LINCKS

—

A Platform for Computer Supported Cooperative Work

Patrick Lambrix Martin Sjölin Lin Padgham
Intelligent Information Systems Laboratory (IISLAB)
Department of Computer and Information Science
University of Linköping, S-581 83 Linköping, Sweden
{patla,marsj,linpa}@ida.liu.se

1994-03-06

Abstract

This paper presents the object-centered database system LINCKS as a platform for applications in Computer Supported Cooperative Work (CSCW). LINCKS is an object-centered database management system with extended support for multiple views, temporal data handling and composite objects. Several features, such as the parallel editing support, make sharing and reuse of objects and cooperation between users easy. The current main application interface to the database is a document editing environement under X11 with a hypertext like browser.

Keywords

CSCW, OODBMS, Multiple Views, Hypertext, Collaborative Writing, Temporal Data Handling, Composite Objects

1 Introduction

An important future area in information systems is computerized support for cooperative agents, where those agents may be either human or computer. The act of cooperation implies a mean of communication. Further, to reach a common goal the individual agents need not only to cooperate[1] but to also *collaborate*. Moreover, the agents need to have access to information, both actual and historical, as well as have support for searching, sorting, filtering and selecting [7] information from large repositories.

We identified the following requirements for a Cooperative Information System.

- Support for *Collaboration* and *Cooperation* among agents.
- *Sharing* and *Reuse* of information.
- *Multiple Views* since information usage is highly context dependent.
- flexible representation of *Complex* information, supporting different reasoning techniques (e.g classification).
- *Temporal* information handling, to ease information reuse, integrate legacy systems, and learn from experience.
- *Intelligent* support for tasks performed by agents.

In this paper we show that LINCKS fulfills these requirements. We describe the LINCKS architecture (section 2) and the object model (section 3). Our version model allows easy sharing and re-use of objects (even between different users) as well as founding a base for handling temporal information (sections 4, 5, 6). Complex information, and the special case of composite objects (section 8) is handled by using templates (section 7). These templates are also used to create multiple views of the same information.

2 Architecture

LINCKS 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.

2.1 Software Layers

LINCKS is made up of essentially four separate layers:

- 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, 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). Same lifetime as the client.
- The second layer, the NODE workspace layer, is encapsulated in the `liblincks` library and works together with layer 1 to form the basic object-centered[2] database management system which we refer to as NODE [22]. The NODE interface library `liblincks` is described in [17].
- The third layer, the Composition Management 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* which is used for displaying (composite) objects (see section 7).
- 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. Editing of the leaf items is done directly on the display using *emacs*–like commands.

  Editing of the interior structure is done via structure editing commands, which can be divided into to two different classes – *simple* commands (insert, remove etc) operating one terminal item or one on branch of the reference structure and *complex* commands (cut, copy and paste) operating one or more branches of the references structures at the same time (see [11] and [25]).

2.2 Workspaces and Sessions

LINCKS database is divided into a common shared area and a private area for each user. A user can update objects reachable[3] from the objects in the common area and in his own area.

Upon login to the system the user initiates a *session*. A *workspace* is created and it is via the workspace that the user issues all updates to the database. After loading an object, the directly reachable objects are loaded into workspace upon *reference*. The result of an object update is only available to other users (or in the same user's other sessions) after issuing an explicit store command[4].

3 Object Model

An *atomic object* in LINCKS is represented by two kinds of nodes (see figure 1) [2]: *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 5).

A node has three distinct parts [22, 2]. These 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 [18]). 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).

Nodes are identified by unique *node identification numbers*. An object is then identified by the node identification number of its history structure node. A particular version of an object is identified by a pair of node identification numbers: the object identifier and the node identification number of the actual version[5].

LINCKS supports both *dynamically bound links* and *statically bound links* (see figure 1). The dynamically bound link is a link between a version of an object and another object. The binding of the target object to a particular version of that object occurs on demand. Thus it is not necessary to introduce new links every time the target object changes. The version referenced by a dynamically bound link is time dependent and is in LINCKS by default the most recent version of that object. It is however possible for the user to interactively specify the desired version. In contrast to dynamically

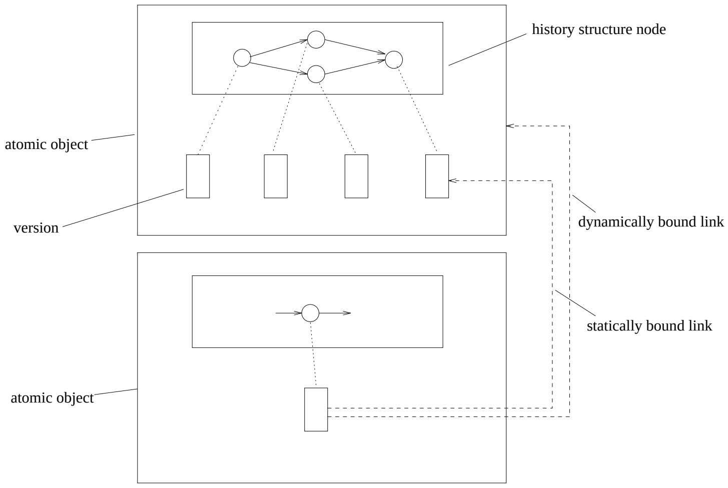

Figure 1: Atomic object in LINCKS

bound links one can also have statically bound links which link directly to a particular version of an object and do not require any further resolution.

4 Versions and Change Notification

Versions of an object have a status. *Transient versions* are versions which have been created by a user in his workspace, but which have not yet been stored into the database. After an explicit store command occurs, the version is promoted to a *working* version. A working version cannot be changed anymore. This means that whenever we change the working version, a new transient version is created.

4.1 Change Notification

LINCKS provides a basic change notification mechanism. A marker is created in the server when we create the transient version in our workspace. We notify users who retrieve objects having a marker in the database.

When a user A accesses an object which is being updated by user B, the user A receives a notification (see figure 2) that user B is already editing this object. If user A then starts to modify the object, user B will receive notification that user A have started to modify the object (see figure 3). The case is similar when several users update a certain object simultaneously. The notification consists of a message and visual aid (see figure 4). The objects (or parts of objects in the case of composite objects) which are affected are shaded.

5 Historical Information of Objects and Actions

An important aspect in object-centered database management systems is the ability to handle temporal data [27]. In [21, 22, 16] we argued that at least two kinds of temporal information are necessary: information about creation order and information about which versions are used to create other versions. There we also showed that we can do more extensive reasoning using these two kinds of historical information than would be possible if we had only one of them available. Our current implementation of LINCKS supports these different kinds of temporal information. It has a graphical interface to browse through the historical information and allows to promote older versions of an object to be the current version. In this section we introduce these two kinds of object histories and discuss historical information of actions.

5.1 Temporal History

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. In our model we allow several users to take out an object at the same logical[6] time and to create a 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. Consequently the temporal history is a partial order between the versions of an object. Versions that cannot be ordered following creation time are said to be *parallel*.

By allowing this kind of parallelism in the temporal

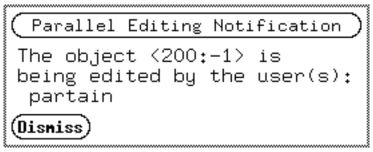

Figure 2: 'Another user is already editing' notification

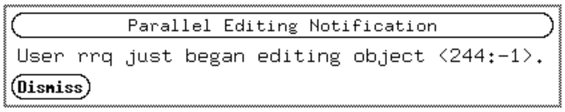

Figure 3: 'Somebody started modifying one of your modified objects' notification

history, several users can work at the same time using and even updating the same piece of information in the shared database without disturbing each other. Although this results in the potential problem of parallel versions which at some time must be merged this may be preferable to locking, given the fact that transactions are relatively long. If desired, constraints can be placed at a higher level to enforce locking and a sequential history. This would be just a simplification of the temporal model which we take as a baseline.

The model also allows that several users connect with the same database, take a copy, and work further disconnected from that database. When the user reconnects with the database all new information in the workspace is integrated into the database.

5.2 Development History

The *development history* is an ordering which reflects information about what version of an object is used to create a new version of a (possibly different) object. The new version is then called the *child*; the original version is the *parent* of the new version. As one can use something several times to create something new, the development history is also a partial order.

There is a natural constraint that the parent must be created temporally before the child. This constraint can be checked using the temporal history as long as the parent and the child belong to the same object. (In the case where another object provides the parent, the constraint can be checked using the *command history*.)

5.3 Command History

The *command history* [10] 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.

For each session a session object is created. The versions of this session object represent the different commands which have been issued by the user. We consider two kinds of commands. The commands which have an immediate effect on the database, such as a store command, are called *database commands*. The other commands, called *workspace commands*, have an effect on versions of objects which are in the workspace but not directly on the database, such as the updating of a transient version. The commands in one session constitute a linear temporal order. For other users it is interesting to know what changes have been done to the database during their own sessions. Therefore we also maintain information about the temporal order between database commands in different sessions.

6 Transactions on Atomic Objects

We can perform the following transactions on atomic objects. We state their exact meaning.

- **Access:** The objects reachable from the accessed objects are brought in into the workspace if necessary.
- **Update:** In the case the update is made to a transient version, nothing changes for the temporal history of the object. When the update is made to a working version, a new transient version is created which is a copy of the working version. The update is made to the new transient version. The new transient version comes after the working version in the

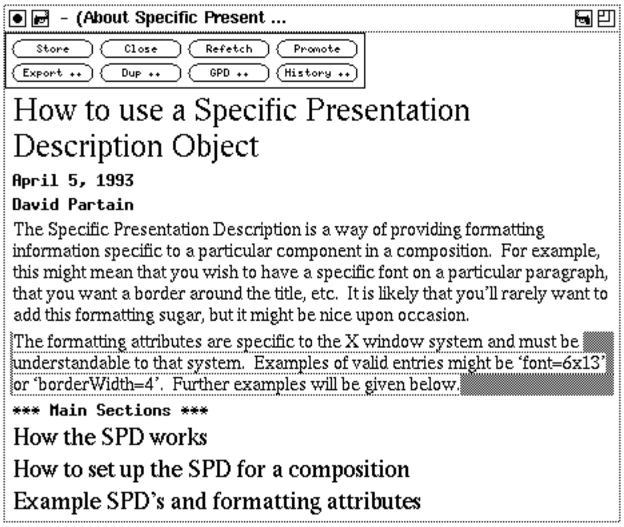

Figure 4: Window shading for parallel editing

development history and after the working version and any versions parallel to it, in the temporal history. The new transient version occurs only in the histories in the user's workspace.

- **Store:** Storing a working version has no effect. When storing a transient version, the transient version is promoted to a working version. The version appears from now on in the histories of the object in the database.

- **Copy:** (the original version and the copy version can belong to different objects) If the original version is a working version then the following happens. The transient versions of the copy object are promoted to working versions (by storing). Then a new transient version is created which is a copy of the original version. The new transient version comes after all working versions of the copied object in the temporal history and after the original version in the development history. (Development history may cross object boundaries.) The new transient version occurs only in the histories in the user's workspace. If the original version is a transient version, then this version is first promoted to a working version and is then copied as described above.

7 Presentation Description Objects and Reference Structures

A *reference structure* represents a view of a particular object as it exists at a particular time. It is built using a template, described in a *presentation description object*, and actual data objects, the *content objects*. A reference structure is thus an instantiated template and has the form of an ordered tree with named branches. Objects are displayed using the information in a reference structure.

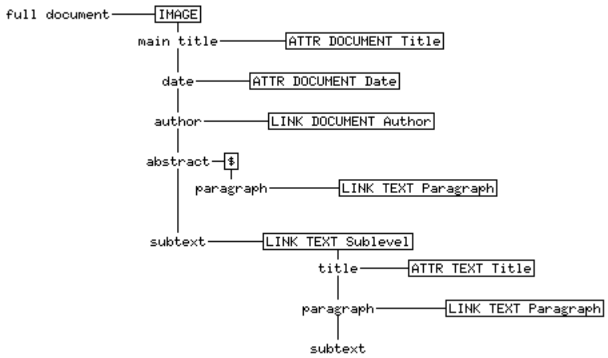

Figure 5: A template

The presentation description object[7] contains no actual data but provides the framework for both retrieving and storing the different parts of an object. Usually, a presentation description object is specified to apply to a whole class of objects. The information in

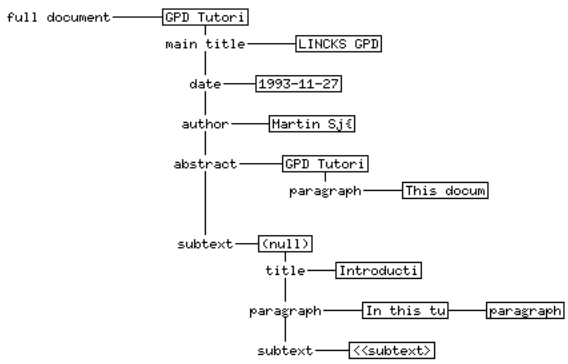

Figure 6: The Reference Structure

the presentation description objects is divided in essentially two parts [28, 23]. The logical structure of the template is given by the *structure* part. Both composite, linked and terminal atomic elements in the template are defined. Composite elements are themselves composed of sub-elements, which may be either composite or terminal atomic elements. The information for locating linked objects and the parts of the objects to instantiate the template is found in the *access* part. This information is also used to determine which database objects should be updated when performing updating operations towards a displayed object.

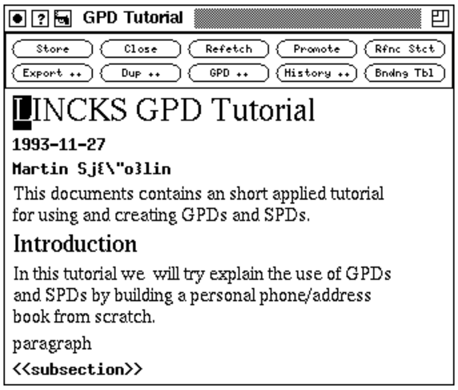

Figure 7: The resulting view.

When LINCKS is asked to create a reference structure to display a particular object, a presentation description object is chosen. This can be done in several ways. The most specific place to find the name of the presentation description object is to look whether the object itself defines which presentation description object to use. If this is not the case we look in the presentation description object of the object from which the desired object is originating. This can be seen as information belonging to the definitional context of the originating object. The final possibility is to use the type name of the object directly to find the presentation description object irrespective of context.

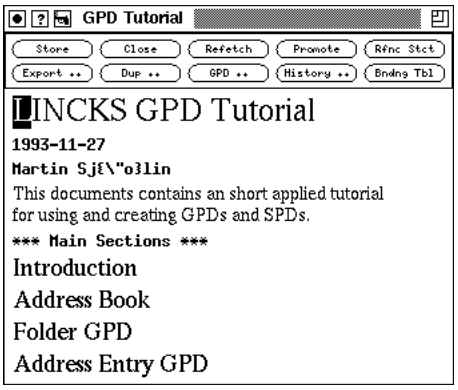

Figure 8: An overview view of the same document.

We allow for different presentation description objects to have the same name. LINCKS has a predefined set of presentation description objects, but users can add their own set of presentation description objects. User defined presentation description objects always override system defined presentation description objects with the same name. In other cases the override order is user and context dependent.

In LINCKS it is possible to have partially instantiated reference structures. This happens when the template forces the system to follow a link which is not found (a 'missing component'). In that case the system puts a *place holder*[8] in the reference structure and the displayed object (see figure 6). The place holder can be seen as an indication that the user can create something new or connect something existing there if desired.

Objects can be viewed and displayed in several ways. Using the same contents objects, but different presentation description objects, we can build different reference structures and therefore present the information in different ways (see figure 7 and figure 8). If we change a common content object through a view, we propagate the changes through the reference structures to the other views based on the changed content object.

8 Composite Objects

Composite objects in LINCKS are built using presentation description objects, which provide a template for a composite class (i.e. 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 allows us to share contents objects between different compositions and allows different views of compositions sharing the same root.

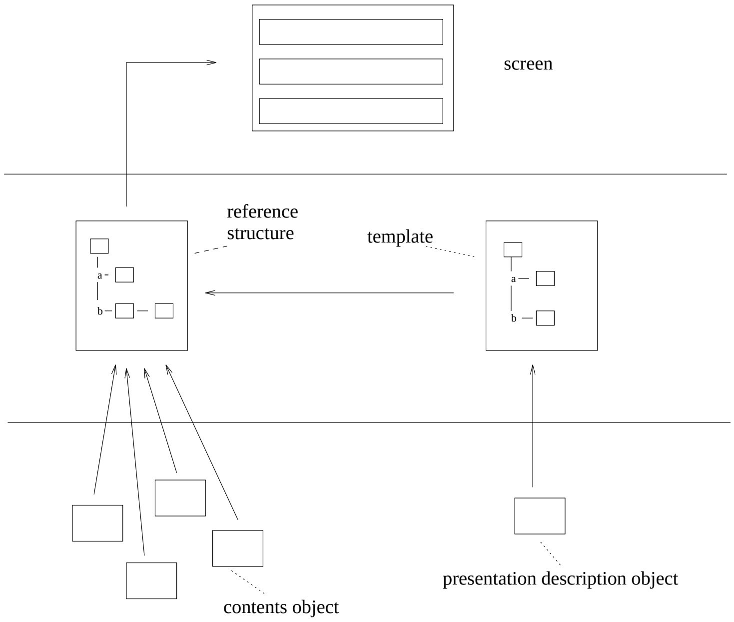

Figure 9: Content Object, Template, and Reference Structure

8.1 Version Management of Composite Objects

In [16] we have shown that it is not sufficient to know the historical information of the components of a composition to deduce the historical information of the composition itself. Therefore we have introduced a representation for a composite object which includes the presentation description object, the root of the composition and a special kind of object called a *binding table*. The binding table versions link together a root version and the component versions of a composition which existed at the same time. In that way the historical information of the composite object can be found in its binding table.[9]

We have implemented a lazy propagation policy for the compositions. This means that we do not always update the binding tables *immediately* whenever a component in a composition changes. We only update the binding table without delay if something in the composition changes while accessed *through* that specific composition. Also before accessing a composition we make sure that the history of the composition is updated if changes have been made to a component since the last access to the composition.[10] If several possibilities exist to do this updating the user is notified.

This policy allows us to maintain the benefits of dynamic link resolution while maintaining all relevant historical information. It also solves the question of the scope of the propagation of a change in a component to a composition in a multi-layered composition [13]. By delaying the propagation for compositions through which the component is not accessed, we defer the propagation to higher levels until we access the higher level.

8.2 Transactions on Composite Objects

- **Access:** When we access a composite object, the system checks whether components have changed (e.g. through another composition sharing the components) and does a delayed update of the composite object and its binding table if necessary. If several possibilities exist to update the composition, which are equally preferable (e.g in the case of new parallel versions of a component), the user is notified and can select which possibility is to be his current version of the composite object.

- **Store:** A store of a composite object gives rise to storing its binding table and all objects involved in the composite object. This means that we commit to a new version of the composite object upon this explicit store. Several new versions of component objects may have been created before the composite store. (This policy allows us to solve the path ambiguity problem [13].)

- **Update:** Upon update of a working version of a component [11] the change is propagated to the composite object through which the component was accessed. If the update occurred to a transient version of the component, nothing changes for the composite object.

When a component is added or deleted we are actually changing the root of the composite object or a component on a higher level (see former case).

9 Discussion

According to the time and space taxonomy described in [6] LINCKS supports collaboration by users at:

- different times and different places,
- different times and the same place, and
- the same time and different places

9.1 Sharing and Views

The object-centered database (layer 1) makes it very easy to reuse and share information on the atomic level. Our view mechanism also allows sharing and reuse of complex objects. The hypertext browser which allows to build dynamic views of information[12] supports different editing and copying operations and modes. We have an immediate updating facility i.e. when an object is used in different views in a session then updates to one view are propagated to the other views as well. When a user commits to a version (i.e. does an explicit store) the version becomes available to other users. This can be seen as draft passing.

9.2 Disconnected Workspaces

Our model also allows for *disconnected workspaces*. A user can then take out the objects he wants to work on and disconnect his workspace from the database. Within the workspace all the usual operations can be performed on these objects. Upon reconnection the database is updated. That includes storing of new objects, new versions and update the temporal information of the objects.

9.3 Coordination and Notification

The parallel editing notification can be seen as a warning signal telling the receiver that he coordination with other users regarding continued editing might be desirable. However it does not force the users to synchronize their work if they do not want to do so. Although users have received notification they may decide to update the same object at the same time and parallel versions will be created. (LINCKS does not have a locking mechanism.) Merging of parallel versions where needed is left upon the user who currently will need to use another channel of communication for agreeing to a possible solution. One way to communicate is having shared objects as a blackboard.

9.4 Access Control

The current implementation allows a user to modify all objects that can be reached from the private database. A basic mechanism for protection based on groups is implemented. A future version will include a mechanism which is also based on roles, and has protection on the object level.

10 Related Work

10.1 Database Systems

For the sake of brevity we have selected only three well known object-oriented database systems to compare: IRIS [4], ORION [15] and Version Server [5]. They are all object-based database management systems and we compare only with respect to the features described in this paper. (One should keep in mind that the other systems may well have other components such as access control or schema evolution not discussed here.)

In none of the three systems which we compare do we find the temporal history. Usually these systems do have a time stamping mechanism instead. We opted for the temporal history as a partial order because time stamping forces a linear order in creation time which may be misleading.

Dynamic binding in IRIS is resolved by a *context* mechanism based on user defined rules in the form of *daemons* which may be triggered. ORION resolves dynamic links on the basis of a time stamp ordering on the creation of versions while the Version Server resolves dynamic binding by specifying *environments*.

ORION supports a flag-based parallel editing notification mechanism [3]. ORION-2 [14] also supports a message-based notification approach where the system sends messages to notify users of potentially affected objects. The users can be notified immediately or at some later time specified by the user.

With respect to composite objects [4] observe that it is sometimes desirable to propagate a change of a component to a composition. This can be done by specifying that a particular component is *significant* for implicit version creation. A problem which then still remains is the proliferation of new versions. In the ORION system the compositions are divided into four classes [15] depending on whether components can be shared and whether a component can exist without the composition. As support for the historical information of composite objects they introduce rules. However these rules still leave them with cases where several developments in time are possible. Consequently they are unable to go back in time to show what a composition really looked like. In the early version of the Version Server propagation from component to composition was carried out automatically to the root. There was no way in limiting the effect. In [8] however the authors describe a graphical representation system for the compositions which can be used to hold a dialog with the user. An interesting notion to resolve the problem of ambiguity is *group-check-in*. This mechanism ensures that only one version of the composition is created. The problem of consistency is tackled by introducing a *layer* system. If two versions of different objects must be used together, then they are placed in the same layer. A *context* defines then a search order over the layers. A drawback of this method is that it is still possible to combine the layers in ways that do not make sense. There are no explicit constraints that state that particular versions of particular objects cannot exist at the same time

10.2 Co-authoring Systems

Another kind of system which has properties in common with LINCKS are the co-authoring systems.

The focus in PREP [19] is on enhancing the effectiveness of loosely-coupled collaboration. As in LINCKS the information in a document is divided into chunks. The chunks can contain text, grids, trees or arbitrary images. They are shared by the users in a database. An important issue for PREP is the possibility of commenting. A user can define 'drafts'. A draft defines an area in the workspace that an author intends others to access and consists of a sparsely filled grid of chunks. Each column in a grid is used to store different workspace content such as document content, plan and comments. Collaboration occurs then by commenting on drafts. The authors find it also important to allow revisions of drafts to exist as distinct versions such that old information is not lost.

The SEPIA system [9] is a hypertext authoring environment using a hypertext database (HyperBase) constructed on top of an commercial relational database. SEPIA supports synchronous and asynchronous collaborative editing of hypertext documents. It supports the creation of hyperdocuments by providing 'activity spaces' which can be seen as task-specific browsers. Information can be shared or private. SEPIA provides annotations nodes and thus supports collaboration via draft passing. Each author may see the actions of other users but is free to navigate through the document. SEPIA forces one to select objects before being able to execute an operation on them. That way a user can lock an object. A coloring mechanism is used to give a user information about locked objects.

GROVE [6] is a real-time outline editor allowing several people to edit at the same time. Users can enter and leave a GROVE 'session' at each moment. Each user in the session knows via 'group windows' which other user is participating. The users in the session can see and manipulate one or more views of the text being worked on. Views can be private, public or shared. The differences are in the read access for the different users. GROVE also uses a kind of shading mechanism but here the shading means that the user has no write access for the specific object. Each change to an object is directly propagated to the other users in the session. The default mode in GROVE is a mode where everybody can read and edit everything, i.e. there is no locking. The authors claim that (after a learning period) this does not give a chaotic situation, but appears to be quite useful. Our experience with LINCKS goes in the same direction.

11 Future Work

We have presented LINCKS as a system which fulfills several of the stated requirements for CSCW. Ongoing work is done on the following topics.

We can look upon presentation description objects as classes for a particular kind of objects (e.g. the 'document' class). Versioning of presentation description objects is then actually doing *schema evolution*. Although no schema evolution is implemented as yet, it seems that applying the versioning mechanism for composite objects to presentation description objects gives us attractive schema evolution features. It gives us the ability of updating objects and classes as well as using the old versions of objects and classes if desired.

We feel that it is important to provide the users with help for merging parallel versions of objects. Following our experience the versioning mechanism takes well care of merging different document versions on section level. (When section 1 is updated by one user and section 2 by another user, for instance, the latest version of the document will include both updates.) However when parallel updates on the same paragraph occur we would usually like to merge the two versions of the paragraph into one. The method discussed in [12] on how to propagate a change in an older version of an object to newer versions can be used.

In [26] an extension to LINCKS is described, which builds on the notification mechanism to add active database behavior [1]. We can tag objects to be 'interesting' and are notified of any change in those objects.

12 Acknowledgements

A considerable number of people have been involved in the design and implementation of the current system. The most significant of these are Ralph Rönnquist, David Partain, 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.

13 Availability

LINCKS Version 2 as well as an extensive user manual is available by anonymous ftp from `ftp.ida.liu.se (130.236.30.131)` in the directory `pub/lincks/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`.

Notes

[1] This paper was jointly written using LINCKS and then exported in LaTeX format.

[2] LINCKS is object-centered. It does not include the message paradigm for reasons outlined in [24].

[3] An object in another user's area may be reachable by a link, such that we also have semi-private areas for groups of users.

[4] This can be seen as committing to this version.

[5] These node identifiers are unique with respect to the database. However for each session only the nodes which are reachable can be brought into the workspace. For reasons of information hiding the nodes receive a local identification number, very likely different in each session, which is mapped to the database node identification number using an interface table. This will also allow us in the future to implement easily *disconnected* workspaces on portable computers.

[6] 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.

[7] Presentation description objects are objects in the database. Therefore they are versionable and can be composite. For instance, it is natural to have as part of a presentation description object for documents a part which is a presentation description object for sections. Different versions of a presentation description object refer to changes in how a particular kind of object display is viewed. We can also view presentation description objects using other presentation description objects.

[8] displayed as << something >>.

[9] The binding tables are also used for the resolution of links to particular versions.

[10] This way of working, i.e. delaying the updating until access time is called *screening* in [20] in contrast to *conversion* where all updates occur immediately. The advantages of screening are performance and a limitation of the propagation of the changes. A disadvantage is the possibility of inconsistency between two accesses.

[11] Currently, all components are version significant in the terminology of [4]. Allowing non-version significant components is a trivial extension to the system.

[12] Application building using LINCKS is described in [23]

References

[1] Special issue on active databases. *IEEE Bulletin of the Technical Committee on Data Engineering*, 15(1–4):1–56, December 1992.

[2] Peter Åberg. Design of multiple view presentation and interaction manager. Licentiat thesis no. 177, University of Linköping, Sweden, 1989.

[3] D. S. Batory and W. Kim. Modeling concepts for VLSI CAD objects. *ACM Transactions on Database Systems*, 10(3):322–346, sept 1985.

[4] 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.

[5] E. E. Chang, D. Gedye, and R. H. Katz. The design and implementation of a version server for computer-aided design data. *Software - Practice and Experience*, 19(3):199–222, march 1989.

[6] Clarence Ellis, Simon Gibbs, and Gail Rein. Groupware: Some issues and experiences. *Commmunications of the ACM*, 34(1):38–58, January 1991.

[7] Williem J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, pages 57–70, Fall 1992.

[8] D. M. Gedye and R. H. Katz. Browsing the chip design database. In *Proceedings of the 25th ACM/IEEE Design Automation Conference*, pages 269–274, 1988.

[9] J.M. Haake and B. Wilson. Supporting collaborative writing of hyperdocuments in sepia. In Jon Turner and Robert Kraut, editors, *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 138–146, Toronto, Ontario, Canada, 31 October–4 November 1992. ACM Press.

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

[11] 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.

[12] Michael Jansson. Propagation of change in an intelligent information system. Licentiat thesis no. 381, University of Linköping, Sweden, 1993.

[13] R. H. Katz. Toward a unified framework for version modeling in engineering databases. *ACM Computing Surveys*, 22(4):375–408, dec 1990.

[14] W. Kim, N. Ballou, J. F. Garza, and D. Woelk. A distributed object-oriented database system supporting shared and private databases. *ACM Transactions on Information Systems*, 9(1):31–51, jan 1991.

[15] W. Kim, E. Bertino, and J. F. Garza. Composite objects revisited. In *Proceedings of the Conference on Management of Data - SIGMOD 89*, volume 18 of *SIGMOD Record*, pages 337–347, 1989.

[16] Patrick Lambrix. Aspects of version management of composite objects. Licentiat thesis no. 328, University of Linköping, Sweden, 1992.

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

[18] 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.

[19] Christine M. Neuwirth, David S. Kaufer, Ravinder Chandhok, and James H. Morris. Issues in the design of computer support for co-authoring and commenting. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 183–195, Los Angelese, CA, USA, 7–10 October 1990. ACM SIGCHI and SIGOIS.

[20] G. T. Nguyen and D. Rieu. Schema evolution in object-oriented database systems. *Data and Knowledge Engineering*, 4:43–67, 1989.

[21] Lin Padgham. Linköping intelligent knowledge communication system. In *IFIP Working Conference on Methods and Tools for Office Systems*, pages 1–15, Pisa, Italy, 1986. IFIP.

[22] 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.

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

[24] 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.

[25] Erik Sandewall. Theory of information management systems. Lectures Notes, 1983. Department of Computer and Information Science, University of Link ping, S-581 83 Link ping, Sweden.

[26] Martin Sjölin. Activation of LINCKS – adding an active aspect. In *Proceedings of Third Annual IDA Conference on Computer and Information Science*, pages 81–87, University of Linköping, Sweden, 1992. Department of Computer and Information Science.

[27] R. Snodgrass and I. Ahn. Temporal databases. *IEEE Computer*, 19(9):35–42, sept 1986.

[28] Tim Teitenberg. *Preliminary Documentation of the Application Interface Manager (AIM) for the LINCKS System.* Deparment of Computer and Information Science, University of Linköping, 1990. System Documentation 34.1.