solution until such time as these programs can be replaced by newer ones designed explicitly for the internet.

With the aide of CICS, end users can now easily browse through VSAM files and relational databases and select functions to be performed on the data. This represents an entirely new approach to conventional data processing where the user was restricted to following a predefined process using fixed forms for data input and display. With the new environment-CICS & EJB- the user now defines the process adhoc as he goes along, combining data and functions as he deems fit.

Of course, the policy of IBM is based on marketing strategies. Whether the end user really wants so much freedom to decide depends on the type of application he is working on. The point here is that the world leading mainframe vendor is pushing the internet as a successor to the prevailing online transaction processing. Obviously IBM views the Web as the standard man/machine interface of the future, giving a clue to the rest of us.

The basis of the IBM solution is a product called Net.Data. Net.Data allows web developers to easily build dynamic internet applications using special „web macros“ which combine the simplicity of HTML with the power of dynamic SQL. It provides database connectivity to a variety of data sources including DB2, IMS and VSAM. Web-based solutions with Net.Data can contain INCLUDE members as well as Web URLs, allowing

modularization and sharing of application components. For data persistence between web pages and across web transactions, Net.Data supports HTML variable substitution and Netscape cookies which keep track of related visits to the web site for caching. In addition, Net.Data features macro capabilities including conditional logic for more flexible development. Finally, Net.Data offers full support for JAVA, the language of choice for the web. It is fully compatible with VisualAge for JAVA and supports JAVA Script in validating data entered at the client's web browser. Furthermore Net.Data is connected with NetObjects Fusion, a graphical template based website authoring tool for generating web pages. Together with NetObjects Fusion, Net.Data allows users to create visually rich web pages, connecting them to existing business data and applications. (9)

IMS Web Studio

IBM offers web gateways for both IMS and CICS. Both are members of IBM's e-business Enterprise Connectors family. IBM Connectors are gateway products that enable one to access legacy applications and data over the internet using the web browser. The product for IMS is IMS Web Studio. IMS Web Studio uses information available from IMS to generate code necessary for mapping between the web and IMS. The main source of information is the IMS Message Format Services - MFS - from which the map definitions are extracted. From the downloaded maps, IMS Web Studio generates a set of C++ classes for parsing and communicating with IMS transactions. For this purpose, the generated C++ classes provide a common gateway interface-CGI-for communicating from the web browser to IMS. They also create a HTML form for inputting data as well as an output form for displaying the results of a transaction on the web browser. (10)

IMS Web Studio can be downloaded from the IBM Website and run in any Windows NT or Windows 95 environment. It allows the user to browse and download the MFS maps from the mainframe. From them it then produces the C++ conversion classes as well as the HTML forms for the new web transactions. Finally, it provides a makefile to compile and link the generated C++ classes to create an executable CGI binary. With IMS Web Studio it is possible to access both IMS transaction programs and IMS databases.

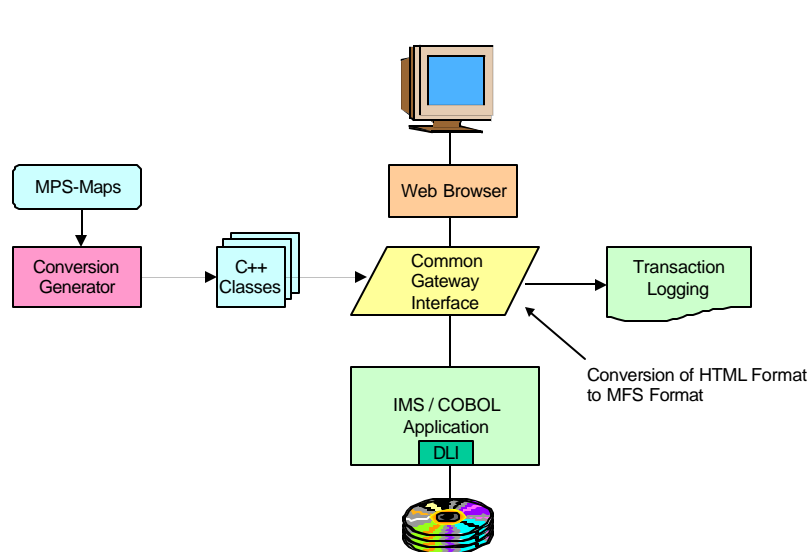


Figure 6: IMS Web Studio

CICS Internet Connector

The product for CICS is the CICS Internet Connector, a CGI script that makes a web browser appear like a 3270 terminal. This common gateway provides an interface between the web server and the CICS application,

converting 3270 data streams into either HTML or XML formats. The CICS application can then be accessed by any web browser with no change to either the browser or the CICS program. This is the classical screen scratching or facelifting approach. Of course the HTML or XML form could be reused or totally redesigned, but in this case the target program would also have to be reengineered or rewritten. (11)

The CICS Internet Connector enhances the functions of the web server by exploiting a standard web server mechanism known as the common gateway interface, dynamically building web pages containing CICS data for transmission to and from the web browser. It also provides automatic translation between CICS 3270 data streams and HTML using the CICS Extended Presentation Interface. The final result is a virtual terminal or CICS-BMS proxy on the mainframe which simulates the original CICS transaction. In this way the web browser is able to send and receive messages from the CICS program as if it were a standard Web application.

CGI-7

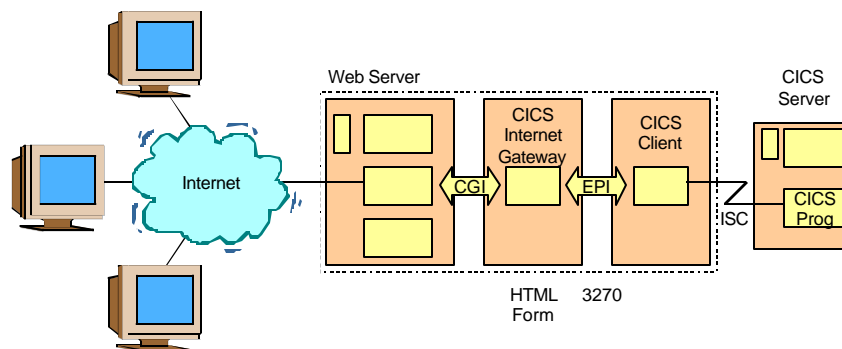


Figure 7: CICS Internet Gateway

COBOLCGI

It is obvious from the products described above that there are no real technical barriers to the reuse of legacy programs and databases in the internet. The question is whether to leave the programs as they are and adjust the web pages to the programs or to design new web pages and adjust the legacy programs to them. Ron Langer, a Fujitsu COBOL/internet consultant, advocates the second solution. (12) He recommends a product from England Technical Services called COBOLCGI which allows COBOL programs to communicate directly with the web.

COBOLCGI provides a CGIWRITE routine to display HTML information on a Web browser and a CGIREAD routine to obtain information from the Web browser much in the same way as the COBOL DISPLAY and ACCEPT commands work. To send a message one would use the CALL command:

```
CALL "CGIWRITE" USING BY CONTEXT
    "<p> Content. </P>",
    & x "00".
```

It is also possible to send variables in the same manner as depicted in the following command:

```
CALL "CGIWRITE" USING BY CONTENT
    "<P>",
    Data-Name,
```

```
“</P>“,
& X“00“.
```

On the other hand it is possible to receive data extracted from the HTML form as shown below:

```
CALL „CGIREAD“ USING
PARAM-1,
PARAM-2,
.....,
PARAM-n.
```

By placing these calls to COBOLCGI in a new control module, it is possible to invoke the individual sections of an existing COBOL program via the Linkage Section as if they had always been intended to serve as subroutines. Of course, they must first be stripped of their own original input/output operations, so as not to conflict with those of CGI.

The structure of the control module could be as follows:

CGI-CONTROL SECTION.

```
CALL “CGISETUP“ USING      File-Name
.....
CALL “CGIWRITE“ USING     HTML-Line.
.....
CALL „CGIREAD“ USING     Func-Code,
                          Param-1,
                          Param-2,
                          Param-3.
.....
CALL „CHECK“ USING       Func-Code,
                          Param-1,
                          Param-2,
                          Param-3,
                          Ret-Code.

IF Ret-Code = ZERO
    EVALUATE Func-Code
        WHEN 1
            CALL “Func-1“ USING      Param-1,
                                    Param-2,
                                    Param-3,
                                    Ret-Code
        WHEN 2
            CALL “Func-2“ USING      .....
        WHEN OTHER
            CALL “Error-Func“ USING  Ret-Code
    END-EVALUATE
END-IF.
IF Ret-Code > ZERO
    MOVE ERROR-MSG (Ret-Code) TO ERROR-LINE-MSG
```

CALL „CGIWRITE“ USING ERROR-LINE
END-IF.

The databases are accessed via the existing SQL or DLI macros which have been factored out into a separate data access shell. They are called from the processing modules where they originally occurred. There the EXEC SQL macros are replaced by CALL commands to the data access shell just as the original EXEC CICS macros have been replaced by CALLs to the internet gateway shell. The final architecture of the reengineered programs is depicted in Figure 8.

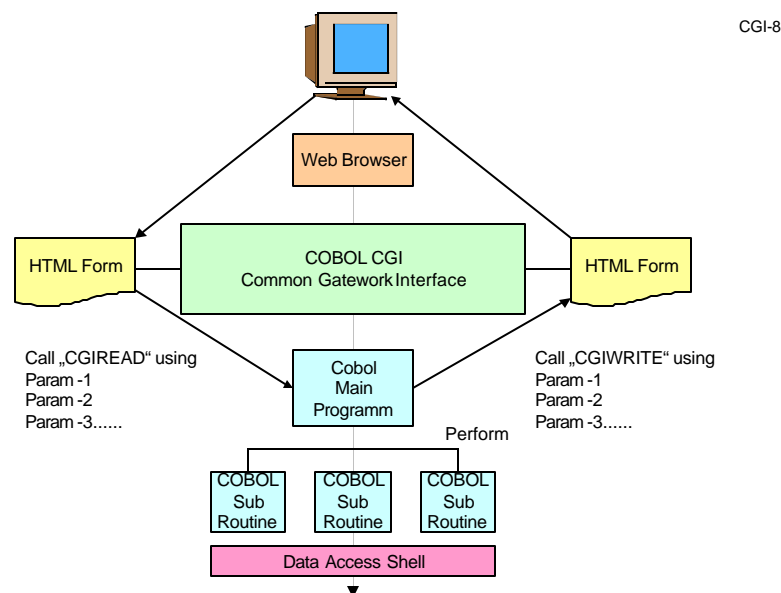


Figure 8: COBOL-CGI Internet Gateway

There are now four layers of code. The top layer is the internet gateway, below it is the control module, then the processing modules extracted from the original program and, at the bottom, the data access modules. In this CGI model, a new internet architecture is introduced while preserving the original business logic.

XML Wrapping Solution

Another solution to the internet connection of mainframe COBOL programs is the one developed by the author himself. This solution combines the facilities of the CICS Internet Gateway from IBM with the use of XML forms. XML, the extended markup language, has become a worldwide standard for exchanging documents. It has evolved out of the original SGML of IBM and the HTML language for presenting web pages. XML combines the data structure specification with the data itself using markers. An XML document has a header, in which the general attributes of the document are described, and a body for the individual data elements or texts being passed. In many respects it resembles a map. The advantage of these markup documents is that they are both human readable and machine processable. What is more, they are producible and consumable by almost any web browser, since this is the W3C standard for exchanging information. (13)

In the old transaction processing programs information was passed from the terminal to the programs by means of a map. Maps reflected the physical characteristics of the terminal. There was a direct interaction between the maps and the programs. Programs could set the color of fields, underline them, cause them to blink and even change the field formats dynamically. A significant portion of the programs was devoted to processing the maps. In principle, similar operations can be carried out using an XML form. However, this is not the task of the server program. This should be left to the client. Only the pure data should be passed to the server with one exception - the name of the function to be processed. By placing the name of the function to be processed in the header of the XML form, the client is able to control the server. In addition to a given program being loaded, also a specific

section of that program can be referred to. The old CICS Handle Aid Condition can even be emulated in this manner.

In the COBOL/CICS program RECEIVE and SEND macros are replaced automatically by CALLs to the XML wrapper routines. In the case of RECEIVE the XML form passed to the server in the Linkage Section is parsed and the data extracted and converted. The ASCII character data is converted to the corresponding COBOL data types depending on the COBOL data description structure read by the conversion routine when it is compiled. The result is a COBOL data structure in the Working Storage Section of the target program replacing the previous map. In the case of SEND the COBOL data structure in the Working Storage Section is converted to an XML form in the Linkage Section for return to the client. The conversion is handled in the same way as the XML to COBOL conversion only in inverse mode.

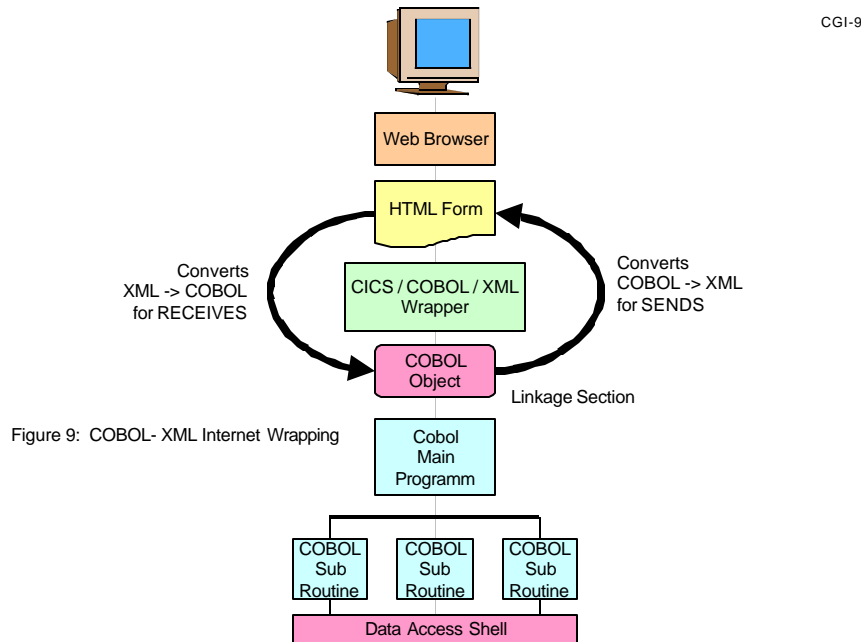


Figure 9: COBOL- XML Internet Wrapping

The advantage of this approach is that the original CICS program can still be reused with a minimum of change, while at the same time allowing the program to be restructured. By inserting a central control section, it is possible to invoke specific subsections depending on the function name indicated in the header of the XML form. This wrapper solution is data driven and based on a standard data interchange format which even COBOL programmers can learn to deal with. Furthermore, it leaves all the presentation details to the client side, making much of the original COBOL code redundant. These parts can be deleted or commented out, leaving only the business logic and data access operations. If the data access logic is also factored out into an access shell and replaced by CALLs, this further reduces the size of the COBOL program. Experience with the first CICS programs has shown that less than 1/3 of the original code is left. Thus, this solution promises to significantly reduce the size and complexity of the existing mainframe programs. The presentation details are taken over by the client which may be a JAVA applet, a C++ windows component or a standard web browser. The data conversion is delegated to the data access modules.

It is recommended to also use XML forms for communicating with the data access layer. Rather than calling the access module with the usual COBOL data structures, the parameters can be converted to an XML form and the form passed by content. In this manner, the access shell can be called directly from the client using the same interface as the application programs to perform SELECTS, UPDATES and INSERTS upon the database. XML then becomes the standard interface between all layers of the software system, a major step in achieving a totally flexible architecture. (14)

Conclusions

In summary, the same barriers to encapsulating legacy mainframe programs in client/server architectures also apply to the internet. It is not advisable to simply replace the CICS maps or the COBOL linkage area with a HTML

or XML form, at least not in the long term. Furthermore, it is not possible to discuss accessing presentation, business logic or data access levels if these levels do not exist. The legacy programs have to be reengineered either before or after they are wrapped if they are going to be viable. The obsolete teleprocessing operations have to be removed, the business logic dissected into isolated, independently executable modules, and the data access operations moved out to separate data access modules. Finally, a control procedure should be inserted to govern the calling sequence of the business subroutines. The result will be a quite different program than the original one, a revised version which may resemble a totally different program. This is where one must decide whether it is worth the effort to reengineer or if it may not be cheaper to redevelop the programs in a new language specifically designed for the internet. The outcome of this decision will depend on many factors which are beyond the scope of this paper. (15) Most important is the extend of the business logic relative to the other layers. If it is significant, it may be worthwhile to preserve it in COBOL. If it is not, it probably should be reimplemented.

References

- 1] Sneed,H.: „Dealing with the dual Crisis– Year 2000 and Euro“ in Proc. of 4th WCRE, IEEE Press, Amsterdam, Oct. 1997
- 2] Miller,H.W.: Reengineering Legacy Software systems, Butterworth-Heinemann, Woburn,Mass.,1998
- 3] Curran,t./Keller,G./Ladd,A.: SAP-R/3 Business Blueprint, Prentice-Hall, New Jersey, 1999
- 4] Brodie,M./Stonebraker,M.: Migrating Legacy systems – Gateways,Interfaces and the Incremental Approach Morgan-Kaufman Pub., san Francisco, 1995
- 5] Plaisant,C./Rose,A./Sheideman,B.: „Low Effort, High Payoff User Interface Reengineering, IEEE Software Magazine, Juli 1997, p. 66
- 6] Graham,I.: Migrating to Object Technology, Addison-Wesley, Wokingham, G.B., 1995
- 7] Garlan,D./Allen,R./Ockerbloom,J.: „Architectural Mismatch– Why Reuse is so hard“, IEEE Software Magazine, Nov. 1995, p. 26
- 8] Tibbetts,J./Bernstein,B.: „Legacy Applications on the Web“ American Programmer, Vol. 9, No. 12, Dec.1996 p. 19
- 9] Keyes,J.: DataCasting – how to stream databases over the internet, McGraw-Hill, New York, 1998, p. 241
- 10] IBM: A Survey of Object-oriented technologies on MVS/ESA, Document GG24-2505-00, Poughkeepsie N.Y., Feb. 1997
- 11] Sellink,A./Sneed,H./Verhoef,C.: „Restructuring of COBOL/CICS Legacy Systems “ in Proc. of 3rd CSMR, 99, IEEE Press, Amsterdam, March, 1999
- 12] Langer,P.: „Why use COBOL for Web Database Applications“ Fijitsu Technical report, in DataCasting, Ed. Keyes,J., M cGraw-Hill, New York, 1998, p. 439
- 13] Graham,I./Liam,Q.: XML– Specification Guide, John Wiley & Sons, New York, 1999
- 14] Harmon, P.: „Distributed Systems, CORBA and Compound Documents“, Cutter Info Corp - Newsletter, Vol. 9, No. 8, Aug. 1999
- 15] Sneed,H.: „Risks involved in reengineering Projects“, in Proc. of 6th WCRE, IEEE Press, Atlanta, Oct 1999 p. 204

Experiences Migrating to a Bilingual Web Site

Scott Tilley

stilley@cs.ucr.edu

Shihong Huang

shihong@cs.ucr.edu

Department of Computer Science
University of California, Riverside

Abstract

This paper outlines some of our experiences migrating from an English-only Web site to a bilingual Web site. This included adding support for Chinese text and streaming audio. The migration process was not particularly smooth, due in part to our inexperience creating content in two languages simultaneously. We also found that there are many small details (and many large bugs) that one must deal with to produce a widely accessible bilingual Web site.

Keywords: internationalization, English, Chinese, migration, Web site

1 Introduction

SIGPC (www.srtilley.com/sigpc) is an online publication that explores the impacts of personal computing on computer science, information technology, and software engineering. It has been in publication since early 1997 and now goes out to several hundred subscribers. When a new article is released (usually once a month), a short summary of the text is sent by email to all SIGPC subscribers, as well as the URL for accessing the complete story and its corresponding audio version.

As with most of today's Web sites, the SIGPC site was English-only. We decided to migrate it to a bilingual Web site to attract a larger audience, and to better serve some of our newer subscribers – many of whom are international graduate students. Going bilingual for us meant adding Chinese language support, which includes text in Kanji and streaming audio in Mandarin.

Adding a second language to a Web site can be challenging. Adding an Asian language such as Chinese is particularly challenging. The usual issues such as logical and physical site structure must be addressed to incorporate the new dual-path through the site. There are also issues specific to languages such as Chinese. These include creating content (which editor to use?) and viewing content (which browser supports this?).

Perhaps most importantly, one must endeavor to create two versions of the same material that are true to one another. This challenge is present whenever there are two or more languages used to represent the same thing. We have tried to make our translation as true as possible as to the Chinese philosophy of “xindaya”: the translation should be accurate, understandable, and elegant. Given the technical nature of SIGPC, this is not always an easy goal to meet. This short paper outlines some of our experiences in trying to achieve this particular Web site evolution task. 信，达，雅

2 Creating and Viewing Content

We found that creating and viewing content for a Chinese Web site is somewhat challenging. Keeping two versions of the same material synchronized with one another is a classic maintenance problem. We also found that there are many small details (and many software bugs) that one must deal with to produce a widely accessible bilingual Web site.

2.1 Creating Content

We use Microsoft FrontPage2000 under the Windows98 operating system for the SIGPC Web site. Given this configuration, there are several options for creating content in Chinese. The most full-featured option is to use a Chinese version of Windows. This is an acceptable solution if you are going to do most of your work in Chinese and if you have the resources available to support a second development environment. In our case, most of the content would still be in English (we use the same equipment for several other Web sites), so this solution seemed too sweeping and disruptive to our other work.

The second option is to use a special software package for inputting Chinese. There are several such packages available on the market. We tried both freeware/shareware programs and one commercial program. The problems we had with such programs were twofold: the result was not very satisfactory, and editing the content to correct errors caused some of the Kanji characters in the file to be incorrectly changed. As shown in Figure 1, the fonts and spacing produced using one such tool are very uneven. The red arrows in the figure show where editing was taking place; the circled blue text indicates where the input program was incorrectly altering existing text as a result of this editing.

在这如雨后春笋般的新技E培粤到四块暇己的速度出现之判断哪些是重要的，
哪些只是时髦是很困难的。依我之见，1999年三个重要的主题是这三字头
缩写的三联星：MP3，XML，和Y2K。对SIGPC的读者们来讲，似乎没有必要解释
它们的含意，因为他们已经在今年的早些E培学会了如何使用这些缩

Figure 1: Uneven fonts and improperly edited text

The third option is to use a plug-in for the Windows environment from Microsoft called the Input Method Editor (IME). We found this to be the best solution for our migration requirements. Besides being free, it doesn't require us to use the Chinese version of Windows. It also uses fairly sophisticated auto-correction and auto-completion technology that makes inputting Kanji text (based on the pinyin spelling) reasonably easy. The result of using IME is Kanji that can be viewed by both major browsers – with a little post-processing on our part.

2.2 Viewing Content

Editing content in all of the scenarios described in the previous section takes place using Microsoft Word. Something that must be considered while creating the content, no matter which method is used, is how the user will see the resultant Chinese text. Our first approach was to create GIF files that captured the Kanji on the screen. The main advantage of this approach is that the user doesn't need to have any special software installed to see the text.

However, there are a few drawbacks to this approach. Every time the original text is changed, the GIF file must be regenerated. Since the GIF file will become unreadable if it is rescaled, it must be created at the proper resolution from the start. This means knowing the width and height of the container page that will hold the GIF file. In some instances, the result is a very wide page with a relatively narrow GIF file, which produces an unpleasant browsing experience for the user. From an accessibility perspective, using GIF files precludes the use of the Web site by sight-impaired users who rely on automated reading tools.

We decided instead to save the Word file produced by IME in native HTML format and use it directly in the Web page. While this solution works fine for the Internet Explorer browser, it doesn't always work for Netscape Navigator. When Word saves files in HTML format, it relies on XML and Cascading Style Sheets to reproduce the exact look of the original Word file. This is usually rendered properly by the latest version of Netscape Navigator, but it won't work for earlier versions. We decided to implicitly require users to have at least version 4.5 of Netscape Navigator to view the Chinese version of the Web site.

There is another niggling issue related to character sets. By default, the character set used by Word when creating HTML files is "windows-1252." This is essentially equivalent to the default character set of "Western (ISO-8859-1)" used by Netscape Navigator. When the Chinese text is included in the SIGPC file,

Internet Explorer will properly render the HTML as Kanji characters, but only when the user manually sets the encoding to "Simplified Chinese (GB2312)." If this manual change is not made, the result is the unreadable text shown in Figure 2.

MP3

Y2K iE iã »ðEí² Óí ×ÅÄ½iã£- µ«iÖEi Í MP3 iÖ i6 ÊÇ+¾ÄêBÖD×i ÁiEE ÐÐEEµÄ»° iã j£ ÓÄMP3 Ñ'EðEã" ±ãÄêµÄÖðÄÖ»ðÖi
Öð±£³ÖÄE'ó²; Ö µÄÖiÖðÖæÊµ¶E£-µ«Ö»Ö¼¾YÄEÍ'¾-N'E5µÄ'æ'£ Ö¼ãµÄ¾, ÖÖÖ»j£ ÄýEç£-Ö»ÖÄÖÄÖCD EÍµÄ 4MB
Ö»Ê×Çú×Ö¼E¾ÖEÜµ¾4MB¶¶iPA+iÖµÄÖðÖÊE#Ê§j£

Figure 2: Simplified Chinese without GB2312 enabled

Manually setting the character set encoding in Netscape Navigator will still not produce readable text. To be displayed properly, we added the directive <META HTTP-EQUIV="content-type" CONTENT="text/html; charset=gb2312"> to the head section of the HTML file. This tells the browser to use the proper character set for Simplified Chinese, which is "gb2312." Finally, Netscape Navigator users must do one more thing: they must select a font to be used when a non-Western character set is used. We selected "Arial Unicode MS" for Simplified Chinese. When all these details are addressed, the result is properly displayed Chinese shown in Figure 3.

MP3

Y2K问题或许统治着媒体，但我认为MP3现象是本年度中最令人感兴趣的话题。用MP3压缩算法编码的音乐或语音保持了大部分语音的真实度，但只占据了未经压缩的存储空间的几分之一。例如，一张音乐CD上的40MB一首曲子可减少到4MB而无明显的音质丧失。

Figure 3: Simplified Chinese displayed properly

3 Comments

There are several other details concerning our experiences migrating to a bilingual Web site that we will document in a future paper. For example, the physical and logical structure of the Web site should be changed in such a way that adding additional languages should be relatively easy. We will follow the example set by well-known sites such as Yahoo!, which have adopted a naming convention for their language-specific sites. For example, their Chinese-language site has a URL of cn.yahoo.com, and their French-language site has a URL of fr.yahoo.com.

An example of end result of our migration is shown in Figure 4 and Figure 5. Figure 4 is the English language version of the year-end issue of SIGPC, while Figure 5 is the translated Chinese version. Note that both versions have a streaming audio component, one in English and one in Mandarin. The creation of the audio presented several challenges by itself, which we will also document in a future paper.

Many of the issues we encountered in adding Chinese language support to the SIGPC Web site would no doubt be viewed as relatively minor to those who had done a similar task before. But, as with most things, it's only easy in hindsight. There are many details and software idiosyncrasies that one must address to make the final result acceptable. There is little doubt that if we decide to add a third language to SIGPC, such as Greek, more thought must be given to the evolution task as a whole.



Figure 4: SIGPC V3N9 -- English version



Figure 5: SIGPC V3N9 -- Chinese version

Lexical scanners for 4GL-source maintenance of a corporate web site

B. Toeter, btoeter@ic.uva.nl
Informatiseringscentrum, Universiteit van Amsterdam
Turfdraagsterpad 9 1012 XT Amsterdam
The Netherlands

Abstract

In this paper we report on a case study discussing practical aspects of maintaining a corporate website of a large university. The website is implemented in a fourth generation language, with all its accompanying typical maintenance problems, like frequent updates of the product, frequent modification to the syntax of the language, and lack of tool support for maintenance purposes. A dialect conversion of the 4GL is described, complete with file management modifications. Moreover, a tool for generating documentation for maintenance purposes is discussed. This tool significantly improves productivity in daily updates of the enterprise website.

1. Introduction

The increasing popularity of the Internet has lead many businesses to the World Wide Web. The initial goal was web presence, which was defined by the business having a corporate web site. Typically early web sites displayed information about the company and its products. Over time, companies realized that the web had the potential to open up new markets, and support traditional customer bases. A transition occurred from information-only to interactive sites. Software manufactures reacted to this by releasing smart development tools for creating web sites. In a short time span several 4GL languages were created which enabled rapid development of web sites that incorporate vast amounts of corporate data.

One of the more popular 4GL languages is the ColdFusion Markup Language (CFML) by Allaire Corporation. Developing applications with ColdFusion does not require coding in the sense of a traditional programming language like Perl, C/C++, Visual Basic, or Delphi. Instead, applications are created by combining standard HTML files with high-level database markup tags, that are easy to understand and use. ColdFusion 1.0 was released in July 1995, making it the first 4GL tool for database integration in web sites. By 1999 it had been adopted by nearly half the Fortune 500 companies [1]. Over time many updates of ColdFusion have been released. These new releases are not only restricted to bug-fixes, but comprise mainly of enhancements and extensions to serve the needs of its users. Due to changes in the syntax of the ColdFusion Markup Language, code that runs on version 1.0 may not run on version 2.0 or later versions. In order to benefit from language and performance enhancements of new versions of ColdFusion and its accompanying CFML, it is necessary to perform a source code migration.

The rest of this paper is structured as follows: In the next section we shall discuss the process and tools to aid in such a migration. Besides aspects of source code management driven by technological changes on the server side, user induced changes have to be dealt with also. Therefore we discuss in section 3 how to deal with changing requirements for web applications while keeping the source code in good shape. We shall discuss a tool that improves productivity of the ColdFusion programmer.

Finally we make some concluding remarks.

2 Source code migration

Several updates to the ColdFusion language and the ColdFusion Application Server have lead to syntactical changes in the ColdFusion Markup Language. The University of Amsterdam owns a corporate website that contains 6 ColdFusion applications. These applications have various functions, among which are: an online course information system and an up to date daily menu of the University owned restaurants. The source code is distributed among 236 source files adding up to 12.000 lines of code. The language used is CFML version 1.0. Due to updates in the product the code had to be upgraded to version 4.0.

The syntactical differences we dealt with follow changes of a nature that can be categorized as:

- Technological changes
- Changes in ColdFusion system standards
- Non-backwards compatible language changes

What follows is a discussion of the above points.

2.1 Technological changes

In the first releases the ColdFusion Application server interfaced with the web server by means of the Common Gateway Interface (CGI) [2]. In this approach an executable is placed in a certain directory on the web server and can be executed by requesting a certain URL. For instance a request for:

```
http://www.uva.nl/courseinfo/dbml.exe?template=coursedetails.dbm&courseId=02131
```

invokes the Application Server by starting a dbml.exe process. The question mark separates the path component from the searchpart component of the URL[3]. The searchpart contains of variable instances, which are separated by an ampersand(&). These variable instances are passed to the Application Server, which reads the template file `coursedetails.dbm` and executes it with the remaining variable instance `courseId=02131`.

A more efficient interface was adopted in version 2.0 of ColdFusion by using the web server Application Programmers Interface [4] (API). This technique allows the ColdFusion Application Server to be loaded into memory by the web server at start-up. When the web server receives a request for a template file it hands the contents of the template file over to the Application Server which executes the code. A request using this method could be:

```
http://www.uva.nl/courseinfo/coursedetails.dbm?courseId=02131
```

This way a variable instance `template` is no longer needed. The template filename has become part of the path component of the URL. We discuss how to implement these changes in a moment. Using the web server API greatly improves performance compared to the CGI approach. This is mostly due to the fact that the CGI approach spawns a new process for every request for a ColdFusion template. With the web server API approach, requests are queued to the in-memory Application Server, which will spawn a new thread to service incoming requests.

A drawback of adapting to the web server API approach is the change in the URL syntax of the template files. We were able to deal with this syntax change by executing a search and replace operation on the ColdFusion templates and every HTML file that included links to the ColdFusion applications. The replace operation consisted of:

```
Removing dbml.exe from all affected files.  
Changing .dbm& to .dbm? (indicating that what follows is the first variable)
```

Because in the CGI approach the template name is passed to the Application Server as a variable, the template name can be used as an input value in an HTML form. For instance, a pull down menu can be used to determine to which template the form should be submitted. In the web server API integrated ColdFusion Application Server this is no longer possible. This is because the name of the template to which the form has to be submitted now has become part of the path component of the URL, and therefore can no longer be defined dynamically. To accommodate this style of programming a stub template was inserted to which these types of forms were submitted. The stub template consisted of an include statement to include the template that was selected in the HTML form.

2.2 Changes in ColdFusion system standards

Two standards were changed in release 3.0 of the ColdFusion Application server. First, the file extension `dbm`, which stands for Database Markup, was changed to `cfm`, which stands for ColdFusion Markup. Second, the tag names were changes from `<db` to `<cf`. Changing the filenames was as straightforward as executing `rename *.dbm *.cfm` in all directories containing ColdFusion files. The tag names were updated using the search and replace function in a programmer's editor.

2.3 Non backwards compatible language changes

Starting at version 4.0, the syntax of the query tag became obsolete. The following CFML code fragments define a query that returns the name of a course from the database. After checking if the form variables `Semester` and `Level` have non-empty values they are included in the query.

Obsolete CFML 1.0 query syntax

```
<dbquery name="getCoures" datasource="wdbl"  
  sql="select name from courses">  
  
<intertwined HTML>  
  
<dbif #form.Semester# neq "">  
  <dbsql="where course_sem=#form.Semester#">  
</dbif>  
  
<intertwined HTML>  
  
<dbif #form.Level# neq "">  
  <dbsql="where course_level=#form.Level#">  
</dbif>  
  
<dbsql="order by 1">
```

New CFML 4.0 query syntax

```
<cfquery name="getCourses" datasource="wdb1">
  select name from courses

<cfif #form.Semester# neq "">
  where course_sem=#form.Semester#
</cfif>

<cfif #form.Level# neq "">
  where course_level=#form.Level#
</cfif>

  order by 1
</cfquery>
```

In the old syntax, the SQL part of the query could be appended to by the `<dbsql="...">` tag. HTML was allowed to be in between parts of the query, this is referred to as intertwined HTML. In the new syntax SQL and HTML are strictly separated. Statements in between the `<cfquery>` and `</cfquery>` are either SQL or ColdFusion flow control statements.

This syntax change allows for more efficient parsing by the ColdFusion Application Server and improved readability of the code. To convert the old CFML code to the new version, we used a lexical scanner that performed the following functions:

- 1) scan for a query tag
- 2) take out the sql attribute and move it outside the tag
- 3) remove `dbsql="` and its associating `">` leaving the SQL in place
- 4) close the query when there are no more `dbsql` tags that belong to this query

In case of source code that does include intertwined HTML, the lexical scanner would have to be extended so that it scoops up the intertwined HTML and places it at the end of the query. Up until version 3, HTML is also allowed within the flow control tags together with the `dbsql` tags. This was not the case in the code we were dealing with. It could however be dealt with by copying the flow control statements together with the HTML to the outside of the query leaving only the SQL statements in place

So it's a little more work when intertwined HTML is in the query tags, but it does not invalidate the approach that we discuss in this paper.

This concludes our discussion of the source migration process, next we look into aspects of daily code maintenance.

3 Source code management tools

Some programmers estimate that they spend as much as 40 percent of their time editing source code [5]. At least in our situation it was worthwhile to create a tool that supported us in rapid structured editing. For this purpose we focused on:

- Generation of system navigation
- Separation of views from database code
- Detection of dead parts

3.1 Generation of system navigation

ColdFusion stimulates developers to reuse code by providing an include tag that makes it easy to include other ColdFusion files anywhere in the code. The advantage of code reuse has the potential problem that the organisation of the files becomes more difficult to grasp. Therefore, proper and up to date documentation is needed to present a clear image of the structure of the source files. Without this kind of documentation the maintainer is more likely to make changes to a query in a file that is included by more files than he realises. This means that his changes may have an unwanted effect on the application. This is called the ripple effect. Keeping documentation up to date is a daunting task, as many legacy problems have shown already. According to the Principle of Proximity [6] this is especially the case with documentation that is kept beside the code.

In order to solve such problems we have developed a tool that generates up to date information on the file call structure of a particular file. This information is kept with the source file so that when a maintainer modifies it, it is known up-front which other files are possibly affected. We call such a file a system navigation chart. As indicated, documentation is often not kept up to date. However, with a lexical scanner that looks for the include tags it is fairly easy to generate the system navigation charts. When the source code has been updated, rescanning the source files is sufficient to update the system navigation charts. If the rescanning is integrated into the maintenance process, for instance by rescanning automatically after a file has been changed, keeping the documentation up to date requires hardly any effort at all.

3.2 Separation of views from database code

One of the reasons for having to edit source code is a change in requirements put forward by the customer. Often these changes concern the layout of the application. The source code maintainer can alleviate from this task by separating the database code from the layout, also called view, of the application. The customer can be given access to the files containing layout and gains a more direct control over the view of the application. Some of the applications hosted by the University of Amsterdam have not been set up this way, where the CFML is physically separated from the HTML. A time effective way of achieving this separation is by using a lexical scanner that takes out individual blocks of CFML code and stores each block in a separate file. The resulting CFML source files can then be included in the file that now contains HTML only. This is the kind of file a customer can easily edit without risking to damage the application by accidentally removing CFML statements.

3.3 Detection of dead code and loose links

The use of a lexical scanner to create a system navigation chart can reveal the occurrences of source files that are not referenced anywhere else in the application. Typical legacy applications contain up to 30% dead code [7]. Dead code in a web application can still be executed when a request for such a file is answered by the web server. Since dead code may not be up to date it can result in unexpected behaviour of the application.

Lexical scanning did not reveal all dead code residing within the applications. Scaffolding code that was still in the application could cause another form of undesired behaviour of the application. This code is meant for testing purposes and should have been removed by the programmer before releasing the code into the production environment. In our case the code consisted of queries that did not serve a purpose anymore. Because the ColdFusion application server executes all queries that are specified in a source file, unused queries unnecessarily degrade performance. Finding these types of queries can be left to a lexical scanner that makes sure each query is referenced to elsewhere in the code.

4 Conclusion

Keeping source code compliant with the platform it is supposed to run on, becomes an issue when a change in the platform leads to changes in the syntax of the language. Due to the relatively simple nature of the syntax changes in the ColdFusion language we were able to use a lexical scanner to perform the source code migration from version 1.0 to version 4.0.

The use of a lexical scanning tool, to migrate source code, saved us time and prevented us from having to perform the error prone task of manually investigating all the source files and making the changes ourselves. Because the software migration only had to take place once, we created a separate tool for this purpose. For the navigational charts mentioned below another tool was developed.

The use of system navigation charts saves a lot of time, that would otherwise be spent investigating the call structure of the source code manually. Besides saving time, the ripple effect is less likely to occur, which decreases the time needed for debugging. By generating the system navigation charts automatically we can ensure that these charts are always up to date.

References

- [1] H. Hamilton, *ColdFusion research project*
<http://www.talltech.com/student/imos99/studentweb/Heather/RESEARCH/Res.htm>
- [2] D. Robinson, *The WWW Common Gateway Interface Version 1.1*
<http://www.poli.studenti.to.it/info/cgi-ssi/cgi-spec.htm>
- [3] T. Berners-Lee, L. Masinter, M. McCahill, *Uniform Resource Locators (URL)*,
RFC 1738 <ftp://ftp.uva.nl/pub/rfc/rfc1738.txt>
- [4] Web server API: <http://www.cs.rpi.edu/~noel/Security/WebAPI.html>
Netscape specific: <http://developer.netscape.com/support/faqs/champions/nsapi.html>
- [5] W. Ratliff, Interview in *Solutions Systems*, 1987
- [6] S. McConnell, *Code Complete*, Microsoft Press 1993, p. 483
- [7] Capers Jones, *Software Cost Estimation*, McGraw-Hill, 1998

Localizing and Using Services in Web-Enabled Environments

Ying Zou and Kostas Kontogiannis

University of Waterloo
Dept. of Electrical & Computer Engineering
Waterloo, ON, N2L 3G1
Canada

Abstract

The Web browser technology has revolutionized in the 1980's the way Internet is explored and utilized for gathering and presenting information. With the emergence of distributed object technologies and new programming languages Internet is now moving from a world-wide information pool towards a service providing facility. This position paper examines the origin and architecture of the "services-based" Web and, discusses techniques for searching and integrating content and services with the Web as the primary medium.

1 Introduction

The Web browser technology, the so-called first-wave of the Internet [1], brings the explosive usage of the Internet in terms of world-wide information shared resources. The convergence of the Internet and distributed-object technologies, extend this "information-based" Internet to the "services-based" Web. This evolution is referred to, as the Internet's second-wave [1], where software services and content are distributed over the Internet, Intranet, or Extranets. In this context, services are denoted by the behavior of software that is accessible as a component in the Web domain. As more data and software components are made available on the Web, these software services have unique addresses and relate to processes that can be dynamically executed on a server upon the request of arbitrary Web clients.

As an example, consider a scenario of the "services-based" Web, whereby a global infrastructure enables software components that have been developed independently, be integrated with each other in order to facilitate complex programming tasks. In this way, content (data) located virtually everywhere in the world and, software components can be

combined. This integration can happen dynamically by allowing requirements, functionality descriptions and, signatures of components be published on the Internet so that client processes invoke them without necessarily downloading all the components to the local client machine. In this context, a software agent can search for available software services on the Internet, select them to compose new applications, invoke the services remotely, and obtain the results utilizing the Web. Such an infrastructure offers the opportunities for the Web-based enterprise integration and e-commerce to function.

2 Distributed Object Technologies

Essentially, the "services-based" Web relies on the emerging concept of "object Web" [12] which assumes universal connectivity, broadly distributed environments, and cross-platform interoperation of both infrastructure and applications [2]. The "services-based" Web adopts the distributed object technologies, and can be constructed from the interoperability among the language and platform-independent applications. Therefore, it moves the distributed applications beyond the boundaries of the simple client-server communication and message passing, towards a larger scale collaboration infrastructure.

Definition of Distributed Object

In [11] a distributed computing system can be defined as a system of multiple autonomous processes that do not share the primary memory, but that cooperate by sending messages over a communication network. This definition denotes the behavior of physically separated components and logically autonomous

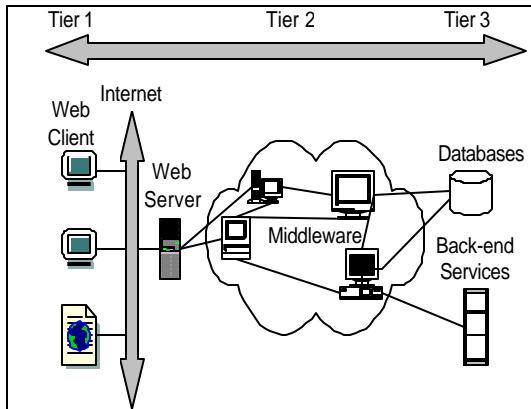


Figure 1: Three-tier Architecture

modules communicating via message passing. On the other hand, a distributed system can also be defined as a system whose services and functionality are encapsulated in objects so that only specific interfaces allow to mediate message passing between processes. These objects are referred to, as distributed objects.

The distributed objects encapsulate software components at different level of granularity and detail. For example, the coarse-grained components can be executable legacy applications and the fine-grained component can be specific software components, which provide specific functionality. Although the fine-grained components are intended to be reused arbitrarily are often too small for practical use and, they may require considerable work from programmers for being customized to a given application context [7]. For the sake of reusability, it is more practical that legacy software systems and be wrapped as distributed objects and be easily integrated with other applications.

In real life, the distributed objects can be Java servlets, Enterprise JavaBeans and, CORBA objects, which communicate by Java RMI or, IIOP protocols.

Three-Tier Architecture

The three-tier architecture is fundamental to the deployment of the distributed objects on a Web-based environment. A Web server acts as a front-end between client processes (Tier 1) and to the middleware (Tier2). The middleware in its turn facilitates the interoperability of many disparate software applications or distributed objects (Tier3). The Web clients (Tier1) send the

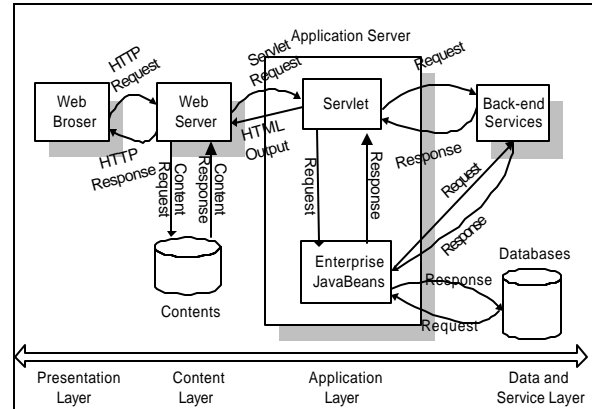


Figure 2: Control Flows in Three-Tier Architecture

requests to the Web server for the software services to be invoked over the HTTP protocol. The Web server invokes the requested service on behalf of the clients (shown in Figure 1). Currently, the major middleware technologies include Java RMI from SUN, CORBA from OMG, and DCOM from Microsoft.

3 An Infrastructure for Locating Web Services

Control Flows in Three-Tier Architecture

From another point of view, the generic three-tier architecture (Figure 1) is further separated into four layers [1]: *a*) presentation layer (Web browser); *b*) content layer (Web server); *c*) application layer (application server) and; *d*) back-end data and services layer.

The Web server and the application server can be located in the same machine or on different machines. The Web server is responsible to accept HTTP requests from Web clients, and delivers them to the application server, while the application server is in charge of locating the services and returning responses back to the Web server. For example, IBM's HTTP server, which is a Web server, uses IBM WebSphere application server as its front-end.

On the other hand a thin client in the presentation layer has little or no application logic. It sends the request via HTTP to the web server for an interactive view of the static and dynamic information.

To provide the static information, the Web server can maintain a content repository, or a file

system, where the information-based resources are stored and serve the static HTML pages. Upon receiving a request from the client, the Web server can retrieve the requested document from the content repository and send it to the client. In this case, the client entirely relies on the Web server. Programming languages, such as Java, and scripting languages, like JavaScripts and CGI, can be used to access databases and other on-line resources.

To provide the dynamic information, generated by software services, the Web server needs to constantly interact with the application server. A servlet, which is a Java program, provides the dynamic HTML content to clients. When the Web server receives the request for a servlet, it re-directs the client's request along with the parameters to the application server, which loads the servlet and runs it. The servlet, an actual Java class, can make calls to back-end services, other servlets or, to the Enterprise JavaBeans. In the end of the process, servlets construct the response in the HTML format and passes the results to the Web server for transmission to the client. [13]

In particular, the EJBs (Enterprise JavaBeans) are server-side components which act as building blocks for the other EJBs and servlets inside the application server, or for other Java applications and Java applets available in the Web via IIOP.

Categories of Software Services

According to the client processes, the software services via the Web can be divided into two categories. The first category focuses on clients for which the servlets provide the end services via Web browsers. As stated before, the servlets rely on the Web server to present parameters and other information required for a service to be invoked. Consequently, the Web server transmits the input to the servlet on the client's behalf. After execution, the servlets generate results that can be presented or rendered in HTML or any other form.

The second category focuses on Intranet or Extranets where, software services can be provided as distributed objects in the form of servlets, Enterprise JavaBeans, JavaBeans or back-end services. These distributed objects are reused to construct another applications without the aid of the Web server.

Specifications of Software Services

To facilitate the "service-based" Web, a meta-level description for the software services should be published at the same time as the distributed objects are deployed onto the application servers. The specification of the software services includes the following information:

- General compile-time and runtime properties, including keywords for describing its functionality, execution environment.
- HTTP URL links for the servlets Web page, the IIOP hostname for the name server along with the port number, in which the name server listens to the naming request.
- Interfaces of the distributed objects, specifying the available behaviors of the objects, the expected parameters and the types of the return value.
- Inputs for servlets are simply the HTML forms. It is necessary to describe the range of the parameters.

Such specifications supply rich information for the available software services, and ease the way to understand and locate them.

Search Engine for Software Services

A search engine is needed to locate distributed software services much like a search engine locates data in Web pages. In its current infrastructure, the description of Web documents is not stored in any structured way. Therefore, it is very common to get low precision and low recall results using the current Web search engines, especially when the semantic content of the queries is not precise. The main reason for this problem is that the search engines scan the flat abstract information from the Web servers to locate keywords without taking into account the particular query context.

In a similar way, searching and dynamically linking content and services using the Web as the medium, require that description information for components be represented in a structured but flexible way. Signatures of components, pre- and post-conditions as well as, triggering mechanisms need be specified. XML provides a natural way to represent the specification of software services using the hierarchy relationship inherent in its tags. Service localization engines should be modified accordingly. With the DOM

(Document Object Model) API, the localization engine can easily navigate the hierarchy structure of the XML document tags, and extract the different parts of such information. It is noticeable that the description keywords for the functionality can be structured according to the context. It allows the clients to search the services by functionality not just static descriptions. Such an environment is under development at the University of Waterloo in collaboration with IBM Canada, Center for Advanced Studies. The application area deals with the integration of services in e-commerce and e-business to business environments. The invocation of services is based on scripts encoded in XML and is enacted using the Event-Condition-Action paradigm.

4 Conclusion

With the aid of service description and service localization mechanism, the “services-based” Web transforms the traditional way of pulling static information from repositories, to pushing dynamic information to clients upon request. As a result, the new infrastructure makes the reuse of software components available in a much larger scale, shortens the time to architect new applications, and eases the enterprise integration and e-commerce operations. Meanwhile, it provides a new market for e-commerce and e-business by allowing user configurable and customized content and services be delivered using existing World-Wide-Web infrastructure.

References

- [1] Paul Dreyfus, “The Second Wave: Netscape on Usability in the Services-Based Internet”, IEEE Internet Computing, March/April 1998.
- [2] Cynthia McFall, “An Object Infrastructure for Internet Middleware: IBM on Component Broker”, IEEE Internet Computing, March/April 1998.
- [3] Gary R. Voth, Charles Kindel, and Jon Fujioka, “Distributed Application Development for Three-Tier Architectures: Microsoft on Windows DNA”, IEEE Internet Computing, March/April 1998.
- [4] Ellis Horowitz, “Migrating Software to the World Wide Web”, IEEE Software, May/June 1998.
- [5] Vivekanand P. Kochikar, “The Object-Powered Web”, IEEE Software, May/June 1998.
- [6] Hans-W. Gellersen and Martin Gaedke, “Object-Oriented Web Application Development”, IEEE Internet Computing, January/February 1999.
- [7] Israel Ben-Shaul, James W. Gish, and William Robinson, “An Integrated Network Component Architecture”, IEEE Software, September/October 1998.
- [8] Israel Ben-Shaul, Gail Kaiser, “Coordinating Distributed Components Over The Internet”, IEEE Internet Computing, March/April, 1998.
- [9] Writing Enterprise Applications, <http://developer.java.sun.com/developer/onlineTraining/J2EE/Intro/session.html#setup>.
- [10] Israel Ben-Shaul, James W. Gish and William Robinson, “An Integrated Network Component Architecture”, IEEE Software, September/October 1998.
- [11] Reaz Hoque, “CORBA 3”, P573, IDG Books Worldwide Inc., 1998.
- [12] Robert Orfali, Dan Harkey, and Jeri Edwards, “Instant CORBA”, John Wiley & Sons, 1997.
- [13] Joquin Picon, et al, “Enterprise JavaBeans Development Using VisualAge for Java”, <http://www.redbooks.ibm.com>.