ACE

a collaborative editor

Report Evaluation Algorithms

Berne University of Applied Sciences
School of Engineering and Information Technology

Date:	21.04.2005
Version:	0.3
Projectteam:	Mark Bigler (biglm2@hta-bi.bfh.ch) Simon Räss (rasss@hta-bi.bfh.ch) Lukas Zbinden (zbinl@hta-bi.bfh.ch)
Receivers:	Jean-Paul Dubois (doj@hta-bi.bfh.ch) Claude Fuhrer (frc@hta-bi.bfh.ch)
Location:	Subversion Repository

Contents

List of Tables

List of Figures

Version Control

Version	Date	Author	Remarks
0.1	12.04.2005	rasss	first version
0.2	20.04.2005	zbinl	algorithms SOCT2, TIBOT, SDT, Li 04
0.3	22.04.2005	project team	revision
0.4	25.04.2005	project team	revision after meeting
0.5	26.04.2005	project team	review, completion
1.0	27.04.2005	project team	release

Table 1: Version Control

1 Introduction

Real-time cooperative editing systems allow multiple users to view and edit the same document at the same time from multiple sites connected by communication networks. Consistency maintenance is one of the most significant challenges in designing and implementing real-time cooperative editing systems.

1.1 Requirements

The following requirements have been identified for such systems.

Real-time: The response to local user actions must be quick, ideally as quick as a single-user editor, and the latency for reflecting remote user actions is low (determined by external communication latency only).

Distributed: Cooperating users may reside on different machines connected by communication networks with nondeterministic latency.

Unconstrained: Multiple users are allowed to concurrently and independently edit any part of the document at any time, in order to facilitate free and natural information flow among multiple users.

1.2 Preliminaries

In this section, some basic concepts and terminologies are introduced. Following Lamport[12], we define a causal (partial) ordering relation on operations in terms of their generation and execution sequences as follows.

Definition 1 *Causal ordering relation* $\rightarrow$

Given two operations O_a and O_b generated at sites i and j, then $O_a \rightarrow O_b$, iff:

1. $i = j$ and the generation of O_a happened before the generation of O_b
2. or $i \neq$j and the execution of O_a at site j happened before the generation of O_b
3. or there exists an operation O_x such that $O_a \rightarrow O_x$ and $O_x \rightarrow O_b$

Note that the causal ordering relation is a *partial ordering*.

Definition 2 *Dependent and independent operations*

Given any two operations O_a and O_b:

1. O_b is *dependent* on O_a iff $O_a \rightarrow O_b$
2. O_a and O_b are *independent* (or *concurrent*), expressed as $O_a \| O_b$ iff neither $O_a \rightarrow O_b$ nor $O_b \rightarrow O_a$

Intuitively we can say that two operations are dependant if there exists a path from the generation of one message to the generation of another message. So for example in figure 1 operation O_1 and O_3 are dependent, that is $O_1 \rightarrow O_3$. Operation O_1 and O_2 are *independent*.

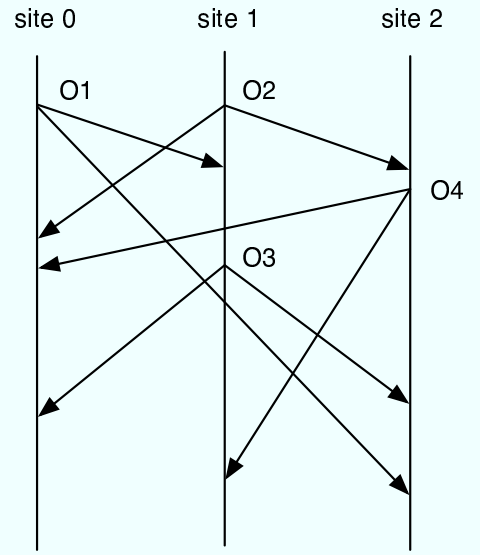

Figure 1: A scenarion of a real-time cooperative editing session

1.3 A shared document model

Consider n sites, where each site has a copy of the shared document. The shared document model we take is a *text document* modeled a by sequence of characters, indexed from 0 up to the number of characters in the document. It is assumed that the document state (the text) can only be modified by executing the following two primitive editing operations: (i) $Insert(p, c)$ which inserts the character c at position p; (ii) $Delete(p)$ which deletes the character at position p.
It is important to note that the above text document model is only an abstract view of many document models based on a linear structure. For instance the character parameter may be regarded as a string of charcters, a line, a block of lines, an ordered XML node, etc.

1.4 Three inconsistency problems

To illustrate the challenges researchers are facing, consider a scenario in a cooperative editing system with three cooperating sites, as shown in figure 1. Suppose that an operation is executed on the local replica of the shared document immediately after its generation, then broadcast to remote sites and executed there in its *original form* upon its arrival. Three different inconsistency problems have been identified by Sun et. al[28].

Divergence: Operations may arrive and be executed at different sites in different orders, resulting in different final results. As shown in figure 1, the four operations in this scenario are executed in the following orders: O_1, O_2, O_4 and O_3 at site 0; O_2, O_1, O_3 and O_4 at site 1; and O_2, O_4, O_3 and O_1 at site 2. Unless operations are commutative, which is generally not the case, final editing results will diverge. The divergence problem can be solved by any serialization protocol, which ensures the final result is the same as if all operations were executed in the same total order at all sites.

Causality violation: Due to the nondeterministic communication latency, operations may arrive and be executed out of their natural cause-effect order. As shown in figure 1, operation O_3 is generated after the arrival of O_1 at site 1, the editing effect of O_1 on the shared document has been seen by the user 1 at the time O_3 is generated. Therefore, O_3 may be *dependent* on O_1. However, since O_3 arrives and is executed before O_1 at site 2, confusion may occure to the system as well as to the user at site 2. For example, if O_1 is to insert a string into a shared document, and O_3 is to delete some characters in the string inserted by O_1, then the execution of O_3 before O_1 at site 2 will result in O_3 referring to a nonexistent context.

Intention violation: Due to concurrent generation of operations, the *actual effect* of an operation at the time of its execution may be different from the *intended effect* of this operation at the time of its generation. As shown in figure 1, operation O_1 is generated at site 0 without any knowledge of O_2 generated at site 1, so O_1 is *independent* of O_2, and vice versa. At site 0, O_2 is executed on a document state which has been changed by the preceding execution of O_1. Therefore, the subsequent execution of O_2 may refer to an incorrect position in the new document state, resulting in an editing effect which is different from the *intention* of O_2.
For example, assume the shared document initially contains the following sequence of characters: "ABCDE". Suppose $O_1 = Insert[''12'', 1]$, which intends to insert string "12" at position 1, i.e. between "A" and "BCDE"; and $O_2 = Delete[2, 2]$, which intends to delete the two characters starting from position 2, i.e. "CD". After the execution of these two operations, the *intention-preserved* result (at all sites) should be: "A12BE". However, the actual result at site 0, obtained by executing O_1 followed by executing O_2, would be: "A1CDE", which apparently violates the intention of O_1 since the character "2", which was intented to be inserted, is missing in the final text, and violates the intention of O_2 since characters "CD", which were intended to be deleted, are still present in the final text.
Even if a serialization-based protocol was used to ensure that all sites execute O_1 and O_2 in the same order to get an identical result "A1CDE", but this identical result is still inconsistent with the intentions of both O_1 and O_2.

The three inconsistency problems are independent in the sense that the occurence of one or two of them does not always result in the others. Particularly, intention violation is an incosistency problem of a different nature from the divergence problem. The essential difference between divergence and intention violation is that the former can always be resolved by a serialization protocol, but the latter cannot be fixed by any serialization protocol if operations were always executed in their originial forms.

1.5 A consistency model

A cooperative editing system is said to be consistent if it always maintains the following properties:

Convergence: when the same set of operations have been executed at all sites, all copies of the shared document are identical.

Causality-preservation: for any pair of operations O_a and O_b, if $O_a \rightarrow O_b$, then O_a is executed before O_b at all sites.

Intention-preservation: for any operation O, the effects of executing O at all sites are the same as the intention of O, and the effect of executing O does not change the effects of indepentent operations.

In essence, the *convergence* property ensures the consistency of the final results *at the end* of a cooperative editing session; the *causality-preservation* property ensures the consistency of the execution orders of dependent operations *during* a cooperative editing session; and the *intention-preservation* property ensures that executing an opertion at remote sites achieves the same effect as executing this operation at the local site at the time of its generation, and the execution effects of independent operations do not interfere with each other.
The consistency model imposes an execution order constraint on dependent operations only. The execution order of independent operations is left open as long as the convergence and intention-preservation properties are maintained. The consistency model effectively specifies, what assurance a cooperative editing system gives to its users and, what properties the underlying consistency maintenance mechanism must support.

1.6 Operational Transformation

Ellis and Gibbs [6] proposed a new kind of algorithm for consistency control, called *Operational Transformation* (OT). This kind of algorithm transforms operations to include/exclude the effects of other operations. Intuitively, transformation shifts the position parameter of an operation before execution to incorporate the effects of previously executed operations that it was not *aware* of (or that are concurrent) at the time of generation. Operational transformation helps to solve the problem of intention violation.
In general there are two different types of operational transformation, inclusion transformation (IT) and exclusion transformation (ET). All OT algorithms use inclusion transformation, whereas exclusion transformation is not needed by some algorithms.
A transformation function has to be defined for every combination of operations. So for a text editor with the primitive operations *insert* and *delete*, there would be a total of four transformation functions for IT and another four for ET. That is, given a transformation function T, T(*insert, insert*), T(*insert, delete*), T(*delete, insert*) and T(*delete, delete*) must be defined.

1.6.1 Definitions

Conceptually, an operation O is associated with a *context*, denoted as CT_O, which is the list of operations that need to be executed to bring the document from its initial state to the state on which O is defined (*definition context*). The significance of context is that the effect of an operation can be correctly interpreted only in its own context. If the current context (called *execution context*) is different from the definition context of an operation, the operation has to be transformed so that it can be executed in the current context.

Definition 3 *Context equivalent relation* $\sqcup$

Given two operations O_1 and O_2, associated with contexts CT_{O_1} and CT_{O_2} respectively, O_1 and O_2 are *context-equivalent* iff $CT_{O_1} = CT_{O_2}$. Apparently, the context equivalent relation $\sqcup$ is transitive.

Definition 4 *Context preceding relation* $\mapsto$

Given two operations O_1 and O_2 associated with contexts CT_{O_1} and CT_{O_2} respectively, O_1 is *context preceding* O_2 iff $CT_{O_2} = CT_{O_1} + [O_1]$. Note that the contex preceding relation $\mapsto$ is not transitive by definition.

1.6.2 Inclusion Transformation

Inclusion Transformation (IT) transforms an operation O_1 against another operation O_2 in such a way that the impact of O_2 is effectively included.

Specification 1 $IT(O_a, O_b) : O'_a$

1. Precondition for input parameters: $O_a \sqcup O_b$
2. Postcondition for output: $O_b \mapsto O'_a$ where O'_a's execution effect in the context of $CT_{O'_a}$ is the same as O_a's execution effect in the context of CT_{O_a}.

Most important, it was recognized that the correctness of IT relies on the condition that both O_1 and O_2 are defined on the same document state so that their parameters are comparable and can be used to derive a proper adjustment to O_2, i.e. $O_1 \sqcup O_2$.

1.6.3 Exclusion Transformation

Exlusion Transformation (ET) transforms an operation O_1 against another operation O_2 in such a way that the impact of O_2 is effectively excluded from O_1.

Specification 2 $ET(O_a, O_b) : O'_a$

1. Precondition for input parameters: $O_b \mapsto O_a$
2. Postcondition for output: $O_b \sqcup O'_a$ where O'_a's execution effect in the context of $CT_{O'_a}$ is the same as O_a's execution effect in the context of CT_{O_a}.

Both transformation functions must meet the *reversibility* requirement as defined next.

1.6.4 Reversibility

Definition 5 *Reversibility Requirement*

Given two operations O_1 and O_2.

1. if $O_1 \sqcup O_2$ and $O'_1 = IT(O_1, O_2)$, then it must be that $O_1 = ET(O'_1, O_2)$
2. if $O_2 \mapsto O_1$ and $O'_1 = ET(O_1, O_2)$, then it must be that $O_1 = IT(O'_1, O_2)$

Achieving reversibility is not a trivial task. This is because IT/ET functions may loose some information, so reversing the effect of a transformation may not be possible.

1.7 Transformation Properties

It was shown in [19] that transformation functions must satisfy two conditions, called $TP1$ and $TP2$. These transformation properties are sufficient and necessary for OT algorithms to guarantee convergence along arbitrary transformation paths.

Transformation Property 1: The transformation property 1 ensures that the effect of executing O_1 followed by the transformed request O_2 is the same as executing request O_2 followed by the transformed request O_1.

Definition 6 *Transformation Property 1:* $O_1 O'_2 \equiv O_2 O'_1$

Transformation Property 2: Transformation property 1 is a necessary and sufficient condition to ensure that the groupware system with two users is correct. When there are more than two users, the situation is more complex. An operation can be transformed along different, albeit equivalent paths, not necessarily yielding the same result. In the simplest case, an operation can be transformed along the two paths of a simple transformation step. Operation O_1 may be transformed first with respect to O_2 and then to O'_3 yielding $IT(IT(O_1, O_2), O'_3)$, or it may be transformed first with respect to O_3 and then to O'_2 yielding $IT(IT(O_1, O_3), O'_2)$. Note that different sites might choose different paths for O_1 to be transformed. So we have to make sure that both paths lead to the same resulting operation:

Definition 7 *Transformation Property 2:* $IT(IT(O_1, O_2), O'_3) = IT(IT(O_1, O_3), O'_2)$

1.8 Groupware Architecture Analysis

We consider the architecture of a groupware application from two different point of views. On the one hand, the focus lies on the document replication and on the other, we consider the type of group communication between the participating sites.

1.8.1 Document replication

Centralized architecture In a centralized architecture all data, i.e. the document, resides on a central machine. Client processes at each site are only responsible for passing requests to the central program and for displaying any output sent to them from the central program. The advantage of a centralized scheme is that synchronization is easy. Document state information is consistent since it is located in one place, and events are handled at clients in the same order because they are serialized by the server. Its main drawback is latency, as the message corresponding to any action must pass from the client to the server and back again before response to the action is shown.

Replicated architecture In a replicated architecture the document is replicated at all participating sites. Client processes at each site (replicas) must coordinate explicitly both local and remote actions, synchronizing all copies of the document. Replicas need only exchange critical state information to keep their copy of the document current. While remote activities may still be delayed, local activities can be processed immediately. Processing bottlenecks are less likely, because each replica is responsible for drawing only the local view. The most

significant cost of replication is increased complexity as issues of distributed systems like conflict management, concurrency control, etc. must be handled.
The *real-time* requirement has led most researchers to adopt a replicated architecture.

1.8.2 Group Communication

We consider two types of group communication architectures: unicast and multicast:

Unicast communication Unicast communication is a two way communication where client processes at each site communicate bidirectionally with a centralized server. The server forwards information from one client to all other clients.

Multicast communication Mulitcast communication is an n way communication where client processes at each site communicate with all the participating sites directly. It has an enormous growth in terms of the number of communication paths. They grow at the rate of $n(n-1)/2$, where n is the number of clients in the system. As compared to this, systems that use unicast communication have a linear growth.

2 History

Ellis and Gibbs [6] were the first to propose an *Operational Transformation* algorithm in 1989. The algorithm is called *dOPT* and is implemented in the *Grove* system. Soon however a flaw was discovered in the original *dOPT* algorithm (by Cormack[29]). The scenario where *dOPT* failed is called the *dOPT* puzzle. Ressel[19] proposed a new algorithm *adOPTed* in 1996 that solved the original *dOPT* puzzle. Sun et. al[28] proposed another algorithm called *GOT* that similarly to *adOPTed* solved the *dOPT* puzzle. Sun et. al[27] developed some transformation functions for string-wise operations.
Later research groups[1][3] proved the transformation functions of both Ressel[19] and Sun[28] to fail to hold TP2 in certain situations. They proposed new transformation functions they developed using a theorem prover.
Proving the transformation property 1 (TP1) seems to be rather straightforward. However, proving that a given transformation function holds TP2 appears to be difficult. There are over 100 cases that have to be analyzed (according to Imine et. al[3]). Imine et. al showed that many proposed transformation functions do not hold TP2.
Recently, two different ways have been taken to deal with the TP2 problem. One kind of algorithms tries to avoid the need to comply with TP2 altogether (GOT [28], SOCT3/4 [22], TIBOT[16] and NICE [23]). Other research groups [17] [3] try to correct the problems in the original transformation functions of GOTO [27], adOPTed [19] and SDT [15].

3 Algorithms

In this section we give an overview of the *Operational Transformation* algorithms we could gather. Two important properties on such algorithms are described first.

The OT algorithm approach consists of two main components:

1. The *integration algorithm* which is responsible of receiving, broadcasting and executing operations. It is independent of the type of replica and application.
2. The *transformation function* is responsible for merging two concurrent operations. It is application dependent. For example, a text editor has different operations than a whiteboard application.

The integration algorithm calls the transformation function when needed. The correctness of the OT approach relies on both the correct integration algorithm as well as on the correct transformation function.

3.1 dOPT

dOPT (Distributed Operational Transformation) is the first operational transformation algorithm developed by Ellis and Gibbs[6] in 1989. It was soon discovered that in some situations the document replicas did not converge. This situation is known as the *dOPT* puzzle.
The algorithm fails in cases where there is more than one concurrent operation from a user (see figure 2). Given operation O_a from site a and operations O_{b_1} and O_{b_2} from site b, where $O_a \parallel O_{b_1}$, $O_a \parallel O_{b_2}$ and $O_{b_1} \rightarrow O_{b_2}$, *dOPT* incorrectly applies inclusion transformation with O_{b_1} and O_{b_2} in sequence against O_a at site a. Note however that O_{b_2} is not context-equivalent with O_a and therefore $IT(O_a, O_{b_2})$ violates the precondition of inclusion transformation.

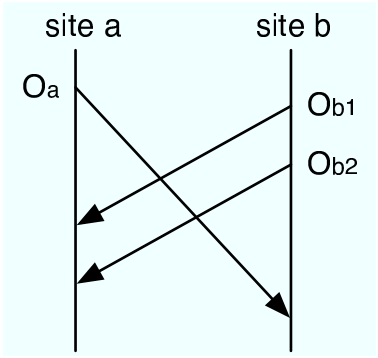

Figure 2: dOPT puzzle

Cormack[29] detected this case and proposed a new algorithm (see 3.2) that is only suitable for two sites connected by a point-to-point communication channel. He stated that there does not appear to be a simple and efficient correction to *dOPT* that maintains its suitability for broadcast operations.
However Cormack showed in [9] that using several point-to-point communication channels forming a tree it is possible to derive a consistent solution for an arbitrary number of sites (see also 3.4).

3.1.1 Properties

- algorithm is incorrect

- uses state vectors to determine causality relations (dependent or independent operations)
- uses linear history buffer (called request log)
- architecture: replicated, multicast

3.2 CCU

CCU (a Calculus for Concurrent Update) derives from the *dOPT* (see 3.1) algorithm. It was developped by Gordon V.Cormack in 1995 that discovered earlier [29] that *dOPT* is incorrect. The algorithm specifies a concurrent model based on a sequential model augmented with definitions of all possible pairs of elementary operations (e.g. insert, delete). The concurrent model is implemented by a set of objects: one for each source of events.
Although there is a description of the algorithm in [9], many implementation details are left out. Interestingly no references to this algorithm are found in other research papers.

3.2.1 Properties

- correctness never confirmed by other researchers
- uses state vectors to determine causal relations
- architecture: replicated, unicast

3.3 Jupiter

Jupiter is a multi-user, multimedia virtual world intended to support long-term remote collaboration. In particular, it supports shared documents, shared tools, and, optionally, live audio/video communication. It is basically a collaborative windowing toolkit. The low-level communication facilities (operational transformation) are described in [21].
Jupiter's algorithm is derived from *dOPT*. The centralized architecture and thus the reduction of point-to-point connections allows them to simplify the *dOPT* algorithm. Several point-to-point connections are used to build a tree-structured n-site algorithm.
Jupiter solves the *dOPT* puzzle. It uses a two dimensional state space instead of a linear history buffer (request log) to save operations. It transforms saved messages against conflicting incoming messages. Unfortunately, simply transforming saved messages does not work for the n-way case, since the next message can come from a third site that is in an inconvenient message state. See *adOPTed* 3.5 for a solution to the n-way case.

3.3.1 Details

The general tool for handling conflicting (concurrent) messages is the transformation function, called *xform* in [21].

$$xform(c, s) = c', s'$$

It takes a message c from the client and a message s from the server and returns two transformed messages c' and s'. The messages c' and s' have the property that if the client applies c followed by s', and the server applies s followed by c', both client and server wind up in the

same final state. The *xform* function is basically a combined inclusion transformation (IT) function that returns the result of both $IT(c, s)$ and $IT(s, c)$.
It is helpful to show the two dimensional state space that both client and server pass through as they process messages (see figure 3). Each state is labelled with the number of messages from the client and server that have been processed to that point. For instance if the client is in the state $(2, 3)$, it has generated and processed two messages of its own, and has received and processed three from the server. The client and server messages are displayed on different axis in the state space.
If there is a conflict, the paths will diverge, as shown in figure 3. The client and server moved to the state $(1, 1)$ together by first processing a client message, and then a server message. At that point, the client and server processed different messages (concurrently), moving to state $(2, 1)$ and $(1, 2)$ respecitvely. They each received and processed the other's message using the transformation function to move to state $(2, 2)$. Then the server generated another message, sending it to the client and both were in state $(2, 3)$.

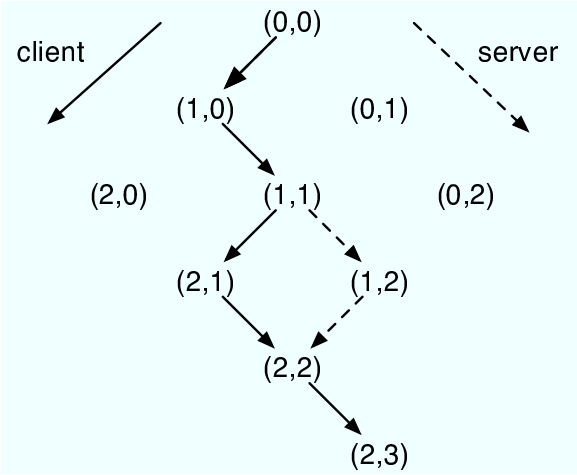

Figure 3: Two dimensional state space example

The algorithm labels each message with the state the sender was in just before the message was generated (state vector). The recipient uses these labels to detect conflicts. Two concurrent messages have to be transformed, but they can only be transformed directly when they were generated from the same state of the document.
If client and server diverge more than one step, the transformation function cannot be applied directly. Consider figure 4. The client has executed c and receives the conflicting message $s1$ from the server. It uses the transformation function to compute $s1'$ to get to the state $(1, 1)$. The server then generates $s2$ from the state $(0, 1)$, indicating that it still has not processed c. What should the client do now? It cannot use the transformation function directly because c and $s2$ were not generated from the same document state.

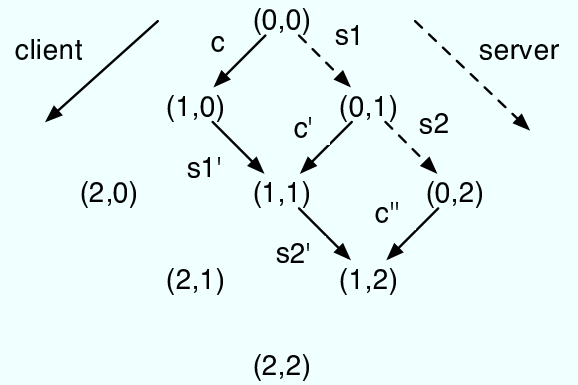

Figure 4: Conflicting messages

The solution to this situation is as follows. When the client computes $s1'$ it must also remember c'. This represents a hypothetical message that the client could have generated to move from the state $(0, 1)$ to $(1, 1)$. When $s2$ arrives, the client can use c' to compute c''. It executes $s2'$ to get to the state $(1, 2)$. If the server has processed the client's message, it will be in the state $(1, 2)$ as well. If not, its next message will originate from $(0, 3)$, so the client saves c'' just in case.
The algorithm guarantees that, no matter how far the client and server diverge in state space, when they do reach the same state, they will have identical states (so convergence is achieved).

3.3.2 Properties

- uses state vectors to determine causality relations
- uses multiple 2-way synchronization protocols to create a n-way protocol
- free of TP2
- very simple algorithm
- two dimensional state space graph
- architecture: replicated, unicast

3.4 NetEdit Consistency Algorithm

NetEdit is a collaborative text editor ([30]) that uses a replicated architecture with processing and data distributed across all clients. It uses an n-way synchronization protocol derived from the algorithm of the *Jupiter* (see 3.3) collaboration system. The algorithm is called *NetEdit Consistency Algorithm*.
The 2-way synchronization protocol developed for *Jupiter* was the starting point. That algorithm was extended to a multi-way protocol using multiple 2-way connections. In [31], the multi-way protocol that was already mentioned in [21] is described in detail.

3.4.1 Properties

- uses state vectors to determine causality relations
- uses multiple 2-way synchronization protocols to create a n-way protocol
- free of TP2
- architecture: replicated, unicast

3.5 adOPTed

adOPTed was devised by Ressel and Gunzenhaeuser and described in [19]. It is an improved version of the *dOPT* algorithm. A multi-dimensional model of concurrent interaction is the core of this approach. This model allows direct communication with n sites. The algorithm is conceptually similar to *Jupiter* (see 3.3), but extends the two way communication in *Jupiter* to a multi way communication. Both use a state space graph to track operations. The state space graph in *adOPTed* is n-dimensional wherelse in *Jupiter* it is two dimensional.

3.5.1 Algorithm

This algorithm uses a so called *L-Transformation* function. This L-Transformation function is the equivalent of the $xform$ function in *Jupiter* (see 3.3). It is defined as:

$$tf(r_1, r_2) = (r_1', r_2')$$

An L-transformation function must satisfy TP1, TP2 and must be symmetric (that is, if $tf(r_1, r_2) = (r_1', r_2')$, then $tf(r_2, r_1) = (r_2', r_1')$). Requests and L-transformations can be represented as grid-based diagrams. The axes represent users, grid points represent states with a certain state vector, and arrow represent requests, being either original or the result of some transformation.
Each user process manages the following local data structures: the application state s, a counter k for locally generated requests, a site's state vector v, its request queue Q, a request log L and an n-dimensional interaction model G. The *state vector* v holds the number of execution for each user. The *request queue* Q is used to store generated and incoming requests that have to wait for execution. The *request log* L stores a copy of each original request so that a request can be easily accessed by its key consisting of user id u and serial number k. The *interaction model* G is mainly used to store transformed requests that might be needed later. It would be possible to store application states like in the formal definition of the interaction model, but this is not necessary for the algorithm to work.

3.5.2 Transformation Functions

In [19] they also propose a set of transformation functions for text editing. The transformation functions were proved to be wrong by [1]. That is, they do not hold transformation property 2 (TP2).
Note however that it has been proved by [18] that the control algorithm of *adOPTed* is correct as long as the transformation functions hold TP2. Using the proposed transformation functions from [3] results in a correct system that achieves the three properties convergence, causality preservation and intention preservation.

3.5.3 Properties

- proved to be correct if transformation functions hold TP2
- uses state vectors to determine causality relations
- uses only IT
- user undo described in [20]
- n-dimensional interaction graph
- architecture: replicated, multicast

3.5.4 Known implementations

Ressel implemented a prototypical group editor named *Joint Emacs*. Another group editor that uses *adOPTed* is *Gclipse* [8], a collaborative editor plug-in for eclipse.

3.6 GOT

Sun et. al were the first to define the three consistency properties, convergence, causality preservation and intention preservation in [28]. The GOT (generic operation transformation) algorithm separates the parts that achieve each of these properties.

3.6.1 Achieving Causality Preservation

To achieve causality preservation, a quite standard approach is used. An operation O is only executed if all operations that causally precede O have been executed at the local site. It can be shown that if a remote operation is executed only when a remote operation satisfies the above condition, then all operations will be executed in their causal orders, thus achieving causality preservation.

3.6.2 Achieving Convergence

The causality preserving scheme imposes causally ordered execution only for dependent operations and allows an operation to be executed at the local site immediately after its generation. This implies that the execution order of independent operations may be different at different sites. So how is the convergence property ensured in the presence of different execution order of independent operations? GOT defines a total ordering relation to solve this problem.

Definition 8 *Total ordering relation* $\Rightarrow$

Given two operations O_1 and O_2 generated at sites i and j and timestamped by SV_{O_1} and SV_{O_2} respectively, then $O_1 \Rightarrow O_2$ iff

1. $sum(SV_{O_1}) < sum(SV_{O_2})$ or
2. $i < j$ when $sum(SV_{O_1}) = sum(SV_{O_2})$

sum is simply the sum of all components of the state vectors SV. In addition each site maintains a linear *history buffer (HB)* for saving executed operations at each site. Based on this total ordering relation, the following *undo/do/redo* scheme is defined. When a new operation O_{new} is causally ready, the following steps are executed.

1. **Undo** operations in HB which totally follow O_{new} to restore the document to the state before their execution
2. **Do** O_{new}
3. **Redo** all operations that were undone from HB

Note that the undo/do/redo scheme is an internal operation only. The user interface should only show the final result.

3.6.3 Achieving Intention Preservation

To achieve intention preservation, a causally-ready operation has to be transformed before its execution to compensate the changes made to the document state by other executed operations. *GOT* uses both inclusion transformation (IT) and exclusion transformation (ET). As there is a diverse and irregular dependency among operations, a sophisticated control algorithm is needed to determine when and how to apply IT/ET to which operations against which others. See [28] for a detailed description of this control algorithm.

3.6.4 Integration

The achievement of the three properties, convergence, causality preservation and intention preservation must be integrated to get a solution that satisfies all three properties. This results in a new modified *undo/transform-do/transform-redo scheme*. This new scheme is described in detail in [28].

3.6.5 Properties

- uses state vectors to determine causality relations
- uses IT and ET
- global ordering using undo/do/redo scheme to achieve convergence
- free of TP1 and TP2, through the use of global total ordering
- complex control algorithm (many transformations needed)
- architecture: replicated, multicast

3.6.6 Implementations

The *GOT* control algorithm was implemented in the *REDUCE* [7] prototype. It was however replaced later by the *GOTO* algorithm.

3.7 GOTO

The *GOT* (see 3.6) control algorithm integrated with the undo/do/redo scheme achieves both intention-preservation and convergence. Sun et. al [27] modified the *GOT* control algorithm in such a way that the IT/ET functions must satisfy TP1 and TP2. The resulting algorithm is called *GOTO* (GOT optimized).
The original *GOT* algorithm, given IT/ET functions that satisfy TP1 and TP2, can ensure both intention preservation and convergence without integrating with the undo/do/redo scheme or using a multi-dimensional graph. This results in a simplified control algorithm that uses fewer IT/ET transformations compared to *GOT*.

3.7.1 Properties

- uses state vectors to determine causality relations
- uses IT and ET
- TP1 and TP2 must be ensured by IT/ET functions
- architecture: replicated, multicast

3.7.2 Implementations

The *GOTO* control algorithm was implemented in the *REDUCE* [7] prototype.

3.8 SOCT2

The SOCT2 concurrency control algorithm is proposed for a centralized and a distributed environment. For details, see [24] [25].

3.8.1 Centralized Environment

In the centralized environment, a collaborative system is constituted of a set of user sites (one user per site) connected by a reliable network to a server site. The server site is responsible for the storage of the objects (i.e. documents, texts, graphics,...) shared by the users collaborating in a common activity. Each object may be manipulated by the users by means of specific operations that must be transmitted to the server to be executed. The progress of an operation is decomposed into four steps: generation, sending, reception and execution. After execution of an operation on the server, it is supposed that the results on the object are immeditately visible to all users sharing that object.
When an operation $< op, i >$ generated from the object state O_i is sent to the server, the server integrates the operation by transposing it forward with the sequence of concurrent operations in the history buffer (if any). The resulting operation is then executed and added to the history buffer.
At the server, causally ready operations are executed immediately. To serialize concurrent operations with respect to the users intentions, the server uses a forward transposition function (equivalent to IT). The forward transposition is specific to the application and to the semantic of the operations (e.g. insert and delete operations for text editing). This transposition makes use of the object history in order to determine if there are concurrent operations to which the operation to be executed must be forward transposed.

The object history is the sequence of all operations executed on the object and ordered in their execution order. To achieve the correct execution order of the operations, timestamps are used. Timestamps allow to determine if an operation precedes another one or is concurrent with it.

3.8.2 Distributed Environment

The main difference to the centralized environment is that the shared objects are distributed and replicated. Thus the algorithm is more complex.
Each participating site is both a user and a server site. For each site, there exists a copy of the object O. Every operation intended to be executed on O must be executed on all the copies. For that, an operation generated by a site must be broadcast to all the other sites. The four events associated with an operation are: generation, broadcast, reception and execution. Each site has to respect the following constraints:

- immeditate execution of local operations after receiving them
- respect of the causality between the operations
- guarantee of the consistency of the copies by respecting intentions associated to the operations

To each copy of an object O on a site S is associated an operation history. So the problem is to construct, for every site S and for every object O, a history respecting the constraints above in such a way that if the object copies were identical at the beginning of the work session, they will also be identical each time the system is quiescent, that is to say when there are no more messages circulating in the network.
To guarantee the consistency of the copies, the forward transposition is used. To handle the problem that comes from the conflict between two insertion operations of a character at the same position, the insertion operation contains two additional parameters b and a. b (a), is a set of operations that have deleted a character before (after) the inserted character. These parameters, which show how the operations were generated, help to determine the correct solution. Additionally, a priority (e.g. the character's code) to each character is used in the forward transposition function for two insertion operations.
An operation history H is maintained for each object O. The history H verifies that:

- all operations that precede Operation op are integrated in H
- op does not precede any operation of H

The causality is thus respected, and it remains to transpose op forward with its concurrent operations to obtain an operation whose execution respects the user intention and guarantees the data consistency. However in some cases (i.e. partially concurrent operations) the history must be re-ordered by means of backward transposition (changes the order of operations in the history while preserving equivalence) so that the forward transposition can be applied correctly (i.e. executed on two operations transformed in such a way as though they were defined on the same object state). Thus the history is reordered in such a way that the operations which precede op, the operation to be integrated, are regrouped before the operations wich are concurrent to op. By that, H will be separated into two sequences seq_1 and seq_2, so that all the operations of seq_1 precede op and all the operations of seq_2 are concurrent

to op. The forward transposition for op can then be applied correctly to seq_2 and finally be executed on the object.

3.8.3 Properties

- proposed algorithm in centralized environment uses only forward transposition
- replicated environment uses forward and backward transposition (without undoing and redoing operations)
- uses timestamps (centralized env.) and state vectors (replicated env.) respectively to determine causality relations
- uses linear history buffer (called history)
- the management of the sets a and b is difficult and transferring the insertion operation is not efficient
- no undo mechanism proposed
- transformation functions were proved wrong by counterexamples by Imine et al. in [3]
- architecture in centralized environment: centralized, unicast
- architecture in distributed environment: replicated, multicast

3.9 SOCT3

Verifying that a given set of transformation functions satisfies TP2 is not trivial. There are over a hundred cases to be checked depending on the various parameters. So the inventors of *SOCT3* and *SOCT4* (see 3.10) decided to go a different way in order that TP2 must not hold ([22]).
They propose the implementation of a global serialization order such that the operations can be delivered in this order. The global serialization order is achieved by the use of a sequencer (see 3.10.2). A sequencer is an object which delivers continously growing positive integer values, called timestamps.
A local operation O is executed immediately to respect the real-time constraint. Next, the call to the function *Ticket* returns a timestamp N_O which is assigned to the operation. The quadruplet $< O, S_O, V_O, N_O >$ is then broadcast where S_O is the generating site and V_O is the state vector associated with O and N_O.
The reception procedure ensures a sequential delivery of all operations with respect to the ascending order of the timestamps. Upon receiving an operation it delays its delivery until all operations with lower timestamps have been received and delivered. The state vector is of no use for the reception procedure, but it enables to determine which operations are concurrent to O during the integration step.
SOCT3 uses both IT (called forward transposition) and ET (called backward transposition) in the integration step. For details, see [22].

3.9.1 Properties

- uses state vectors to determine causality relations
- uses linear history buffer (called history)
- uses a unique global ordering to abandon TP2 (by using a sequencer)
- no known user undo algorithm
- architecture: replicated, multicast

3.10 SOCT4

In *SOCT4* [22] as in *SOCT3*, the operations are ordered globally using a timestamp given by a sequencer. They are then delivered on each site in this order thanks to the sequential reception. The originality of *SOCT4* comes from the fact that IT (called forward transposition) is now made by the generator sites of the operations. According to [22] this results in three major advantages:

- the receiver site does not have to separate history anymore; thus backward transposition becomes unecessary
- the received operation can be stored as it is in the history without further transformation
- state vectors are no longer needed

To achieve this, the broadcast of an operation must be deferred until it has been assigned a timestamp and all the operations which precede it according to the timestamp order have been received and executed. As usual, local operations are executed immediately without delay.

3.10.1 Properties

- does not need any state vectors
- uses a unique global ordering to abandon TP2 (by using a sequencer)
- no ET needed
- no known user undo algorithm
- architecture: replicated, multicast

3.10.2 Sequencer

As noted before, both *SOCT3* and *SOCT4* use a sequencer to globally serialize operations. In *SOCT4* operations are broadcast sequentially. This makes collaboration difficult when the propagation delay of an operation on the network is high. This characteristic makes *SOCT4* particularly adapted to fast networks.
More information about sequencers can be found in [10]. Various methods for implementing sequencers are described in [11] (circulating sequencers) and [5] (replicated sequencers).

3.11 TIBOT

The *TIBOT* algorithm presents a novel interaction model that is based on time intervals. It was presented in 2004 by Rui Li, Du Li and Chengzheng Sun in [16]. In this summary, firstly some necessary concepts are introduced. Secondly the consistency control algorithm is explained.

3.11.1 Concepts

Time intervals based on logical clocks Every site in the replicated architecture maintains a linear logical clock. All clocks are initialized to a common value. A time interval is the period between two consecutive clock ticks. Time interval lengths are assumed to be the same. Every operation is timestamped by the current local clock value to indicate which time interval it belongs to. Function $TI(O)$ denotes the time interval of operation O. Two operations, O_i and O_j, are in the same time interval iff $TI(O_i) = TI(O_j)$, no matter whether $i = j$ or not.

Total ordering and operation context The total ordering is defined by means of the time interval function. In addition, *TIBOT* uses a linear history buffer in which the operations are stored in the total order at each site. The notion of operation context is the same as defined in 1.6.1. The operations are allowed to be executed in any order as long as their causality is preserved and the following synchronization rules are observed.

Propagation and synchronization rules

- Propagation rule 1: A local operation is propagated only when the current time interval is over and the operation has been transformed with all operations that have been executed in earlier time intervals. That is, a local operation always needs to be transfomred against all operations generated in previous time intervals before it can be propagated. Hence operation propagation is normally delayed.
- Propagation rule 2: When a time interval is over, if no operation has been generated locally, a control message is propagated to notify other sites that there was no operation generated in the past time interval.
- Synchronization rule 1: Any remote operation received during a time interval can be synchronized only when (1) the local time interval is over; (2) the time interval of this operation is the same as the current local time interval; and (2) all operations totally preceding this operation have been executed.
- Synchronization rule 2: When a remote operation O is to be executed, all operations that have been executed but out of the total order must be first undone, then perform O, and then redo these undone operations in sequence. These operations to be done or redone need to be transformed before they are executed (undo/do/redo scheme). Local operations are always executed immediately once generated.
- Synchronization rule 3: Operations with the same timestamp and site id are always synchronized as an entirety (i.e. are processed as entirety in transformations against local operations (concept of group operation)).

The above rules reduce the complexity of communication and interaction compared with other consistency models for group editors. E.g. ET is not necessary in this model.

3.11.2 The TIBOT control algorithm

Due to the rules defined above and the properties derived from these rules (see [16]), the concurrency problem is simplified in this model. It can be reduced (from the three cases discussed in [27]) to the following two cases. Assume that O is a execution-ready operation. $[O_1, O_2, ..., O_n]$ are a sequence of operations that have been executed locally and stored in the local history buffer (HB). Suppose HB = $[O_1, O_2, O_3]$.

- Case 1: $O_1 \rightarrow O, O_2 \rightarrow O, O_3 \rightarrow O$
  All operations in HB are preceding O. Therefore O' = O, i.e., no transformation is needed.
- Case 2: $O_1 \rightarrow O, O_2 \parallel O, O_3 \parallel O$
  Operations preceding O are stored in HB before operations that are independent of O. Then O' is obtained by transforming O against concurrent operations, O_2 and O_3, in sequence.

The top-level control algorithm *TIBOT* is based on these two cases and the undo/do/redo scheme. *TIBOT* is called only when a remote group operation is ready for synchronization.

3.11.3 Properties

- simplified and efficient approach based on time intervals instead of state vectors
- uses only IT
- is free from TP2
- implementation of time interval concept not clear
- architecture: replicated, multicast

3.12 NICE

Haifeng Shen and Chengzhen Sun devised a new operational transformation control algorithm in combination with a notification component in 2002 [23]. In this paper they describe a flexible notification framework that can be used to implement a wide range of notification strategies used in collaborative systems.
In the proposed framework, the notification policy that determines when and what to notify is separated from the notification mechanism that determines how to notify. The parameters *frequency* and *granularity* are provided to define various notification policies.
The frequency parameter determines the *when* aspect of notification, that is, when a notification is propagated/accepted. The granularity parameter determines the *what* aspect of notification, that is, which updates are going to be propagated/accepted. Further there can be a separate policy for input and output direction.
The notification mechanism determines the *how* aspect. Both outgoing and incoming messages are put in distinct buffers (input buffer IB and output buffer OB). An outgoing notification executor (ONE) and an incoming notification executor (INE) are needed to carry out various outgoing and incoming notification policies respectively.

A very important component in the notification mechanism is the notification propagation protocol (NPP), which is needed for propagating updates from the OB at the notifying site to the IB at the notified site.
The notifications are contextually serialized. This is achieved by use of a central notification server, which acts as a centralized serialization point and message relaying agent (all messages pass through this central server).

3.12.1 Notification Propagation Protocol

Before propagating a notification, the notifying site sends a *Token-Request* message to the *Notifier* (a central notification server), waiting for the *Token-Grant* message from the *Notifier*. After being granted the token, the site propagates the notification piggybacked with the *Token-Release* message to the *Notifier*. When the *Notifier* receives the notification and the *Token-Release* message, it forwards the notification to all interested sites. By using the notifier as a message relaying agent, causal relationships among notifications are automatically guaranteed.
This sequential propagation simplifies concurrency control. However it is also inefficient in supporting notification policies for meeting real-time collaboration needs. For propagating one notification that may contain only one operation, three extra messages have to be sent.

3.12.2 Concurrent Propagation

The proposed solution consists of a protocol that allows a site to propagate its notification without first requesting a token, thus effectively eliminating the *Token-Request* message. This operation propagation protocol is called *SCOP* (symmetric contextually-serialized operation propagation).

3.12.3 Properties

The transformation control algorithm is called *SLOT* (symmetric linear operation transformation). Together with the *SCOP* protocol it has the following properties.

- no state vectors needed
- no ET
- free of TP2
- architecture: replicated, unicast

The reason why *SLOT* is free of TP2 is that under no circumstance an operation could be transformed against the same pair of operations in different orders. The operations are always ordered uniquely.

3.13 SDT

The algorithm *SDT* was presented in 2004 by Du Li and Rui Li [15]. They detected defects in existing inclusion transformation and exclusion transformation functions. The proposed solution (*SDT*) tries to fix these.

3.13.1 Defects of traditional transformation functions

The problem is related to inclusion transformation between two insert operations and one delete operation with close position parameters. In some of these cases, the result of inclusion transformation is not deterministic.

Example: Given the initial document state "abc". The three editing sites 1, 2 and 3 generate $O_1 = insert(''1'', 2)$, $O_2 = insert(''2'', 1)$ and $O_3 = delete(''b'', 1)$ respectively. The three operations are independent. Now consider what happens at site 3. After the deletion of character "b" at position 1 the document state becomes "ac". The next message arriving is then O_2 which inserts "2" at position 1. The resulting document state is "a2c". The next operation O_1 arrives at site 3 and is transformed against O_3 and then O_2. The result of the second transformation is non-deterministic. It could be $insert(''1'', 1)$ or $insert(''1'', 2)$. However, the original intention of O_1 is to insert "1" after "b" and the intention of O_2 is to insert "2" before "b". Then "1" should appear after "2" in the resulting document state ("a21c"). So depending on the chosen priority scheme, the result could be violating the intention of the original operations. Another even more severe problem results from the fact, that the document state of site 3 could diverge from site 1 and 2. A similar problem arises with traditional exclusion transformation functions.
The conventional transformation functions use the site id as priority scheme if two inserts happen at the same position. This was identified as the source of the problems as described above by Du Li and Rui Li. This new algorithm tries to delay the use of site ids.

3.13.2 Proposed Solution

The algorithm works conceptually as follows. For each operation the original intention is recovered by computing its β value against a well-known document state (the latest synchronization point). Then in performing inclusion transformation, the β values are compared. An algorithm to compute β is given in the paper. The approach is based on a new concept called state difference, hence the name SDT (state difference transformation).
The user intention is always achieved through performing operations that generate certain effects on a given document state. The effect of an operation O on its definition context is trivially itself, either to insert or delete a character. However, the effect of operation O on S_i is not as obvious if S_i precedes the definition context. To characterize the effect of an operation on a prior document state more accurately, two notations β and δ are introduced. Read [15] for a general overview and [13] for the implementation details. The former document does not specify important implementation details (e.g. it is not explained how to obtain the *latest synchronization point*). We did not read [13] because SDT was proved incorrect [3] and we did not find the document on the Internet.

3.13.3 Properties

- transformation functions must hold TP2
- no undo mechanism
- proposed IT/ET functions proved wrong by Imine et. al [3], i.e. they do not hold TP2 in all cases

- architecture: replicated, multicast

3.14 Li 04

This paper was written 2004 by Du Li and Rui Li in [17] [1]. In fact, the algorithm *Li 04* replaces SDT (3.13). It proposes a novel transformation based approach to preserving the correct operation effects relation in group editors. Due to its root in effects relation violation (see example in 3.13), the divergence problem is then solved automatically so that convergence is achieved in the presence of arbitrary transformation paths. In this summary, firstly two consistency model issues are introduced and secondly the control algorithm with its transformation functions are explained.

3.14.1 Operation effects relation

The position of every character in a string is unique. Therefore the position relation between all characters in any string is a total order $\prec$. Characters in any document state are eventually a result of editing operations. Every characterwise operation O has an effect character $C(O)$. Hence the total order $\prec$ of characters is extended to operations. Given any two operations O_x and O_y, it is $O_x \prec O_y$ iff $C(O_x) \prec C(O_y)$. Relation $\prec$ is not defined between any pair of inverse operations because the effect characters of inverse operations are the same, their position relation is not a total order.

3.14.2 Definition of the CSM consistency model

A group editor is CSM-consistent iff the following three conditions hold:

- **C**ausality preservation: all operations are executed in their cause-effect order.
- **S**ingle-operation effects preservation: the effect of executing any operation in any execution state achieves the same effect as in its generation state.
- **M**ulti-operation effects relation preservation: the effects relation of any two operations maintains after they are executed in any states.

3.14.3 Transformation functions

Operational transformation functions are used for achieving CSM consistency in group editors.

Inclusion Transformation The IT function is defined in the straightforward way except that it uses the operation effects relation $\prec$. If the correct relation $\prec$ between the effects of all concurrent operations is known, then remote operations can always be executed correctly while preserving S and M conditions. The problem is how to determine the effects relation. Therefore, the concept of last synchronization point (LSP) is introduced. Let V_1 and V_2 be the state vectors of operations O_1 and O_2, respectively. Let V_{min} be a state vector, each element of which is equal to the minimal value of the corresponding elements of V_1 and V_2. Then S_{lsp} is the state that corresponds to V_{min}. State S_{lsp} is the last common state for two operations to be transformed inclusively. By means of the LSP the effects relation $\prec$ between two concurrent operations can be determined.

[1] Their approach was not named in [17], so we name it *Li 04* for convenience.

Exclusive Transformation The conceptual ET function is defined similarly to the IT function described above based on the effects relation $\prec$. The question is again how to know $\prec$. For the ET function to work correctly, position parameters are used as far as possible. Three different cases are identified to determine $\prec$ (see 4.3 in [17] for further details). The three cases reveal the following important fact: Given two operations, $O_i^{S^{i-1}}$ and O^{S^i} where $O_i \rightarrow O$, and O does not depend on O_i, only when S^i is the generation state of O, is it safe to use position parameters to determine the effects relation, and are the original ET functions of [28] equivalent to the conceptual ET functions proposed. Therefore effects relation cannot always be determined correctly. This is allowed for in the control algorithm described next.

3.14.4 The control algorithm

The control algorithm partly follows the structure of *GOTO* (see 3.7). A history buffer is used at each site. When a remote operation O is causally-ready for execution, a copy of HB in SQ is made. SQ is transposed such that all operations in the left subsequence causally precede O, and all operations in the right subsequence are concurrent with O. After that, O is inclusively transformed against the concurrent subsequence in SQ. After the result O' is executed in the current state and appended to HB. The details of the proposed algorithm as well as the differences from GOTO are presented in section 5 in [17]. The algorithm makes use of concepts described above: operation effects relation and LSP to perform IT and ET correctly. The concept of state difference SD (see also algorithm *SDT* in [15]) is proposed further and used in relation to excluding operation effects (ET). SD allows for the ET problem implied above so that the effects of contextually preceding operations can be excluded correctly.

3.14.5 Properties

- IT and ET functions are based on a novel consistency model
- control algorithm similar to *GOTO* (see 3.7)
- proves to satisfy TP1 and TP2
- no user undo functionality proposed
- architecture: replicated, multicast

4 Comparison of Algorithms

4.1 Overview

The following tables compare different aspects of the examined algorithms. The first table explains what is meant by the aspects.

Aspect	Description
Year	The year the research paper was published.
Correct?	No if another research paper has proved the algorithm (or its OT functions) wrong, otherwise Yes
Architecture	Which architecture is the algorithm designed for?
Available Information	Is there enough information for an implementation?
Intention Preservation	How is the user's intention for an operation be preserved? (see 1.4)
Causality Preservation	How is causal ordering relation on operations preserved? (see 1.4)
Copies Convergence	How is copy convergence on all replicated objects achieved? (see 1.4)
Broadcast	When is the broadcast i.e. the emission of a generated operation carried out?
Delivery	In what order are the operations at each site executed?
Undo	Is a user undo functionality for the algorithm available?

Table 2: Aspect Explanation

	dOPT	Jupiter	adOPTed	GOT	GOTO
Year	1989	1995	1996	1998	1998
Correct?	no	control algorithm	control algorithm	yes	control algorithm
Architecture	replicated, multicast	replicated, unicast	replicated, multicast	replicated, multicast	replicated, multicast
Available Information	enough	enough	enough	enough	enough
Intention Preservation	dOP Transformation	Transformation and two-dimensional graph	L-Transformation and multi-dimensional graph	IT and ET	IT and ET
Causality Preservation	state vectors	state vectors	state vectors	state vectors	state vectors
Copies Convergence	TP1 (but no convergence achieved)	TP1	TP1 and TP2	non-continuous global order and undo/redo	TP1 and TP2
Broadcast	immediate	immediate	immediate	immediate	immediate
Delivery	causal order	causal order	causal order	causal order	causal order
Undo	no	no (but could be derived from adOPTed)	yes	no	yes

Table 3: Comparison Matrix 1

	SOCT2	SOCT3	SOCT4	SDT	TIBOT
Year	1997	2000	2000	2004	2004
Correct?	no (transformation functions)	yes	yes	no (transformation functions)	yes
Architecture	replicated, multicast	replicated, multicast	replicated, multicast	replicated, multicast	replicated, multicast
Available Information	enough	implementation of sequencers	implementation of sequencers	not enough for implementation	not enough
Intention Preservation	IT and ET	IT and ET	only IT	IT and ET	IT
Causality Preservation	state vectors	timestamps	timestamps	state vectors	time intervals
Copies Convergence	TP1 and TP2	TP1 and continous global order	TP1 and continous global order	TP1 and TP2	TP1, propagation and synchronization rules
Broadcast	immediate	immediate (as soon as timestamp is assigned)	deferred, in timestamp order	immediate	deferred, after time interval
Delivery	causal order	continous global order	continous global order	causal order	?[2]
Undo	no	no	no	no	no

Table 4: Comparison Matrix 2

	NICE	LI04	CCU		
Year	2002	2004	1995		
Correct?	yes	yes	probably yes		
Architecture	replicated, unicast	replicated, multicast	replicated, unicast		
Available Information	enough	not enough [3]	not enough		
Intention Preservation	IT	IT and ET	?		
Causality Preservation	central notification server	state vectors	?		
Copies Convergence	TP1 and unique global order	TP1 and TP2	TP1 and TP2		
Broadcast	immediate	immediate	immediate		
Delivery	causal order	causal order	causal order		
Undo	no	no	no		

Table 5: Comparison Matrix 3

4.2 Selection Criteria

From the set of available algorithms we want to make a pre-selection. This pre-selection is based on the criteria set forth in this section.

[2]Due to lack of information we are not capable of determining the answer definitely.
[3]The paper [14] with the necessary implementation details should be released around May 2005.

Correctness: This is obviously the most important criteria. If an algorithm is not correct, it is not worth being implemented.

Availability of information: Some papers do not provide enough information for an implementation.

Availability of user undo: Users of collaborative applications expect the same commands as in a single user application. Without user level undo, the user experience will not be satisfactory.

Algorithmic complexity: Some algorithms are strikingly simple, others are very complex (too complex). Simplicity is a selection criteria.

4.3 Selection of Algorithms

4.3.1 Correctness

Based on the first selection criteria (correctness) *dOPT* algorithm is deemed unsuitable. Further the transformation functions used by some algorithms have been proved incorrect [3]. These include *adOPTed*, *GOTO*, *SOCT2* and *SDT*. Note that the transformation functions can be replaced as they are independent of the control algorithm. By using the proposed IT function of [3] these control algorithms would be correct again (at least the control algorithm of *adOPTed* and *GOTO* have been proved correct). As *GOTO*, *SOCT2* and *SDT* also use ET functions and [3] only proposed and proved IT, we cannot assume them to be correct for an implementation yet. There have to be ET functions developed and proved analogous to the IT functions in [3] first.
The following algorithms meet the correctness criteria:

- *Jupiter* (see 3.3)
- *adOPTed* (see 3.5)
- *GOT* (see 3.6)
- *SOCT 3/4* (see 3.9 and 3.10)
- *TIBOT* (see 3.11)
- *NICE* (see 3.12)
- *LI04* (see 3.14)
- *CCU* (see 3.2)

It is important to note that *CCU*, *TIBOT* and *LI04* have not been approved or successfully implemented by a third party as far as we know.

4.3.2 Availability of Information

For the algorithms in the next list we do not have enough information available for an implementation.

- *CCU* (see 3.2)
- *SDT* (see 3.13)
- *TIBOT* (see 3.11)

For the implementation of *SOCT3* and *SOCT4* some more papers are needed concerning the implementation of sequencers. Though referenced, these could not be found on the Internet. For *Li04* we do not have enough information yet. But there will be a paper with implementation details published around May 2005 according to Mr. Du Li.

4.3.3 Complexity

SOCT2 is considered to be very complex [2] [15] due to management of the sets s_i and b_i associated with each *Insert* operation. *SOCT3* and *SOCT4* use a sequencer that simplifies the implementation of the algorithm but requires this additional component which complicates the implementation. The theoretical aspects of *GOT* as well as *GOTO* are relatively complex, but the implementation itself should be straightforward. *NICE* uses a central notification server that simplifies the algorithm significantly. *Jupiter* is a very simple algorithm, because it restricts itself to point-to-point communication. *adOPTed* is more complicated than *Jupiter* because it adds n-way communication. However, we consider the implementation as relatively straightforward.
The following algorithms are considered to be too complex to be accounted for an implementation.

- *SOCT2* (see 3.8)

4.3.4 User Undo

The following algorithms have a user level undo defined. This does not imply that it would be impossible to devise an undo mechanism for other algorithms. Note however that adding an undo mechanism to an existing algorithm is a non-trivial task.

- *Jupiter* (see 3.3)
- *adOPTed* (see 3.5)
- *GOTO* (see 3.7)

4.3.5 Selected Algorithms

Based on the above observations but excluding the user undo requirement, the following algorithms remain as implementation choice:

- *Jupiter* (see 3.3)

- *adOPTed* (see 3.5)
- *GOT* (see 3.6)
- *NICE* (see 3.12)

If we take the user undo functionality into account, the selection shrinks to three algorithms:

- *Jupiter* (see 3.3)
- *adOPTed* (see 3.5)

All the above algorithms fulfill the selection criteria sufficently well.

5 Transformation Functions

In section 3, we described several control algorithms along with their proposed transformation functions. Additionally, there exist research papers which focus only on transformation functions which are considered to be used with existing control algorithms. In this section we summarize the research papers we have found on that topic.

5.1 IT function based on position words

A new approach to achieving convergence is presented by A. Imine et al. in [3]. They have formally proved the proposed solution to be correct. Firstly, the key concept of position words is introduced. Secondly, the novel IT function is explained.

5.1.1 Position Words

Definition p-word Let $\Sigma = N$ be an alphabet over natural numbers. The set $\mathcal{P} \subset \mathcal{N}^*$ of words, called p-words, is defined as follows: (i) $\epsilon \in \mathcal{P}$; (ii) if n $\in$ N then n $\in \mathcal{P}$; (iii) if ω is a nonemtpy p-word and n $\in$ N then n$\omega \in \mathcal{P}$ iff either n = Current(ω) or Current(ω) $\pm$ 1. Current(ω) is denoted the first symbol of ω. Thus, Current(1232) = 1.
The position word (p-word) is a vector of numbers which is denoted by ω and defined only on insertion operations. This vector keeps track of the insertion positions of an operation. E.g. p-words are $\omega_1 = 00$, $\omega_2 = 3454$, but $\omega_3 = 3476$ is not.

5.1.2 IT function

The insert operation is extended with a new parameter p-word giving the positions occupied before every transformation step. Thus an insert operation becomes: $Insert(p, c, w)$ where p is the insertion position, c the character to be added and w a p-word.
The OT function is redefined using the p-word concept. A function PW is defined that enables to construct p-words from editing operations. Hence, given an insert operation op, PW(op) gives a p-word which restores all positions occupied by the insert operation op since it was generated.
When the OT function for two insertion operations is called, the PW values of $Insert(p_1, c_1, w_1)$ and $Insert(p_2, c_2, w_2)$ are first compared. If their p-words are equal, then their character codes are compared. When the two operations have the same character to be inserted in the same position then the OT function gives the null operation nop, i.e. one insert operation must be executed and the other one must be ignored [25].

5.1.3 Properties

- IT function extended with concept of p-words
- first (inclusion) transformation function which was thoroughly formally proved ever

5.2 IT and ET for stringwise operations

The OT functions from [28] are enhanced to stringwise operations. They have been implemented in an Internet-based protoype system called REDUCE.

A Undo in multiuser collaborative applications

The availability of undo in a multiuser collaborative application is valuable because features available in single-user applications should also be available in corresponding multi-user applications. However, supporting undo in a multiuser collaborative application is much more difficult than supporting undo in single-user interactive applications because of the interleaving of operations performed by multiple users in a collaborative computing environment.

A.1 An Example

To illustrate the difficulties, let us consider a collaborative editing session with two users and a shared text document containing the string "abc". Suppose user 1 issues the operation $O_1 = Insert[1, X]$ to insert the character "X" at position 1 after the character "a". The resulting document state is "aXbc". After this, user 2 issues operation $O_2 = Insert[1, Y]$ to insert character "Y" after the character "a" to get the resulting document state "aYXbc". Suppose user 1 wants to undo his last operation, i.e. O_1. His intended effect is to remove the "X" thus resulting in a document state "aYbc". First, blindly picking the *globally* last operation, i.e. O_2 will undo the wrong operation. Second, simply executing the inverse operation $\overline{O_1} = Delete[1, X]$ to delete the "X" at position 1 in the current document state "aYXbc" will delete "Y" instead of "X".
This means that any group undo solution must meet the challenge of undoing operations in a nonlinear way and must be able to achieve the correct undo effect in a document state that has been changed by other users' operations. It can be clearly seen that undo in a groupware system is highly related to operational transformations.

A.2 Types of Undo

Generally a user expects an undo to reverse their own last operation (*local* undo) rather than the globally last operation (*global* undo). So an undo framework for groupware systems needs to allow selection of an operation to undo based on who performed it. Undoing the globally last action can be problematic as just before one user presses undo, another user may issue an operation. The effect of the operation may get executed at the first users site just before he effectively invokes the undo action. This would not result in the undoing of an operation which the first user intended to undo.
Further an undo mechanism may be classified by whether it allows to undo an arbitrary operation (called *selective* undo) or it is restricted to the *chronological* order.
It is still an open question how to select operations to be undone in a selective undo system. To undo the chronologically last operation of the local user, the single familiar undo menu or shortcut is sufficient as it is always clear wich operation has to be undone.

A.3 Existing Undo Solutions

A.3.1 DistEdit System

The DistEdit system [4] allows operations to be undone in any order. However, it may be that an operation is not undoable because it conflicts with a later executed operation. The conflicts occur if a later operation happens at the same index as the operation to undo. They managed to solve some conflicting situations, but still a few remained in the finished system.

A.3.2 adOPTed undo

In the *adOPTed* algorithm (see 3.5) undo is supported by operational transformation. It allows to undo the chronological last operation of a user (it is implemented only for the local user but it would be possible to extend it to an arbitrary user). To undo an operation, an inverse of this operation is generated by a special *mirror* operator. This inverse operation must be placed at a valid location in the corresponding dimension of the interaction graph. No conflict can occur in undoing any operation. Another special operator called *folding* is used for a correct working system. See [20] for a detailed description of this undo system.
The transformation function must hold the *undo property* so that the proposed undo mechanism works correctly:

Definition 9 *Undo Property*

Let $\overline{r_1}$ be an undo request for r_1. For all requests $r_2 \in R$: If $tf(r_1, r_2) = (r_1', r_2')$ and $tf(\overline{r_1}, r_2') = (\overline{r_1}', r_2'')$ then $r_2 = r_2''$ and $\overline{r_1}' = \overline{r_1'}$. If the undo property is satisfied, transformations are invertible:

$$tf_1(r_2, r_1) = r_2' \Longleftrightarrow tf_1(r_2', \overline{r_1}) = r_2$$

Where tf_1 simply returns the transformed first argument to tf (for a definition of tf see 3.5.1.

A.3.3 GOTO undo

Sun [26] described an undo mechanism for the *REDUCE* prototype. It is using the *GOTO* algorithm (see 3.7) for *do* and an algorithm called *ANYUNDO* for *undo*. The algorithm allows undoing arbitrary operations, so it is a selective undo system. The algorithm is described in great detail along many possible problems and how they were solved in this algorithm.

B Vector Time

Most algorithms use *vector time* to determine causal relations. Each site maintains a state vector v that has n components. n is the number of participating sites. S is the local site. The i-th component of v denoted as $v[i]$ represents the number of operations executed from site i at site S.

Definition 10 *For two time vectors u, v*
$u \leq v$ *iff* $\forall i : u[i] \leq v[i]$
$u < v$ *iff* $u \leq v$ *and* $u \neq v$
$u \parallel v$ *iff* $\neg(u < v)$ *and* $\neg(v < u)$

Notice that $\leq$ and $<$ are partial orders. The *concurrency relation* $\parallel$ is reflexive and symmetric. The above definition gives us a very simple method to decide whether two events e and e' are causally related or not: We take their timestamps (vector times) and check whether $C(e) < C(e')$ or $C(e') < C(e)$ where $C(x)$ determines the timestamp of x. If the test succeeds, the events are causally related. Otherwise they are independent.

References

[1] G. Oster A. Imine, P. Molli and M. Rusinowitch. Development of transformation functions assisted by a theorem prover. September 2003.

[2] G. Oster A. Imine, P. Molli and M. Rusinowitch. Proving the correctness of transformation functions in real-time groupware. 2003.

[3] G. Oster A. Imine, P. Molli and M. Rusinowitch. Achieving convergence with operational transformation in distributed groupware systems. May 2004.

[4] Michael J. Knister Atul Prakash. A framework for undoing actions in collaborative systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, pages 295–330, 1994.

[5] Zimmermann H. Banino J.S., Kaiser C. Synchronization for distributed systems using a single broadcast channel. *Proc. 1st Int. Conf. on Distributed Computing Systems*, 10 1979.

[6] S. J. Gibbs C. A. Ellis. Concurrency control in groupware systems. *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 399–407, 6 1989.

[7] Yanchun Zhang Chengzhen Sun, Yun Yang and Xiaohua Jia. Real-time cooperative editing on the internet. 2001.

[8] Marco Cicolini. Gclipse - a collaborative editor plug-in for eclipse. 2 2005.

[9] Gordon V. Cormack. A calculus for concurrent update. *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, 1995.

[10] Reed D.P. and Kanodia R.K. Synchronisation with eventcounts and sequencers. *Commun. ACM*, 22:115–123, 2 1979.

[11] Le Lann G. Algorithms for distributed data sharing systems which use tickets. *Proc. 3rd Workshop on Distributed Data Management and Computer Networks*, 8 1978.

[12] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. 1978.

[13] Du Li and Rui Li. Ensuring consistency on arbitrary transformation paths in real-time group editors. August 2003.

[14] Du Li and Rui Li. Ensuring consistency in real-time group editors. *ACM Transactions on Computer-Human Interaction*, April 2004.

[15] Du Li and Rui Li. Ensuring content and intention consistency in real-time group editors. 2004.

[16] Rui Li, Du Li, and Chengzheng Sun. A time interval based consistency control algorithm for interactive groupware applications. In *ICPADS '04: Proceedings of the Parallel and Distributed Systems, Tenth International Conference on (ICPADS'04)*, page 429, Washington, DC, USA, 2004. IEEE Computer Society.

[17] Du Lir and Rui Li. Preserving operation effects relation in group editors. *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 457–466, 2004.

[18] Brad Lushman and Gordon V. Cormack. Proof of correctness of ressel's adopted algorithm. *Information Processing Letters*, 86:303–310, 2003.

[19] Doris Nitsche-Ruhland Matthias Ressel and Rul Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 288–297, 1996.

[20] Rul Gunzenhäuser Matthias Ressel. Reducing the problems of group undo. *Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 131–139, 1999.

[21] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. *roceedings of the 8th annual ACM symposium on User interface and software technology*, pages 111–120, 1995.

[22] Jean Ferrie Nicolas Vidot, Michelle Cart and Maher Suleiman. Copies convergence in a distributed real-time collaborative environment. *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 171–180, 2000.

[23] Haifeng Shen and Chengzheng Sun. Flexible notification for collaborative systems. *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 77–86, 2002.

[24] Maher Suleiman, Michele Cart, and Jean Ferrie. Serialization of concurrent operations in a distributed collaborative environment. *Proceedings of the international ACM SIGGROUP conference on Supporting group work : the integration challenge: the integration challenge*, pages 435–445, 1997.

[25] Maher Suleiman, Michele Cart, and Jean Ferrie. Concurrent operations in a distributed and mobile collaborative environment. *Proceedings of the Fourteenth International Conference on Data Engineering*, pages 36–45, 1998.

[26] Chengzhen Sun. Undo as concurrent inverse in group editors. *ACM Transactions on Computer-Human Interaction (TOCHI)*, pages 309–361, December 2002.

[27] Chengzheng Sun and Clarence Ellis. Operational transformation in real-time group editors:ăissues, algorithms, and achievements. *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68, 1998.

[28] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, pages 63–108, 1998.

[29] Gordon V.Cormack. A counterexample to the distributed operational transform and a corrected algorithm for point-to-point communication. 8 95.

[30] Ali A. Zafer, Clifford A. Shaffer, Roger W. Ehrich, and Manuel Perez. Netedit: A collaborative editor. 2001.

[31] Ali A. Zafer, Clifford A. Shaffer, Roger W. Ehrich, and Manuel Perez. Netedit: A collaborative editor (master thesis). 2001.