A WWW Front End to an OODBMS

Martin Sjölin

Intelligent Information Systems Laboratory (IISLAB)
Department of Computer and Information Science
University of Linköping, S-581 83 Linköping, Sweden
marsj@ida.liu.se

Abstract

We have created a prototype World Wide Web (WWW) front end to an OODBMS, LINCKS. LINCKS is a multi user object store with support for composite objects, database histories, multiple views, information sharing by linking, and parallel editing notification.

The prototype generates HTML documents on the fly from information pieces in the database. A document's logical structure is described by a simple grammar, as well as *where* in the database the different parts of the logical structure should be retrieved.

We can use LINCKS for tracking owner, source, and creation date of information pieces and documents. Instead of using "copy-and-paste" for information duplication, we are using links to the *real* source and thus maintain trace-ability.

Moreover, by applying different views to the *same document* root object we are able to provide: abstract; table of contents (of different levels); a chapter, a section, or a subsection; the full article; with or without footnotes; or any combination of the above.

Using any of the current crops of server (CERN, NCSA, Plexus, etc) we must create several different files for each of these views. These files must be created by hand or created by a program like LaTeX2HTML [Dra94] which makes it yet more difficult to maintain the documents.

This article is available at `http://www.ida.liu.se/labs/iislab/people/marsj/www94/www.html` or in postscript `http://www.ida.liu.se/labs/iislab/people/marsj/www94.ps.gz`.

1 Introduction

The World Wide Web (WWW) [BLCL+94] increases in size and usage all over the world. In fact, WWW is one of the fastest growing services on the internet [Pit94]. We make our papers, articles, and other pieces of information available on the WWW marking up their contents (text, graphics, sound, movies) in HTML [BLC93]. We fill our WWW servers with huge amounts of information – but what about maintenance?

When making a longer article available in HTML we would like to give readers a choice between: reading the whole paper, a section, a subsection, or only the table of contents, i.e. giving them a choice of different *granularity*. Using the current HTTP [BL93] servers we have to create separate files for each different choice! To have an article in LaTeX and make it available on WWW, we need to convert it using a tool like LaTeX2HTML, which creates yet another set of files from the original source. A possible solution to the maintenance problem is to use software like MOMspider [Fie94] which can be used to *verify* the consistency of a local web (link verifications). However it only operates on the *information structure*, not on the *information content*.

We have a partial solution to the problems of maintenance, sharing and re-use of information, and viewing of information at different levels of granularity. Our first prototypical WWW front end to LINCKS [LSP93], enables readers to browse the contents of a LINCKS database using any HTML browser. The underlying features of LINCKS are able to solve a few of the mentioned problems.

In this paper we give an overview of our prototype front end, starting with a brief description of LINCKS' architecture as well as the data model. Then we give a more thorough overview of the "document" structure used in LINCKS before we describe the WWW front end's architecture. Finally, we discuss experiences learned from the realization of the prototype and possible future extensions.

2 LINCKS Overview

LINCKS [LSP93, PL93] is based upon a client server, multi-users architecture, where the database resides on a single server machine and the clients are on workstations connected to the server by a local area network. LINCKS is made up of essentially four separate software layers:

1. The first layer, the NODE database layer, resides on the server machine and is made up of three processes:
   - The `monitor` process serializes creation of objects, imposes access control (it allows you to define which groups of objects a particular user has *read* and *write* access to), and implements notification upon object creation.
   - The `netserv` process handles connections from clients and user validation before creating the 'dbs' process.
   - The `dbs` interacts with a client – one per connected workspace. It retrieves and stores objects in files using the monitor (to ensure unique object identifiers, etc). The `dbs` has the same lifetime as the client.
2. The second layer, the NODE workspace layer, is encapsulated in the `liblincks` library (described in [LIN88]) and works together with layer 1 to form the basic object-centered[1] database management system which we refer to as NODE [Pad88].
3. The third layer, the Composition Management (CIM) layer, filters out substructures in the database and represents these as a finer grained structure of items. The central data structure of this layer is what we call the *reference structure*, see section 3.
4. The last layer (4), the Interface layer, displays the leaf items of the reference structure in a window display. It provides the interface to the editing of the reference structure and via that, the information components. Currently the main user interface for editing `xlincks` [IIS93] is built using X11 and where editing of the leaf items is done directly on the display using *emacs*–like commands. The WWW front end is also built on top of the third layer.

2.1 LINCKS Data Model

An *object* in LINCKS is represented by two kinds of nodes [Pad88] *versions* and a *history structure* node. Versions represent the actual appearances of objects at certain times. The history structure nodes contain object information and historical information of the objects. The historical information is represented by different partial orders (see section 2.2). A node has three distinct parts [Pad88] which are:

- the *image*, which contains uninterpreted information (textual, graphic, audio information etc.) which is application dependent,
- a number of *attributes* with values (reminiscent of a frame-like representation or semistructured objects [MGL+87]). Attributes are single valued and describe the object's properties,
- and a number of *links* which describe the object's relationships to other objects. Links can have multiple values, (in a document each section is implemented as 'section link'). Links are used to build composite objects, or to simply establish a connection or a relation (e.g. a hypertext link).

[1] LINCKS is object-centered. It does not include the message paradigm for reasons outlined in [PR86].

2.2 Historical Information

NODE [Pad88] maintains information regarding the history of objects, composite objects and actions within the system. Past versions of objects can be accessed and reactivated. The *development history* informs us about what version of an object is used to create a new version of a (possibly different) object (also in ObjectStore [LLOW91], Iris [BM88], and Ontos [Cat91]). The *temporal history* of an object is an order between the different versions of that object and reflects the order of the versions' creation time. We allow several users to take out an object at the same logical[2] time and to create new versions of that object. In this case we say that logically all the new versions come after the latest common version, but there is no order between the new versions. Versions that cannot be ordered following creation time are said to be *parallel*.

The *command history* [Hal92] is a partial order which gives information regarding the temporal relation between commands that have been issued by various users of the system. It also contains information about what objects are affected by each command.

2.3 Linking

LINCKS supports both *dynamically bound links* and *statically bound links*. A *dynamically* bound link is between a source object and a target object where the binding to a particular version of the target object occurs on demand (by default the most recent version). Thus, it is not necessary to introduce new links every time the target object changes. We also have *statically* bound links which link directly to a particular version of a target object. Link resolution is in this case neither needed nor wanted.

3 Documents & Views

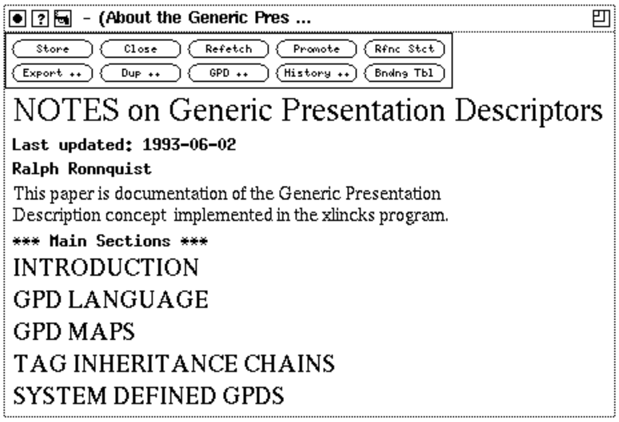

Figure 1: LINCKS Table of Contents View

A document (or rather a document view) corresponds to the content of one HTML page or one window in `xlincks`. A typical instance of a table of contents from `xlincks` can be found in figure 1 and the corresponding page from WWW is in figure 4. The view is constructed from several objects. Each section title is contained in a separate object, thus the root object is a *composite object*. In LINCKS we construct a view by applying a general presentation descriptor (GPD) to a root object (often a composite object).

[2] Two users take out an object at the same logical time if the object they access has the same state at both accesses, ie. there has been no change in the database involving that object between both accesses.

3.1 General Presentation Descriptor (GPD)

The general presentation descriptor[3] contains no actual data but provides the framework for both retrieving and storing the different parts of an object. The information in the general presentation descriptor is divided in following parts [PL93]:

- The `STRUCTURE` part which defines the logical structure of the view and its decomposition into items or structures. It similar to the DTD in SGML [Gol90], but much simpler.
- The `ACCESS` part defines *where* in the database we find the information we need to construct the reference structure.
- The `EXPAND` part defines which GPD should be used when following a hypertext link (e.g. expanding an object).
- The `EDIT MODE` part defines restrictions on the terminal items – if reading, writing, expanding and linking from is allowed.
- The `FORMAT` specifies X11 format attributes used by **`xlincks`** – font, borders, margins etc
- The `FORMATX` specifies conditions and actions to generate textual output (HTML or LaTeX[Lam86]) from the reference structure.

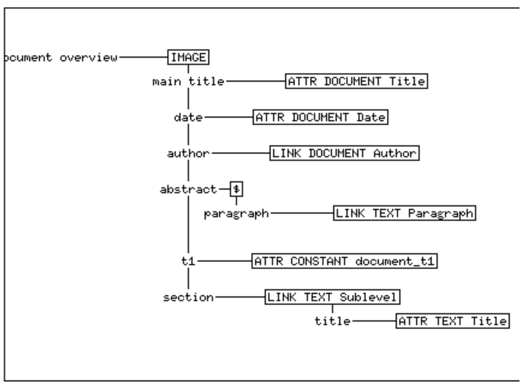

Figure 2: GPD Template for 'TOC View'

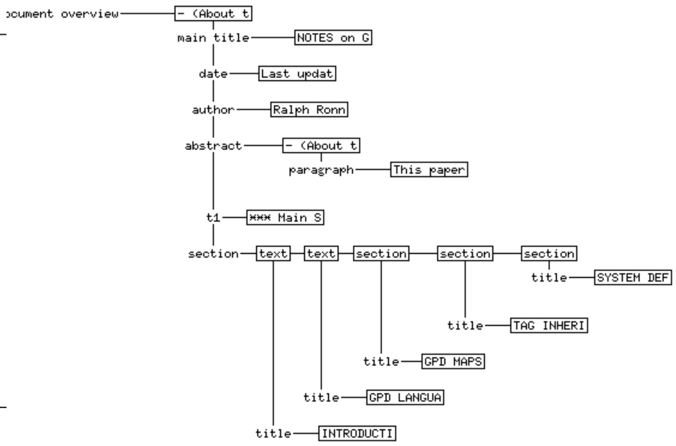

Figure 3: Reference Structure for 'TOC View'

The *reference structure* represents a view of a particular object as it exists at a particular time. It is built using a *template* and actual data objects, the *content objects*. The template itself is constructed from the *structure* and *access* parts in the GPD. If we apply the template in figure 2 to a root object, we end up with the reference structure in figure 3. A reference structure is thus an instantiated template and has the form of an ordered tree with named branches. The view in figure 1 is the result of displaying the leaves of the corresponding reference structure (in figure 3).

3.2 Composite Objects & Multiple Views

Composite objects are built using a general presentation descriptor, which provides a template for a composite class (e.g. the links have the part-of intuition), and contents objects. The contents objects are the root and the components of the composition. This way of handling compositions gives us a strong flexibility. It

[3] General presentation descriptors are objects in the database and have their own GPD to create and change themself.

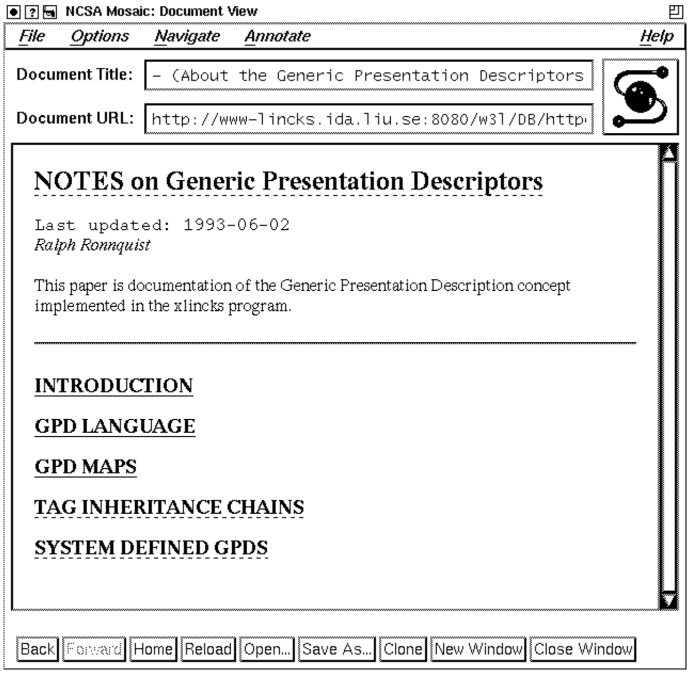

Figure 4: WWW Table of Contents View

allows us to share contents objects between different compositions and allows different views of compositions sharing the same root. It also allows us to use different GPD with the same root object, giving us different compositions (or views of a composition) sharing the same root – see figure 4 and figure 5.

4 WWW

The current prototype server is implemented via a "CGI-script" [McC93] running under a normal NCSA httpd server [McC94]. Today, only the *GET* method is implemented. Since we envisage to extend the set of supported methods, we are writing our own HTTP [BL93] server. In figure 6 we have shown the processes after the creation of the CGI process following the GET request. The HTML is generated from actions specified in the `FORMATX` part of the GPD for each item, terminal or non-terminal (as defined in the `STRUCTURE` part). The actions are written in a small application language[4] using the parser included in the CIM layer.

4.1 The CGI Program

The CGI program `cgi2lincks` takes one parameter - a path to a `key file`. The key file contains the host where LINCKS client `w3lincks` is running, the port number to connect to, current LINCKS user name for this incarnation of `w3lincks`, and a random key generated by `w3lincks`. This setup was inspired partly by the SQL gateway [Ng93].

After extracting the URL and the "virtual" path from the environment (per the CGI specification [McC93]), `cgi2lincks` connects to `w3lincks`, sends one line containing the key and the URL, and waits for the

[4] For the second prototype, which is in preparation, we envisage to switch to either TCL or Python.

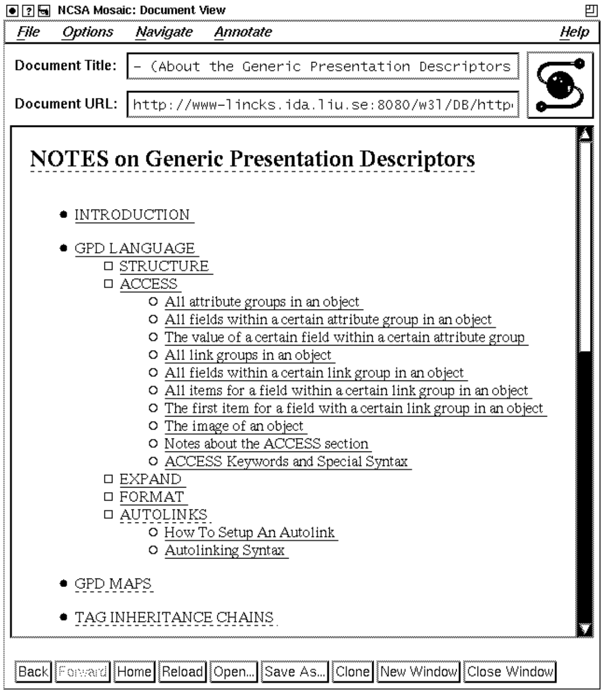

Figure 5: WWW Alternate Table of Contents View

response. The CGI program copies any information sent by `w3lincks` to standard output. If there was no output generated by `w3lincks`, a warning message is sent.

4.2 The LINCKS Client

The LINCKS database client `w3lincks` is similar to the X11 client `xlincks`. We removed all X11 related code and added a new general purpose module for generating HTML (or *any* text).

When starting `w3lincks`, it connects to a LINCKS database and login as a normal user. It writes the **`key file`** file, and waits for a connection from `cgi2lincks`. Notice that the `w3lincks` can only process one

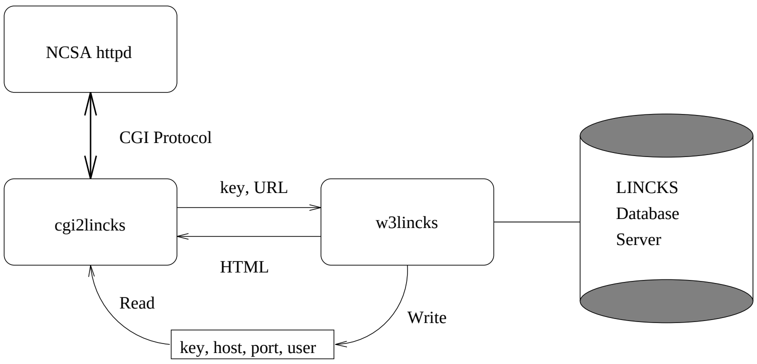

Figure 6: WWW Front End Architecture.

request at a time. When receiving a connection from the `cgi2lincks`, `w3lincks` verifies that the supplied key matches its own key. Moreover, we can restrict `w3lincks` to only accept connections from a specific host[5]. After the verification step, we extract the view (GPD) from the URL as well as the path to a root object. The current syntax of the URL is[6]:

```
URL         ::= /&lt;segment&gt;/ ... /&lt;segment&gt;-<gpd>
&lt;segment&gt;   ::= <name> | <name>:<position>
<name>      ::= <shortname> | <group> : <field>
<gpd>, <group, <field>, <shortname> ::= <id>
```

The "shortname" case is an internal mapping from a pair (*group*, *tag*) to a unique letter combination. This was initially implemented to keep the generated URL short (cf. 'TP' with 'Text:Paragraph'). The `group:field` is a name of an attribute or a link. The position parameter is only valid for links, and it enumerates the different object that the `group:field` points to. The position parameter defaults to zero (0), i.e the first link.

The starting point for the search is the 'home page' of the login user (of `w3lincks`). Users can specify a special 'home page' if they would like to avoid exporting their normal home page. Starting from the 'home page', we traverse the path, following the links to the next object, looking up any "shortname". At the end of the path, we should have a root object and a view (GPD). To create a reference structure tree (section 3), we apply the GPD to the root object. Then, we do an infix traversal of the reference structure to generate output. For each of the nodes of the reference structure tree, we execute the statements in `FORMATX` part. Whenever the conditions in the `FORMATX` part is true, we execute the corresponding action which could result in the output of HTML or the extraction of images/sounds from the database.

4.3 Actions & Conditions

The small language in the `FORMATX` part has a very simple syntax (lisp reminiscent) and basicly consists of expressions and function applications.

```
<FORMATX> ::= <formatxs>
<formatxs>::= <formatx> | <formatxs> , <formatx>
<formatx> ::= <function> ( <formatxs> )   -- function
              <string>   ( <formatxs> )   -- FORMAT( .... )
              <string>   | <integer> | <identifier> | <var> | <attr>
```

When traversing the reference structure in infix order, the attributes below are used to restrict when and where the actions (e.g. mostly `FORMAT(...)`) are executed. The attributes are restrictions on the current position in the reference structure (tree). Notice that the two last attributes are special cases since they causes side effects. The `ONCE` attribute is used for executing an action only once and sets a flag in the reference structure. The `NOREMAP` attribute avoids re-mapping of the input characters (e.g. '<' to '<' for HTML) in the leaf items.

```
AFTER/BEFORE   -- before and after treating children
FIRST/NOTFIRST -- first/not first for sibling nodes
LAST/NOTLAST   -- last/not last for sibling nodes
LEAF/ROOT/NODE -- is it a leaf, root, or an internal node?
PLACEHOLDER    -- is it a place holder (i.e. no data in database)?
ONCE           -- action is executed only once
NOREMAP        -- disable remapping (to HTML, depends on EXPORT setting)
```

[5] Using the key and restricting the allowed host seems to give us a reasonable level of security.

[6] In the second prototype the syntax will be changed and the GPD part will be determined from the context (the path taken to reach the final object).

For the actions we have the following set of functions which have been sufficient so far:

```
CASE(integer-expr, 0-formatx, 1-formatx, ...)
IF(integer-expr, formatx, formatx)    -- true if non-zero
EQ/GE/GT/LE/LT/NE(integer-expr,integer-expr)
ATTR(grp0,fld0[,pos],grp1,fld1, ...)  -- get an attribute value
LINK(grp0,fld0[,pos],grp1,fld1, ...)  -- get an binary image value
I2STR(image)                          -- binary image to string
N2STR(integer)                        -- number to string
UNESCAPE/ESCAPE(string)               -- for handling URLs
FORMAT([printf-string] [,attribute]* [,string-expr]*)
MACRO(name)                           -- execute macro 'name'
EXPORT(name)                          -- calls ROOTBEFORE and ROOTAFTER
```

Notice that the `EXPORT` function is only executed for the root node of the reference structure. The purpose is to specify: the output mode and thus character set mapping, default macros, default short name, as well as to execute prolog and epilog actions. The `FORMAT` corresponds to the 'C' library function `printf`, but we are only using "%s". It is used to output the strings in leaf items in the reference structure.

We have the following set of variables for retrieving information, which can be used in any expression:

```
%GPDEXP%   -- expansion GPD (view to use)
%GPD%      -- name of current GPD (used in current view)
%ROOT%     -- path from user root object to root object for this view
%RPATH%    -- relative path from view root object to this item
%PATH%     -- equal to %ROOT%/%RPATH%
%SELF%     -- displayed item value (image or attribute value)
%SIZE%     -- number of bytes in %SELF%
%TYPE%     -- type of leaf item (LINKITEM, LINKGROUPTAG, ...)
%LINE%     -- number of lines in the current item
%POS%      -- current number of siblings (starting from zero)
%LEVEL%    -- level in the reference structure/tree
%GROUP%    -- group name for item
%FIELD%    -- field name for item
```

For example, when retrieving an image in `xbm` format from the database, we generate the appropriate `Content-Type` and `Content-Length` MIME headers using %SELF% to retrieve the image and %SIZE% to get the size.

5 Conclusion

LINCKS fits very well in as a document or image server on a single site, and in particular the support of multiple views generated from the same information. Further the underlying database server keeps all versions, and allows us to trace the original source and the owner of the information stored in the database. The linking mechanism allows easy sharing of information between several documents (e.g. headers, footers, personal information etc.).

The implementation of the first prototype was rather straight forward, since we could build on top of the CIM layer (section 2). The CIM layer creates the reference structure and we only had to add the extra statements in `FORMATX` for all GPDs and the small application language. Based on the short comings discovered during the implementation of the prototype, we have started the work on a second front end:

- Will be implemented as a stand-alone server (as well as CGI program), to reduce the number of connections/pipes.
- Replacing our home grown language with TCL [Ous93] or Python [vR94].
- Support for HEAD and other methods by making the `FORMATX` part independent of the GPD.
- Form support, where we will generate a HTML form from the reference structure. Fields to fill in corresponds to what we call "place holders". The field names will be mapped to positions in the reference structure using 'cursors'.
- Integration with future search mechanisms in LINCKS database server with search mechanisms in WWW.
- Today LINCKS (and `xlincks`) has support for parallel editing where users receive notifications when parallel editing of items occurs. We will add support for notification messages on the generated HTML pages.
- Markup language to allow basic markup within a paragraph. We intend to investigate SGML [Gol90], HTML and LaTeX during the close future.

6 Acknowledgments & Availability

A considerable number of people have been involved in the design and implementation of LINCKS. The most significant of these are Lin Padgham, Ralph Rönnquist, David Partain, Patrick Lambrix, Thomas Hall, Leif Running, Michael Jansson, Tim Teitenberg, Nilgun Mat, and Peter Åberg. A significant number of undergraduate students has also been involved to varying extents, but are too numerous to mention.

LINCKS Version 2 as well as an extensive user manual is available in the directory ftp://ftp.ida.liu.se/pub/lincks/ as `lincks-2.x.tar.gz`. LINCKS requires some sort of UNIX (tm) and X11R5 and it is available under the terms of the GNU General Public License version 2. For more information and/or questions, please send mail to `lincks@ida.liu.se`.

References

[BL93] Tim Berners-Lee. *Hypertext Transfer Protocol (HTTP)*. CERN, Geneva, Switzerland, 2.0 edition, 5 November 1993. http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html .

[BLC93] Tim Berners-Lee and Daniel Connolly. *Hypertext Markup Language (HTML)*. CERN, Geneva, Switzerland, 13 July 1993. Online documentation http://info.cern.ch/hypertext/WWW/MarkUp/HTML.html .

[BLCL+94] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The world-wide web. *CACM*, 37(8):76–82, August 1994.

[BM88] D. Beech and B. Mahbod. Generalized version control in an object-oriented database. In *Proceedings of the 4th IEEE Conference on Data Engineering*, pages 14–22, 1988.

[Cat91] R. Cattell. *Object Data Management: Object-Oriented and Extended Relational Database Systems*. Addison-Wesley, Reading, MA, 1991.

[Dra94] Nikos Drakos. From text to hypertext: A post-hoc rationalisation of latex2html. In *Proceedings of The First World Wide Web Conference*, Geneva, Switzerland, 25–27 May 1994. CERN. http://cbl.leeds.ac.uk/nikos/doc/www94/www94.html .

[Fie94] Roy T. Fielding. Maintaining distributed hypertext infostructures: Welcome to momspider's web. In *Proceedings of the First World Wide Web Conference*, Geneva, Switzerland, 25–27 May 1994. CERN.

[Gol90] Charles F. Goldfarb. *The SGML Handbook*. Clarendon Press, 1990.

[Hal92] Thomas Hall. Maintaining a command history in LINCKS. Final undergraduate thesis, Linköping University, Sweden, 1992.

[IIS93] IISLAB, Department of Computer and Information Science, University of Linköping, Sweden. *The xlincks User's Manual for Version 2.0 of the LINCKS Database System*, first edition, 1993.

[Lam86] Leslie Lamport. LaTeX*: A Document Preperation System*. Addison-Wesley, 1986.

[LIN88] LINCKS Project Group, Department of Computer Science, University of Linköping. *Application Interface to LINCKS on Sun; Programmer's Manual*, third edition, 1988.

[LLOW91] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The ObjectStore database system. *Communications of the ACM*, 34(10):50–63, 1991.

[LSP93] Patrick Lambrix, Martin Sjölin, and Lin Pagdham. LINCKS – a platform for cooperative information systems. Technical report, Department of Computer and Information Science, LiTH, Sweden, 1993. Research Report.

[McC93] Rob McCool. *The Common Gateway Interface*. National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 1993. Online documentation http://hoohoo.ncsa.uiuc.edu/cgi/overview.html .

[McC94] Rob McCool. *NCSA httpd Overview*. National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 1994. Online documentation http://hoohoo.ncsa.uiuc.edu/docs/Overview.html .

[MGL+87] Thomas W. Malone, Kenneth R. Grant, Kum-Yew Lai, Ramano Rao, and David Rosenblit. Semistructured messages are surprisingly useful for computer-supported coordination. *ACM Transaction on Information Systems*, 5(2):115–131, April 1987.

[Ng93] J. Ng. *A Mosaic-SQL Gateway*. National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 1993. Online documentation http://www.ncsa.uiuc.edu/SDG/People/jason/pub/gsql/starthere.html .

[Ous93] John K. Ousterhout. *An Introduction To Tcl and TK*. Addision-Wesley, 1993.

[Pad88] Lin Padgham. NODE : A database for use by intelligent systems. In *Proceedings of Third International Symposium on Methodologies for Intelligent Systems*, pages 190–199, Torino, Italy, 12–15 October 1988.

[Pit94] James Pitkow. Graphics, visualization, and usability center: Nsfnet backbone statistics page., March 1994. http://www.cc.gatech.edu/gvu/stats/NSF/merit.html .

[PL93] Lin Padgham and Jonas Löwgren. A user interface management approach for object oriented database applications. *Journal of Systems and Software*, 1993. in press.

[PR86] Lin Padgham and Ralph Rönnquist. From a technical to a humane environment : A software system supporting cooperative work. In *Proceedings of the GDI Internationl Conference on User Interfaces*, Rushlikon, Switzerland, 1986.

[vR94] Guido van Rossum. *The Python Programming Language*. CWI, Department CST, The Netherlands, 1994. Online tutorial, reference manual, library manual and source found at http://www.cwi.nl/cwi/people/Guido.van.Rossum/Python.html .

Author Biography

The author is currently preparing his licentiate thesis (corresponds to a MS) in computer science at the Department of Computer and Information Science, Linköping University, Sweden. Before settling down in Linköping, he worked as computer consultant with PC, Windows, Xenix etc, for one year. During the academic year 1989/1990, he studied computer science at the technical school/university ENSIMAG (Grenoble, France) financed by a one year scholarship from the French state. He has a BS in Mathematics from Uppsala University (1989) plus took courses in psychology. In parallel with his studies at Uppsala University he worked part time with RDBMSs and as a system manager of VAXs (VMS) during a period of 4-5 years. Native of Uppsala, Sweden, but has spent almost two years in Grenoble, France - re-learning French, studying, working, skiing and enjoing 'La cuisine Française'!

The current e-mail address is `marsj@ida.liu.se` and http://www.ida.liu.se/labs/iislab/people/marsj/