LiTH-IDA-R-88-26 ISSN-0281-4250

INSTITUTIONEN FÖR DATAVETENSKAP
UNIVERSITETET OCH TEKNISKA HÖGSKOLAN
LINKÖPING

NODE:
A Database for Use by Intelligent Systems

by

Lin Padgham

RESEARCH REPORT
RKLLAB, September 1988

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

The Department of Computer and Information Science
Linköping University

PhD theses:

* Available at: University of Microfilms Intl., 300 N. Zeeb Road, Ann Arbor, MI 48106, USA.

(*Linköping Studies in Science and Technology. Dissertations.*)

*No 77	**Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982.
No 94	**Hans Lunell:** Code Generator Writing Systems, 1983.
No 97	**Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983.
*No 109	**Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984.
*No 111	**Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984.
*No 155	**Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons. 1987.
No 165	**James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning. 1987.
No 170	**Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems. 1987.

Licentiate of engineering Theses:

(*Linköping Studies in Science and Technology. Theses.*)

No 48	**Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching. 1985.
No 52	**Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems. 1985.
No 60	**Johan Fagerström:** Simulation and Evaluation of an Architecture based on Asynchronous Processes. 1985.
No 71	**Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System. 1987.
No 72	**Tony Larsson:** On the Specification and Verification of VLSI Systems. 1986.
No 73	**Ola Strömfors:** A Structure Editor for Documents and Programs. 1986.
No 74	**Christos Levcopoulos:** New Results about the Approximation Behaviour of the Greedy Triangulation. 1986.
No 104	**Shamsul I. Chowdhury:** Statistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology. 1987.
No 108	**Rober Bilos:** Incremental Scanning and Token-based Editing. 1987
No 111	**Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments. 1987.
No 113	**Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge. 1987.
No 118	**Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs. 1987
No 126	**Dan Strömberg:** Transfer and Distribution of Application Programs. 1987
No 127	**Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems. 1987
No 139	**Christer Bäckström:** Reasoning about Interdependent Actions. 1988.
No 140	**Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing. 1988.

LiTH-IDA-R-88-26 ISSN-0281-4250

NODE:
A Database for Use by Intelligent Systems

by

Lin Padgham

We describe here a database, NODE (Network Object-Oriented Database Environment), which has been built as the base layer for LINCKS (Linköpings INtelligent Communication of Knowledge System), an ongoing research project in the Knowledge Representation Laboratory at Linköping University.
NODE is a networked molecular database, which is object oriented, and has within the database historical information regarding the development of objects over time. We describe the features of the database, give examples of its use, and explore aspects which make it particularly suitable for intelligent systems.
The database was designed and built especially for the LINCKS project, which is a project aiming at a highly integrated intelligent system for use by end users in a wide variety of application areas. We give a brief overview of some of the goals in the larger project which motivated the design of the database. We also discuss some A.I. applications which are being built using NODE.

Accepted for publication in Proceedings of ISMIS '88
Third International Symposium on Methodologes for Intelligent Systems
Torino, Italy, October 12-15, 1988.

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

INTRODUCTION

Traditional database systems have grown up to fill certain needs. They should be able to contain large amounts of information, though often of a relatively homogeneous and non-complex nature. Access of that information should be fast in answering queries that relate directly to the contents of the data base (i.e. retrieving the information that is directly there). They should be reliable.

These systems are however lacking when it comes to the needs of databases for A.I. systems. Here the sort of information needed is often less homogeneous in nature and requires a richer representation than that offered by traditional databases. The use of the information is also more complex, requiring reasoning beyond mere retrieval. This in turn requires that extra information be available to reason with. Much of this information is relatively routine, but is not kept within traditional databases (e.g. historical information). While efficiency is still an issue for A.I. databases, it does not override these above mentioned issues.

In response to the deficiencies of traditional databases there have emerged several trends. One is the many expert system shells which provide facilities for building up a specialised specific purpose database. Another is systems such as 'Notecards' [6] which are more general purpose, but still lack important features such as multi user capacities and history information.

DESCRIPTION OF NODE

We were motivated to build NODE because we were interested in building a research system, which had access to a general purpose database which could include documents, and could also house the kinds of information needed in the knowledge base of 'expert systems', or as we prefer to call them 'intelligent services'. Our aims are to explore the kinds of intelligent services needed in a relatively non-specialised real world information management system such as an office system, or medical system, and the kinds of knowledge representation needed to build such intelligent services.

In examining our needs we discovered that none of the existing database systems which we knew of could meet our requirements. One possibility was to build a knowledge based system on top of an existing database system, the other was to build our own database system. We chose the latter approach. Following is a description of the resulting software.

Networked Molecular Database

We have often described NODE as a networked molecular database: networked because objects within the database are connected by means of links, which can be of many different kinds; molecular because of the basic data structure we use throughout the system, which we call a molecule. The name molecule was chosen because it has the connotation of a relatively small, basic building block. It also has within itself a certain structure. The molecule structure we use has three distinct parts:

* An *image* of the object. This is uninterpreted information, usually textual or graphic (e.g. a paragraph in a report, or a diagram). It is expected that the textual part of a

single molecule will be relatively small - the size of a single paragraph or thought.

* A number of *attributes* with values (e.g. name or colour). These attributes are properties of the object which are not related to other objects.

* A number of *links* which describe the relationship of the object to other objects (e.g. footnote or reference). These links are pointers from one object denotion to another.

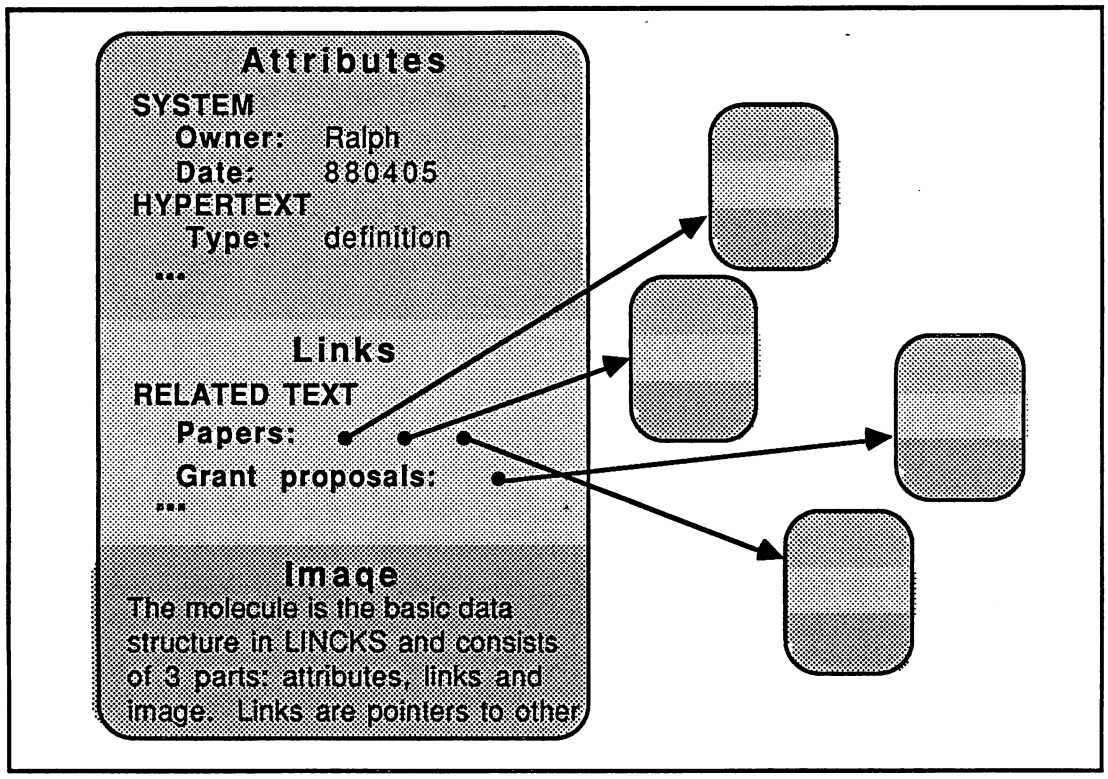


Diagram of a sample molecule

The choice of this data structure was an attempt to find a representation that was sufficiently rich that it would be adequate to represent the wide variety of objects found in a complex environment, but also sufficiently structured that it could be interpreted by appropriate software.

The links within the database are essentially pointers to objects. They can be pointers to an object without reference to time, in which case when the link is followed one will obtain the current version of the object, or they can be pointers to an object as it existed at a particular time. In this case what is pointed to is fixed, and on following the link one does not see if the object has in fact developed further since the time the link was set up. It is expected that

most common linkage will be to objects without reference to a time point, so that changes and development are automatically propogated throughout the system. However the need for explicit reference to an object as it existed at a certain time can also be envisaged. (Further details regarding implementation of versions of objects are given in a following section).

Links have a two layer labelling scheme, where the first label is a group name in which there exist some number of field names. Field names are local to the group making it easy to achieve unique group.field tags within any molecule. Each group name must occur only once within each molecule. A group.field tag can have as many links to other objects as desired. These links can be ordered by position if desired.

Attributes have a similar two layer labelling scheme to links. However there is only one value for each atribute group.field tag. This value can be of any type desired by the application (e.g. number, string, bitmap, record, array), giving great flexibility in the way the database is used.

Object Oriented Database

NODE is object oriented, in that real world objects (such as papers, filing cabinets, forms etc.) can be represented directly within the computer system. Object attributes and relationships to other objects are described within the basic data structure of the database.

The expectation is that most objects are complex objects and can best be represented by a structure of other objects. For example a cabinet can be seen as being composed of many drawers, each of which in turn contains a number of files or other entities, which in turn contain papers, etc. Similarly a paper or report can be seen as being composed of sections, subsections and paragraphs.

It is also expected that applications will define within the database certain prototype or type descriptor objects. These can then be linked to in order to facilitate inheritance of default information. They can also be compared to in order to reason about what type of object something may be seen as (within a particular context), on the basis of its described characteristics and how well these characteristics fit with available prototypes.

The model for the LINCKS system does not include message passing, or methods which are strictly attached to types, for reasons which we believe are theoretically sound [2]. A result of this definition of operations separately from objects (though related via a context mechanism) is that actions themselves become distinct entities within the system, about which it is possible to keep various kinds of information. This is in many ways more suitable for expert system type applications, as it very easily allows actions to carry information about pre-conditions for execution, expected post-conditions, abnormal effects, alternative mechanisms, etc. Within the database actions are just another type of object, part of which is a connection (directly or indirectly) to code which does something. The mechanism which interprets a user command, identifies the appropriate action within the database, and extracts and executes the appropriate code is not yet implemented. One can see this interpretor program as part of the system built on top of the database. However if some of the database code itself should be actions within the database, then the action retrieval mechanism needs

eventually to be integrated at a lower level.

The disadvantage of integration of the view of actions as separate objects within the low level database is that the database then becomes fixed to our admittedly controversial interpretation of object oriented. If the database has no knowledge of actions, where they are stored and how they are instantiated, then it is equally easy to build up a traditional type object oriented system on top of NODE. Messages would simply be attributes of an object type declaration, which are then interpreted in a particular way by the overlying software. The resulting system would still have the benefits of a multi user database, with parallelism and development over time, which are lacking in most (if not all) current object-oriented systems.

It may be questionable where the line should be drawn (if at all) between the database and the eventual larger system. However we believe that it is very important for a system which is to be used for building intelligent applications to have an explicit and rich representation of actions. This is not a part of the current database, but is very much a part of the model for which the database is a starting point.

Another consequence of separating actions from type descriptions is that end users are free to set up and modify type hierarchies - something which we believe is very individual, and user/task dependent - without having to concern themselves with programming the actions appropriate for those object types. Of course if an action is needed which cannot be composed of already existing operations, then it must be written. However any existing operation or combination of operations can be used on the new object type, and can if desired be given a new name, more meaningful to the envisaged context.

The current database does not give any interpretation to particular types of links, or have any routines for following inheritance hierarchies or collecting up the scope of an object. Such routines and interpretations are of course planned for the larger system.

Database with History

To deal intelligently with situations which arise in the real world it is often necessary to have information not only about the current state of affairs, but also about past states and actions. Even quite simple requests often have to do with information about past as well as present states. (e.g. Where is the folder which usually sits on the top shelf?) A database which is used by an intelligent system, which attempts to reason about the objects in the database, needs to contain historical information concerning its objects.

Another advantage of having history contained within the database is that it supports the relatively smooth and easy evolution of a system. By having operations and actions also as objects within the database, it is possible to continually evolve and improve one's actions. A simple mechanism can then determine whether a given action is compatible with some older object one wishes to use it with. In the case that it is not, the older compatible version of the action can be accessed and used instead. This avoids the problem of having to update many old objects when one changes the functionality of an action in some way.

We have identified three types of history information which we considered important in a database which should be used by an intelligent system.

Object history is the history of an object as it has developed over time. It enables reconstruction of the object as it was at any past point in time. The information within this type of history can be used in the answering of queries such as: What was the most recently printed and distributed version of document A? or: What was the version of action X which was current on July 1 1985?.

Edit history is the history of what was used as a base for this particular version of an object. When development of an object is straightforward (using always the previous version of the object as starting point for the next version) this will be identical to the object history. However it is not uncommon, particularly in document preparation, to use something other than the most recent version as a base. Text or formatting attributes for example may be taken from some other document, or from some earlier, now preferred version of the document. The information about what was used as the base for a version is not available in the object history, which gives only development information over time. In those cases where the two histories differ, it can be useful to trace what was used as a basis for a given version of an object.

Command history is the information about what actions have been done on what objects, and by whom. The command history is to be implemented as a structure of user actions, partially ordered over time, with links to those objects affected by the actions. A plan could well be represented as a similar structure of actions partially ordered over future rather than past time [3]. The keeping of command history will amongst other things enable plans to be checked against actual outcomes, so that plan revision can be initiated if an outcome is not as expected.

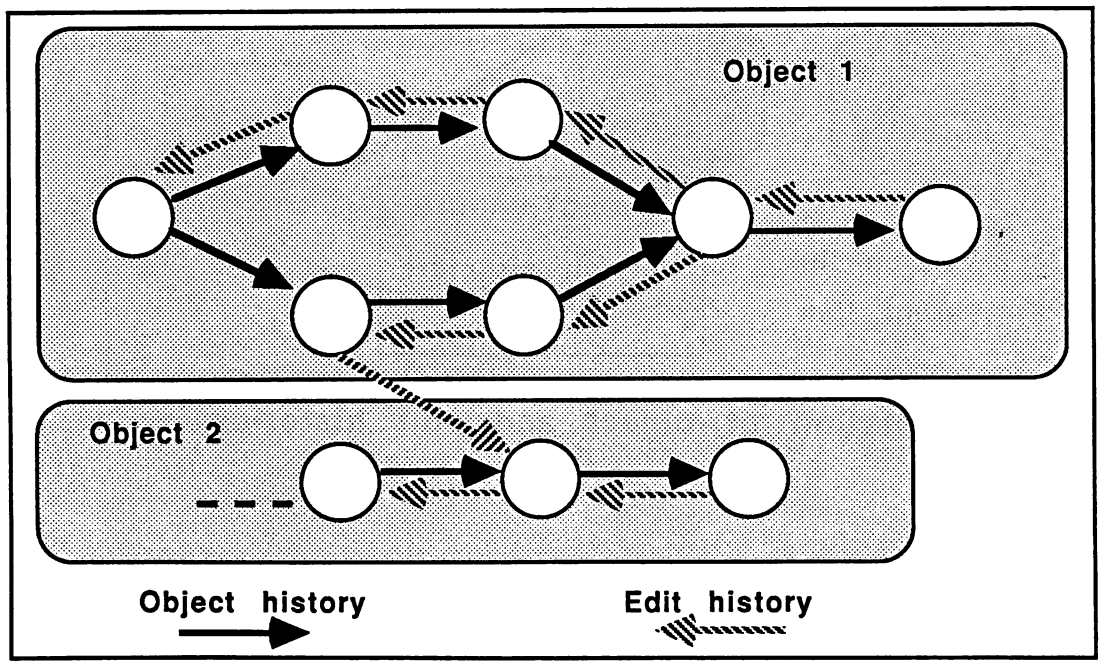
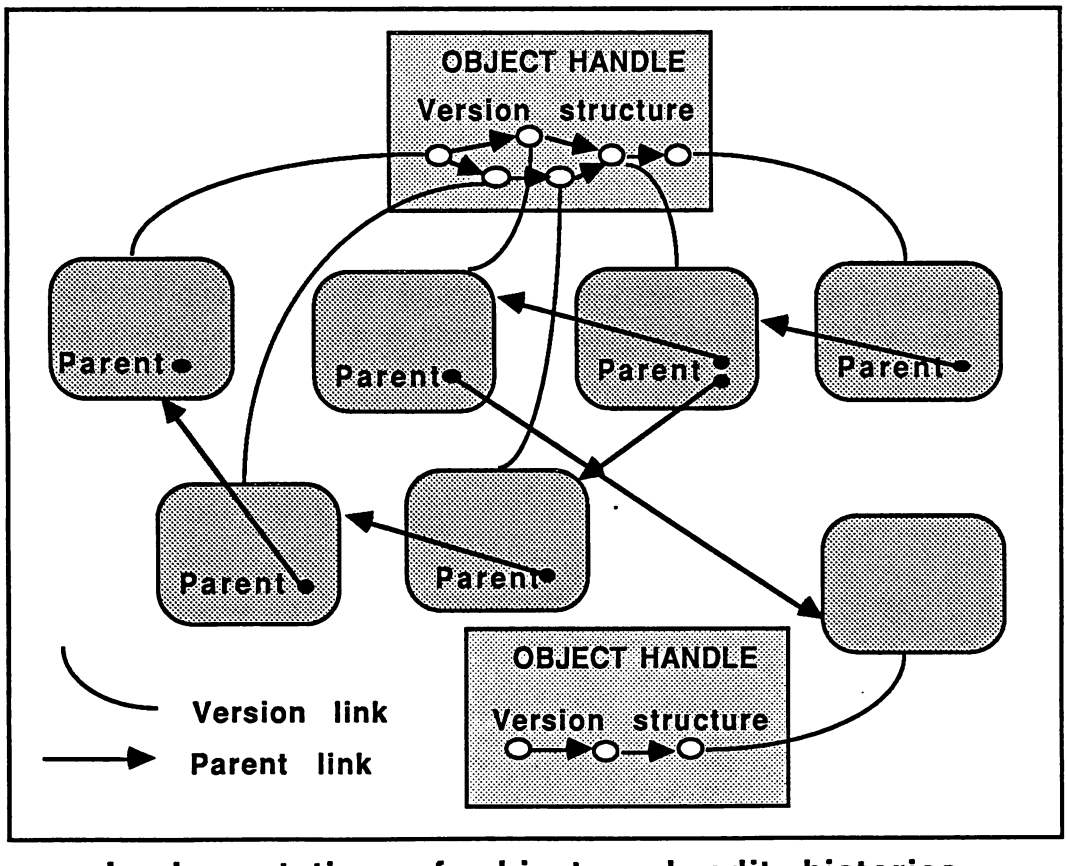


Diagram of difference between edit and object history

NODE implements the object and edit history structures, but not yet the command history. The object history is implemented by having one molecule which is the 'handle' to an object, and which contains a partially ordered (over time) graph of links to the versions of an object as they existed at various time points. Because the database is designed with the idea that objects will be made up of parts, which are in themselves objects, the primitive or smallest grain objects are expected to be fairly small, about the size of a single paragraph. The picture of a complex object at a particular time point is then built up from its pieces. The history saved is only the change of the relatively small piece. Links to an object where one wants to link to whatever is the latest version, are then a link to the 'handle' molecule, with default version. Links to an object at a particular time point are links to the handle molecule plus its specific version.

The edit history is implemented as a link to the 'parent' of a version. The command history is seen as a non-complex addition to the existing system. The reason that it was not implemented along with the other two types of history was that it was at a higher level of abstraction and could be implemented on top of the existing system.



Implementation of object and edit histories

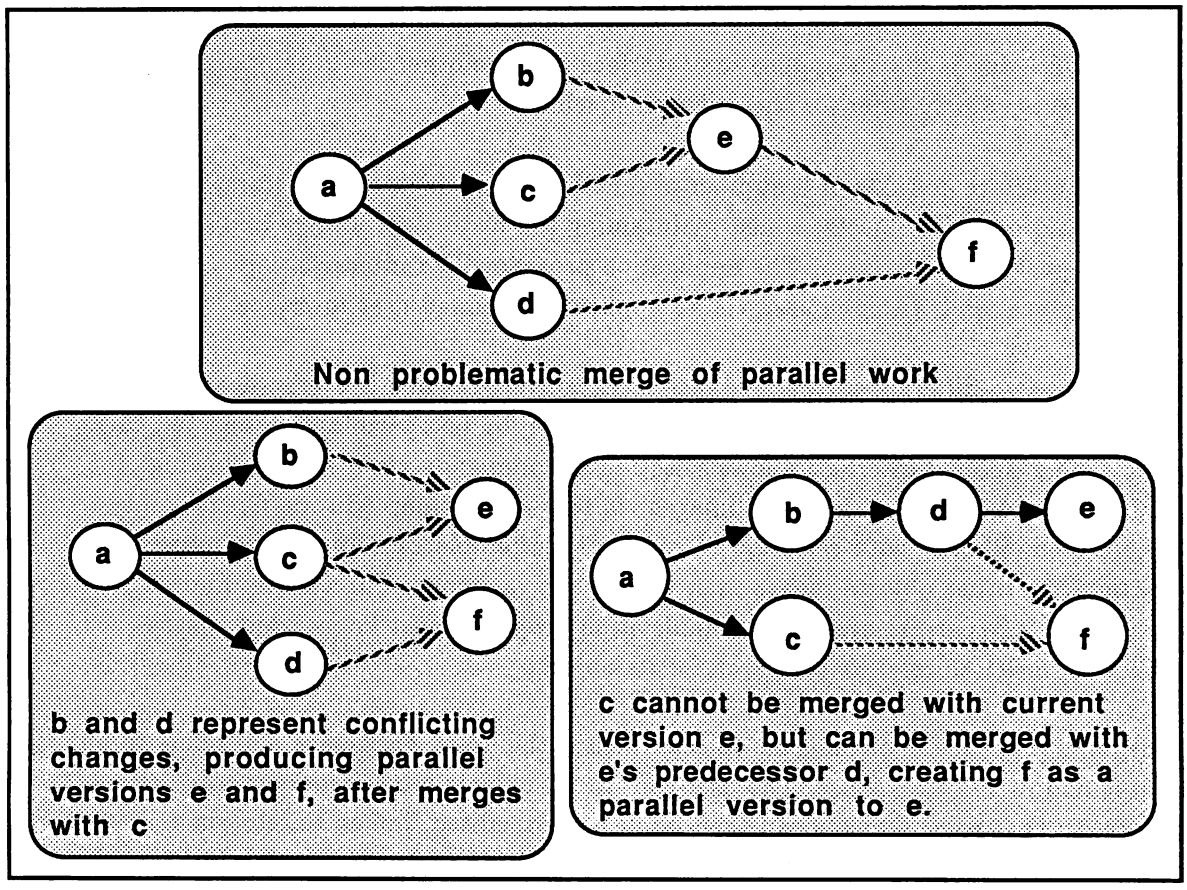
The database does not as yet contain any routines for binding the links from a chosen historical object to refer only to the versions of other objects which existed at that historical point in time. This is an obvious extension which is necessary for the history mechanism to be used effectively. The information to do this exists in the database, allowing the routines to be written at an application level. However it is more appropriate that such routines eventually

exist within the database itself.

Support for Parallel, Co-operative and Distributed Work

All modification of database objects takes place within a workspace which is logically, and possibly physically separated from the database. Labelling of objects in the workspace is local, thus avoiding the problems of different workspaces having to either communicate regarding labels for new objects, or having some global labelling scheme. There is an interface layer between workspace and database which translates object labels.

When an object is in a workspace and being modified it is not locked in the database. Consequently it is possible for two or more different users to modify the same object at the same (logical) time. When the modified objects are stored back into the database, the result is a merged version containing all changes, where that is possible. If the changes were made on the same small piece of an object so that they cannot reasonably be merged, then the result is parallel versions. If two objects a and b are taken out at the same logical time and a is put back and developed further before b is stored back, then b is merged with the latest version of a if possible. If this is not possible an attempt is made to merge with a's predecessor (or pre-predecessor) creating a parallel version to the latest development of a. This gives the same effect as merging b with the original version of a and propogating as far as possible along the development chain. The current implementation most often creates parallel versions, but we are working on methods for preferring merging as well as methods for specifying appropriate limits on merging.



Some possible parallel work situations

At the point that an object is modified within a workspace, the database is notified. This information is potentially available to an application, so that if desired a user could be notified that someone else was editing the same object. This allows a user to decide in the specific situation whether parallel work would be a desirable , or whether to work on something else instead.

The ability to have multiple versions of an object also allows for multiple, possibly contradictory versions of a 'fact'. These contradictory versions can contain auxiliary information, such as who entered this 'fact', other information supporting the 'fact', etc. This ability to allow for representation of contradictions, while leaving it up to the application as to how to deal with them, is an advantage for many real world situations. If one has contradictory information it is often preferable to keep the contradiction, than to choose one version. It is expected that this facility will be useful in building applications and tools that are non-monotonic in their reasoning.

Interface to NODE

The database storage program is written in 'C' and runs under Unix on a Sun III system. The workspace program which manages editing and viewing of objects runs on a Xerox lisp machine and on a terminal connected to a Sun system. The workspace programs are written in lisp and 'C' respectively. It is also planned to have a version of the program running on DG1 portables and Macintosh. These are both currently under development.

Because NODE was built as the base layer for the LINCKS system it is within LINCKS that we are planning the user interface to NODE. There are currently however two ways to interface to NODE - one is via an application interface and the other is via temporary interactive interfaces, which differ for the different machines.

The application interface consists of a number of function calls which create, store, edit, and retrieve objects and parts of objects, whilst maintaining the history structures and managing the multi user aspects of the system. The application program then interacts with NODE via the program interface, and with the user via an application defined user interface.

We currently have two different interactive interfaces to NODE - one is a graphical interface on the lisp machines, the other a command based interface for text terminals connected to a Sun network. The graphical interface allows selection of commands via menus, following of links via pointing, and editing of text and images via text and bitmap editors. Creation of objects is via a menu choice, with selection of various type templates. Objects are retrieved by following paths from other objects rather than via a query language. Thus a user finds objects by 'browsing' as in a hypertext system. An application program must know which paths (i.e. links) to follow in order to retrieve or search for its objects.

The text terminal interface maintains an object which is the users workspace and refers to all objects he last had in the workspace. The user can list all objects in the workspace by names which are local to that user. Objects can then be displayed, edited, linked to, etc via a command interface combined with emacs text editor for editing textual images.

We expect the LINCKS system to have a number of tools which aid in accessing objects from the database, or locating areas in the database where one may wish to browse. One of these tools will undoubtedly be a query language, but this is not yet designed in detail.

APPLICATIONS USING NODE

NODE was very much planned and built as a part of an overall project. However it is already evident that even without the sophisticated system software which is planned as a part of LINCKS, NODE is of potential interest for a number of applications. There are three applications currently in progress, which are making use of the database. The first is a prototype version of a programming environment for a project team, the second is a medical information system to be used by general practitioners, and the third is a material selection support system for mechanical engineering.

In the programming environment application the most important benefits of NODE are the ability to have parallel versions, and the flexibility of the linking between objects. One important use of parallel versions is to enable different persons using the programming environment to have a shared environment but to be able to have individual arrangements of various objects found within that environment. The flexibility of the linking system is important in setting up many different sorts of links such as default characteristic links, contents links, etc.

Both the medical application and the material selection application are decision support systems requiring a significant amount of 'intelligent' software. It is thus important for the database to contain the information needed for these intelligent services. The NODE data structure, the molecule, can be likened to a frame which has been shown to be suitable in many expert system applications.

Previous experience in the LIMEDS [5] project at Linköping University, has shown that in order to build sophisticated and intelligent support systems in medical informatics, it is necessary to have support for historical information regarding the development of objects, and also to have a very flexible mechanism for building relationships between objects. NODE appears to offer both of these facilities.

The material selection support system was implemented in a first prototype version[1] with a relational database, but as the complexity of the information, and amount of reasoning in the system increased it became evident that the use of an object-oriented database supporting structured objects was desirable.

SUMMARY

Time is an essential aspect of information about the real world. Much information about, for example causality, is implicit in, or relies on, information regarding time. In an ACM article on temporal databases, Richard Snodgrass gives examples of time related capabilities which he considers necessary but lacking in traditional database systems [4]. In his terms NODE is a

rollback database, allowing for viewing of the database as it was at previous points in time. It also has facilities for incorporating user defined time via attribute fields. Particularly the first of these is extremely important in answering even fairly simple queries about the real world. It is crucial information for any kind of reasoning about the world represented by the information.

A time facility which NODE does not have directly is the ability to retroactively or proactively enter facts. There is currently no direct method to enter something which will not take effect (be visible?) until some later time point, or to correct information which was wrongly entered, so that the correction is directly related to the time point of the original wrong entry. These facilities could of course be provided for particular applications, using the general purpose links and attribute mechanisms.

The facility for multi user parallel access, including ability to modify the database, without restrictive locking mechanisms, makes NODE an attractive tool for a multi user environment.

The ability for parallel versions gives a possibility for users to have differing versions of essentially the same object. This is important if a system is to be both a multi user system encouraging co-operation, and also a system which is non restrictive and allows for personal preferences.

The object oriented view where actions are also fully fledged objects appears to be promising for A.I. applications where it is usual to have information about actions (e.g. pre and post conditions for rule instantiation).

NODE has attempted to incorporate the concepts of time information, partial ordering, object orientation, and flexible connections between objects. These are seen as crucial lacks in traditional database systems if one wishes to incorporate them into an intelligent system.

REFERENCES

[1] Nyberg, Mats, A Computerised Materials Selection System, LiTH-IKP-R-495,1987.

[2] L. Padgham R. Rönnquist, LINCKS: An Imperative Object Oriented System. *Proceedings of the 20th Hawaii International Conference on System Sciences,* Jan 1987, vol. 1, pg. 516

[3] E. Sandewall and R. Rönnquist, A Representation of Action Structures, *Proceedings of American Association of Artificial Intelligence Conference* 1986.

[4] Temporal Databases, *Communications of the ACM*, September 1986, p.35-42

[5] Timpka, T. LIMEDS. Knowledge-based decision support for general practitioners; an integrated design. Proc. in SCAMC, H. F. Orthner, ed.; Washington, 1986, IEEE Computer Society 73, pp 394- 402.

[6] NoteCards Release 1.1 Reference Manual, Xerox Special Information Systems, March 1985

Avdelning, institution, fakultet
Division, department, faculty

Department of Computer and
Information Science
Linköping University

ISBN:

ISSN: ISSN-0281-4250

Rapportnr:
Report no: LiTH-IDA-R-88-26

Upplagans storlek:
Number of copies:

Datum:
Date: September 1988

Projekt:
Project:

Titel:
Title: NODE: A Database for Use by Intelligent System

Författare:
Author Lin Padgham

Uppdragsgivare:
Commissioned by:

Dnr.:
Call no:

Rapporttyp:
Kind of report:

- ☐ Examensarbete/Final project
- ☐ Delrapport/Progress report
- ☐ Reserapport/Travel report
- ☐ Slutrapport/Final report
- ☒ Övrig rapport/Other kind of report

Rapportspråk:
Language:

- ☐ Svenska/Swedish
- ☒ Engelska/English
- ☐ _____

Sammanfattning (högst 150 ord):
Abstract (150 words):

We describe here a database, NODE (Network Object-Oriented Database Environment), which has been built as the base layer for LINCKS (Linköpings INtelligent Communication of Knowledge System), an ongoing research project in the Knowledge Representation Laboratory at Linköping University.
NODE is a networked molecular database, which is object oriented, and has within the database historical information regarding the development of objects over time. We describe the features of the database, give examples of its use, and explore aspects which make it particularly suitable for intelligent systems.
The database was designed and built especially for the LINCKS project, which is a project aiming at a highly integrated intelligent system for use by end users in a wide variety of application areas. We give a brief overview of some of the goals in the larger project which motivated the design of the database. We also discuss some A.I. applications which are being built using NODE.

Nyckelord (högst 8):
Keywords (8):

Bibliotekets anteckningar

A Selection of Previous Research Reports.

LiTH-IDA-R-88-17 **Bengt Lennartsson:** Multi-Level Software Architectures - Industrial Experiences and Research Directions. This report is a revised version of a paper in *Advance Working Papers of The Second International Workshop on Computer-Aided Software Engineering.* Cambridge, Massachusetts. July 12-15, 1988.

LiTH-IDA-R-88-16 **Lin Padgham:** A Model and Representation for Type Information and Its Use in Reasoning with Defaults. Also in *Proc. of AAAI'88, American Association for Artificial Intelligence*, 1988.

LiTH-IDA-R-88-15 **Lin Padgham, Ralph Rönnquist:** LINCKS: An Imperative Object Oriented System. Also in *Proc. of the 20th Annual Hawaii Int. Conf. on System Sciences*, Hawaii, 1987.

LiTH-IDA-R-88-13 **Erik Tengvald:** Ett kartorienterat planeringssystem för autonoma farkoster, en design diskussion.

LiTH-IDA-R-88-12 **Erik Sandewall, Johan Hultman, Erik Tengvald:** Software Architecture and Programming Paradigms for Robotics Applications.

LiTH-IDA-R-88-11 **Mats Wirén:** An Incremental Chart Parser for PATR.

LiTH-IDA-R-88-10 **Mats Wirén:** A Control-Strategy-Independent Parser for PATR. Revised version of paper appearing in *Proc. of First Scandinavian Conf. on Artificial Intelligence*, Tromsö, Norway, March 9 - 11, 1988.

LiTH-IDA-R-88-09 **Peter Fritzson:** Incremental Symbol Processing.

LiTH-IDA-R-88-08 **Erik Sandewall:** Formal Semantics for Reasoning about Change with Ramified Causal Minimization.

LiTH-IDA-R-88-07 **Erik Sandewall et al:** IDA Annual Report 1987.

LiTH-IDA-R-88-06 **Christer Bäckström:** A Representation of Coordinated Actions Characterized by Interval Valued Conditions.

LiTH-IDA-R-88-05 **Christer Bäckström:** Keeping and Forcing: How to Represent Cooperating Actions.

LiTH-IDA-R-88-04 **Wlodek Drabent, Simin Nadjm-Tehrani, Jan Maluszynski:** Algorithmic Debugging with Assertions. Also in *Proc. of META 88 - Workshop on Meta Programming*, Bristol, June 21-24 1988.

LiTH-IDA-R-88-03 **Staffan Bonnier, Jan Maluszynski:** Towards a Clean Amalgamation of Logic Programs with External Procedures. Accepted to the *Joint Fifth Int. Logic Programming Conf. and Fifth Logic Programming Symp.*, Seattle Washington, August 15-19, 1988.

LiTH-IDA-R-88-02 **Ulf Nilsson:** Inferring Restricted AND-Parallelism in Logic Programs using Abstract Interpretation.

LiTH-IDA-R-88-01 **Jacek Leszczylowski, Staffan Bonnier, Jan Maluszynski:** Logic Programming with External Procedures: Introducing S-Unification. A shorter version is to appear in *Information Processing Letters*, North Holland.

LiTH-IDA-R-87-26 **Jonas Löwgren:** Applying a Rapid Prototyping System to Control Panel Dialgues.

LiTH-IDA-R-87-24 **Sven Moen:** Drawing Dynamic Trees.

LiTH-IDA-R-87-22 **Lars Ahrenberg, Arne Jönsson:** An Interactive System for Tagging Dialogues. Also in *Proc. of 14th Conference of the Association for Literary and Linguistic Computing*, Göteborg 1-5 June, 1987.

LiTH-IDA-R-87-21 **Harold W. Lawson, Jr.:** Challenges and Directions in Computers and Education.

LiTH-IDA-R-87-20 **Krzysztof Kuchcinski, Zebo Peng:** Parallelism Extraction from Sequential Programs for VLSI Applications. This paper is to appear in *Microprocessing and Microprogramming, the Euromicro Journal*, 1988.

LiTH-IDA-R-87-19 **Zebo Peng:** Construction of Asynchronous Concurrent Systems from their Behavioral Specifications. Also in *Proc. of the 10th World Computer Congress*, Dublin, Ireland, Sept. 1-5, 1986.

The Department of Computer and Information Science
Linköping University

organizes undergraduate and graduate studies in *Computer Science*, *Telecommunication and Computer Systems*, and *Administrative Data Processing*. Research activities have an emphasis on advanced software technology and computer systems design and are organized in a number of research laboratories:

- *ACTLAB - Laboratory for Complexity of Algorithms*, which is concerned with the design and analysis of efficient sequential and parallel algorithms, and complexity theory, especially in the areas of computational geometry, data structures on bounded domains and graph algorithms.
- *ASLAB - Application Systems Laboratory,* which studies design of advanced support systems for interactive use of computers, including tools for automated construction of applications software.
- *CADLAB - Laboratory for Computer-Aided Design of Electronics*, which concentrates its research activities around tools for integrated development of hardware and software, graphics-based modelling and simulation techniques.
- *LIBLAB - Laboratory for Library and Information Science,* which studies methods for access to documents and the information contained in the documents, concentrating on catalogs and bibliographic representations, and on the human factors of library use.
- *LOGPRO - Laboratory for Logic Programming*, which concentrates its research activities around foundations of logic programming, relations to other programming paradigms and methodology.
- *NLPLAB - Natural Language Processing Laboratory*, which conducts research related to the development and use of natural language interfaces to computer software.
- *PELAB - Programming Environments Laboratory*, which works with design of tools for software development, specific functions in such tools and theoretical aspects of programs under construction.
- *RKLLAB - Laboratory for Representation of Knowledge in Logic*, which covers issues and techniques such as non-monotonic logic, probabilistic logic, modal logic and truth maintenance algorithms and systems.

Research Reports 1988

LiTH-IDA-R-88-26 **Lin Padgham:** NODE: A Database for Use by Intelligen Systems. Also in *Proc. of ISMIS '88,* Torino Italy, October 12-15, 1988.

LiTH-IDA-R-88-23 **Johan Fagerström, Yngve Larsson:** Two Contributions on Debugging Distributed Systems. Also in *Proc. of ACM SIGPLAN/SIGOPS, Workshop on Parallel and Distributed Debugging*, Madison, Wisconsin, May 5-6, 1988.

LiTH-IDA-R-88-22 **Johan Fagerström:** Design and Test of Distributed Applications. Also in *Proc. of the International Conference on Software Engineering, ICSE*, Singapore, April 1988.

LiTH-IDA-R-88-21 **Johan Fagerström, Lars Strömberg:** A Paradigm for Distributed System Design and Test. Also in *Proc. of COMPCON'SPRING 88*, San Fransicso, Febr-March 1988.

LiTH-IDA-R-88-19 **Lin Padgham, Ralph Rönnquist:** From a Technical to a Humane Environment: A Software System Supporting Cooperative Work. Also in *Proc. of the GDI Internationa Conference on USER INTERFACES,* Rüsc

LiTH-IDA-R-88-18 **Nils** anding - a Com