

Cooperative database system: A constructive review of cooperative transaction models *

Heri Ramampiaro and Mads Nygård
Norwegian University of Science and Technology (NTNU)
N-7491 Trondheim, Norway
{heri, mads}@idi.ntnu.no

Abstract

The use of transactions to provide reliable and secure information processing and data management has increasingly gained the attention of the CSCW community. Traditional transaction models have, however, been claimed to be too restrictive in the context of cooperation. Research on transaction support for cooperative work has therefore particularly aimed at finding mechanisms and frameworks to overcome this limitation. In this paper we survey the work done in the development of flexible and cooperative transaction models. This survey is divided into two categories. The first consists of approaches mainly based on the CSCW perspective, while the second category consists of database methods extended to support cooperation. The paper also provides an evaluation of the models with respect to relevant criteria and requirements for cooperative work settings.

1. Introduction

Computer Supported Cooperative Work (CSCW) is a multidisciplinary field that examines how groups of people work and attempts to find a way to provide appropriate technological support for the work process. Research on CSCW has gained a lot of attention over the last two decades [18] due to changes in the use of computers. In modern organisations, especially those involving design and product manufacturing, close cooperation between co-workers is often needed in order to get the work done. New tools and applications have been developed to enable such cooperation. Examples of these are groupware [8] and workflow systems [14]. However, as organisations evolve, requirements change. It is widely accepted that effectiveness or performance is no longer sufficient for the organisations to achieve their goal. Also, reliability has been shown to be crucial. Bearing this in mind, we believe it is important that cooperative information systems consider the definition and use of correctness criteria, as well as to provide support for consistent handling of failures and exceptions. These issues

have been widely addressed in traditional database systems through the adoption of the concept of transaction.

Transactions provide mechanisms that can solve many of the problems incurred by concurrent access to shared objects. In the past few years, several transaction models have been suggested for advanced applications. The main purpose of this paper is to give a constructive review of these models. Here we will devote special attention to the models that have been suggested and developed to support cooperative activities. Our main contribution is on the evaluation of the models with respect to their appropriateness and viability for cooperative work settings.

The rest of this paper is organised as follows. Section 2 provides a general overview of cooperative work characteristics. Section 3 briefly introduces the concept of transaction. It also outlines some of the traditional extended transaction models that we have selected based on their potential applicability in advanced applications. In Section 4, we give a survey of newer cooperative transaction models, that we later evaluate in Section 5. Finally, in Section 6 we briefly present some suggested approaches to satisfy a minimal set of requirements of cooperation.

2. Cooperative work characteristics

A precise definition of cooperative work is hard to find. However, several attempts can be found in the literature. One of the earliest definitions is the following given by Marx [18]: "... *multiple individuals working together in a planned way in the same production process or in different but connected production processes.*"

This is not a complete definition by far. It is too narrow since it assumes that cooperative work is always planned. However, as we will see later, such an assumption is not accurate. Because of this, the definition needs to be extended. An intuitive and a more generic definition could be: a work process involving several people acting together, in a shared context to perform some tasks in order to achieve a pre-specified common goal.

Just as a precise definition for cooperative work does not exist, neither do classifications and characterisations. However, based on the investigation done by Schmidt and Ban-

*© 2000 IEEE Computer Society

non [18], cooperative work may have at least one of the following main characteristics:

(1) The degree of cooperation

between involved parts may differ depending on the situation requirements. Some times a tight cooperation (or collaboration) may be required in order to get the work done; in other cases cooperation may not be needed at all. As illustrated in Figure 1a, the spectrum may therefore span from individual activities to collaborative activities.

(2) Variable number of users:

The size of the ensemble is not fixed, and may change from time to time. This means that the users involved in an activity may vary from a single user to many users, yielding the spectrum in Figure 1b.

(3) Separation

in both time and space is quite usual. The interaction between the involved parts may be synchronous (real-time interaction) or asynchronous (non-real-time interaction). In addition, they may be co-located or dispersed. Figure 1c illustrates this diversity [8].

(4) Variable length of interaction:

The duration of the interaction or work activity is not always possible to predict in advance. Some times, a complete set of actions can be specified before the task is initiated. For such a type of activity, an approximated duration of the involved interaction can be estimated. Otherwise, this is not possible. See Figure 1d.

(5) Dynamic interaction:

The spectrum of the cooperative activities may span from structured (e.g. activity that involves the use of workflow), via semi-structured (e.g. software development) to ad-hoc (e.g. publication activities). Figure 1e gives a graphical representation of this property.

3. “Standard” advanced transaction models

The transaction concept has been traditionally used in database systems to deal with concurrency and failure anomalies. Informally, a transaction is defined as a basic work unit executed on a database or a shared system resource. It is characterised by *atomicity*, *consistency*, *isolation*, and *durability* properties (also referred to as the ACID properties) [3, 12]. Here, atomicity means that a transaction should be run in its entirety or not at all (the “all-or-nothing” property), consistency is the property that ensures a transaction always transforms a (database) system from a consistent state to another consistent state, isolation means that any transient states of the (database) system caused by a specific transaction should be invisible to other transactions until it is committed, whereas durability insures that once a transaction is committed, its result remains permanent. Each transaction that has all of these properties is referred to as ACID transaction.

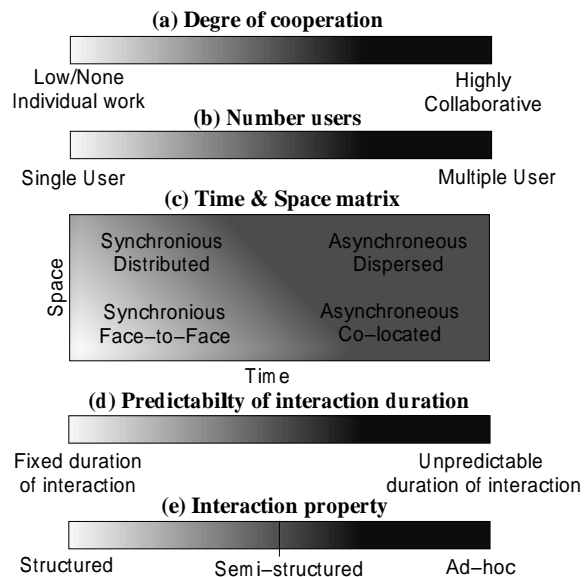


Figure 1. Graphical illustration of the cooperative work characteristics

It is widely known that traditional ACID (normally flat) transactions provide well-defined correctness criteria (e.g. *Serializability* [3]) and efficient support for failure and exception handling (e.g. *Recoverability* [3]). However, with the ACID properties they are too restrictive and too simple for advanced and cooperative applications. This has resulted in a call for more advanced and more flexible transaction models, which we will now outline.

Nested Transaction model was introduced by Moss [12] to extend the flat transaction by splitting transactions hierarchically into several recursive sub-transactions. A child transaction may start after its parent has started, and may commit locally. The committed local result is, however, released only when all of its parents up to the root have successfully terminated. A parent transaction, on the other hand, may terminate only when all of its children have terminated, and it may execute another alternative transaction if a child fails. However, if it aborts, all of its children must also rollback independent of their current state.

The nested transaction does not address cooperation since full local and global isolation are required. However, it allows increased modularity, more concurrency and finer recovery granularity than the traditional flat transaction.

Saga was introduced by Garcia-Molina and Salem [10] primarily to deal with long transactions, by using the concept of compensating transactions. A Saga is a transaction that consists of a *sequence* of several ACID (sub-) transactions and associated compensating transactions. Each sub-transaction is allowed to commit individually. A compensating transaction is then used to explicitly undo its effect

if the whole Saga transaction has to abort. Finally, a Saga can terminate only when all of its sub-transactions including any compensating sub-transactions have run successfully.

By allowing sub-transactions to commit, thus revealing their partial result(s), Saga relaxes the full isolation requirement. This means that some degree of cooperation is permitted. However, the main drawback of the concept is that it is not fully implemented [16]. This means that it is difficult to provide a complete evaluation of its performance and usefulness. Nevertheless, the idea has been used to implement other models like transactional workflow [2], which shows its potential applicability.

Cooperative Transaction Hierarchy is a transaction model proposed by Nodine and Zdonik [17] for design environments. It is like the nested transaction model, a tree-based approach. In this case, the tree is restricted to three main levels: a *root*, one or more *transaction groups (TG)* and several *cooperative transactions (CT)*. The cooperative transactions correspond to the leaf nodes, which are grouped into transaction groups. They are associated each with a designer in the environment and can, within a transaction group, cooperate on some task. Cooperative transactions do not need to be serializable; instead for each transaction group, *patterns*, being a set of rules for how operations can be interleaved, and *conflicts*, being a set of rules that specify which operations are not allowed to run concurrently, are used as correctness criteria. Both can be tailored to the needs of the application.

Although cooperative transaction hierarchy addresses cooperation, its main weakness is the need to define both patterns and conflicts in advance. This is because all sets of operations must be known a-priori, which limits the use of the model to applications with a well-defined work structure.

Open nested transaction and its specialisation multilevel transaction models were both proposed by Weikum and Schek [20]. They were suggested to improve the nested transaction model, by allowing sub-transactions to issue final commit and by using *compensation* as in Saga to further enhance inter-transaction parallelism. As in the nested transaction model, they are both tree-based approach but the tree for the multilevel model is balanced.

With both open nested and multilevel transaction models, all instantiated sub-transactions do not need to run successfully for the transaction to terminate. This means that the atomicity property is compromised, thus allowing flexibility. They also permit a higher degree of cooperation than the previous nested transaction model by relaxing the global isolation. Both models allow running transactions to reveal their partial results to other concurrent transactions, though these partial results can be used by operations that commute

only. This is managed by using so-called semantic-base concurrency control. The main drawback of both models is that both stick to Serializability as their correctness criterion, making full flexible sharing of tentative data impossible. Finally, the implementation of the model is, for the authors knowledge, still limited.

Split/Join Transaction was proposed by Keiser and Wu [9]. Its main goal is to provide transactions the ability to share resources by allowing dynamic reconstruction of running transactions. It was originally developed for activities with uncertain duration, unpredictable developments, and interaction with other activities. The principle is basically to *split* a running transaction into two or more transactions and later *join* other transactions by merging their resources. The model then addresses cooperations among users by allowing transfer of resources from one transaction to other transactions. Further, it introduces an adaptive recovery mechanism which allows part of the work done to be recoverable, and since a committing transaction part may release some of its resources, isolation may somehow be reduced. The main drawbacks are however the emerging complex merging mechanisms. Moreover, the two resulting transactions from the split command still have to obey a Serializability criterion which implies that the two transactions still must be seen as two isolated transactions while running.

ACTA is a transaction framework that allows formal reasoning about the properties of transaction models. It is not a transaction model in that sense, but it can be used to specify transaction models by determining the effects of transactions on other transactions (interaction between transactions) and the effects of transaction on objects. ACTA [4, 5] uses first-order logic to capture properties of transactions, such as visibility, consistency, recovery, and permanence. A new extended transaction model can be specified by using *ACTA-axioms*. Its main building blocks are history, inter-transaction dependencies, transaction view, conflict set, and delegation, which can be used to "invent" new transaction models or just extend existing models by methodically modifying their definition. Using ACTA a Saga can, for instance, be extended by giving it a new nested structure [4]. In view of this, the usefulness of the ACTA framework depends on which models we want to specify and modify. Its power lies in the ability to represent structural behaviour of transactions and the dependencies between them. However, implementation of ACTA-models are not always straightforward.

4. Towards cooperative transaction models

As seen in the previous section the "standard" advanced transaction models do not cover all of aspects of collabora-

tion. In the past few years, researchers have therefore tried to develop transaction models and frameworks that further improve the support of cooperative work. This effort can be divided into two categories. The first category consists of developments based on cooperative work requirements, and thus focusing on the CSCW perspective, while in the second category we can find models that have been developed based on extensions of existing database fundamentals (e.g. locking protocols and timestamp ordering). We now give a general description of these models.

4.1. Coo

Coo is a research project aiming to develop a framework to support cooperation between software developers, by encapsulating software processes into long and cooperative transactions [11]. The transaction model was developed based on the software development processes requirements. A new model with relaxed atomicity and relaxed isolation was suggested.

In Coo, relaxing the atomicity basically means that long transactions may save their *intermediate results*, thus minimizing losses in the case of crashes, while relaxed isolation allows several software processes to access these intermediate results without violating the correctness criterion.

Intermediate results are managed by applying three different object consistency levels. These are *stable*, *semi-stable*, and *unstable*. An object is stable when it is fully consistent (e.g. a result from a successfully committed transaction). Semi-stable objects are those some processes may generate as tentative data, and can be seen as consistent enough, but may violate some of the correctness criteria. For this reason, access to such objects is restricted. That is, only processes that satisfy the semantic rules and integrity constraints encoded in the software process description are allowed to use semi-stable objects. Finally, unstable objects are those that do not satisfy the correctness criterion at all, and are currently locked by some processes. It is inaccessible until it becomes stable or semi-stable.

The three types of object are stored in three different databases, which are managed through the use of a workspace concept. Figure 2 illustrates the relation between object consistencies and the three database types. The double arrows show which databases contain which object types. In view of this, each database can be seen as a workspace for the processes. Data exchanges between workspaces are carried out using the following set of operations. These are *CHECK OUT*, which is used by a process to transfer an object from the public database to the private database, and *CHECK IN*, which is issued when a process is terminated and it transfers its final result to the public database. Other operations are *UPWARD COMMIT* and *REFRESH*. The first is used by a process to make its

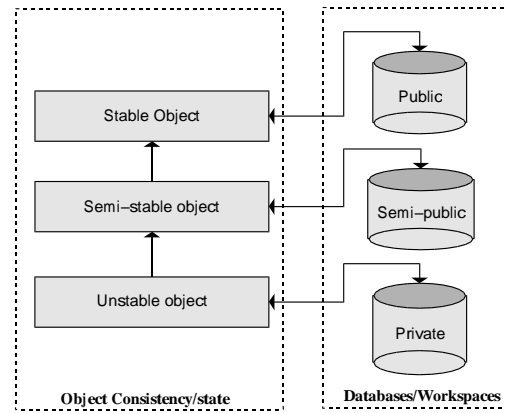


Figure 2. Relation between object consistencies and database types

intermediate results available to other processes by storing it in a semi-public database. The second is issued when the results have new values.

4.2. Collaborative databases

Agrawal et al. [1] suggest a new database approach to manage concurrent activities in collaborative environments. Their work was inspired both by the attempts by database researchers to relax the traditional issues of atomicity and isolation, and by the proposals by software engineering and design environments. Their main goal was therefore to develop a transaction model by merging flexible transaction models from collaborative environments and semantic based correctness criteria.

They introduce the notion of *relative atomicity* of co-actions¹ as a relaxation of the traditional atomicity property (or absolute atomicity). It is used to specify how a co-action can be interleaved relative to other co-actions without breaking the overall atomicity requirement for the collaborative activity. Their main idea is that before a collaborative activity takes place, first a *collaboration channel* is established. Then, by connecting to the channel, different transactions may cooperate on the same data objects following the relative atomicity constraints. Correctness of execution is checked against a so-called *relative serializability (RSR)* correctness criterion, a more relaxed criterion than the traditional serializability (SR). That is, any execution obeying the RSR criterion would preserve the consistency of the database even if it is not serializable. Figure 3 illustrates how different transactions can specify their atomicity relative to other transactions, and how this specification is related to the relative serializable criterion. In this example, we consider three transactions T_1 , T_2 , and T_3 and their respective relative atomicity relations. Execution S_1 is rela-

¹A co-action is a sequence of read and write operations executed on data objects.

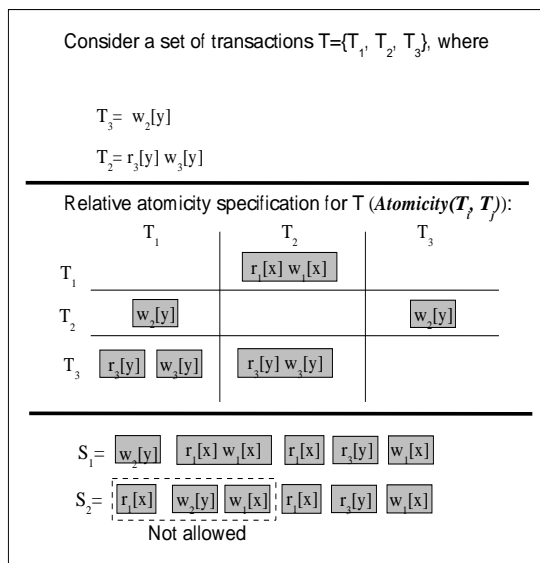


Figure 3. Illustration of the relative serializability specification

tive serializable with respect to the relative atomicity specification, while S_2 is not. To guarantee the relatively serializable execution of co-actions a new lock protocol is suggested. It is an extension of the standard 2-phase lock protocol (2PL) [3], by introducing the notions of *push-forward* and *push-backward* locks. In the event of potential conflict, both of the locks have to be acquired before an involving operation sets a normal lock. Push-forward lock causes the conflicting operation to be delayed until the last operation of the atomic units of co-actions with which it conflicts is run, whereas pull-backward lock is used to “pull” operations backward before the start of the atomic units. These are possible if the cause of pulling or pushing the operation does not change its effect.

4.3. EPOS transaction model

EPOS [6] was a research project at the Norwegian University of Science and Technology to develop a framework for quality assured software engineering. To support different software engineers, a database called EPOS-DB was designed to manage resources produced in the development process. For consistent access to the database, a flexible transaction model is proposed. This is another extension of the standard ACID transaction to support cooperation between co-workers in a Software Engineering Environment (SEE). It shares some of the ideas of Coo in the sense that it is based on the use of workspaces (both private and common workspaces). The transaction model also uses check-in/ check-out operation for interaction through/with the workspaces. As in Coo, Conradi et al. assumed that transactions for SEE are generally long-lived. The use of

transactions and workspaces with nesting structure were therefore suggested. Figure 4 illustrates this structure for a parent transaction T with N children T_i .

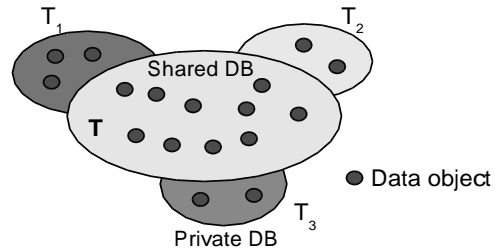


Figure 4. Illustration of a workspace with nesting structure in EPOS [6]

To deal with concurrency, two types of lock were suggested; mandatory and cooperative locks. Mandatory locks are used to prevent or restrict operations from accessing shared data, while cooperative locks allow different operations to access a shared object without any restriction. Awareness support is then provided to allow users to know about the event that may affect their work. Correctness of execution is achieved by applying mechanisms for conflict change detection. Finally, the users working on the EPOS-DB are given the ability to define a set of operations before the transactions are run [19], which is used as a baseline for the conflict detection mechanism. In this case, serialization is no longer needed.

4.4. TransCoop/CoAct

TransCoop was a basic ESPRIT research project between the database research group at GMD-IPSI (Germany), University of Twente (The Netherlands) and VTT Information Technology (Finland) [7]. The goal of the project was to develop a transaction model and a specification language to enable effective information sharing. In this presentation, we will consider the TransCoop transaction model only. The transaction model was called CoAct and was developed based on an extension of existing advanced transaction model. Their motivation was to overcome the limitations imposed by the use of a standard ACID model. Then by using four application scenarios such as

- cooperative authoring, being based mainly on ad-hoc processes,
- design for manufacturing, being characterized by structured activities,
- software engineering, being characterized by semi-structured processes and
- workflow, focusing mainly on automated business processes,

the requirements for the transaction model were defined. They agreed with other researchers in the field that both relaxed atomicity and relaxed isolation are strongly needed. In addition, the model should allow the users to explore several alternatives to solve a problem and to revise erroneous actions (retraction of decision). Moreover, it should provide support for management of alternative versions of data objects (private and shared data). Finally, the transaction model should allow the use of execution constraints to coordinate individual and joint work.

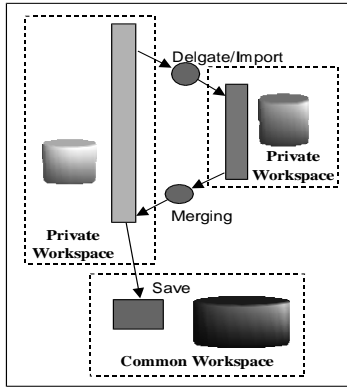


Figure 5. TransCoop workspaces and exchange operations

Basically, the ideas of CoAct share some of those of the “standard” advanced transaction models described in the previous section. Example of these are *compensation and semantic-base concurrency control* from the open nested and multi-level transaction models, *resource exchange* from the split/join transaction model, and *delegation* in ACTA. They then extended the Check-In/Check-out, versioning and workspace models with sophisticated *history merging* mechanisms.

Figure 5 illustrates how different workspaces are used in CoAct. It is worth to note that in CoAct, exchange of operations between workspaces is used instead of exchange of data (as in Coo and EPOS). The correctness of interactions is checked by validating the history produced after each exchange (delegation, import or merging).

4.5. New Timestamp ordering approach

Zhang et al.[21] suggest a new timestamp ordering(TO) approach that allows both traditional short transactions and long cooperative transactions to be run within the same system. Their method is in the same category as that of Agrawal et al. in Section 4.2 in the sense that they also use database fundamentals as a starting point for their approach.

Using this novel timestamp ordering, long cooperative transactions can abort without blocking other short transaction. Their main idea is that when two operations within a traditional transaction and cooperative transaction are in conflict, we should neither abort the cooperative transaction nor block the traditional one. Rather, we assign the operation a new global timestamp and let the transactions continue as long as the serializability among all the transactions involved can be preserved. The long cooperative transaction will then incorporate the recent updates into its own processing, thus avoiding abortion and loss of all of the partial results. Figure 6 illustrates how a long cooperative trans-

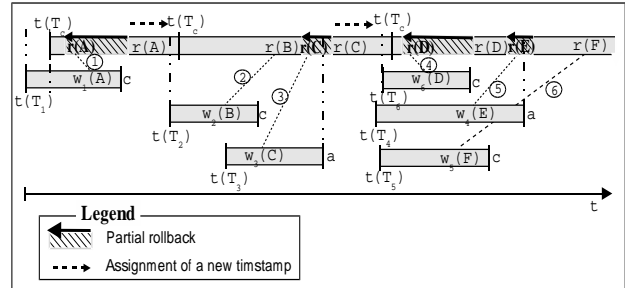


Figure 6. Illustration of the use of the new timestamp ordering approach

action T is run together with several short (ACID) transactions and uses the suggested timestamp ordering approach to handle conflicts. As shown in the figure, the following are examples of situations that may occur:

- (1) T reads a value A that later is updated by T_1 before it commits. T_1 has started before T and therefore has a smaller timestamp than T . With traditional TO, T_1 would need to abort and restart with a bigger timestamp. Instead, the scheduler partially rollbacks T , assigns it a new timestamp, and lets it rerun the read operation $r(A)$.
- (2) Later T tries to read a value B written by T_2 after T_2 has committed. Since T now has a timestamp smaller than T_2 , we should abort T . Instead, T is assigned a new timestamp bigger than T_2 and T may proceed.
- (3) Now, T_3 updates C , which T tries to read before T_3 has terminated. If T_3 had committed, we would have the same situation as in (2). However, since T_3 aborts, $r(C)$ is inconsistent and the scheduler has to partially rollback T so that the read can be reprocessed.
- (4) T has read a value that T_6 updates later. Since the newest timestamp for T is still smaller than for T_6 , T must abort some of its actions, acquires a new timestamp as in (2), and incorporates T_6 's changes.
- (5) We have a conflict between $r(E)$ and $w_4(E)$, and this seems to happen before T_4 has terminated. Here, T_4 aborts, then $r(E)$ becomes inconsistent. Therefore, T has

to partially rollback, and rerun $r(E)$. Otherwise, if T_4 had committed we would have the next situation.

(6) Finally, $r(F)$ is run by T which obviously conflicts with T_5 's write operation. Since, the timestamp for T_5 is smaller than for T , we have a legal situation and T may proceed as normal.

According to the authors, these six cases cover all types of situation and that the corresponding TO rules produce a correct transaction execution. To prove the correctness, the notion of final serializability, also called *f-serializability* is introduced, with which only the last read or write conflicts are given a priority.

5. Model evaluation

In this section, we give an evaluation of the models presented in the previous section.

5.1. Transactional support requirements

Before giving the evaluation of the models, we first define a set of requirements we believe transaction models must satisfy in order to support cooperative work. These requirements are divided into three categories; transaction properties, transactional support, and services provided. They have been mainly derived and extracted from the characteristics described in Section 2.

* Transaction properties

As already mentioned, ACIDity is inappropriate for cooperative environments, especially because of the atomicity and isolation properties. We however believe that both consistency and durability should be preserved to be able to have reliable data management.

R1: Compromised atomicity.

To prevent long transactions from losing lots of work effort in the presence of failures or exceptions, the transaction model should support *partial rollback*. That is, instead of discarding all changes, we should only undo part of it. This means that we need a compromised atomicity. Moreover, since cooperation between co-workers or/and operations is important, flexible interleaving between transactions is needed. This implies that transactions should no longer be seen as an atomic unit of work.

R2: Sharing of intermediate results.

Sharing of information or artefact is crucial in order to cooperate. Requiring operations to run as isolated units of work would therefore be unsuitable. For this reason, the isolation property should be relaxed to enable sharing of tentative results. However, concurrency control mechanisms are still vital.

* Transactional support

The nature of cooperation may affect the transactional support that should be provided. These are:

R3: Open-ended running.

The fact that we are not able to predict the composition of the ensemble involved in a cooperation and that the interaction may have an uncertain duration and form, the transaction should be open-ended. The transaction should, for example, be able to provide support for earlier commit if this is desired or needed.

R4: Scalability.

Since the number of users involved in a cooperative activity is not fixed, scalability is crucial. The framework or the model should therefore be able to support a seamless transition from single operation to multiple cooperative operations and vice versa.

R5: Distribution and heterogeneity.

CSCW introduces a high degree of diversity and distribution in both time and space. It is therefore crucial that the transaction model/framework takes both distribution and heterogeneity into consideration.

* Services provided

A minimal set of services should be provided in order to support cooperation [15]. The following are example of these:

R6: Awareness.

In flexible sharing of information, notification about changes done on common objects is necessary. As a result of the relaxed isolation, awareness services should be provided in order to handle potential conflict(s).

R7: Temporal data management.

It is usual that members of a cooperative ensemble may leave and join the group without wanting to face any restriction. To deal with this, the members should be able to know about the changes made in the environment and/or the shared information while they were away. Temporal data may, for instance, serve as a means to support extraction of different alternative decisions or be used to resume actions that have been interrupted.

R8: Access control.

Though sharing of artefacts is important in cooperative settings, we may not want to share all of the objects. A set of rules defining which data/objects can be accessed by whom should therefore be specified. This may also hinder interference between the users. For this reason, a sophisticated access control service should be provided.

We believe that cooperative transaction models or frameworks that satisfy these requirements can provide successful support of cooperation. However, *flexibility* should be considered for all of the dimensions. First, the framework

should be able to seamlessly switch or integrate different form of support depending on the need. Some situations may, for example, need strict atomic transactional support in order to have reliable processing, other may see atomicity as just a burden. Moreover, to be able to support different kind of situations, it should be possible to tailor the model or the support provided to the needs of the cooperative settings.

5.2. The evaluation

This section presents the evaluation of the newer cooperative transaction models outlined in Section 4. The following is a list of criteria that has been used:

Requirement: How the models are positioned with respect to the list of requirements?

Application/domain covered: Are the models applicable on a wide application area?

Formality: Is the method used based on formal provable foundations? (We should however consider the trade-off between implementability and the formality).

Correctness: Do the model apply any type of correctness criteria?

The positions of the models with respect to the list of requirements are summarised in Table 1.

Coo introduces a formal framework. However, their use of perhaps too formal temporal logic may impose difficulties in practical implementations. Nevertheless, a working prototype is provided. As mentioned, Coo was specifically developed for Software Engineering Environment(SEE). Some aspects of cooperation may not be well supported. For instance, flexibility with respect to ad-hoc process support is restricted. However, the idea of using three degrees of correctness allows more collaborations than with the standard transaction models.

Collaborative database also uses formal models. They extend the traditional databases with more flexible correctness criteria to support collaboration. A typical application area is design environments. However, the use of the notion of co-action and relative atomicity and serializability requires that the transactions must know all of their operation sets before they can be executed. This implies that ad-hoc applications are not well supported because the transaction may not be created dynamically.

The EPOS transaction model shares many of the assumptions underlying Coo. It introduces a nice framework that allows co-workers in a SEE to cooperate with high degrees of flexibility. The requirements for the transaction model were derived from those applied in software development. Because of its flexibility, EPOS can be used to support other type of applications. However, the framework doesn't provide explicit guidelines for how this can be used

effectively. It uses instead the underlying process model to define the degree of cooperations. Possibility for integration is also restricted. Moreover, no new formal mathematical foundation is directly provided. This makes the correctness of the execution used in the model difficult to validate with formal methods. Nevertheless, a working prototype is available.

TransCoop/CoAct is one of the models that has attempted to cover the most broad application area. They based their approach on four application scenarios, that include both structured and ad-hoc activities. Moreover, formal foundations based on provable mathematical formalism are provided. This implies that the approach can be formally validated. However, the merging mechanisms introduced in the model may impose some complexity, which may affect the overall performance of the system. The available prototype shows only its use in co-authoring applications.

New Timestamp ordering approach is a quite new approach, which bases the support of cooperative transactions on the extension of the existing database concurrency control mechanism. Support for ad-hoc activities is better than for the relative serializability method since the definition of operation sets is not needed before execution. Inconsistencies are checked during run-time. However, implementation is not yet available, which makes the approach difficult to be fully evaluated.

6. Approach towards a generic framework and architecture

By using the result of the evaluation presented in the previous sections we may conclude that the available transaction models may somehow still be unsatisfactory for use in a wide set of cooperative applications. The following are suggestions to further extend the existing systems:

(1) Framework rather than a single model. Because of the diversity of the cooperative settings, we do not believe that a single transaction model would be able to cover all types of cooperation. Therefore, instead of introducing a new model we suggest the use of a framework for transaction models. A successful framework will provide a collection of different transaction models. It then gives the user a guideline to choose appropriate models for different situations. The system might, for instance, in one hand suggest the use of a standard ACID model when the need of sharing is very low, while on the other hand, when the user need to share their data, it might suggest a suitable flexible transaction model.

(2) Groupware with transactional theoretical basis and reliability. Groupware systems provide high computing flexibility. Its main weakness is however the lack of security and reliability. By using ideas from database transactions we may develop a groupware system which is reliable

Requirements/ Models	Coo	Collaborative Database	EPOS	TransCoop	Novel TO
<i>Compromised atomicity</i>	Saving of intermediate results	Relative atomicity, given additional semantic	Supports partial rollback	Supports partial rollback	Supports partial rollback
<i>Relaxed isolation</i>	Controlled exchange of intermediate results	Relative serializability	Exchange of intermediate results	Sharing of tentative objects by exchange of operations	Incorporation of tentative results controlled by f-serialization rules
<i>Open-ended running</i>	Limited: Goal-oriented processes, termination upon goal-achievement.	Limited: All prespecified operation sets must be run.	Supported depending on the underlying process model.	Supported depending on the termination constraints	Supported as in traditional TO.
<i>Scalability</i>	-	-	-	-	-
<i>Distribution and Heterogeneity</i>	Distributed, homogeneous	Distributed, homogeneous	Distributed, homogeneous	Distributed, homogeneous	Distributed, homogeneous
<i>Awareness</i>	-	-	Change and lock notification	-	Commit notification
<i>Temporal data management</i>	-	-	-	History management; decision retraction	-
<i>Access Control</i>	Not explicitly considered	Depending on the underlying database, but not explicitly considered	Not explicitly considered	Not explicitly considered	Depending on the underlying database, but not explicitly considered
<i>Flexibility</i>	-	-	Supported without explicit guidelines but with flexible process models	Supported as long as the scenarios for the actions/activities are defined	-

Table 1. Comparison w.r.t cooperative work requirements. (The label ”-” means not known for the authors or not provided/supported).

and secure, while at the same time flexible enough to handle cooperation.

(3) Heterogeneous Autonomous Distributed(HAD) architecture. To handle the diverse nature of cooperative systems, we suggest an architecture based on HAD assumptions. Figure 7 shows a highlevel architecture that could be used. Here both the applications and the stores are heterogeneous. They are connected through a CORBA-like [13] middleware.

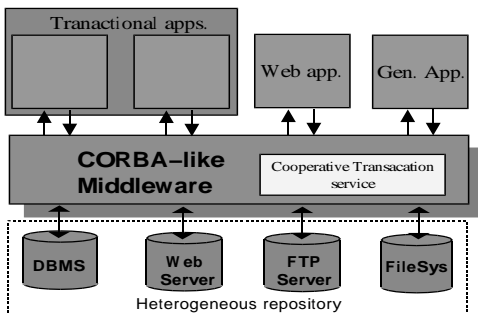


Figure 7. A highlevel HAD architecture

7. Concluding remarks

This paper has given a general survey of the “standard” transaction models that have been mainly suggested to handle long and complex interaction. Because of the general

lack of implementation and their limitations in covering most of the aspects of cooperation, several research projects have been conducted with the aim of extending the models. The results from these efforts are newer cooperative transaction models. These have been given a special consideration in this paper. Table 2 summarizes the models presented, including the result from our evaluation. Based on this result, we believe all of the models can be still extended in order to fully support cooperative work. From here our work is proceeding in three main directions. First, we develop a framework using a collection of existing transaction models rather than suggesting yet a new model. Second, we are investigating the use of the transaction concept combining with the ideas from groupware. Third we are developing an architecture based on the HAD assumption.

8. Acknowledgment

We would like to thank our colleagues in CAGIS for their suggestions and criticisms. We also would like to thank Babak Farshchian, Monica Divitini and John Mariani for their contributions in reviewing this paper. This research is partly supported by the Norwegian Research council (NFR). Finally, part of this work has been done at the Computing Department, Lancaster University, U.K.

References

- [1] D. Agrawal, J. L. Bruno, A. E. Abbadi, and V. Krishnaswamy. Managing concurrent activities in collaborative

Models	Coo	Collaborative Database	EPOS	TransCoop	Novel TO
Basic structure	Based mainly on check-in/check-out model	Not applicable	Based mainly on check-in/check-out model	Nested, check-in/check-out w/ delegation	Not applicable
Correctness Criteria	3-level, definable	Relative Serializability	Locking and conflict change detection	Valid History merging	Final Serializability
Formality	Uses formal temporal logic for transition and integration constraints	Uses formal database fundamentals	Limited; found only in the process model and post conditions	Applies mathematical foundations for the history merging rules	Uses formal database fundamentals
Flexible interaction	Ad-hoc process not well supported	Limited	High but lacking usage guidelines	Wide range of application support	Much depending on the underlying DBMS
Application Area(s)	SEE	Design Environment	SEE	Workflow, Cooperative Authoring, SEE, and Design	Unlimited
Implementation	Prototype	Prototype	Prototype	Prototype (Cooperative authoring only)	Not available

Table 2. Summary of the newer transaction models presented

- environments. In *CoopIS 1995*, pages 112–124, 1995.
- [2] G. Alonso, D. Agrawal, A. E. Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced transaction models in workflow contexts. In *ICDE 1996*, pages 574–581, 1996.
- [3] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
- [4] P. K. Chrysanthis and K. Ramamritham. ACTA: The saga continues. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 350–397. Morgan Kaufmann, 1992.
- [5] P. K. Chrysanthis and K. Ramamritham. Synthesis of extended transaction models using ACTA. *ACM Transactions on Database Systems*, 19(3):450–491, Sept. 1994.
- [6] R. Conradi, J.-O. Larsen, M. Nguyen, A. I. Wang, and C. Liu. Transaction models for software engineering database. In *Dagstuhl Workshop on Software Engineering Databases*, 1997.
- [7] R. A. de By, W. Klas, and J. Veijalainen. *Transaction Management Support for Cooperative Applications*. Kluwer Academic Publ., 1998.
- [8] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):9–28, January 1991.
- [9] A. K. Elmagarmid, Y. Leu, J. G. Mullen, and O. Bukhres. Introduction to advanced transaction models. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 35–52. Morgan Kaufmann, 1992.
- [10] H. Garcia-Molina and K. Salem. Sagas. In *SIGMOD International Conference on Management of Data*, pages 249–259, May 1987.
- [11] C. Godart. Coo: A transaction model to support cooperating software developers coordination. In *4th European Software Engineering Conference, Garmisch, LNCS 717*, pages 361–379, 1993.
- [12] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.
- [13] O. M. Group. *The Common Object Request Broker: Architecture and Specification*. OMG, v2.2 edition, 1998.
- [14] D. Hollingsworth. Workflow management coalition; the workflow reference model. Technical report, The Workflow Management Coalition, 1995.
- [15] J. A. Mariani and T. Rodden. Cooperative information sharing: Developing a shared object service. *The Computer Journal*, 39(6):455–470, 1996.
- [16] C. Mohan. Tutorial: Advanced transaction models - survey and critique. In *ACM SIGMOD 94*, page 521, May 1994.
- [17] M. H. Nodine and S. B. Zdonik. Cooperative transaction hierarchies: Transaction support for design applications. *VLDB Journal*, 1(1):41–80, 1992.
- [18] K. Schmidt and L. Bannon. Taking CSCW seriously. supporting articulation work. *Computer Supported Work. An International Journal*, 1(1-2):7–40, 1992.
- [19] A. I. Wang, J.-O. Larsen, R. Conradi, and B. P. Munch. Improving cooperative support in the EPOS CM system. In *The sixth European Workshop in Software Process Technology*, pages 75–91. Springer, 1998.
- [20] G. Weikum and H.-J. Schek. Concepts and applications of multilevel transactions and open nested transaction. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 350–397. Morgan Kaufmann, 1992.
- [21] Y. Zhang, Y. Kambayashi, Y. Kambayashi, Y. Yang, and C. Sun. On interactions between co-existing traditional and cooperative transactions. *International Journal of Cooperative Information Systems*, 8(1):1–24, 1999.