Transaction Models Supporting Cooperative Work – The TransCoop Experiences –

K. Aberer, J. Klingemann, T. Tesch, J. Wäsch, and E. J. Neuhold

Integrated Publication and Information Systems Institute (GMD-IPSI)
GMD – German National Research Center for Information Technology
Dolivostr. 15, D-64293 Darmstadt, Germany

{aberer, klingem, tesch, waesch, neuhold}@darmstadt.gmd.de
http://www.darmstadt.gmd.de/~{aberer, klingem, tesch, waesch, neuhold}

Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'96), Kyoto, Japan, December, 5-7, 1996.

Transaction Models Supporting Cooperative Work – The TransCoop Experiences –

K. Aberer, J. Klingemann, T. Tesch, J. Wäsch, E. J. Neuhold

Integrated Publication and Information Systems Institute (GMD-IPSI)
GMD – German National Research Center for Information Technology
Dolivostraße 15, D-64293 Darmstadt, Germany
{aberer, klingem, tesch, waesch, neuhold}@darmstadt.gmd.de

Abstract

Cooperative work on shared information requires different kind of computing system support to coordinate the work of multiple users, to establish mutual awareness and to ensure consistency. These issues are currently tackled separately in various loosely related areas, like workflow systems, groupware, and advanced transactional models. We present a transactional model that provides a core functionality for information sharing in cooperative systems, that explicitely supports cooperation primitives and at the same time ensures consistency of results. The model has been derived from a thorough analysis of various cooperative appliciation scenarios. It is currently being implemented as an extension of an object-oriented database management system and evaluated for a cooperative document authoring application.

1 Introduction

The global information infrastructure, both open networks as well as intranets, induces a rapid growth in opportunities to perform joint work efforts in locally distributed environments and within virtual organizations. This requires to support human interaction in cooperative working environments increasingly at a computing system level, in addition to conventional means, like personal interaction, phone or mail. Examples of applications were globalization of cooperative work is taking place are design applications, like cooperative document authoring, CAD/CASE or design for manufacturing, real-time groupware, like conferencing, shared whiteboards or joint editing, or business workflow management in administration and production.

Most of these activities are based on some common information bases, providing documents, design data or business data. Database systems managing this data thus have to support the typical modes of interaction of cooperating users with each other and with the computing system. These interactions involve aspects, like multi-user cooperation on shared documents, support for long duration activities, or interactive user control. At the same time basic consistency requirements need to be ensured. Ensuring consistency of data in multi-user environments is the classical problem of transaction management. However, in order to support cooperation, the classical paradigm of competition for resources needs to be replaced by the paradigm of semantically correct exchange and sharing of information.

The goal of the ESPRIT III project TRANSCOOP[1] is the development of a cooperative transaction model and a corresponding specification language that are applicable for a wide spectrum of cooperative applications. The TRANSCOOP specification language allows the specification of workflow-like cooperative scenarios. It provides a sound basis for verification, since it is based on a formal description technique for process algebras (LOTOS) [BB89]. In this paper we focus on the development of the TRANSCOOP cooperative transaction model COACT [RKT+95, WK96, KTW96b] which provides the basic transactional support to ensure consistent management of shared data in cooperative applications.

The research activities in TRANSCOOP are started from a former analysis of different cooperative application scenarios, namely cooperative authoring [TW95], Design for Manufacturing [VFSE95] and workflow applications [JLP+95] and from an analysis of existing approaches to support cooperative work, both of which are presented in Section 2. We describe the cooperative transaction model COACT which builds the core of the TRANSCOOP runtime environment for cooperative applications in Section 3 and compare its properties with other transaction models devised for the support of cooperative work. The

[1]This work is done in the ESPRIT III BRA project TRANSCOOP (EP8012) which is partially funded by the Commission of the European Communities. The partners in the TRANSCOOP project are GMD (Germany), University of Twente (The Netherlands), and VTT (Finland).

TRANSCOOP transaction model supports alternating periods of individual and joint work and allows to exchange and share information consistently. To support this flexibly, we take an operation-oriented view. We determine consistency of shared work results on the basis of the semantics of operations performed to obtain these results. The model has been implemented as an extension of the object-oriented database management system VODAK [GI95] developed at GMD-IPSI and is evaluated for the application scenario of cooperative document authoring (Section 4).

2 Requirements Analysis and Related Approaches

2.1 Cooperative Application Domains Investigated in TransCoop

In the TRANSCOOP project, we investigated three kinds of cooperative application scenarios with respect to requirements for a cooperative transaction model and its specification language, namely *Cooperative Hypermedia Document Authoring* (CDA), *Design for Manufacturing* (DfM), and *Workflow applications* [TV95, VT95].

Cooperative Hypermedia Document Authoring [TW95]. CDA is characterized by multiple authors interactively working on shared hypermedia documents. Hypermedia document authoring can be considered as a design problem solving process [HF86], mainly characterized by the decomposition into smaller subproblems and their solution by interacting activities. An important characteristic of these processes is that the documents to be produced can be described only vaguely in advance. Authoring activities require a high flexibility in choosing the next actions to end up with the aimed document.

Design for Manufacturing [VFSE95]. DfM can be seen as a variant of Concurrent Engineering. The scope of DfM is the engineering process of discrete complex industrial artifacts, usually separated into upstream processes (product design) and downstream processes (production realization, including engineering, planning, and manufacturing) [SRN93]. The essential part of DfM is the early involvement of specialists from downstream processes in the upstream design process. Thus, the strengthening of the design process by *overlapping* design phases requires extensive cooperation and coordination facilities. In comparison to cooperative authoring, there exists more knowledge of the processes in different phases as well as of the sequence of processing.

Workflow applications [JLP+95]. Workflows are used to define complicated business processes, e.g., to accomplish the production of goods or services. A workflow consists of a collection of tasks that are partially ordered by control and data flow dependencies. Tasks are the basic units of work that are processed by one responsible actor. Thus, workflows focus mainly on the coordination of activities.

The three scenarios can be classified as cooperative applications, emphasizing different aspects of the communication, collaboration and coordination properties of CSCW systems [EGR91]. We have identified the following common properties (for details we refer the reader to [TV95, VT95, TW95, VFSE95, JLP+95]):

- Multiple concurrent users are involved in multiple activities to satisfy a common goal or to produce a common product or artifact.
- Activities are processed interactively by humans and are usually of long duration.
- Cooperative work is characterized by alternating periods of individual and joint work.
- Exchange and sharing of persistent data between different users performing several activities is a basic feature.
- There is a need to ensure consistency both for the work of a single user as well as for the cooperative effort.
- Between activities there exist control flow dependencies that are derived from the application domain.
- It is likely that co-workers are geographically distributed and only partially connected, as mobility has growing importance.

The relevance of these properties varies for the three application domains. For example, in workflow scenarios, control flow dependencies are of higher importance than in the cooperative authoring domain. We can identify a spectrum, where the solutions become more fixed and prescribed and the problem solving process and termination conditions can thus be more deterministically described, starting from CDA over DfM to workflow applications.

2.2 Requirements for a Cooperative Transaction Model

A central objective of a cooperative transaction processing system is to provide a cooperative transaction model that supports the key requirements of a broad spectrum of cooperative applications. From the above analysis, we have identified the following requirements for a cooperative transaction model.

Relaxed atomicity. The rollback of the whole cooperative work process in case of a failure is generally not acceptable. It is required that a cooperative activity should be able to proceed (and eventually succeed) even if other parts of

the cooperative process fail. A failure within one user's activity should not imply the rollback of another user's work in their joint effort.

Retraction of decisions. To support the interactive user control of activities, a cooperative transaction model has to provide services that allow to retract decisions taken by the cooperating authors, for example by compensation. This allows, for instance, to explore several alternatives to solve a problem.

Relaxed isolation. The sharing of final, as well as of intermediate artifacts among co-workers is a prerequisite for most cooperative applications. Instead of isolation, a cooperative transaction model has to guarantee that no anomalies are introduced although tentative working results are exchanged. The model should offer appropriate primitives for the semantically correct exchange of information between co-workers.

Version support. To explore different solutions of the same problem, different co-worker should be able to work at the *same* time on the *same* data. To avoid interference from co-workers, the cooperative transaction model should be able to manage alternative versions of objects. On user's demand, it should be possible to exchange versions and to combine them into a commonly accepted version. Moreover, to support geographical distribution and mobility the model should be able to deal with multiple copies of data.

Execution constraints. To describe the pre-planned parts of cooperative scenarios and the decomposition of the overall work into smaller subactivities, a cooperative transaction model should allow the definition of execution constraints. Execution constraints should be able to describe the possible structure of a single user's activity as well as the overall cooperative work process. This includes the assignment of subactivities to co-workers and constraints on the occurrence of particular tasks and on their execution order within the overall cooperative process.

Of course, further requirements are imposed on cooperative systems. For example, to support the mutual awareness of co-workers, notification mechanisms are needed. Additional communication facilities like e-mail or audio may be required for direct negotiation of co-workers. Such services are not covered by a cooperative transaction model since it aims at synchronizing the cooperative access to persistent data. Nevertheless, a cooperative transaction processing system satisfying the above requirements can provide an application-independent nucleus to build cooperative systems on top of it.

2.3 Approaches to Support Cooperative Work

The issue of synchronization of multiple cooperating users is treated in several fields, including groupware, workflow systems, and advanced transaction models.

Groupware. Most groupware systems [EGR91] synchronize cooperative access to shared data in a more or less ad-hoc manner. Concurrency control in most cooperative hypertext systems is based on mechanisms like explicit user-controlled locking of objects, different lock modes, extended lock semantics, and notifications [WL93, GS87]. Some systems are using floor passing protocols [GS87] to synchronize concurrent operations on shared data, thereby limiting the availability of data. Some systems do not provide any concurrency control at all and rely on social protocols [EGR91]. Other approaches in the CSCW area (e.g., [EG89]) are only applicable to real-time groupware systems like shared whiteboards and synchronous group editors. Most of these systems are based on replication of data and use multicast protocols like ISIS [BC91, BSS91] for synchronization purposes. Real-time groupware systems do not address the issues of persistency of data and recovery to ensure fault-tolerant processing.

Workflow systems. Workflow management is gaining popularity, although the current generation of workflow management systems (WFMS) has several limitations [GHS95, JLP+95]. Most of these deficiencies result from the purely process-centric approach of WFMS neglecting a data-centric view. This results in lack of support for correctness and data consistency in case of concurrent workflow tasks and insufficient recovery mechanisms. Most of the commercial WFMS concentrate on relatively static workflows. Though some of the applications we have studied also have a rather statically defined structure at a higher-level, the order of executions of the constituent activities is determined dynamically at runtime by the participating users. Current WFMS do not have adequate modeling and execution support for this.

Advanced Transaction Models. Transaction models in general are guaranteeing fault tolerance and synchronize concurrent access to shared persistent data. To support cooperative applications, several advanced transaction models have been proposed in the recent years. For an overview of these models we refer the reader to [Elm92, Hsu93, Kai95].

A general approach to support cooperation is to divide the database in public and private areas [KW84, LP83]. Objects are copied from the public database by check-out into private areas. When the transaction is finished the modified objects are checked-in into the public database. Check-out models appear often in tandem with versions and configurations [Kat90]. CAD transactions [BKK85, KSUW85] enhance the basic check-out model by introducing a hierarchy of public, semi-public, and private databases. For CASE applications, several extensions of the basic checkout model have been developed by taking advantage of the opportunity of generic software consis-

tency checking [Hon88, KPS89], but they are not generally applicable to other domains.

The split/join transaction model [PKH88, KP92] supports the dynamic restructuring of ongoing transactions. A split-operation allows to split a running transaction into two new (serial or independent) transactions while a join-operations allows to incorporate two transactions into a new transaction. These mechanisms enable a cooperative behaviour by exchanging parts of transactions between concurrent users.

The participant transaction model [Kai95] defines each transaction as a participant in a specified domain. The domain represents the set of user transactions controlled by users collaborating on a common task. Participant transactions in the same domain need not to be serializable. Only transactions of different domains have to be serializable.

Group-oriented transaction approaches describe the overall working process as a transaction hierarchy consisting of group transactions. Individual user transactions form the leaves of the transaction hierarchy. Visibility between transactions is supported by extended lock schemes [KSUW85] or by following predefined access patterns that define the application-specific correctness criteria [FZ89, NRZ92].

ACTA [CR90, CR92] is a transaction meta-model that can be used to specify the types of dependencies between transactions. One of the ACTA building blocks is the delegation primitive [CR93]. A transaction can delegate the responsibility for committing or aborting parts of transactions to another transaction. This primitive is useful to control the (partial) visibility of results.

The area of transactional workflow approaches [RS95] usually comprises specification languages to express various execution constraints for a set of tasks. This can be done either by supporting a script language like in the ConTract model [WR92], by a declarative specification of the execution structure in terms of externally visible execution states [ARSS93], or by ECA rules [DHL90]. Cooperation is characterized in these models by passing results between workflow tasks in a predefined manner.

3 The TransCoop Model

As described in Section 2, cooperative work has several dimensions and the requirements for a cooperative system largely vary depending on the application. In TRANSCOOP it was our aim to satisfy these requirements by introducing a set of complementing mechanisms which can be used by the application designer to tailor the system for specific needs. In the following we will describe these building blocks of TRANSCOOP together with the aspects of cooperative work that are supported by them.

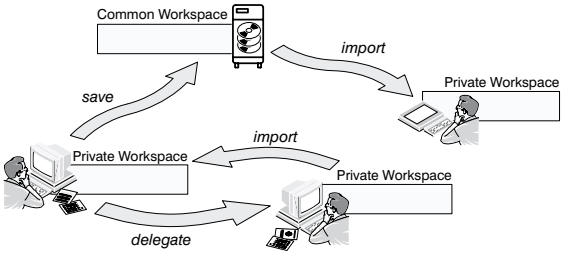

Figure 1: Workspaces and exchange facilities in COACT.

3.1 CoAct

The *Cooperative Activity Model* (COACT) [RKT+95, WK96, KTW96b] comprises the core part of the TRANSCOOP transaction model. It is based on the observation stated in Section 2 that cooperative work is characterized by alternating periods of individual and joint work [TV95, TW95, Kai95]. During individual work periods, users try out alternative problem solutions while co-workers may work simultaneously on the same subject. Access to and use of shared data should neither block other users nor should it affect co-workers unintendedly. During joint work, co-workers should be able to exchange information and to share final as well as intermediate results. Moreover, dynamically formed subgroups among co-workers should be possible.

Therefore, we assign in COACT a *private workspace* to every user who takes part in a cooperative activity. By default, the private workspaces of the co-workers are isolated from each other. Additionally, there exists a *common workspace* for each cooperative activity. This workspace is isolated from the private workspaces and is not assigned to a single actor in a cooperative activity. The common workspace contains the data items available when a cooperative activity is *started* and the results of the cooperative activity when *committed*. Figure 1 gives an overview of these constituents of the COACT model.

To achieve isolation of workspaces, we (conceptually) copy the data items initially contained in the common workspace to all private workspaces. From these copies the actors can create throughout the working process their private versions of data items that can be manipulated independently. Hence, modifications to data items done by different co-workers do not interfere. For each workspace, we keep a log of the modifications in a *workspace history*.

To model the correct interaction, i.e., cooperation of users involved in a cooperative activity, we introduce the concept of *cooperative activity types* (*CAT*). Each cooperative activity is specified by a single *CAT*. First, a *CAT* describes a set of constituent activity types that can be invoked by a user in his private workspace. We denote the

execution of one of these constituent activities by the term *activity instance*. Second, a *CAT* describes a set of merging rules that exploit the semantics of activity instances to guide the process of information exchange (*history merging*).

Information exchange in CoAct is based on the exchange of activity instances. This is an explicit act that is initiated by an actor through invoking one of CoAct's *exchange operations*. The exchange operations are generic meta operations of the CoAct model (like starting, aborting, or committing of a cooperative activity) and are all based on the paradigm of merging workspace histories. The CoAct model provides two different options for exchanging information:

1. Co-workers can directly exchange activity instances between their private workspaces by means of *import* and *delegate* operations. The import operation is used by a co-worker to incorporate activity instances executed in the scope of another workspace into his own workspace. The importing user is responsible for resolving conflicts that may occur during the merge. The delegate operation is used to pass on a set of activity instances to a co-worker who is then responsible for merging them into his own workspace.

2. Co-workers can exchange activity instances through the common workspace by means of *save* and *import* operations. An actor can use the save operation to incorporate activity instances of his workspace into the common workspace, thus, making (parts of) his results public to all co-workers. The user who invokes the save operation is responsible for the resolution of conflicts. Other co-workers can retrieve this information using the import operation described above.

The merge mechanism makes sure that only consistent parts of workspaces are exchanged. We identify such consistent units of work by examining the backward commutativity relation [Wei88, LMWF94] between activity instances contained in a workspace history. The incorporation of activity instances is then realized by the *re-execution* of the activity instances in the respective destination workspace. In this way, the effects of these activity instances are reflected in the private versions of the data items in the destination workspace. The semantic correctness of the exchange of activity instances is guaranteed by ensuring that the re-execution of an activity instance has an equivalent "view" on the history in the destination workspace as in the source workspace. Hence, the behavior of activity instances in terms of output parameter values is indistinguishable from the initial execution. We use the forward commutativity relation [Wei88, LMWF94] to check this. If the merge process cannot be performed without violating the semantical correctness, the merger offers different consistent sets of activity instances. The controlling user then selects one of the offered solutions. This may result in reverting previous decisions which is done by compensation in CoAct. Compensation in combination with backward commutativity can also be used to implement a flexible undo/redo mechanism on top of CoAct. To facilitate the merge process for the user, the selection task can alternatively be performed within a software module without requiring user interactions. In this case, certain merge policies can be specified providing different conflict resolution strategies.

If an activity instance has been successfully incorporated into another workspace, it is conceptually the same activity instance which is present in *more than one workspace*. The presence of identical activity instances in several workspaces enables us to establish a close cooperation between co-workers. The degree of cooperation is scalable depending on the exchange frequency.

Those parts of a cooperative activity that are reflected in the common workspace after its completion (commit) are considered as its final result. It is assumed that all users integrate their relevant contributions into the common workspace such that there is a single result of the cooperative activity. The merge mechanism and its properties are discussed in detail in [KTW96b, WK96].

3.2 Structuring the Work Process

The merge mechanism of CoAct enables the participants of a cooperative activity to work concurrently on the same data items without blocking each other while at the same time ensuring the consistency of the result. This is a basic requirement for all cooperative applications and may even be sufficient for creative work which is performed in a more or less ad-hoc fashion. Other cooperative applications like workflow require additional mechanisms which govern the work process. These have to ensure for example that certain activity instances are performed in a specific order or that certain tasks are performed at all. Such mechanism are available within the TransCoop approach at two levels.

With regard to single workspaces the application designer can specify a set of *execution rules*. These rules pose workflow-like restrictions on the order and existence of activity instances and define termination states. Execution rules are enforced for each private workspace and the common workspace separately. This allows the enforcement of execution rules even for those applications where the workspaces are not permanently connected, e.g., in mobile environments. We use a language-based specification mechanism for the execution rules. This means that the application designer specifies certain grammars

and the allowed sequences of activity instances are equal to the words in the generated language. To specify this language multiple grammars can be used whose combination results in the desired restrictions on sequences of activity instances. In contrast to [Ska89, NRZ92] our mechanism avoids dead ends caused by interdependencies of different grammars. For details see [KTW+96c, FEdB96].

To structure the overall work process, i.e., establish restrictions which have to be obeyed across all workspaces, TRANSCOOP provides a step mechanism which has been developed by the TRANSCOOP team at the University of Twente [FEdB96]. The step mechanism controls whether a user is allowed to execute an operation or not. This is performed by explicitly enabling the allowed operations. The set of operations which are controlled by the step mechanism include constituent activities, exchange operations as well as meta operations to terminate the step. In contrast to execution rules, there are no restrictions on the order or existence of operations *within* a step. It is left to the users whether they want to execute the enabled operations and in which order. The enabling mechanism is complemented by the possibility to specify user roles. Hence, the permission to execute an operation can be restricted to a certain role. Steps can be combined in various ways forming the organizational structure of the work process. The constructs to group steps range from sequential and repetitive execution of steps to nested and parallel steps. For further details see [FEdB96].

3.3 Related Models

In this section, we compare the COACT model to the extended transaction models presented in Section 2.3 with regard to applicability for cooperative application scenarios.

In the basic check-out model [KW84, LP83] objects which are checked out are reserved for exclusive access until a later check-in. This may result in blocking of objects over long periods of time which is obviously not suitable for cooperation. In COACT we avoid this behavior by providing each user with his own version in his private workspace. Approaches which combine the check-out mechanism with versioning go into a similar direction. In contrast to COACT, they suffer from the fact that their support for the required merging of different versions is only limited. Most merge mechanisms provided by versioning schemes force the user to choose among one of the generated versions and drop the others. Otherwise, the user has to perform the cumbersome task of integrating the different versions by hand. In contrast to this, COACT is able to merge the work of different users automatically by using the provided operation semantics. Only in cases where users have done irreconcilable work, manual interference is necessary. Even in this case, our merge algorithm can propose the user different solutions which minimize the amount of lost work. Another drawback of check-out approaches is that all objects and subobjects have to be checked-out explicitly whereas in COACT this is transparent for the users.

The split/join transaction model [PKH88, KP92] is in contrast to COACT restricted to serializable executions. This limits the possibility to exchange operations between transactions by means of the the provided split and join primitives. Since the model supports only single data copies, the access to these data items has to be synchronized, e.g., by a locking protocol which limits the availability of data. In addition to this, by splitting or joining a transaction the former owner of the operations loses control over these operations. In contrast to this, in COACT both users can continue their work based on the exchanged results. Another limitation is the lack of higher level operation semantics. The split/join transaction model considers only read and write operations. Hence, it can be extended by using our merge criteria to be more powerful and flexible in splitting and joining, e.g., merging ongoing transactions by utilizing the semantics of operations and thus enabling a closer cooperation between transactions.

The participant transaction model [Kai95] is based on read/write actions and no concurrency control applies to a domain. This can lead to inconsistencies of data accessed in a domain (e,g., updates based on data no longer valid) and unintuitive behavior of the system from the user's viewpoint. In COACT we guarantee that the view of each activity instance remains valid by exploiting its semantics. The system detects if the exchange of activity instances would introduce inconsistencies and proposes alternative sets of activity instances that guarantee consistency of data.

Group-oriented transaction approaches [FZ89, NRZ92] structure the overall working process by means of a transaction hierarchy and application-specific correctness criteria. The usability of these models is restricted because significant parts of the cooperative application have to be pre-specified in order to describe whether a particular non-serializable execution is correct or not.

Delegation in ACTA [CR93, CR90, CR92] means that the responsibility for committing or aborting actions can be delegated from one transaction to another transaction. In our model both the delegating user as well as the delegatee keep separate responsibilities for the delegated actions. This is possible since our model assumes isolated workspaces where copies of objects reside.

Cooperation in transactional workflow approaches [RS95, ARSS93, WR92] is characterized by passing results between workflow tasks in a predefined manner. There is no opportunity for flexibly passing results back and forth between co-workers. This is needed in non-workflow scenarios, e.g., cooperative authoring.

4 Implementation Aspects and TransCoop Demonstrator

4.1 The TransCoop System

To prove the applicability of the results, one important goal of the project was to build a demonstrator system realizing the TRANSCOOP concepts. The TRANSCOOP reference architecture [dBLP+95] separates between the specification environment providing means for the specification and verification of cooperative scenarios and the runtime environment offering support for the execution of cooperative scenarios. The TRANSCOOP runtime system presented here is an instantiation of the TRANSCOOP reference architecture and it is implemented as an extension of the object-oriented DBMS VODAK developed at GMD-IPSI [GI95], i.e., all cooperation facilities are supported as DBMS services.

In the remainder of this section, we report on the experiences we made during the system design and give an impression how the new functionality is presented to the users of the selected demonstrator application SEPIA (cooperative authoring of hypermedia documents) [SHH+92].

The design of the TRANSCOOP runtime environment required an extension of the traditional centralized OODBMS architecture, as used for VODAK, to meet the architectural and conceptual requirements posed by the TRANSCOOP cooperative transaction manager [KTW96a]. In detail the following problems were considered and solved:

- **Workspaces.** The aimed workspace functionality requires the maintenance of multiple private versions of an object instead of a single shared version. Instead of implementing a special purpose version management extending the object manager of the DBMS, we decided to revise the DBMS memory architecture. The object management initially based on a single centralized shared object buffer was replaced by multiple private object buffers which can be distributed across the network.

- **History maintenance.** The merge algorithm on which the TRANSCOOP exchange facilities are based uses workspace histories of different workspaces. To allow an efficient access to *all* workspace histories of an ongoing scenario, the histories are centrally stored in the TRANSCOOP system.

  When an operation is invoked manipulating the local workspace state, the corresponding history entry is stored persistently in a dedicated Status Database which maintains the current state of the scenario execution. This design decision requires permanent availability of the database server, but, in case of a failure on the client, it allows the recovery of the clients workspace state. An alternative design with local storage of histories would increase the clients independence of the database server, e.g., for the purpose of mobility.

- **Specification.** For the execution of cooperative scenarios, VODAK database schemas are enriched by a cooperative scenario specification. In particular, this captures (1) information about which schema methods are available at the application interface, (2) which orders of operation invocations are allowed in a particular workspace (execution rules), (3) the structure of the overall scenario execution (steps), (4) information how certain method executions can be compensated, and (5) predicates describing how to evaluate backward and forward commutativity at runtime. All this additional information is stored in the enhanced VODAK data dictionary at runtime.

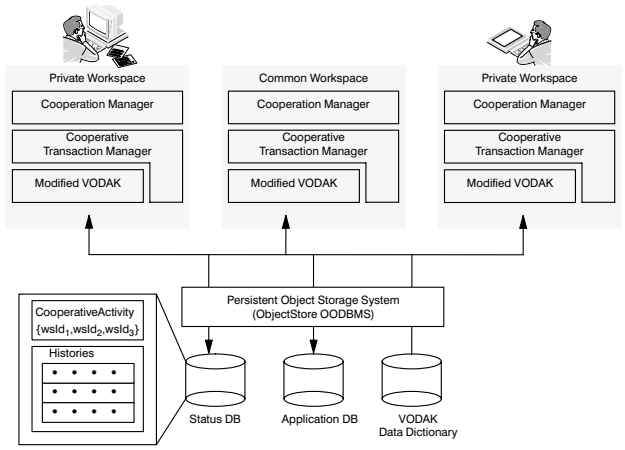

Figure 2: Runtime architecture of the TRANSCOOP system.

Figure 2 illustrates the runtime architecture of the TRANSCOOP system. The workspaces are realized by a modified VODAK system which provides private object buffers, the cooperative transaction manager containing the merge functionality, and the cooperation manager governing the execution of steps.

An important part of the runtime system design is the Status-DB which maintains histories of the participating users and the runtime structure of the step specification and which provides general administrative information among different workspaces, e.g., the current participants of the scenario or the operations enabled by the current steps.

The cooperative transaction manager ensures the correctness of histories by enforcing the execution rules, of-

fers functions to preview the progress of co-workers by means of querying their histories in the Status-DB, and provides operations for both the calculation of possible merge alternatives and the enforcement of a specific merge.

Users can join an ongoing scenario execution as well as leave the scenario before it is finished. When the current step allows the scenario to terminate, the results in the common workspace represent the commonly agreed upon result. The final history is then applied to the Application-DB.

4.2 The TransCoop Demonstrator Application

The selected application for the TRANSCOOP demonstrator system [dBLP+96] is the cooperative hypermedia authoring system SEPIA [SHH+92], developed at GMD-IPSI.

Because we treat the authoring process from a database point of view, we assume a scenario in which multiple authors are manipulating a collection of shared hypermedia documents that are stored in a database management system. The SEPIA hierarchical hypertext document structure and the corresponding operations for the creation and manipulation of SEPIA document structures are modeled explicitely by means of the VODAK Manipulation Language (VML) in the database schema [WA95, BWAH96].

The TRANSCOOP demonstrator system provides information about the current participants and their work progress to establish group awareness. For example, when a delegation is performed, the delegatee is notified in order to control the integration of the respective piece of work in its workspace. Further notifications are caused if co-authors join or leave the cooperative activity.

The exchange facilities of the COACT model support an ad-hoc working style of groups with no pre-coordination. Conflicts that may occur between the individually carried out problem solutions are semi-automatically resolved within the merge procedure. The cooperative transaction manager offers different consistent alternatives in case of a conflict. The controlling user then selects one of the offered solutions using the graphical user interface.

One challenge building the demonstrator is the design of an appropriate graphical user interface to *import*, *delegate*, and *save* working results. The merge functionality frees the users from concerns of explicitely ensuring consistency when exchanging results. However, the required selection of operations from a history to perform an information exchange is a rather system-oriented view. For a user it is more common to refer to some part of a document in terms of data instead of logged operations. To strive for a human-oriented end-user interface, the system allows the selection of data in terms of document structures visualized, from which the corresponding history entries are then determined by the cooperative transaction manager.

5 Conclusions

Todays workflow systems do not support scenarios with spontaneous cooperation that cannot be prescribed in a specification. In contrast, current groupware approaches support ad-hoc working style but give, from a database point of view, no satisfying execution guarantees. TRANSCOOP approaches both criteria: the growing need for models supporting highly dynamic forms of cooperative work without giving up transactional correctness criteria.

The nucleus of the exchange facilities in the CoAct model is the history merging approach. The flexibility of the approach is mainly achieved by its ability to determine dynamically consistent units of work in terms of performed operations and its consideration of operation semantics for resolving conflicts.

A question that requires further attention is the specification of commutativity relations. Specification tools that compute commutativity information automatically from a formal specification of operations will improve practical applicability of the approach. Commutativity analysis tools that are investigated in the framework TRANSCOOP are a first step in this direction.

The approach fits various application areas and provides directions for solving open research problems in related fields like mobile wireless computing or versioning.

Strategies for replication control and transaction processing are not mature in the area of mobile wireless computing. The possibility of disconnection of mobile units from the stationary host and the limited bandwidth of networks make it necessary to develop new models. As already noted in Section 4, the merge component can be integrated in a system architecture where no durable connection to the database server is required. Possible conflicts occuring in disconnected mode can be resolved by merging results. The fact that transfer of operation logs instead of data is used, which requires far less network bandwidth, makes the model additionally attractive for mobile wireless computing.

Todays version models provide very limited support for merging concurrently derived versions. In contrast to tools forcing users to manually combine attributes from different versions, our approach allows the automatic computation of all correct alternatives. Thus the user only needs to select the desired alternative.

Acknowledgments

The authors would like to thank all members of the TRANSCOOP teams in Enschede and Espoo for many inspiring discussions on the topic.

References

[ARSS93] P. C. Attie, M. Rusinkiewicz, A. Sheth, and M. P. Singh. Specifying and enforcing intertask dependencies. In *Proc. of the 19th Int. Conference on Very Large Databases*, pages 134–145, Dublin, Ireland, August 1993.

[BB89] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1989.

[BC91] K.P. Birman and R. Cooper. The ISIS project: Real experience with a fault-tolerant programming system. *ACM Operating System Review*, 21(2):103–107, 1991.

[BKK85] F. Bancilhon, W. Kim, and H. Korth. A model of CAD transactions. In *Proc. of the 11th Int. Conference on Very Large Databases*, pages 25–33, Stockholm, Sweden, August 1985.

[BSS91] K.P. Birman, A. Schiper, and P. Stephenson. Leightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, 1991.

[BWAH96] A. Bapat, J. Wäsch, K. Aberer, and J.M. Haake. HyperStorM: An extensible object-oriented hypermedia engine. In *Proceedings of the Seventh ACM Conference on Hypertext (HYPERTEXT'96)*, pages 203–214, March 16-20 1996. Washington, D.C.

[CR90] P. K. Chrysanthis and K. Ramamritham. ACTA: A framework for specifying and reasoning about transaction structure and behavior. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 194–203, Atlantic City, NJ, USA, May 1990.

[CR92] P. K. Chrysanthis and K. Ramamritham. ACTA: The saga continues. In Elmagarmid [Elm92], chapter 10, pages 349–397.

[CR93] P. K. Chrysanthis and K. Ramamritham. Delegation in ACTA to control sharing in extended transactions. *IEEE Data Engineering Bulletin*, 16(2):16–19, June 1993.

[dBLP+95] R. de By, A. Lehtola, O. Pihlajamaa, J. Veijalainen, and J. Wäsch. A reference architecture for cooperative transaction processing systems. VTT Research Notes 1694, VTT Technical Research Centre of Finland, 1995.

[dBLP+96] R. de By, A. Lehtola, O. Pihlajamaa, J. Veijalainen, and J. Wäsch. Deliverable III.2: Specification of the TransCoop demonstrator system. Report TC/REP/VTT/D3-2/960425, Esprit Project No. 8012, 1996.

[DHL90] U. Dayal, M. Hsu, and R. Ladin. Organizing long–running activities with triggers and transactions. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 204–214, Atlantic City, NJ, USA, May 1990.

[EG89] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 399–407. MCC, Austin, Texas, May 1989. Portland, Oregon.

[EGR91] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):38–58, January 1991.

[Elm92] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. ACM Press. Morgan Kaufmann Publishers, Inc., 1992.

[FEdB96] F. J. Faase, S. J. Even, and R. A. de By. Deliverable IV.3: An introduction to CoCoA. Report TC/REP/UT/D4-3/033, Esprit Project No. 8012, 1996.

[FZ89] M. F. Fernandez and S. B. Zdonik. Transaction groups: A model for controlling cooperative transactions. In *Proc. of the Int. Workshop on Persistent Object Systems*, pages 341–350, January 1989. Newcastle, New South Wales.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.

[GI95] GMD-IPSI. VODAK V4.0 User Manual. Arbeitspapiere der GMD 910, Technical Report, GMD, April 1995.

[GS87] I. Greif and S. Sarin. Data sharing in group work. *ACM Transactions on Office Information Systems*, 5(2):187–211, April 1987.

[HF86] J.R. Hayes and L. Flowers. Writing research and the writer. *American Psychologist*, 41(10):1106–1113, 1986.

[Hon88] M. Honda. Support for parallel development in the Sun network software environment. In *Proc. of the second Int. Workshop on Computer-Aided Software Engineering*, pages 5–5 – 5–7, Cambridge, Massachusetts, USA, July 1988.

[Hsu93] M. Hsu, editor. *Special Issue on Workflow and Extended Transaction Systems, Data Engineering Bulletin*, volume 16, 1993.

[JLP+95] J. Juopperi, A. Lehtola, O. Pihlajamaa, A. Sladek, and J. Veijalainen. Usability of some workflow products in an inter-organizational setting. In *Proc. of IFIP WG8.1 Working Conference on Information Systems for Decentralized Organizations*, Trondheim, Norway, August 1995.

[Kai95] G. E. Kaiser. Cooperative transactions for multiuser environments. In Kim [Kim95], chapter 20, pages 409–433.

[Kat90] R. H. Katz. Towards a unified framework for version modelling in engineering databases. *ACM Computing Surveys*, 22(4), 1990.

[Kim95] W. Kim, editor. *Modern Database Systems: The Object Model, Interoperability, and beyond*. Addison-Wesley Publishing Company, 1995.

[KP92] G. E. Kaiser and C. Pu. Dynamic restructuring of transactions. In Elmagarmid [Elm92], chapter 8, pages 265–295.

[KPS89] G. E. Kaiser, D. E. Perry, and W. M. Schell. Infuse: Fusing integration test management with change management. In *Proc. of the 13th IEEE Computer Software and Applications Conference*, pages 552–558, Orlando, Florida, USA, September 1989.

[KSUW85] P. Klahold, G. Schlageter, R. Unland, and W. Wilkes. A transaction model supporting complex applications in integrated information systems. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 388–401, Austin, Texas, USA, May 1985.

[KTW96a] J. Klingemann, T. Tesch, and J. Wäsch. Deliverable V.3: Design of the TransCoop cooperative transaction manager. Report TC/REP/GMD/D5-3/512, Esprit Project No. 8012, 1996.

[KTW96b] J. Klingemann, T. Tesch, and J. Wäsch. Semantics-based transaction management for cooperative applications. In *Proc. of the Int. Workshop on Advanced Transaction Models and Architectures*, pages 234–252, Goa, India, August 31 – September 2 1996.

[KTW+96c] J. Klingemann, T. Tesch, J. Wäsch, J. Puustjärvi, and J. Veijalainen. Deliverable V.2: Definition of the TransCoop cooperative transaction model. Report TC/REP/GMD/D5-2/511, Esprit Project No. 8012, 1996.

[KW84] R. Katz and S. Weiss. Design transaction management. In *Proceedings of the 19th Design Automation Conference*, June 1984.

[LMWF94] N. Lynch, M. Merrit, W. Weihl, and A. Fekete. *Atomic Transactions*. Morgan Kaufmann Publishers, Inc., 1994.

[LP83] R. Lorie and W. Plouffe. Complex objects and their use in design transactions. In *Proceedings on Database for Engineering Applications*, pages 115–121. ACM, May 1983.

[NRZ92] M. H. Nodine, S. Ramaswamy, and S. B. Zdonik. A cooperative transaction model for design databases. In Elmagarmid [Elm92], chapter 3, pages 53–85.

[PKH88] C. Pu, G. E. Kaiser, and N. Hutchinson. Split–transactions for open–ended activities. In *Proc. of the 14th Int. Conference on Very Large Databases*, pages 26–37, Los Angeles, California, USA, August 1988.

[RKT+95] M. Rusinkiewicz, W. Klas, T. Tesch, J. Wäsch, and P. Muth. Towards a cooperative transaction model: The cooperative activity model. In *Proc. of the 21st Int. Conference on Very Large Databases*, pages 194–205, September 1995. Zurich, Switzerland.

[RS95] M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In Kim [Kim95], chapter 29, pages 592–620.

[SHH+92] N. Streitz, J. Haake, J. Hannemann, A. Lemke, W. Schuler, H. Schütt, and M. Thüring. SEPIA: A cooperative hypermedia authoring environment. In *Proc. of the fourth ACM Conference on Hypertext*, pages 11–22, 1992. Milano, Italy, Nov. 30 – Dec. 4.

[Ska89] A.H. Skarra. Concurrency control for cooperating transactions in an object-oriented database. *ACM SIGPLAN Notices*, 24(4):145–147, April 1989.

[SRN93] A. Storr, U. Rembold, and B.O. Nnaji. *Computer Integrated Manufacturing and Engineering*. Addison-Wesley Publishing Company, 1993.

[TV95] T. Tesch and P. Verkoulen. Deliverable II.2: Requirements for the TransCoop transaction model. Report TC/REP/GMD/D2-2/207, Esprit Project No. 8012, 1995.

[TW95] T. Tesch and J. Wäsch. Transaction support for cooperative hypermedia document authoring: A study on requirements. In *Proc. of 8th ERCIM Database Research Group Workshop on Database Issues and Infrastructure in Cooperative Information Systems*, pages 31–42, Trondheim, Norway, August 1995.

[VFSE95] P. A. C. Verkoulen, F. J. Faase, A. W. Selders, and P. J. J. Oude Egberink. Requirements for an advanced database transaction model to support design for manufacturing. In *Proceedings of the Flexible Automation and Intelligent Manufacturing Conference*, pages 102–113, Stuttgart, Germany, June 1995.

[VT95] P. Verkoulen and T. Tesch. Deliverable II.1: Requirements for the TransCoop specification language. Report TC/REP/UT/D2-1/014, Esprit Project No. 8012, 1995.

[WA95] J. Wäsch and K. Aberer. Flexible design and efficient implementation of a hypermedia document database system by tailoring semantic relationships. In *Proc. of the sixth IFIP Conference on Database Semantics*, May 30 – June 2 1995. To appear in R. Meersman and L. Mark, editors. Database Applications Semantics. Chapman & Hall, 1996.

[Wei88] W. E. Weihl. Commutativity-based concurrency control for abstract data types. *IEEE Transactions on Computers*, 37(12):1488–1505, 1988.

[WK96] J. Wäsch and W. Klas. History merging as a mechanism for concurrency control in cooperative environments. In *Proceedings of RIDE-Interoperability of Nontraditional Database Systems*, pages 76–85, New Orleans, USA, February 1996.

[WL93] U. K. Wiil and J. J. Leggett. Concurrency control in collaborative hypertext systems. In *Proc. of the fifth ACM Conference on Hypertext*, pages 14–18, November 1993. Seattle, Washington.

[WR92] H. Wächter and A. Reuter. The ConTract model. In Elmagarmid [Elm92], chapter 7, pages 219–264.