Integrating Structured Databases Into the Web: The MORE System[1]

David Eichmann[†] Terry McGregor[‡] Dann Danley[‡]

[†]University of Houston – Clear Lake
2700 Bay Area Boulevard, Houston, Texas 77058 U.S.A.
eichmann@rbse.jsc.nasa.gov

[‡]I–NET, Inc.
1020 Bay Area Boulevard, Houston, Texas 77058 U.S.A.
{tmcgrego, ddanley}@rbse.jsc.nasa.gov

Abstract

Administering large quantities of information will be an increasing problem as the World Wide Web grows in size and popularity. The MORE system is a meta-data based repository employing Mosaic and the Web as its sole user interface. We describe here our design and implementation experience in migrating a repository system onto the Web. A demonstration instance of MORE is accessible at

`http://rbse.jsc.nasa.gov:81/DEMO/`

1 – Introduction

As increasing quantities of information are made part of the World Wide Web [4], it will be increasingly difficult for Web administrators to provide effective access to that information. Support for meta-information concerning web-accessible artifacts will be necessary, particularly when there are large numbers of such artifacts. The Repository Based Software Engineering (RBSE) project addresses just such a scenario, a public repository of thousands of software engineering information and application source artifacts with over one thousand remote users.

The RBSE research and development group, seeking to free the project from a monolithic architecture based upon X-Windows, has developed a new repository - MORE, the Multimedia Oriented Repository Environment. MORE was designed as a set of application programs (more specifically a set of CGI executables [10]) that operate in conjunction with a stock httpd server [11] to provide access to a relational database of meta-data. The entire MORE interface, client browsing and search, repository definition, data entry and other administrative functions, is provided through stock Web clients. (We currently use X-Mosaic [15], WinMosaic [14] and Lynx [13] for most of our interaction.) MORE provides separate hierarchies of meta-classes and collections and support for controlled access to proprietary collections through the definition of user groups. With the single exception of the system front page, the entire user interface is accomplished as dynamically generated Hypertext Markup Language (HTML) [1, 4].

1. This work supported in part by NASA cooperative agreement NCC-9-16, RICIS research activity number RB02.

2 – The Repository Based Software Engineering Project

The Repository Based Software Engineering (RBSE) project is a research and development program whose mission is to provide a technology transfer mechanism to improve NASA's software capability. RBSE is sponsored by the NASA Technology Utilization Division and is administered by NASA's Johnson Space Center and the Research Institute for Computing and Information Systems (RICIS), a part of the University of Houston - Clear Lake. The purpose of RBSE is the support and adoption of software reuse through repository-based software engineering in targeted sectors of industry, government, and academia. The project consists of two principal activities, research into repositories and related issues, and operation of a public facility. The RBSE research and development group is active in a number of areas:

- repository technology,
- internet discovery [8],
- collaboration packaging,
- reengineering process modeling [16],
- ION - the Intermediate Object Notation [9, 12],
- interface slicing [3],
- reusability metrics [6, 7], and
- automated classification of assets.

The RBSE projects contracts with MountainNet, Inc., a small technology firm, to operate the AdaNET repository as its public facility. The AdaNET Repository contains a comprehensive collection of information about all aspects of software engineering and the software development life cycle, as well as an extensive collection of public domain software with related documentation provided to support software development efforts. In addition to software and related documentation contained in the reuse library, AdaNET contains information related to conferences, tools and environments, publications, and references. The AdaNET Repository is available to the general public, however user registration is required.

3 – Our Previous System

RBSE's previous repository mechanism, known as ASV3, consisted of a monolithic X-Windows application that interacted with a relational database, as shown in Figure 1. The data model provided for an inheritance hierarchy of asset meta-classes with asset attributes chosen by librarians as needed. A separate hierarchy of collections allowed for the clustering of heterogeneous sets of assets, and support for related collections. Search mechanisms included boolean expressions, pattern templates, and relevance-feedback natural language. Browsing of both class and collection hierarchies was supported. The resulting client interface proved to be quite powerful and adaptable to a broad variety of user expertise.

The RBSE public facility currently has more than a thousand subscribers spread broadly across industry, academia and government. Access is gained by logging in to the repository host and executing the repository software. Both ASCII and X-Windows interfaces are provided; previous versions of the system were entirely ASCII-based and dominantly over dial-up connections, but current user session counts run over 80% direct Internet connections, largely employing the X-Windows interface.

Performance in our local area network is quite adequate, but geographics have created problems for remote user interaction with ASV3. Many of our clients are quite distant from our server, and the 'presence' of ASV3 suffers from network delays.

Furthermore, while the representation and search mechanisms in the current system are

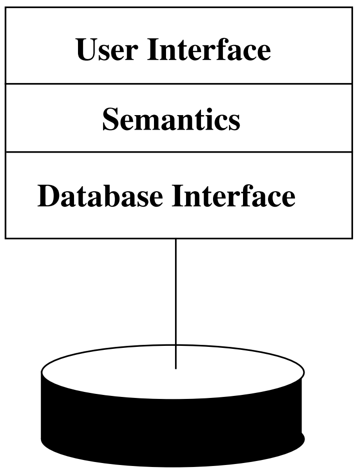

Figure 1: Previous RBSE Repository Architecture

rich, there are definite limitations that we sought to overcome:

- Users must assess deposits with little support beyond group classification and text display of the actual source code (and perhaps a user manual if the author went to the effort of creating one).
- Some assets have over 100 compilation units, and providing a sense of asset system structure was limited by the nature of the interface. Addition of new interfaces promised a massive retesting of system stability.
- Most error reports against ASV3 trace back to the intricacies of X-Windows, not to the particular semantics of the application or to database interaction.
- The system used a single table, made up of fifty attributes, each an eighty-character string, into which all asset metadata was placed. These limits were hard-coded into the system.
- All users, regardless of their local environment, were presented with the same repository model, the schema definition did not support the capability of defining access limitations. Users either were either not allowed access the system, or could access all production artifacts.
- The monolithic architecture required substantial installation effort, and was dependant upon a single DBMS (Oracle). Interaction with the database was spread throughout the application. Installing multiple instances of the repository on the same server required substantial replication of system resources.

4 – The MORE Architecture

Building MORE involved a complete redesign of the repository to support the following goals:

- an adaptable group definition mechanism for managing access to proprietary sub-collections of assets;
- optimization in the storage of metadata concerning assets – each class definition now has a corresponding relation, with each class attribute directly mapping to a relation attribute of the same name and type;
- the World Wide Web, and Mosaic in particular, as our sole user interface (including all administrative access);
- visualization mechanisms using HTTP ISMAP protocols [2]; and

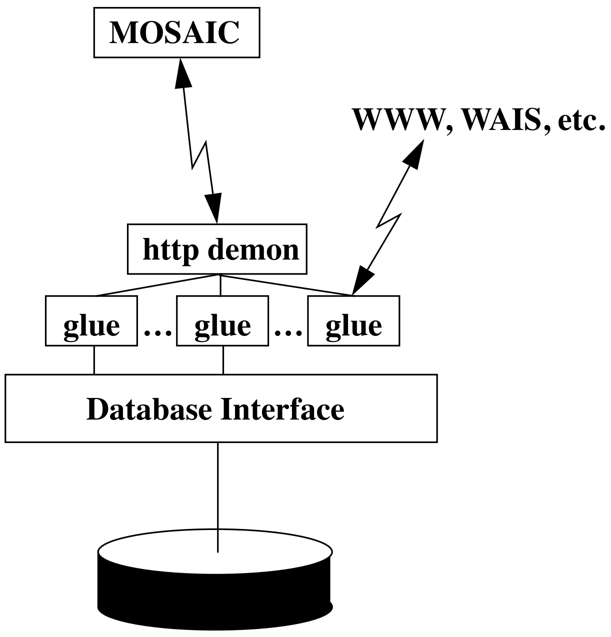

Figure 2: The MORE Architecture

- integration of other Internet resources into our user interface through the use of URLs in repository data.

The resulting architecture, shown in Figure 2, supports repository semantics without the overhead of a low-level X session across the Internet. The redesign has substantially reduced and simplified the overall system structure as well.

The Mosaic / World Wide Web architecture has altered our perception of user interaction. Our means of measuring user activity previously involved session connect hours. Now users connect to the system through sequences of distinct transactions, forced by the stateless interaction with the server. This results not only in a change in how we report user activity, but also in an acceptance that adding URL references as our primary semantic threading of the system result in us not knowing whether a user actually takes advantage of that URL unless it points back at us. URLs followed from a client result in direct interaction with that other server (which, of course, also frees us from having to play intermediary). Heterogeneity supports access to WAIS, WWW, Gopher and other servers without having to concern ourselves with anything other than the composition of a correct anchor. This promises to have great impact on the current efforts in standardization of repository interoperation [17].

Users can access the system at arbitrary entry points by storing URLs through their client software. The repository system is now modular, with execution split between the client (e.g., Mosaic or Lynx) and the server referenced by the current URL. With the exception of the home page, all HTML presented to the client software is generated dynamically by the glue routines. The modular architecture also easily supports a variety of interface models for customization of the appearance of the system, since a typical glue routine execution thread is a couple of pages of code. We have created a compilation dependency graph browser as an example of the type of visual interfaces that we plan incorporating.

MORE is now much more adaptable to multiple platforms. Mosaic currently runs on PCs, Macs and a number of UNIX platforms, and our only requirement for Web client programs is that they support HTML for client access (HTML+ for the search mechanisms) and HTML+ for librarian access. The HTML+ requirement is more specifically support for forms. HTTP demons are available for numerous platforms. Our only requirement here is that the demon support execution of programs and user authentication (this allows us to avoid prompting for authentication with every repository interaction during a transaction sequence).

Table 1: A Comparison of System Sizes

System	Subsystem	Source Lines of Code
ASV3	application	38,468
	library	10,315
	other	13,578
	total	62,361
MORE	application	12,640
	library	16,184
	other	1,264
	total	30,088

We also have a modular DBMS interface as a secondary effect of the redesign. We are using Oracle v.6 for MORE 1.0 and plan Oracle v.7 and Postgres [18] support in MORE 1.1; other SQL-compliant engines are easily added through the separation provided by the database interface layer. Reporting mechanisms are now discrete from the repository itself. The default information provided by the http log files is sufficient for the majority of our needs and have freed us from the need track user activity within the repository itself, as was necessary in ASV3.

MORE is substantially smaller than ASV3, as shown in Table 1. The dramatic reduction in the size of the application code is due primarily to the replacement of complex X-Windows code with printf C calls to mark up data into HTML, and in small part to migration of database specific code into the library. The remaining growth in the library is due to the additional database activity to support group definition and manipulation.

4.1 – Metadata and Classification in MORE

MORE is a meta-data based repository – the information stored in its underlying database is not the artifacts themselves, but rather information concerning the artifact, which is stored using other mechanisms (the file system, another database, or another software package such as a configuration management or CASE tool). MORE supports two distinct representation mechanisms, a class definition hierarchy, allowing homogeneous organization of information and a collection hierarchy, allowing a mix of homogeneous and heterogeneous information.

The class definition hierarchy is single inheritance, with a base class that is customizable at installation time through the database definition scripts (no software changes are required). The semantics of the system require that the base class contain at least an asset id, title, keywords, and abstract – with additional fields added as required. Further definition of the class hierarchy is then carried out completely through the librarian interface, with the database interface generating the calls to the DBMS to dynamically create and destroy classes and their corresponding relations as necessary.

The collection hierarchy supports the aggregation of assets without respect to their defining class. Any given collection can contain a set of assets drawn from any number of classes, as well as sets of subcollections and related collections. Any asset will always be a member of at least one collection in the hierarchy, but can be a member of as many collections as is appropriate, at any level in the hierarchy. Furthermore, each collection can have associated with it

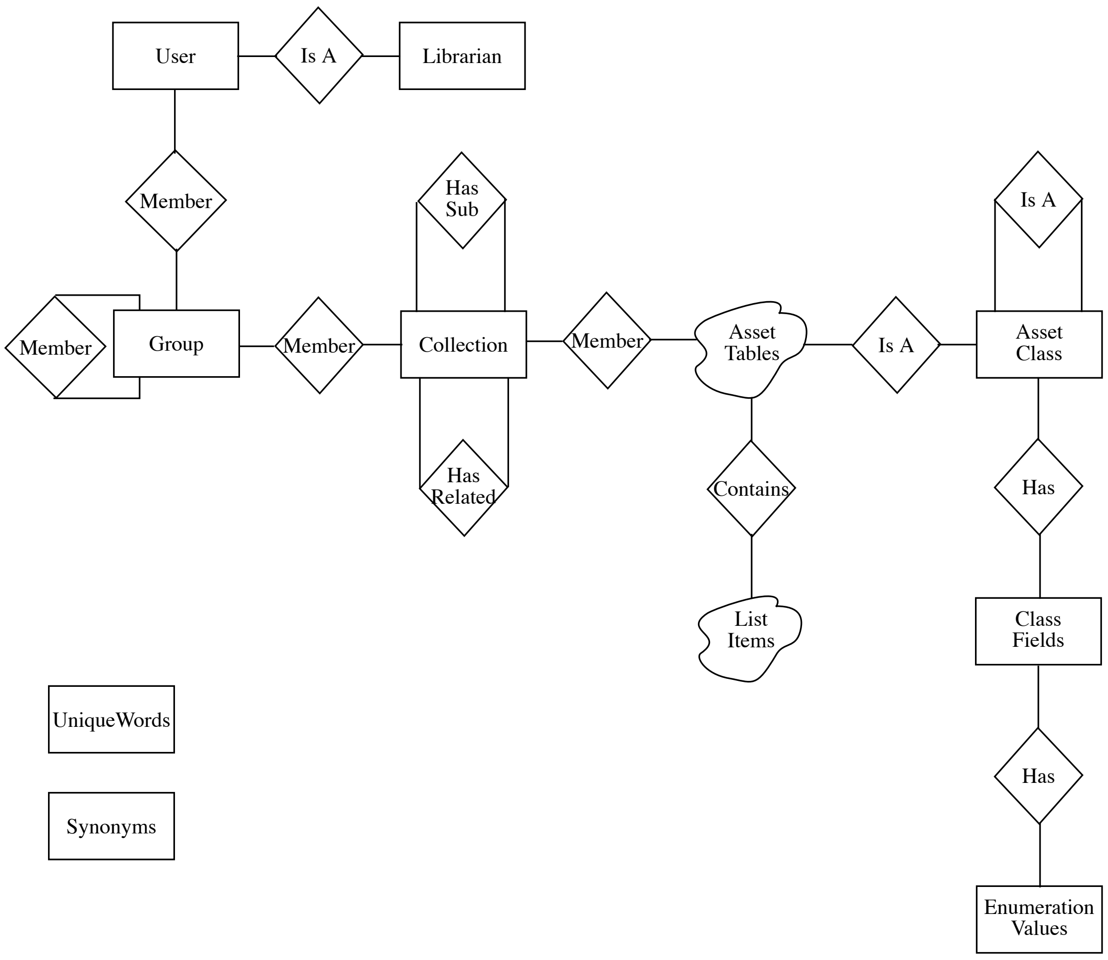

Figure 3: The MORE Schema

one or more groups which are authorized to access the assets and subcollections making up the collection. Groups in turn are made up of sets of users and other groups – all defined through the librarian interface. Users not transitively a member of a designated group for a given collection will never see the collection (or its contents) through any of the browser or search mechanisms.

The related collection mechanism is unary, a given collection can refer to another collection without the referenced collection being required to reciprocate. This was a conscious design choice, as we wish to support work groups referencing more public collections without revealing the contents (or existence) of their own collections to the organization or public at large.

Assets, as mentioned earlier, are characterized by their metadata, which normally includes an address, comprised of a hypertext anchor (a URL and a label) that provides a clickable path to the asset. A special case involves assets that are composites – made up of a number of distinct artifacts. We organize these into directories in a server file space (usually the same server as MORE is using), one asset per directory and one artifact per file. The URL is then a path from the server root directory to the asset's directory. This results in a list of files marked up as links.

4.2 – The Database Schema

Figure 3 show the schema for MORE. As mentioned earlier, much of the structure defined for asset metadata is not static, but rather generated dynamically by librarians through the MORE interface. The two clouds in the E-R diagram denote where this occurs: the asset tables and the associated list item tables that hold repeating field data (such as keyword lists and abstract text lines). Each asset class has a tuple in the asset class table, tuples in the class fields table corresponding to each field that field declares beyond its parent in the inheritance hierarchy and a table that stores the tuples containing the asset metadata. Class fields of type enumeration also have entries in the enumeration values table corresponding to the values making up the enumeration. Enumerations are defined through the existence of one or more pairs of enumerations and values in the enumeration values table, allowing multiple class fields to be defined that share a common enumeration type.

The collection, group and user mechanisms described in the previous section are represented through corresponding tables, with the usual implementation of many-to-many relationships as pairs of database keys. Note that both users and groups can be members of groups, allowing for shorthand inclusion and removal of collection access for entire communities of users.

Librarians are presumed to be users, with access privileges defined as bit masks. Privileges operate at a very fine grain, supporting for instance, specific allowance/disallowance of asset creation, modification and deletion. The administrative interface program matches the librarian's identity against the defined list of capabilities and presents hypertext links to only those administrative functions that their mask permits.

The two decoupled tables, unique words and synonyms, are used for natural language search against the metadata. When an asset is added to a collection, the text of its various fields is merged into a list of unique words, mapped against a stop list to remove common words and then added to the unique words table. (Note that we currently only include terms that appear in the metadata, and not in the asset itself. We plan a separate mechanism to support full-text indexing of assets [8]). When a natural language search is run, the query executes against this table and results, ordered by relevance, are presented as hyperlinks back to the asset's metadata. The natural language search page contains a check box for indication that the unique words should be augmented with any synonyms that appear for them in the synonyms table.

5 – Lessons Learned in Reengineering a Repository for the Web

The architecture of the ASV3 system was split between the X/Motif interface, the semantics layer and the database interface layer. The X/Motif interface and semantics layers of the design composed the ASV3 user interface. The database layer provided an Application Programming Interface (API) to the ASV3 repository. The two interfaces utilized the same API and were dependent on state information for each interaction with the user. The design of the database API was driven in large part by the data requirements of the X/Motif interface.

We became aware of the World Wide Web and the NCSA Mosaic viewer in mid 1993. Over the next few months we discussed the feasibility of using Mosaic as a display mechanism for the new RBSE library system. It was decided in November of 1993 to develop a proof of concept program which would interface with the ASV3 repository through the database API and to attempt to display the information via Mosaic. The learning curve for developing script programs for the generation of dynamic HTML pages was short. The first program generated an HTML page which contained a form with a scrolled list of collections retrieved from the repository through the database API, proving it could be done at least at the user level. One week later a fully functional interface to the ASV3 repository had been constructed which allowed browsing, searching and viewing of assets in the repository. We then held a lessons-learned session and concluded that:

1. We needed to be able to do parameter passing between the dynamically allocated HTML forms and the programs written to access the database. There was key field information needed by the database API which could not be stored anywhere else but the form action anchor.

2. Changing from a state environment to the stateless environment of HTML would require a new architectural design for the system. The system would change from one monolithic program which required users to maintain a sustained session on the server to a series of short disjunct sessions lasting only long enough to generate the next HTML page. Developing complex interface code would no longer be necessary.
3. Asset files were no longer required to be on the server, but could be anywhere on the Internet that was accessible through a hypertext link.
4. Support for download and viewing capabilities would be handled by WWW client applications rather than by the repository software.
5. There would no longer be a single entry point into the system. The user would have the capability to drive the system to information of interest and save the URL allowing entry back into the system at the same point later.

It was clear that the project should change direction and pursue development of the new system using WWW client applications as its interface. The MORE database API was designed and built using an object-oriented approach. Each table in the repository has a set of operations which support data manipulation. The API was developed using C and Oracle Pro*C. The API supports data manipulation at the tuple level and above. The API was greatly influenced by the architecture of the system. The kinds of state information available in ASV3 were not available in the new design. API functions were written to support the retrieval of information previously available as state data from the interface.

The redesign and development of the new database API was also driven by the move to a stateless environment. Previous assumptions about available state information were invalid and a new interface was required. The semantic design of the repository was also changed as features which were previously supported by the system would now be handled by the WWW client applications.

One of the temptations of working in a stateless environment is to create one executable for every page which needs to be created. In the case of the MORE system this approach would have resulted in producing almost 150 programs each of which was 500K bytes in size. (The large size is due in large part to linking with the Oracle interface libraries.) The design of the system is organized around functional systems and sub-systems, with most of the programs are in support of repository administration. Only one-third of all the programs in the system support browsing, searching and viewing. The other two-thirds support repository administration.

An interesting side effects of moving from a state environment to the stateless environment of HTML is that the user interface became more simplistic. The user interface transformed from complex multi-function windows to single function single action HTML pages.

6 – Conclusions and Future Work

Our experience with MORE has been an unqualified success. We have been able to develop and deploy a complete revision of our repository in dramatically less time and with substantially fewer (and simpler to fix!) error reports submitted during testing than our previous system. MORE is comparatively lightweight, currently comprising approximately 12mb of executables (in debug mode, with no sharing of Oracle libraries) and a database requirement one-quarter of that for ASV3. The complexity of the executables is also substantially reduced, implying that maintenance should be relatively easy.

The system is capable of administering an effectively unlimited number of assets (constrained only by the amount of disk space the database engine can manipulate for the metadata) distributed on an arbitrary number of Web servers scattered about the Internet. This leads to an interesting ability for independent repositories to point at one another without requiring substantial alteration to either system. For example, our demonstration system contains an asset with a hyperlink to COSMIC's front page, an asset with a hyperlink directly at COSMIC's author list, and an asset with a hyperlink directly at the description to a particular software system in COSMIC's catalog. Repositories can share assets and support separate organizations and classification schemes in their local systems.

Our development path for MORE includes:

- MORE 1.0: Core Functionality (completing user testing)
  - MORE 1.1: Multiple data engine support [mid '94]
  - MORE 1.2: Asset versioning [mid-to-late '94]
- MORE 2.0: Distributed Servers [late '94]
  - Subcollections spanning servers
  - Related collections spanning servers
  - Searches spanning servers

The planned releases of 1.x will result in a rich, flexible and portable repository mechanism where metadata for a single environment resides on a single server. MORE 2.0 will extend this capability with the ability to have the repository environment seamlessly span an arbitrary number of Web servers. Users only awareness of this distribution will be the URLs that their client might display.

References

[1] Andreessen, M., "A Beginner's Guide to HTML," National Center for Supercomputer Applications, Univ. of Illinois, http://www.ncsa.uiuc.edu/demoweb/html-primer.html.

[2] Andreessen, M., "Graphical Image Map Tutorial," National Center for Supercomputer Applications, Univ. of Illinois, http://wintermute.ncsa.uiuc.edu:8080/map-tutorial/image-maps.html.

[3] Beck, J. and D. Eichmann, "Program and Interface Slicing for Reverse Engineering," jointly appearing in *Working Conference on Reverse Engineering,* Baltimore, MD, May 21-23, 1993, pages 54-63 and the *International Conference on Software Engineering,* Baltimore, MD, May 17-21, 1993, pages 509-518.

[4] Berners-Lee, T. (ed.), "HyperText Mark-up Language," CERN, http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html.

[5] Berners-Lee, T., "The World Wide Web Initiative," CERN, http://info.cern.ch/hypertext/WWW/TheProject.html.

[6] Boetticher, G. and D. Eichmann, "A Neural Network Paradigm for Characterizing Reusable Software," to appear in *Proc. of The First Australian Conference on Software Metrics,* Sydney, Australia, November 18-19, 1993.

[7] Boetticher, G., K. Srinivas, and D. Eichmann, "A Neural Net-Based Approach to Software Metrics," *Fifth International Conference on Software Engineering and Knowledge Engineering,* San Francisco, CA, June 16-18, 1993, pages 271-274.

[8] Eichmann, D., "The RBSE Spider – Balancing Effective Search Against Web Load," *Proceedings of the First International Conference on the World Wide Web,* CERN, Geneva, Switzerland, May 25-27, 1994.

[9] Gunawardena, S., "Mapping Problem Oriented Notations to the Implementation Oriented Notation (ION)," M. S. Thesis, University of Houston – Clear Lake, in preparation.

[10] McCool, R., “The Common Gateway Interface,” National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, http://hoohoo.ncsa.uiuc.edu/cgi/.

[11] McCool, R., “NCSA httpd Overview,” National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, http://hoohoo.ncsa.uiuc.edu/docs/Overview.html.

[12] Mishra, R. R., “ION – An Implementation Oriented Notation for the Graphical Representation of OO Programs,” M. S. Thesis, University of Houston – Clear Lake, December 1993.

[13] Montulli, L., “Lynx Users Guide v2.3,” The University of Kansas, http://www.cc.ukans.edu/lynx_help/Lynx_users_guide.html.

[14] NCSA, “Introduction to NCSA Mosaic for Microsoft Windows,” National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, http://www.ncsa.uiuc.edu/SDG/Software/WinMosaic/HomePage.html.

[15] NCSA, “Introduction to NCSA Mosaic for X,” National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/d2-intro.html.

[16] Perera, D. N., “A Software Reengineering Process for an Object Oriented Environment,” University of Houston – Clear Lake, M. S. Thesis, December 1993.

[17] Reuse Interoperability Group, “Uniform Data Model,” Applied Expertise, Arlington, VA, 1993.

[18] Stonebraker, M. and L. A. Rowe, “The Design of POSTGRES,” *Proc. of SIGMOD '86 International Conference of Management of Data,* Washington, D.C., May 28-30, 1986, p. 340-355.