“Activation” of LINCKS – Adding an Active Aspect

Martin Sjölin

Department of Computer and
Information Science
Linköping University
S–581 83 LINKÖPING, Sweden
E-mail: marsj@ida.liu.se

May 29, 1994

Abstract

There are a number of applications where the need for coordination and synchronisation of the different applications is covered by neither the traditional relational databases nor the object-oriented databases.

We argue for a simple extension of an object-oriented database manager by adding a notification (or triggering) capability to the database manager. The idea is to have the database manager monitoring the changes in the database, and whenever specific changes occurs a corresponding action will be invoked. Thus, the database manager is transformed from a passive entity only storing structured information to an active entity taking part in the exchange between the applications.

We outline a proposed extension along these lines for the object oriented database manager within the LINCKS environment.

1 Introduction

We present a simple extension to LINCKS to raise the level of support for applications like collaborative editing, cooperating information services (agents), etc. We then introduce the concept of an active database. Whenever a certain pattern of data is introduced into a database, the database manager executes an appropriate action, possibly notifying an application about the change. We contrast this with having the application repeatly querying the database for a change (polling).

We begin with a brief description some typical intelligent information systems where object-oriented database are interesting to use. It is in followed section 3 by a brief description of LINCKS, our object-oriented database system. Then, in section 4, a few typical applications such as mail-handling agents and collaborative editing are presented where an active database manager plays a more important role. We introduce the concept of an active database in section 5. The activation of LINCKS is described in section 6. We also sketch how the earlier application(s) can be simplified using an active database as compared to the traditional approach. Finally we touch upon some of the problems and open questions for future research.

2 Intelligent Information Systems

Compared to other systems for supporting work in an office environment, LINCKS has no specific domain in mind (such as mail filtering [Mal87], collaborative editing [Ellis91], etc), but has been developed as a general purpose intelligent information system. The area of research of Office Information Systems [Bra84] (OIS), in which the system originated, can be described as a subset of the more general *Intelligent Information Systems* (IIS). As example application areas for IIS we have:

- Medical Systems.
- Decision Support Systems.
- Information Retrieval.
- Development Environments Assistants.
- Intelligent Communication Systems supporting filtering and Classification of Information (mail, news, reports etc).
- Computer Supported for Cooperative Work (for example coordinating activities and group scheduling)

The common items in the categories above are the diversity and the complexity of data, the need for an adaptive system, and the communication of information and of knowledge between humans and/or services. One goal is to automate the repetitive and tedious routine work and improve the creative environment. Also, the systems should be capable of learning new behaviours and adapting themself to the usage pattern of the individual user's.

3 LINCKS

LINCKS – **L**inköpings **IN**telligent **C**ommunication of **K**nowledge **S**ystem [PaR87, Pad86] – is an experimental object-oriented database system developed at the Intelligent Information Systems Lab of Linköping University.

LINCKS contains a structured but passive database server (NODE), where the *atomic object* is a complex object represented by two kinds of nodes ([Åbe89]) : *versions* and a *history structure* node. A node has three distinct parts ([Pad86, Åbe89]):

- the *image*, which is uninterpreted information (usually textual or graphic)
- a number of *attributes* with values, which describe the properties of the object which are not related to other objects,
- and a number of *links* which describe the relationships of the object to other objects.

The history structure of an atomic object represents historical information about that object. User modification or creation of an object results in a new version being created from the original object. The history structure node is updated to reflect the new version and the old original version is kept in the database. Two kinds of historical information are supported – temporal history and edit history. The edit history describes the partial ordering between the different versions of the object ([Lam92],[Pad86],[Pad88b],[Pad89a]). The temporal history [Lam92] is a partial order of the different versions of an object based on their creation time.

4 Scenario

We take a normal office as our base scenario, with the diverse categories of documents, different users and a rapidly evolving environment in mind. Then we can use the following application as examples:

- Creation and modification of documents (alone or together, e.g collaborative editing)
- Coordination or Scheduling of meetings (external or internal)
- Mail or Message handling (electronic mail and electronic new services)
- Information Retrieval (libraries, databases and other sources)

This a typical environment for LINCKS with the different kinds of information evolving over time. LINCKS is built to support both human agents and computerized agents, as well as the communication between the agents involved.

However, when we would like to build an application coordinating different activities, LINCKS' functionality alone is not sufficient. We have to extend the applications to handle the coordination and synchronisation of different functions. Another choice is to extend LINCKS' database manager, NODE, which is natural as we are interested in keeping the database as the main medium for communication between applications.

5 An Active Database

A "normal" (relational) database is by architecture passive, as we have the applications storing, retrieving or searching for information. The database is only used for storing the information in a structured form, to facilitate access and to search for information. Also, every exchange between the application and the database is in the form of a request to database manager from the application. This request is followed by an answer or a confirmation from the database manager. Thus, every interaction between the database and the application is a synchronous. An "active" database emits spontaneous information (indirectly requested by the applications) to the applications, it takes an *active* part within the system. To indicate this we name it an *active* database.

Let us consider the mail handling application mentioned in section 4 and suggest an implementation based on the active database concept. First, we have one application, the *move-mail* agent, handling incoming mail from the external world. This agent moves the incoming mail from the external file system into the database, and sleeps until there is new incoming mail available. A second agent, *news*, moves groups of articles from a conference system, "news", into the database. The third agent, *prioritize*, orders the incoming mails and articles into different folders according to their importance and their contents. Now, having an active database, the prioritize agent will be activated when the input-folder is updated by movement agents. In a passive database system this would have to be done by querying the database manager for changes.

The traditional passive database is not well suited to applications monitoring conditions or states. To catch the changes of conditions and states, we have the following possibly solutions:

- Polling the database manager for changes, but here we have to select the polling interval carefully. Also, this behaviour takes unnecessary resources from the system.
- When storing or changing information in the database, check if any known limiting case is passed. First, this can violate the modularity of the system, as the knowledge about limits will be distributed in system. Second, we need to communicate the information about the change to the appropriate part of the system. Here we need a second communication medium, as we cannot use the database itself.

A general definition of an **active database** is a database where a pattern of data in database invokes an action (rule). The actual execution of the rule can take part either in the database manager itself or in the application.

An example from the field of relational databases is to use the active database manager to control the integrity of the relations before an insert or update of a row in a table [Stone91]. These integrity rules can be defined by the designer of the database to ensure that only correct information is stored in the database. Further, the pattern–action pair can also be used to handle view[1] updates ([Stone90]). The execution of an action part is often denoted **triggering** an action, where the condition part is the "**trigger**".

We can restrict the conditions to apply only when the underlying data is modified [LohL91, SkaZ86]. Take as example a collaborative editing session running on several workstations. All users are making changes and modifications to a common document. They are able

[1] A view is a "virtual" table, defined by selecting information from several tables, often by an SQL statement. The user sees a "table", but in reality it is a virtual view constructed by a query.

to follow each others modifications on their own screens, and the progress of the common work. When a user has finished his modification to a paragraph, the changes would then be displayed on all other users' screens. Using an active database, we simply add an action rule notifying the applications whenever a paragraph is changed in the database. The application retrieves modified objects and redisplays them when notified about the changes in the database.

6 Active LINCKS

6.1 The Basic Mechanism

The current implementation of LINCKS (and its underlying database manager - NODE [Man88]) only supports a synchronous exchange, namely an RPC-like[2] facility for retrieving and storing objects. We propose transforming the currently passive database server in LINCKS, NODE, into an *active* database server[3] following the outline in section 5. To support the change notification mechanism in a relational or object-oriented database, we would like to know when:

- an item is *changed* in the database.
- an item is *added* to the database.
- an item is *removed* from the database.

We assume that we are talking about change, removal or creation of an atomic entity, e.g. nodes, in LINCKS. The conditions above map to the following for LINCKS:

- Change – *creation* of a new version of an object.
- Addition – *creation* of a new version of an object or a new object.
- Removal – Objects cannot really removed in the usual sense, as the database keeps an edit history of all versions of an objects. The only "real" removal occurs when a object is a part of a composite object. A part is removed by unlinking the link to the parent object. This implies a change in parent's link section, and thus the *creation* of a new version (of the parent object).

Summarized, all actions within the database that we would like to monitor in LINCKS are collapsed into the monitoring of the *creation* of a new version of an object (cf. to Gemstone [ButtO91] or Starburst [LohL91]). On a higher level, we would like to request notification upon modification of attributes or links. This can easily be implemented on a higher level using the basic mechanism.

6.2 Implementation Details

We propose adding a notification capability to LINCKS' database manager, NODE. The only and basic condition would be the *creation* of a new version of an object. The applications specify in a request to the database manager those objects about which they

[2]Remote Procedure Call

[3]One of the 1992 PUM projects, PEM, is extending NODE to support notification of parallel editing operation within LINCKS.

would like receive a creation notification. Let us call this a **registration** in the database manager of an object for the purpose of creation notification.

Here can we distinguish between two types of situations when requesting object notifications. First, the applications can register objects for limited period of time, e.g. the duration of their own execution. This would be useful for the collaborative editing example, as we are only interested in changes during the execution of the editing application.

In the second case, we would like register the objects for an extended period of time, e.g. the object's own lifetime. We have a mapping between different objects and a group of functions for a possibly unlimited period of time. When we receive a creation notification for an object from the database manager, we would like to execute the corresponding function. The first solution, though somewhat unpractical, would be to have several applications executing where each application corresponds to a function. Instead, we propose creating a daemon application, like the UNIX internet daemon "inetd" (for an introduction to network programming see [Stevens90]). This daemon looks like a normal application and makes the proper object registrations on behalf of the different functions. However, upon receiving the creation notification the daemon first maps the notification to the corresponding function. Then, this function is invoked by the daemon and handles the creation notification before terminating itself. If there is no simple mapping from registered objects to functions, we simply add an attribute "tag" to objects in the database. When we have received the notification, we retrieve the object and use the tag to invoke the corresponding function.

This is the case where the daemon is executing all the time (this is the case for "inetd"), but we have reduced the number of concurrently running applications to one, the daemon application.

7 Summary and Future Work

We have proposed extending LINCKS with a mechanism for notification of applications whenever a new version of an object is created.

Whether it is for parallel editing or cooperating agents handling mail, we can use the database for communication and synchronisation between the the different applications. Thus, we avoid creation of a second communication channel and a polling behaviour to check for changes in the database.

7.1 UNIX Signals and Creation Notification

The creation notification mechanism has certain similarities to the UNIX "signal" facility (see [Bach86]). Both are asynchronous, they carry a restricted amount of information, and they pose the following problems:

- Queueing of events - unlimited or limited queue size.
- Filtering (ignoring or blocking) of notifications.
- Pending notifications.
- Default actions.

In BSD UNIX [LefMK89], there are system calls to handle blocking, unblocking and pending signals, as well as a possibility of selecting a queueing or a non-queueing behaviour. We would like to avoid the plethora of different functions which exist in BSD UNIX for handling the signal's semantics. How to solve these problems in the scope of LINCKS as an active database is an open question.

7.2 Function Invocating

Another area of interest is *where* the function should be invoked and by *whom*:

- Should only the application registering the object receive notifications?
- Should any of the instances of the application registering the object receive notifications?
- Should the database manager invoke a specific application upon observing the creation? This implies that the database manager must be able to map an object to a specific application.
- Should the database manager simply ignore the events when no suitable application is executing?
- Should we use the proposed solution in section 6.2 with a daemon application demultiplexing the creation notifications to the proper functions?

Initially, we prefer the last case, as we manage with only one type of registration of objects in the database manager. Also, since the functions will execute in the application's process, and not database manager's process, we minimize the risk of inadvertently bringing down the database manager.

References

[Bach86] Bach, M., 'The Design of the UNIX operating system', Prentice Hall, 1986.

[Bra84] Bracchi, G., Pernici, B., 'The Design Requirements of Office Systems', *ACM Transaction on Office Information Systems*, pp 151-170, April 1984, Vol 2, No 2.

[ButtO91] Butterworth, P., Otis, A., Stein, J., 'The GemStone Object Database Management System', *Comm. ACM*, pp 64-77, Oct 1991, Vol 34, No 10.

[Ellis91] Ellis, C., Gibbs, S., Rein, G., 'Groupware: Some Issues and Experiences', *Comm. ACM*, pp 38-58, Jan. 1991, Vol 34, No 1.

[Hal92] Hall, T., *Command History in LINCKS*, Master thesis, Department of Computer Science, Linköping University, 1992.

[Lam91] Lambrix, P., 'Temporal Aspects of Composite Objects', *Research Report*, LiTH-IDA-R-91-14, Linköping University, 1991.

[Lam92] Lambrix, P., *History Management of Composite Objects*, forthcoming Lic. thesis, Dept. of Computer Science, Linköping University, 1992.

[LefMK89] Leffler, S.J., McKusick, M.K., Karels, M.J., Quarterman, J.S., *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, 1989.

[LohL91] Lohman, G., Lindsay, B., Pirahesh, H., Shiefer, B., 'Extensions to Starburst: Objects, Types Functions and Rules', *Comm. ACM*, pp 94-109, Oct 1991, Vol 34, No 10.

[MalG87] Malone, T., Grant, K., Turbak, F. Brobst, S., Cohen, M., 'Intelligent information-sharing systems.', *Comm. ACM*, pp 390-402, May 1987, Vol 30, No 5.

[Man88] *Applications Interface to LINCKS on Sun; Programmers Manual*, LINCKS Project Group, Department of Computer Science, Linköping University, 1988.

[Pad86] Padgham, L., 'Linköpings Intelligent Knowledge Communication System', in *IFIP Working Conference on Methods and Tools for Office Systems*, pp 1-15, Pisa, Italy, 1986.

[Pad88b] Padgham, L., 'NODE : A Database for Use by Intelligent Systems', in *Proceedings of ISMIS-88*, pp 190-199, Torino, Italy, 1988.

[Pad89a] Padgham, L., *Non-monotonic Inheritance for an Object-Oriented Knowledge-Base*, Ph.D. thesis, No. 213, Department of Computer Science, Linköping University, 1989.

[PaR87] Padgham, L., Rönnquist, R., 'LINCKS : An Imperative Object Oriented System', in *Proceedings of the 20th Hawaii International Conference on System Sciences*, pp 516-522, 1987.

[SkaZ86] Skarra, A., Zdonik,S., Reiss, S., 'An Object Server for an Object-Oriented Database System', in *Workshop on Object-Oriented Dababases 1986*, Pacific Grove, CA, Sept. 1986.

[Stone90] Stonebraker, M., , 'On Rules, Procedures, Caching and Views in Data Base Systems', *SIGMOD Proceedings*, pp 281-290, 1990.

[Stone91] Stonebraker, M., Kemnitz, G., 'The Postgres Next-Generation Database Management System', *Comm. ACM*, pp 78-92, Oct 1991, Vol 34, No 10.

[Stevens90] Stevens, W. R., 'UNIX Network Programming', Prentice-Hall, 1990.

[Åbe89] Åberg, P., *Design of a Multiple View Presentation and Interaction Manager*, Lic. Thesis 177, Linköping University, 1989.