In: Proc. 7th Int. Conf. on Systems Research, Informatics and Cybernetics, Advances in Database and Expert Systems, G.E. Lasker (ed.), Int. Institute for Advanced Studies in Systems Research and Cybernetics, Baden-Baden, 1994, pp. 169-75.

# Supporting Cooperative Work by Conventional Database Transactions

**Norbert Ritter**

Department of Computer Science
University of Kaiserslautern
P.O.Box 3049, 67653 Kaiserslautern, Germany
e-mail: ritter@informatik.uni-kl.de

## Abstract

'Computer-Supported Cooperative Work' (CSCW) is a young, interdisciplinary research area considering applications with strong demands on multiple fields of computer technology, e.g., distributed systems and networks or multi-media systems. In this paper, we will be focusing on the information-sharing and the corresponding activity-control aspect. Here, the conventional transaction concept of database systems lacks adequate cooperation control capabilities. A database transaction is not the appropriate operational unit for the sequence of actions/operations performed by a cooperating user because the isolation property builds protective walls to concurrently active users. Given this background many new approaches to transaction modelling are making these protective walls more permeable. While most advanced transaction models continue to apply a transparent concurrency control, in groupware systems a user-centered approach to cooperation control is needed reflecting the highly dynamic work and the rich patterns of interpersonal cooperation. Therefore, we propose an explicit cooperation control. Although, database transactions do not have cooperative capabilities, we claim that they are still applicable, even in groupware applications. The work of a cooperating user contains several elementary actions. Due to preserving basic consistency, a transaction-protected execution of these elementary actions is needed. Thus, we propose to enhance the traditional transaction processing concept by an additional layer supporting explicit cooperation control. In this paper we will discuss the interplay of implicit concurrency control and explicit cooperation control using our **CONCORD** model (**CON**trolling **CO**ope**R**ation in **D**esign Environments) as a sample approach to activity modeling in groupware systems.

**Keywords:** Computer-Supported Cooperative Work, Transactions, Cooperation Control, Concurrency Control.

## 1. Introduction

By using the term *groupware systems* we usually refer to a huge class of computer systems which cannot be rigidly distinguished from other classes of computer systems. In [EGR91] groupware is defined as:

> *"Computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment."*

A lot of taxonomies of groupware systems can be found in literature, e.g., in [Gr94], often distinguishing between message systems, multi-user editors, group decision-support systems and electronic meeting rooms, computer conferencing, intelligent agents, and coordination systems, e.g., supporting design applications in software development or mechanical/electronic engineering.

Being engaged in a common task (or goal) mostly means working on shared resources [GS87]. Simultaneous operations on shared data cause conflicts which need to be resolved by appropriate concurrency control techniques. Here, major parts of the CSCW community see a philosophical difference between database systems and groupware systems. The reason is the isolation property which is one of the important characteristics of conventional transactions. These characteristics are often called the ACID-properties (atomicity, consistency, isolation, durability) [HR83, GR93]. Beside isolation, the consistency property is a crucial issue. It means that a transaction is an operational unit of consistency. Often it is necessary to perform several database operations in order to move the database from one consistent state to another consistent state. In our opinion this notion of consistency is of a very basic nature and has to be the foundation for cooperation, too. Therefore, it should also be applied in cooperative work arrangements using shared information repositories. For example, in cooperative design-applications there is mostly no need for a design tool to cooperate directly with other tools. Nevertheless, it is definitely important for two designers, each working on a certain part of the overall design task, to cooperatively exchange information about the (preliminary) design-object version (e.g., a software module) created by the design-tool application [Ri94]. Another example can be found in group editing [EGR90]. It is often necessary to isolate the writing/modification of a word/sentence in order to make a semantically consistent modification of the contents. In this example the need for flexible transaction boundaries becomes obvious; but still atomic and isolated units of changes are to be supported due to consistency requirements.

Regarding the different types of activities naturally predominating in groupware applications, we can summarize that neither a single conventional ACID transaction is the appropriate unit of a cooperating user's work, nor the result of each database operation initiated by this user is a reasonable portion of cooperative data exchange. Therefore, we have to develop an adequate infrastructure allowing for the mapping of all kinds of activities occurring in groupware applications to operational units manipulating shared data. In the following section, the major current research directions in cooperative transactions and thereby related work are outlined. In Sect.3 we will discuss our approach which provides an infrastructure for groupware activities on top of the conventional transaction model. The last section gives a conclusion.

## 2. Current Research Directions on Cooperative Transactions

*Serializability* is the notion of correctness for conventional transactions [HR83]. It allows concurrently running transactions to interleave their data-manipulation operations however obeying the restriction that the effect of the schedule resulting from the interleaving has to be the same as the effect of an arbitrary serial schedule. This correctness criterion is well-founded and well understood [BSW79]. Lots of concurrency-control techniques which were proven to enforce serializability have been proposed during the last decade. A very important one is the two-phase locking protocol [GR93]. Serializability asserts semantically consistent database-state transitions. We claim that in almost any application area there are atomic units of work consisting of a sequence of database operations which is to be performed completely in order to make the manipulated data usable at all. On the other hand, it is easy to see that a single conventional transaction cannot carry the entire sequence of actions performed by a cooperating user because serializability is too restrictive. This led the database community to develop cooperative transaction models. [El92] gives a good overview of advanced transaction models including models with cooperative capabilities. Due to space restrictions we will only sketch two major directions of this research which are illustrated in Fig.1.

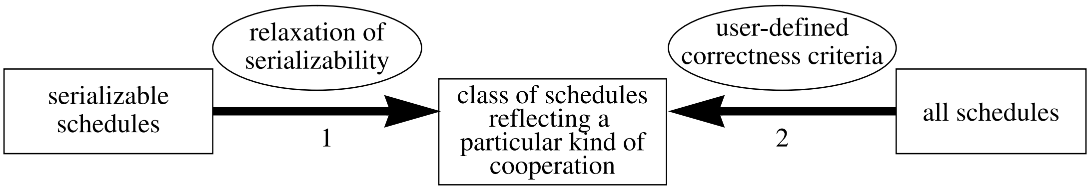


Fig.1: Approaches to cooperative transactions using transparent concurrency control techniques

The first direction starts from the class of serializable schedules and tries to enlarge this class by those schedules reflecting the desired ways of cooperation. An important representative of this class is the "*relative serializability*" approach presented in [ABEK94]. Here, transactions are called *co-action*s due to their cooperative features. The user is allowed to define *collaborative channels* between co-actions in a pairwise manner by splitting up two co-actions into *atomicity units* relative to each other. Now, the notion of *conflicting operations*[1] is only applied to interleaving atomicity units, not to entire transactions, enlarging the class of allowed schedules. In this way, the user is authorized to specify which parts of transactions are to be isolated against each other as well as when operations which would conflict in the serializability case are to be allowed, due to cooperation. If no atomicity units are defined between two transactions, they are isolated against each other like ACID transactions.

The advantage of the relative-serializability approach is to give the user the possibility to specify the allowed ways of cooperation. Furthermore, it has a formal foundation, and the proposed locking protocol, a simple extension of the well

1. In the traditional concurrency control theory [BSW79] two operations of different concurrent transactions conflict if they access the same data item and at least one of them is a write operation. Therefore, concurrency control-techniques enforcing serializability usually prevent conflicts by blocking: If transaction $t_1$ wants to perform an operation which would conflict to an already executed operation of transaction $t_2$, $t_1$ has to wait until $t_2$ is finished.

known two-phase locking protocol, is proven to be correct. The drawback is the necessity of semantic a-priori-knowledge about the transactions allowing the user to specify atomicity units. The specification overhead burdens the user, and low flexibility is achieved due to a-priori specifications.

We would like to discuss the second direction illustrated in Fig.1 by outlining the "*cooperative transaction hierarchies*" approach [NZ92]. Here, the underlying concept doesn't consider the serializability criterion. The notion of *cooperating transaction*s is introduced. Cooperating transactions can be combined to *transaction groups* which, in turn, can be nested.A local database is associated to every node of this hierarchy which is manipulated by the node's direct sub-nodes. These manipulations are synchronized by means of semantic patterns reflecting local, user-defined correctness criteria which can be implemented as deterministic, finite automata accepting sequences of operations preserving the data consistency.

This approach provides very powerful and flexible mechanisms for specifying user-defined correctness criteria which can be changed and supplemented dynamically. It also shows a mature formal foundation. But, again, a lot of specification work has to be done, and the patterns of interactions must be known in advance.

Summarizing the mentioned directions of research on cooperative transactions we can say that both lead to important approaches in specifying correctness criteria reflecting allowed ways of cooperation. To achieve this, on one hand serializability can be relaxed, on the other hand completely user-defined correctness criteria can be used. In both cases the practical relevance cannot be easily judged. Specifications cause a high overhead and burden the users. It is difficult to overlook whether the specifications are reflecting the wanted ways of cooperation or not. Therefore, we raise the question whether the principle of a transparent concurrency control which is still applied by the considered approaches is adequate or whether it should be replaced by an explicit, user-centered approach which is more intuitive to the user and which does not need a-priori specifications. The following section will outline our approach to the latter kind of cooperation control.

## 3. An Infrastructure for Cooperative Work

After having outlined some approaches on new transactional models we will discuss our ideas about providing an infrastructure for those groupware applications depending on shared information management. This infrastructure exploits the achievements of approved transaction concepts and establishes a comprehensive framework. We will first give an overview of our CONCORD model [Ri94] which was originally developed to support design applications, e.g., electronic design. Thereafter, we will discuss those concepts of CONCORD which can be applied in a broader application class than design.

### 3.1 Design-Principles of the CONCORD Model

The CONCORD model [Ri94] captures the dynamics inherent to design processes. To reflect the different requirements, such as *hierarchical refinement, goal orientation, stepwise improvement and team orientation*, three different levels of abstraction are distinguished as illustrated in Fig.2.

**Administration/Cooperation Level (AC level)**

At the highest level of abstraction, we consider the more creative and administrative part of design work. There, the focus is on the description and delegation of design tasks as well as on a controlled cooperation among the design tasks. The central concept at this level is the ***design activity*** (***DA***). A DA is the operational unit representing a particular design task or subtask. During the design process, a ***DA hierarchy*** can be dynamically constructed resembling (a hierarchy of) concurrently active tasks. All relationships between DAs are explicitly modeled, thus capturing task-splitting (cooperation-relationship *delegation*), exchange of design data (cooperation-relationship *usage*), and negotiation of design goals (cooperation-relationship *negotiation*).

**Design-Control Level (DC level)**

Looking inside a DA reveals the *DC level*. There, the organization of the particular actions performed in order to fulfill a certain (partial) design task is the subject of consideration (***work flow***). Fig.2 shows at this level an execution plan (***script***) of a particular design activity. This script models the ***control/data flow*** between several design-tool executions. The operational unit serving for the execution of a design tool is the ***design operation*** (***DOP***). In order to control the actions within the scope of one DA, but without restricting the designers' creativity, flexible mechanisms for specifying the workflow for a DA (scripts, constraints, event-condition-action rules) are provided. Due to space restrictions we cannot detail this layer and have to refer to the literature [DE93].

**Tool-Execution Level (TE level)**

From the viewpoint of the DBMS or data repository, a DOP is a long transaction. A DOP has the consistency, isolation and durability properties of conventional transactions but is not atomic. Because of long duration, it is internally structured by save/restore and suspend/resume facilities as illustrated in Fig.2. A DOP processes design-object versions in three steps. First, the input versions are checked out from the integrated data repository. Second, the loaded object data is processed by the design tool. Third, the finally derived new versions are propagated back to the data repository (checkin operation). Considering that DOPs process versions, the isolation property does not require blocking[2].

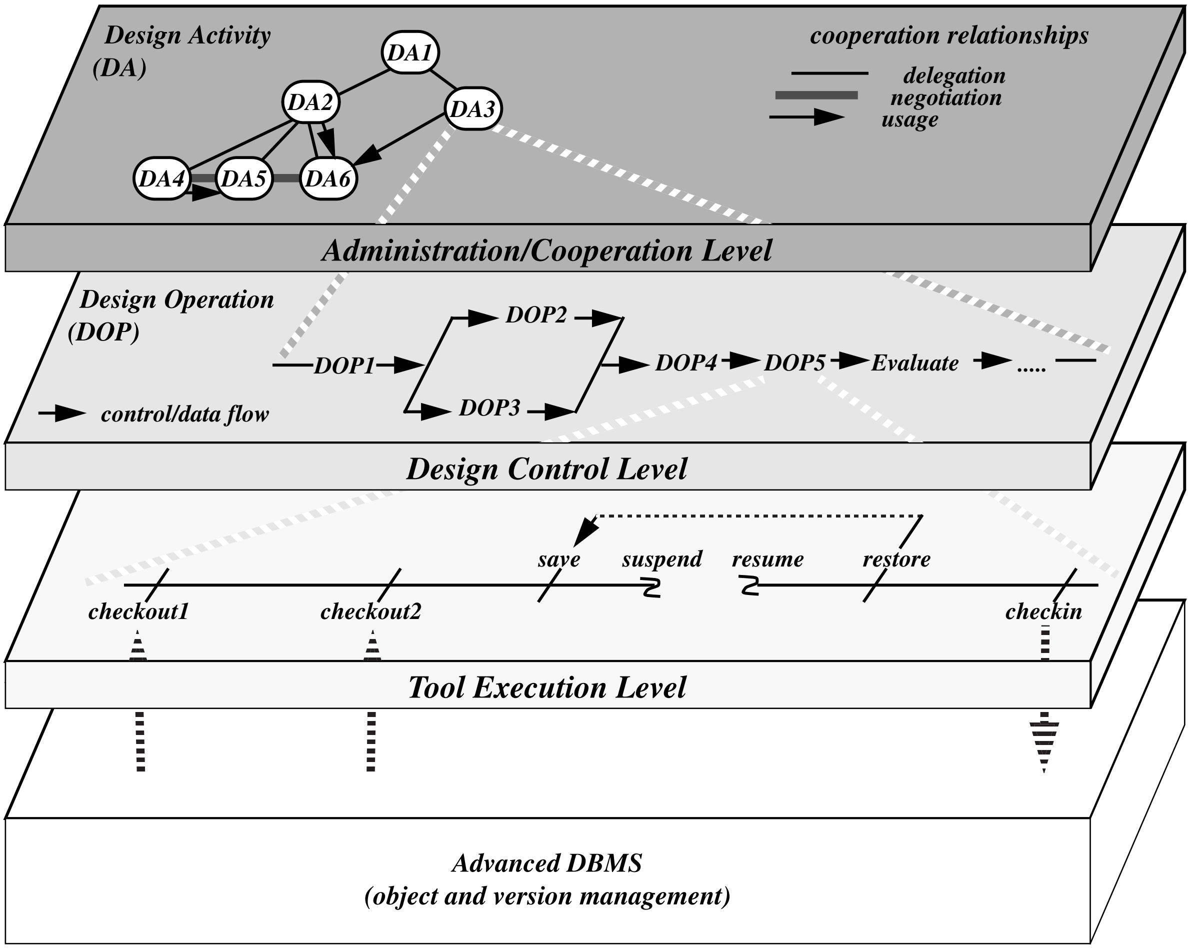


Fig.2: Abstraction levels of the CONCORD model

The three mentioned abstractional layers are provided on top of an object and version management system consisting of an object-oriented database management system and an additional version-management layer. The latter provides explicit versions of complex-structured objects. Thus, the checkout/checkin operations issued by long transactions at the TE level are mapped onto short, conventional database transactions by the version-management system.

Fig.3 summarizes the major concepts of the CONCORD processing model. The user-centered cooperation control facilities (AC level) are provided on top of workflow-management primitives (DC level), such that preliminary infor-

2. Several successors of a single version can be derived by concurrent DOPs. Updating a version, on the contrary, requires locking it for the entire DOP duration.

mation created by workflows can be subject of cooperation. The actions constituting a workflow (e.g., design tool applications), in turn, are themselves long transactions (TE level) manipulating versioned data. Version manipulations (version-management layer) are performed by means of checkout/checkin operations which are realized as short ACID transactions (database management system).

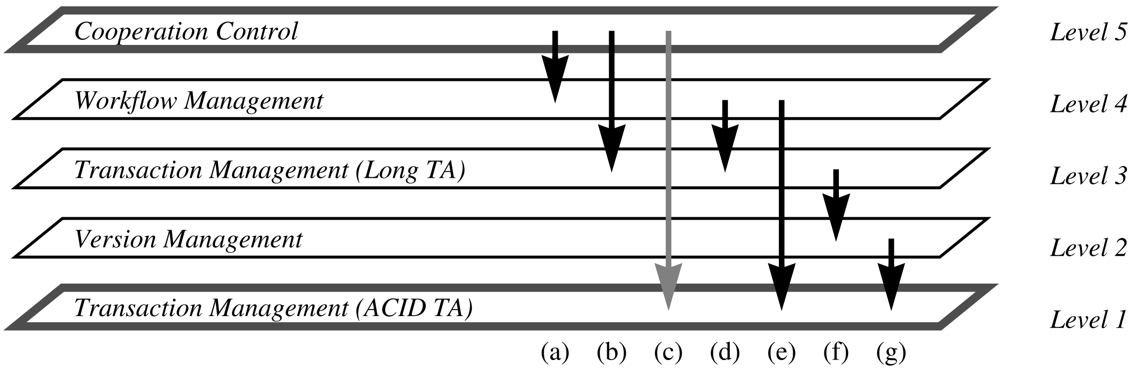


Fig.3: Generalization of the abstractional layers

For further details concerning the CONCORD model we have to refer to [Ri94]. In the context of this paper we want to focus on the interplay between the different types of operational units. Fig.3 illustrates the multiple possibilities of interplay. Each layer can be used as entry point for applications. For example, cooperative work arrangements exploit the cooperation-control facilities which can invoke workflow-management facilities (a) or can directly issue long transactions (b) or even short transactions (c). It is not possible to detail all forms of interplay in this paper. In order to show the applicability of conventional transactions even in cooperative work arrangements we will discuss the direct interplay between the implicit concurrency control of conventional transactions and the explicit cooperation-control facilities provided at the AC level, which, in our opinion, is a simple and intuitive alternative to the complex transactional models mentioned in Sect.2.

### 3.2 Interplay of explicit cooperation control and implicit concurrency control

As already mentioned, the CONCORD model has been developed to support design applications. We claim that major principles of this approach are also applicable in other subclasses of cooperative applications. Therefore, the discussions in this subsection are not limited to design applications.

#### Explicit cooperation control

Due to independence from the application subclass we will talk no longer about *design activities (DA)* but about ***cooperative activities (CA)***. A CA is created by the operation
*Create_CA (super_CA_ID, users): CA_ID.*
The *users* parameter fixes the users which will be allowed to initiate and perform actions associated to this CA. The optional parameter *super_CA_ID* establishes a super_CA-sub_CA relationship[3]. If it is specified, the initiator of the *Create_CA* operation needs to be a user of the indicated *super_CA*. After its creation, the CA's users can start working in the context of the CA. Since we are only considering case (c) of Fig.3, only ACID transactions on the plain database (not versioned data) can be issued. After its successful termination, the data which was manipulated by a transaction becomes visible to all users of that CA. To support cooperation between CAs the selective pre-release of information is supported by the following two operations.

*Permit (permitting_CA_ID, set_of_receiving_CA_IDs, set_of_objects, right).*
By issuing this operation, the CA indicated by parameter *permitting_CA_ID* permits cooperating CAs (specified by parameter *set_of_receiving_CA_IDs*) to manipulate the database objects in *set_of_objects* in a way restricted by parameter *right*. To permit access to all its objects, the permitting CA can invoke the permit operation with an empty set of objects. In our sample scenario of direct interplay between the cooperation control level and the level of ACID trans-

3. When a sub_CA is terminated all its rights on database objects migrate to the super_CA.

actions manipulating not versioned data, *right* can be read or write. The set of objects can also be specified implicitly by referring to a terminated transaction, so that all objects, this transaction has been manipulating, are pre-released.

*Delegate (delegating_CA_ID, set_of_receiving_CA_IDs, set_of_objects).*
This operation causes a complete transmission of all rights, the delegating CA has on the indicated objects, to the receiving CAs. Again, the set of objects can implicitly be specified (delegating all rights a short transaction held) or can be empty, to delegate all objects the delegating CA has been working on.

The set of operations mentioned so far is only a subset of the set of operations which should be provided by the cooperation-control layer. Nevertheless, these operations are best suited to express the character of explicit cooperation control. Further operations (which cannot be detailed in this paper) are needed to make the data exchange more comfortable, e.g., maintaining conditioned data requests between CAs, to support other ways of cooperation, e.g. goal negotiations, or to provide facilities for notifying users after certain activities of cooperating users[4].

**Interplay of concurrency control and cooperation control**

As already mentioned, users perform their work in the context of a CA by issuing transactions. Transactions can be initiated by the operation
*Initiate_Transaction (user, CA_ID, function): TA_ID.*
The specified *user* must be a user of the CA indicated by parameter *CA_ID*. Parameter *function* contains an application-dependent, elementary action, thus abstracting from a transaction program or an interactive database session. Due to flexibility, tools should be supported allowing the user to selectively finish the current transaction and start a new one in order to make the data manipulated so far visible to others.

All concurrently active transactions (without considering to which CA they belong) are performed in isolation. Suppose, serializability is enforced by the two-phase locking protocol. At the successful end of a transaction its locks are not released but inherited by the corresponding CA. This implies, that modified data is, at first, only visible to users of the same CA and still locked for different CAs. Cooperation between CAs is to be managed at the cooperation control level by using the primitives introduced above.

Because of lock inheritance we have to consider the case that a transaction acquires a lock held by a different CA. In this case the transaction cannot be blocked, because CAs are expected to have a long system residence time. Here, notification facilities are to be provided. The user who issued that transaction needs to be notified about which objects, for which his transaction cannot be granted the acquired lock, are locked by which CA in which mode[5]. This information can help the user to take up cooperation with the according CA.

**Example: Group Editor**

Design applications, for which the CONCORD model has been developed, require all activity levels depicted in Fig.3 as shown in [Ri94]. Other kinds of cooperative work-arrangements do not require all these levels. An application scenario where the direct interplay of the cooperation-control layer (level 5) and the transaction-management layer (level 1) suffices, is a group editor. Suppose words, sentences, paragraphs, pages, sections, etc., to be the data granules which are manipulated by transactions. The group work can be organized in two ways. First, each user is assigned to a single CA; second, users who are working more closely together, belong to the same CA. Writing text is done within a transaction, and after having finished a semantically meaningful text fragment the user can terminate the current transaction and the modifications become visible to the other users of his CA. Additionally, he can grant read or write permissions to the cooperating users of different CAs. To modify text the user has to issue a transaction which reads the text fragment which is semantically affected by the modification for update. This requires an exclusive lock. It cannot be granted if one of the following situations exists. First, a concurrent transaction, issued by a user of the same CA, keeps locks on parts of the text. Second, a concurrent transaction, issued by a user of a different CA, keeps locks on parts of the text. Third, a different CA keeps locks on parts of the text and did not grant permissions to the considered CA. In the

4. To support notification services the concepts of active database systems [RIDE94] are well suited.
5. To read a data item a transaction only needs a shared lock allowing the lock-holding transaction and concurrently active transactions to only read and not write it. To write a data item a transaction needs to get an exclusive lock [GR93].

first case the user has to wait (his transaction is blocked) until the semantically atomic set of modifications issued by the cooperating user is completed. Then he has to read the modified text again and to decide whether his intended modifications are still reasonable or not. In the second and third case the user is notified by the system that cooperation facilities must be applied before the text fragment can be accessed.

This example shows that it is appropriate to manipulate text by issuing transactions due to consistency purposes (semantically correct modifications). It is necessary to isolate those parts of the text from other users whose semantics are potentially affected by modifications. It doesn't make sense to let users concurrently modify a text fragment, if modifications of one user change the contents of the complete fragment.

## 4. Conclusions

In this short position paper, we proposed mechanisms of enhancing traditional transaction processing by a layer for explicit cooperation control in order to support groupware applications. The concepts were achieved by generalizing the ideas of the CONCORD model, which has been proposed to support cooperative design applications. The argumentation for applying traditional transactions also in groupware applications is the consistency aspect. Elementary actions performed by cooperating users usually consist of several database operations. Due to consistency, this sequence of operations must be performed completely or not at all. Only the results of these elementary actions are reasonable granules of cooperation. Obviously, the entire set of actions performed by a cooperating user can't be performed within a single ACID transaction. Considering the inherent dynamics of groupware applications and the required flexibility, we decided to combine explicit cooperation control facilities with traditional transaction processing. In order to provide a user-centered approach to cooperation control, the following two aspects should be further explored. First, flexible application tools must be developed that are not implemented as a single implicit transaction, but allow the user to flexibly partition the tool execution into several transactions during runtime. This allows him to make meaningful processing states dynamically visible to cooperating users. Second, dynamic locking granules are needed as shown in the group editor example. It should be possible to lock exactly the amount of data which is semantically affected by the intended modifications.

## 5. Literature


ABEK94 Agrawal, D.; Bruno, J.L.; El Abbadi, A.; Krishnaswamy, V.: Relative Serializability: An Approach for Relaxing the Atomicity of Transactions; Proc. ACM SIGACT/SIGMOD Symposium on Principles of Database Systems, May 1994.

BSW79 Bernstein, P.A.; Shipman, D.W.; Wong, W.S.: Formal Aspects of Serializability in Database Concurrency Control; IEEE Transactions on Software Engineering; Vol 5; No. 5; May 1979, pp. 203-216.

DE93 Bulletin of the Technical Committee on Data Engineering, Special Issue on Workflow and Extended Transaction Management, IEEE Computer Society, June 1993, Vol. 16, No. 2.

EGR91 Ellis, C.A.; Gibbs, S.J.; Rein, G.L.: Groupware - Some Issues and Experiences, Communications of the ACM, Vol. 34, No. 1, Jan. 1991, pp. 38-58.

EGR90 Ellis, C.A.; Gibbs, S.J.; Rein, G.L.: Design and use of a group editor; Engineering for Human Computer Interaction; Cockton, G., Ed.; North Holland, Amsterdam; 1990; pp 13-25.

El92 Elmagarmid, A. (ed.): Transaction Models for Advanced Database Applications, Morgan Kaufmann, San Mateo, CA, 1992.

GR93 Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publ., San Mateo, CA, 1993.

GS87 Greif, I.; Sarin, S.: Data Sharing in Group Work, in: ACM Transactions on Office Information Systems, Vol. 5, No. 2, pp. 187-211.

Gr94 Grudin, J.: Computer-Supported Cooperative Work: History and Focus, IEEE Computer, May 1994, Vol. 27, No. 5, pp. 19-26.

HR83 Härder, T., Reuter, A.: Principles of Transaction-Oriented Database Recovery. ACM Computing Surveys 15, 4, 1983, pp. 287-318.

NZ92 Nodine, M.H.; Zdonik, B.: Cooperative Transaction Hierarchies: Transaction Support for Design Applications; VLDB Journal 1, 1992, pp. 41-80.

RB91 Rodden, T.; Blair, G.: CSCW and Distributed Systems: The Problem of Control, in: Proc. 2nd European Conference on Computer-Supported Work, Sept. 25-27, 1991, Amsterdam, Netherlands

RIDE94 Proc. 4th Int. Workshop on Research Issues in Data Engineering (RIDE), Active Database Systems, February 14-15, 1994, Houston, Texas.

Ri94 Ritter, N.; Mitschang, B.; Härder, T.; Gesmann, M.; Schöning, H.: Capturing Design Dynamics - The CONCORD Approach; Proc. 10th Int. IEEE Data Engineering Conference, Houston, Texas, Feb. 1994, pp. 440-451.