

A Multi-versioning Algorithm for Intention Preservation in Distributed Real-time Group Editors

Liyin Xue and Mehmet Orgun

Department of Computing
Macquarie University
Sydney, NSW 2109, Australia

{lyxue, mehmet}@ics.mq.edu.au

Kang Zhang

Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083-0688, USA

kzhang@utdallas.edu

Abstract

Intention preservation is an important aspect of consistency maintenance in real-time collaborative editing systems. The multi-version approach is able to preserve individual users' concurrent conflicting intentions in a consistent way. Various multi-versioning schemes have been proposed. In this paper, a version composition algorithm is devised for a new multi-versioning scheme that can preserve users' contextual intentions.

Keywords: Intention preservation, multi-versioning, consistency maintenance, group editors.

1 Introduction

Real-time group editors involve a group of geographically distributed users connected by a communication network, e.g., the Internet, who cooperate with each other by concurrently manipulating shared objects (Prakash 1999). They are designed to support the cooperation and collaboration processes between co-authors.

Groupware systems in general, group editors in particular, have inherited many consistency maintenance mechanisms from traditional database systems, where the basic principle is conflict prevention (Dourish 1996). Generally, this is in contradiction to the spirit of collaboration. For example, locking is a widely implemented mechanism in both database systems and groupware systems (Greenberg and Marwood 1994). Locking disallows multiple users to modify the same entity (object) at the same time. On the one hand, this will diminish collaboration if the lock granularity is coarse. On the other hand, fine-grained locking will cause much system or *user overhead*, which is the extra efforts required from end users but not necessary in single-user systems.

With the development of the Internet, online wide area collaboration has become popular. The fundamental challenge to groupware researchers is not the limited bandwidth available (and the delay caused by routers and

operating systems) but the constant speed of light (Yang et al. 2000). Inter-continental communication delay cannot be reduced to being lower than the physical limit. This has a significant impact on the design of groupware systems. First, with centralised groupware systems, users cannot see their own actions immediately, since a message has to go through a round trip. Therefore, local commands (including lock requests) are not of *responsiveness*. Second, remote users cannot see immediately what other users have performed. The WYSIWIS (What You See Is What I See) property no longer holds (Munson and Dewan 1996). Therefore, on the one hand locking is undesirable; on the other hand conflicts seem to be inevitable without locking.

A replicated architecture is usually adopted for maintaining the property of responsiveness, i.e., the effect of a user's action is seen by herself with a negligible delay. Each site has a copy of the shared document under editing. Local operations are executed immediately and then multicast to remote sites. If unconstrained editing is supported in this kind of environments, conflicts will be inevitable.

We believe that conflicts may not necessarily be destructive to the collaboration process. Instead, they contribute to the emergence of a group consensus (or *group intention*). Since it is generally infeasible for the systems to have the knowledge necessary to properly resolve conflicts among users, the systems must be capable of reflecting all aspects of human conflict and cooperation and facilitating human users in conflict resolution. To put it in another way, the systems must preserve individual users' intentions (represented by the operations they issue) that may be conflicting with each other, and provide facilities to help users reach a consensus (Xue, Zhang, and Sun 2000).

The challenge we face is how to maintain the unconstrained multiple individual users' intentions such that all replicas (copies) of the document converge while each user's intentions are preserved. The *multi-version approach* has been proposed to accommodate individual users' concurrent conflicting intentions in multiple versions in the Tivoli (Moran et al. 1995) and GRACE (Chen and Sun 1999, Sun and Chen 2002) systems. However, there are some limitations associated with the current multi-versioning schemes. To overcome those limitations, we have proposed a *contextual intention oriented multi-versioning scheme* that is capable of supporting *editing any version at any time* and preserving

users' intentions (Xue, Orgun, and Zhang 2002). The scheme has been implemented in Java in a research prototype called POLO (standing for P_Ost-L_Ocking, an approach we have proposed for conflict management in (Xue, Zhang, and Sun 2000)).

POLO is a distributed object-based real-time collaborative editor, where a document is modelled by a set of independent objects. An object (e.g., rectangle, ellipse, line, and textbox) in the document is created and deleted by *Create* and *Delete* operations respectively. Each object has a set of attributes such as type, size, position, colour, etc. The attribute values can be changed by *modification* (or *updating*) operations such as *Move* and *Resize*. POLO preserves individual users' concurrent conflicting intentions by multi-versioning and facilitates the negotiation for group intentions by post-locking. This paper presents the multi-versioning algorithm implemented in POLO.

The rest of this paper is organised as follows. We start with a discussion of related work and an examination of the problems in existing work in Section 2. Section 3 introduces the contextual intention oriented multi-versioning scheme. We discuss the design of its version composition algorithm in Sections 4 and 5. An integrated example is presented in Section 6 to illustrate the multi-versioning process. Finally, Section 7 summarises the major contributions of this paper.

2 Related Work

In object-based collaborative editing systems, if two updating operations targeting the same object have been generated from different sites concurrently, they may be executed in different orders at different sites. This may lead to divergent document states at the participating sites. There are some *automatic conflict resolution mechanisms* having been proposed to achieve convergence (i.e., identical document states at all the participating sites). With the *serialization* mechanism implemented in GroupDesign (Karsenty, Tronche, and Beaudouin-Lafon 1993) and LICRA (Kanawati 1997), one of the operations may need to be undone and re-executed in the correct global order (i.e., a *total order* relation of operations) such that replica convergence is guaranteed. In the SYNC framework, the automatic conflict resolution is based on a *merge matrix* rather than a total order (Munson and Dewan 1997). A merge matrix consists of rows and columns labelled by the operations (methods) for an object (class). The merge matrix entry identified by a particular row operation and column operation determines the action the system will take – typically accepting one operation or the other.

The major problem of the automatic conflict resolution approach is *intention violation*. For example, if two users concurrently change the colour of an object to “Red” and “Green” respectively, the final execution effect will be either “Red” or “Green”, depending on the global order (or the merge matrix) defined, thus the intention of one of the two users will be violated.

The only way to preserve the intentions of the concurrent conflicting operations is to make new versions of the

object, and then apply them to the new versions separately. The execution effect is that no conflicting operations are applied to the same version. This is the multi-version approach proposed in the Tivoli and GRACE systems, which can be characterized as *computer supported conflict resolution by human users*. Since GRACE is more recent and elaborated, we briefly introduce its multi-versioning scheme in the remainder of this section.

Given a finite set OP of operations of a collaborative editing session and a set of collaborating sites: $Site = \{1, \dots, N\}$, where N is the number of sites, for any operation O in OP , the following notations are introduced: (1) $Sid(O)$ denotes the identifier of the site where the operation O is generated; (2) $Target(O)$ denotes the identifier of the object which operation O is generated to target; (3) $Att.Key(O)$ denotes the attribute key (or type) targeted by O ; (4) $Att.Value(O)$ denotes the attribute value the operation O is used to change to; and (5) $Oid(O)$ denotes the identifier of O . Each operation is time-stamped based on a vector clock called *state vector* at its originating site (Ellis and Gibbs 1989). The time-stamps of operations are necessary for identifying operations and determining the causal relationship between any two operations (Raynal and Singhal 1996).

In accordance with Lamport's work (1978), a causal ordering relation of operations can be defined as follows (Sun et al. 1998):

Definition 2.1: Causal ordering relation “ \rightarrow ”: Given any operations O_a and O_b in OP generated at sites i and j respectively, O_a is said to be causally preceding O_b , denoted by $O_a \rightarrow O_b$, if (1) $i = j$ and the generation of O_a happened before the generation of O_b , or (2) $i \neq j$, and the execution of O_a at site j happened before the generation of O_b , or (3) there exists an operation O_x , such that $O_a \rightarrow O_x$ and $O_x \rightarrow O_b$. Where *happen before* is well defined in terms of local physical clocks. \square

Definition 2.2: Dependent and independent operations: Given any operations O_a and O_b in OP . (1) O_b is said to be dependent on O_a if $O_a \rightarrow O_b$. (2) O_a and O_b are said to be independent (or *concurrent*), expressed as $O_a \parallel O_b$, if neither $O_a \rightarrow O_b$ nor $O_b \rightarrow O_a$. \square

A pair of *conflict and compatible relations* has been defined in GRACE. Due to the introduction of *spatial (semantic) relations* (see Section 3), we treat the relations presented in GRACE as special types of *intentional relations*, and change the symbols from \otimes and \odot (used in GRACE) to \otimes_{G1} and \odot_{G1} respectively.

The *intentional relations* of operations defined in GRACE are as follows (Sun and Chen 2000).

Definition 2.3 Conflict relation “ \otimes_{G1} ”: Given O_a, O_b in OP , they conflict with each other, expressed as $O_a \otimes_{G1} O_b$, if (1) $O_a \parallel O_b$; (2) $Target(O_a) = Target(O_b)$; (3) $Att.Key(O_a) = Att.Key(O_b)$; and (4) $Att.Value(O_a) \neq Att.Value(O_b)$. \square

Definition 2.4: Compatible relation “ \odot_{G1} ”: Given O_a, O_b in OP , if they do not conflict with each other, they are compatible, expressed as $O_a \odot_{G1} O_b$. \square

The effects of compatible operations can be integrated into the same object without causing intention violation. In a highly concurrent real-time cooperative editing environment, as a group of operations may have rather arbitrary and complex intentional relationships, their combined effect can be expressed by a set of *maximal compatible groups*. Each maximal compatible group (denoted by *mcg*) contains a set of mutually compatible operations. For every pair of maximal compatible groups there exists at least one pair of operations that are conflicting with each other. A *maximal* compatible group will not be a subset of any other in the current set of maximal compatible groups.

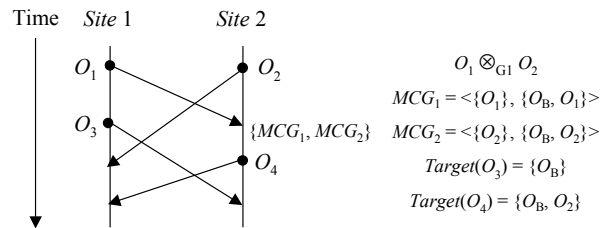
Each maximal compatible group corresponds to an object version (denoted by *MCG*). A Consistent Object Identification (COID) scheme has been defined in GRACE to locate and trace the created versions. The identifier of any object version *MCG*, denoted by $Vid(MCG)$, consists of a set of identifiers of operations, each of which either created *MCG* or has been applied to *MCG*, and is conflicting with an operation which has been applied to some other version. Therefore, each version can be represented by a tuple consisting of a maximal compatible group and an identifier.

Since the operations involved in a conflict are not known in advance, and the users need to know and edit the current versions of the object in conflict, the versions should be constructed incrementally. Consequently, when an operation *O* arrives at a remote site, it may need to be applied to a *collection* of object versions (we called them *target versions*). In contrast, the object that an operation is generated to target is referred to as *targeted object* in order to create new versions. The collection is called the *application scope* of *O*. A Target Object Version Recognition scheme (TOVER) has been designed to locate these versions, which specifies that any version *MCG* within the application scope must have an identifier that is a superset of the identifier of the operation's targeted object, i.e., $Target(O) \subseteq Vid(MCG)$. Furthermore, an algorithm called MOVIC has been devised to incrementally compose maximal compatible groups.

In GRACE, after the first conflict occurs, users are allowed to further edit the versions created from the *base object*. Since versions may be created in different orders at different sites, the targeted version of one operation (O_b) may be a sub-version of the targeted version of a concurrent operation (O_a), that is $Target(O_a) \subset Target(O_b)$. If they change the same attribute to different values, they must also be defined as conflicting.

For example, O_1 and O_2 intentionally conflict with each other (concurrently changing the same attribute to different values) in Figure 1. When they are applied to the same object, two new versions will be created to accommodate their intentions respectively, i.e., MCG_1 and MCG_2 . O_3 is generated to target the base object before the conflict occurs, whereas O_4 is generated to target one of the versions, i.e., MCG_2 . Since the identifier of operation O_2 has been added to the identifier of O_4 's targeted version, we have $Target(O_3) \subset Target(O_4)$.

Obviously, O_3 and O_4 must be defined as intentionally conflicting if they target the same attribute and change it to different values. Nevertheless, this is not defined in Definitions 2.3 and 2.4. A new pair of conflict and compatible relations has been introduced in GRACE to handle the above-mentioned problem.



Note: MCG_1 and MCG_2 are the versions created after the execution of O_1 at Site 2. The first component of the version (either MCG_1 or MCG_2) is its compatible group, and the second its identifier. For simplicity, we use a set of operations to denote a set of identifiers of operations in the versions, e.g., $\{Oid(O_B), Oid(O_1)\}$ is simplified as $\{O_B, O_1\}$, where $\{Oid(O_B)\}$ is the identifier of the base object, and O_B is the operation which created it.

Figure 1. Editing scenario A

The correctness of the scheme and algorithm for the composition of maximal compatible groups has been formally proven *when the set of operations involved in the conflicts are generated to target the same base object* and the intentional relationships are defined as in Definitions 2.3 and 2.4 (i.e., \otimes_{G1} and \odot_{G1}). However, the correctness of the overall multi-versioning scheme (when the intentional relationships between operations targeting different versions of the same object are taken into account) has not yet been fully verified. In other words, it has not yet been fully verified whether the multi-versioning scheme can preserve users' intentions and guarantee convergence (Sun and Chen 2002).

According to a mathematical theorem in set theory, we have separated the discussions on convergence and intention preservation by generalising the version composition scheme proposed in GRACE (Xue et al. 2001, Xue, Orgun, and Zhang 2002). This is briefly introduced as follows:

A binary relation over a set is called *compatibility relation* if it is reflexive and symmetric. Given a set of operations OS , various compatibility relations can be defined, and \odot_{G1} is just an example. However, for any given compatibility relation (denoted by \mathfrak{R}) defined over OS , there exists a unique set of maximal compatible groups. If we assume all operations are executed in the causal order (concurrent operations can be executed in any order), the execution effects of any two maximal compatible groups containing the same set of operations are identical. Therefore, after the execution of the same set of operations, all participating sites see the same set of maximal compatible groups, and thus the same execution effect. In other words, convergence is guaranteed as long as the intentional compatible relation defined is a compatibility relation. We will use $VerCom(\mathfrak{R})$ to represent the general version composition scheme. The versions composed are called *maximal versions*.

Although the $VerCom(\mathfrak{R})$ scheme can guarantee a globally convergent object state, the intentions of users

may not be preserved. Therefore, in order to preserve the intentions of users, we need to constrain the compatibility relation with other conditions. To put it in another way, a compatibility relationship is necessary but not sufficient. This is the starting point of our new multi-versioning scheme (Xue, Orgun, and Zhang 2002). And this scheme calls for a new multi-versioning algorithm that is to be presented in this paper.

3 Contextual Intention Oriented Multi-Versioning Scheme

In this section, we review our multi-versioning scheme. We focus on how the scheme preserves users' intentions.

3.1 Preservation of Concurrent Conflicting Intentions

Preserving concurrent conflicting intentions is supposed to be the basic functionality of the multi-version approach proposed in Tivoli and GRACE. Although the definition of a conflict relation may be artificial, it is generally infeasible to apply two semantically (concurrent) conflicting operations to the same object without causing intention violation. They must always be defined as conflicting with each other. For example, those operations satisfying \otimes_{G1} are in conflict as defined in the GRACE system.

With the introduction of multi-versioning, an operation received from a remote site may be required to target multiple versions that shared the same base object with the targeted object of the operation. Therefore, we introduce spatial relations to represent the semantic relationships among operations targeting different versions of the same base object.

Definition 3.1: Spatial relations¹: Given O_a and O_b in OP , (1) $O_a \boxtimes_0 O_b =_{\text{def}} (Target(O_a) \cap Target(O_b) \neq \emptyset) \wedge (Att.Key(O_a) = Att.Key(O_b))$, (2) $O_a \boxdot_0 O_b =_{\text{def}} \neg(O_a \boxtimes_0 O_b)$, where “ \neg ” denotes logical negation. Notations \boxtimes_0 and \boxdot_0 are called *spatial conflict* and *spatial compatible* relations over OP respectively. \square

If the targeted objects (versions) of the two operations share the same base object (i.e., $Target(O_a) \cap Target(O_b) \neq \emptyset$) and modify the same attribute, they are in conflict in terms of \boxtimes_0 .

Based on the above definitions of spatial relations of operations, basic intentional relations of operations can be defined as follows.

Definition 3.2: Basic intentional relation: Given O_a and O_b in OP , they are said to be *basically conflicting* with each other, expressed as $O_a \otimes_0 O_b$, if (1) $O_a \parallel O_b$; and (2) $O_a \boxtimes_0 O_b$; they are *basically compatible* with each other, expressed as $O_a \odot_0 O_b$, if $\neg(O_a \otimes_0 O_b)$. \square

According to the above definition, \odot_0 is a compatibility relation. If two basically conflicting operations are applied to the same version, intention violation will occur. The execution effect depends on their execution orders. Therefore, their intentional relationship must be defined as conflicting in order to preserve concurrent conflicting intentions. \otimes_{G1} is only a subset of \otimes_0 .

3.2 Preservation of Selecting Intentions

If a new operation is generated by a user to target a particular version (called *intended version*) of an object, it is natural that the operation is compatible with all the operations having been applied to the version; otherwise it could not have been generated. For example, O_4 is compatible with O_2 in Figure 1. All the other versions, which are called *unintended versions*, of the object must have some operations that are not applied to the targeted version. In other words, these operations are conflicting with at least one of the operations in the targeted version. Having seen their execution effects, the user who generates the new operation disagrees with the effects. Hence, from the user's point of view, the new operation conflicts with those operations, and can only be applied to the targeted version at the local site where it is generated. For example, O_4 must be defined as conflicting with O_1 (although they may be spatially compatible), since the user at Site 2 selects version MCG_2 rather than MCG_1 . The two operations must not be applied to the same version since this is not intended by the user. This *selecting intention* is determined by the individual user's decision, and must be preserved by the version composition scheme to be devised.

From the efficient version identification viewpoint, it is desirable that a version identification scheme can guarantee that the application of a new local operation O_{new} to its targeted version will not change the identifiers of any versions in the current version set, otherwise, all the identifiers of operations generated after a conflict has occurred will be used to identify the versions. However, O_{new} must be defined as conflicting with those operations that are in the unintended versions but not in the intended version in order to meet the requirement for the preservation of selecting intentions. Therefore, it is useful to differentiate those conflicts (i.e., direct conflicts) that create new versions from those (indirect conflicts) that do not. The above conflicts do not create new versions and the identifier of O_{new} is not necessary in identifying the versions.

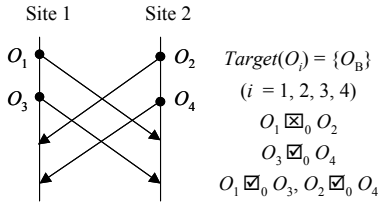
3.3 Preservation of Contextual Intention

An object is characterized by its identifier and state, which can be represented by a set of compatible operations executed in the causal order. Applying an operation to an object state results in a new state. A current state of an object is the application of a sequence of operations to the initial state of the object. Given an operation O , the *generation context* of O , denoted by $GC(O)$, is the object state at the originating site to which O is issued (i.e., the state of the targeted object); the *execution context* of O , denoted by $EC(O)$, is the object state to which O is to be executed (i.e., the effect of O is

¹ In POLO, we ignore the attribute value part of the definitions, i.e., two operations targeting the same object and changing the same attribute to the same value are defined as conflict. This is related to our undo approach. However, all the discussions and conclusions presented in this article are applicable to both kinds of definitions.

to be included in the object), and the *target context* of O , denoted by $TC(O)$, is the object state at executing sites, which O needs to target. Due to multi-versioning, O may have more than one target context, that is, it needs to target multiple existing versions, which share the same base object with the targeted object of O , and may be applied to some of them or to a new version. All of the above contexts can be represented by a set of operations. If $TC(O) = GC(O)$, then $EC(O) = TC(O)$, and directly applying O to the target context will not cause an intention violation. Due to concurrent execution of operations, some of the $TC(O)$'s (or target versions) may not be equal to $GC(O)$. For the case $TC(O) \supset GC(O)$, if O is intentionally compatible with all those concurrent operations belonging to $TC(O) - GC(O)$, it can be directly applied to (or included into) $TC(O)$; if it intentionally conflicts with some operation in $TC(O) - GC(O)$, new versions will be created. O can only be applied to a version (either existing or newly created) with all operations being compatible with O . If O is a *causally ready* operation (an operation is said to be causally ready if all the operations preceding it have been executed) and $TC(O) \supseteq GC(O)$ is not true, then the object version represented by $TC(O)$ does not contain $GC(O)$, thus applying O to it is unintended by the user who issues O . Therefore, O must not be applied to it.

Generally when an operation O is to be executed at a remote site, it must be confined to those target versions that contain all the operations in its generation context. This requirement is referred to as *contextual intention preservation property* (Xue, Orgun, and Zhang 2002).



(a) Editing scenario B

\mathfrak{R}	O_1	O_2	O_3	O_4
O_1	\odot_C	\otimes_{DC}	\odot_C	\otimes_{IC}
O_2		\odot_C	\otimes_{IC}	\odot_C
O_3			\odot_C	\otimes_{IC}
O_4				\odot_C

$$MCG_1: \langle \{O_1, O_3\}, \{O_B, O_1\} \rangle$$

$$MCG_2: \langle \{O_2, O_4\}, \{O_B, O_2\} \rangle$$

(b) Intentional relationships and versions

Figure 2. An illustration of contextual intention preservation

In Figure 2, O_1 and O_2 are spatially conflicting with each other, thus two versions will be created after their execution. Both O_3 and O_4 are generated to target the base object (with the same identifier but different states) before the conflict occurs, and they are spatially compatible. In GRACE, O_3 and O_4 are defined as being intentionally compatible, and the two maximal compatible groups composed for the editing scenario B in

Figure 2(a) are $\{O_1, O_3, O_4\}$ and $\{O_2, O_3, O_4\}$. Obviously, O_3 has been included into the second maximal compatible group that does not contain O_1 . Therefore, GRACE does not maintain the contextual intention preservation property. The problem comes from the fact that there is a pair of conflicting operations (i.e., O_1 and O_2), which are in the generation contexts of O_3 and O_4 respectively, and both O_3 and O_4 have been applied to an (implicitly) unintended version. Although the issuer of O_3 (or O_4) has not seen the conflict explicitly, her/his implicit decision should be taken into account. Therefore, O_3 and O_2 (similarly O_4 and O_1) must be defined as “indirectly” conflicting. This is caused by the “direct” conflict between O_1 and O_2 . In this sense, the contextual intention preservation property is related to the selecting intention preservation property.

3.4 Definitions of Intentional Relationships

The key point of our contextual intention oriented multi-versioning scheme is the definition of the direct conflict relation. Any two operations that directly conflict with each other must be concurrent and spatially conflicting (or basic-intentionally conflicting). Moreover, there must not exist a pair of conflicting operations between their targeted versions (or generation contexts), and neither of them intentionally conflicts with an operation in another’s generation context (otherwise, they are indirectly conflicting). Therefore, the intentional relationship between two operations depends on not only their basic intentional relationship but also the intentional relationships of some other operations. This leads to the following recursive definitions of intentional relations (Xue, Orgun, and Zhang 2002).

Definition 3.3: Direct conflict relation “ \otimes_{DC} ”: Given O_a, O_b in OP , $Target(O_a) \cap Target(O_b) \neq \emptyset$, O_a is said to be directly conflicting with O_b , expressed as $O_a \otimes_{DC} O_b$, if (1) $O_a \otimes_0 O_b$, and (2) $(\forall O_x) (\forall O_y) (F(O_x, O_y, O_a, O_b) \Rightarrow O_x \odot_0 O_y)$, where $F(O_x, O_y, O_a, O_b) = (O_x \in GC(O_a) \cup \{O_a\}) \wedge (O_y \in GC(O_b) \cup \{O_b\}) \wedge (\{O_x\} \cup \{O_y\} \neq \{O_a, O_b\})$, and “ \Rightarrow ” denotes logical implication. The subscript C denotes the “Contextual intention oriented scheme”.

For any two sets of operations OS_1, OS_2 , and a symmetric binary relation R , if we use $OS_1 R OS_2$ to represent $(\forall O_i) (\forall O_j) (O_i \in OS_1 \wedge O_j \in OS_2 \Rightarrow O_i R O_j)$, then the second part of the definition can be expressed as $(GC(O_a) \odot_0 GC(O_b)) \wedge (\{O_a\} \odot_0 GC(O_b)) \wedge (\{O_b\} \odot_0 GC(O_a))$. This is an intuitively more appealing representation.

Definition 3.4: Intentional conflict relation “ \otimes_C ”: Given O_a, O_b in OP , $Target(O_a) \cap Target(O_b) \neq \emptyset$, O_a is said to be intentionally conflicting with O_b , expressed as $O_a \otimes_C O_b$, if $O_a \otimes_{DC} O_b \vee (\exists O_x) (\exists O_y) (F(O_x, O_y, O_a, O_b) \wedge O_x \otimes_C O_y)$.

Definition 3.5: Intentional compatible relation “ \odot_C ”: Given O_a and O_b in OP , $Target(O_a) \cap Target(O_b) \neq \emptyset$, they are intentionally compatible with each other, expressed as $O_a \odot_C O_b$, if $\neg(O_a \otimes_C O_b)$.

Definition 3.6: Indirect conflict relation “ \otimes_{IC} ”: Given O_a, O_b in OP , $Target(O_a) \cap Target(O_b) \neq \emptyset$, O_a is said to

be indirectly conflicting with O_b , expressed as $O_a \otimes_{IC} O_b$, if $O_a \otimes_C O_b \wedge \neg (O_a \otimes_{DC} O_b)$.

Obviously, \otimes_C is a compatibility relation. Therefore, we can now assume the version composition scheme is $\text{VerCom}(\otimes_C)$ in the sequel. Under this scheme, every $GC(O)$ presented in the definitions becomes a maximal compatible group, and can be replaced with $mCG(O)$, which denotes the maximal compatible group of the maximal version that O is generated to target.

The direct conflict relation is introduced to detect those operations that create new versions. No new version will be created due to an indirect conflict. Therefore the identifiers of directly conflicting operations are sufficient in identifying the versions.

The $\text{VerCom}(\otimes_C)$ scheme can maintain the properties of selecting intention preservation and contextual intention preservation (Xue et al. 2001). For example, in Figure 2(a), although both O_3 and O_4 are generated to target the base object, and are spatially compatible, from the definitions, they are indirectly conflicting with each other, since there exists a pair of operations (O_1 and O_2) in their respective targeted versions (generation contexts), i.e., $O_1 \in GC(O_3)$, $O_2 \in GC(O_4)$, which are directly conflicting with each other. The versions created are given in Figure 2(b). Obviously the contextual intentions are preserved, and $Oid(O_3)$ and $Oid(O_4)$ are not included in the versions.

The properties of the contextual intention oriented scheme have been formally verified (Xue et al. 2001). However, the technical detail of version composition and identification has not yet been fully discussed, which is the focus of this paper.

4 Target Version Location Scheme

Suppose the maximal version set for a set OS of operations is $MCGS$. Our task is to construct the new maximal version set $MCGS'$ for $OS \cup \{O_{new}\}$ from the current $MCGS$, where O_{new} is a new causally ready operation with respect to OS . To this end, we first need to locate the set of versions (i.e., $MCGS$) from the current objects in the document.

Definition 4.1: Target Version Location scheme (VerLoc_I): Suppose the set of objects (or versions) contained in the current document is represented by DOC and O_{new} is a causally ready operation. The set of maximal versions which O_{new} needs to target, denoted by $MCGS_I(O_{new})$, can be expressed as $\{MCG \mid MCG \in DOC \wedge (Target(O_{new}) \cap Vid(MCG)) \neq \emptyset\}$. \square

The above scheme defines the current set of maximal versions (i.e., $MCGS = MCGS_I(O_{new})$). For any $\text{VerCom}(\mathfrak{R})$ scheme (e.g., $\text{VerCom}(\otimes_{G1})$), it is generally necessary to compare an operation with *all operations* in OS to obtain their intentional relationships and construct the consistent set of maximal compatible groups. In other words, a causally ready operation O_{new} needs to target every version of the $MCGS$.

Since the $\text{VerCom}(\otimes_C)$ scheme maintains the property of selecting intention preservation, we can directly apply a

local operation to its targeted version (or intended version) without considering the other versions. Furthermore, as our $\text{VerCom}(\otimes_C)$ scheme maintains the contextual intention preservation property, it is not necessary to target a remote operation to all existing versions sharing the same base object with the operation's targeted object in order to incrementally create maximal compatible groups. To put it in another way, the integration of $\text{VerCom}(\otimes_C)$ with TOVER will be sufficient for the purpose of convergence of object states (but not identifiers). Therefore, we can formally define another target version location scheme that is equivalent to TOVER as follows:

Definition 4.2: VerLoc_{II}: Given a maximal version set $MCGS$ constructed out of operation set OS , and O_{new} is a causally ready operation with respect to OS (for any $O \in OS$, $Target(O) \cap Target(O_{new}) \neq \emptyset$), the set of maximal versions which O_{new} needs to target, denoted by $MCGS_{II}(O_{new})$, can be expressed as $\{MCG \mid Target(O_{new}) \subseteq Vid(MCG) \wedge MCG \in MCGS\}$. Particularly if O_{new} is a local operation, $MCGS_{II}(O_{new}) = \{<mCG(O_{new}), Target(O_{new})>\}$. \square

In fact, the above target versions can be located directly from the set of objects in the current document rather than from the set of maximal versions involved in the conflicts, which are unknown before the execution of O_{new} . We present the scheme in this way such that all the objects except those involved in the conflicts can be ignored.

It is worth noting that convergence of maximal compatible groups cannot be guaranteed by the integration of $\text{VerCom}(\otimes_{G1})$ and VerLoc_{II} (or TOVER).

Since some versions in the $MCGS_{II}(O_{new})$ may not contain O_{new} 's generation context, the VerLoc_{II} scheme can be optimized if the knowledge of O_{new} 's generation context is available.

Definition 4.3: VerLoc_{III}: Given a maximal version set $MCGS$ constructed out of operation set OS , and O_{new} is a causally ready operation with respect to OS (for any $O \in OS$, $Target(O) \cap Target(O_{new}) \neq \emptyset$), the set of maximal versions which O_{new} needs to target, denoted by $MCGS_{III}(O_{new})$, can be expressed as $\{MCG \mid mCG(O_{new}) \subseteq Ocg(MCG) \wedge MCG \in MCGS\}$. Particularly if O_{new} is a local operation, $MCGS_{III}(O_{new}) = \{<mCG(O_{new}), Target(O_{new})>\}$. Where $Ocg(MCG)$ denotes the maximal compatible group of version MCG . \square

VerLoc_{III} specifies that it is not necessary to target an operation to those versions that do not contain its generation context. Obviously $MCGS_{III}(O_{new})$ is a subset of the one specified in Definition 4.2.

5 A Generic Incremental Version Composition Algorithm

In GRACE, an algorithm called MOVIC has been devised to construct compatible groups incrementally (Sun and Chen 2002). It is designed for the intentional relationships \otimes_{G1} and \otimes_{G1} , which can be determined directly by comparing the temporal and semantic

relationships of operations. Theoretically, the algorithm is applicable to any multi-versioning scheme based on $\text{VerCom}(\mathfrak{R})$ if the VerLoc_i scheme is integrated. Unfortunately, the algorithm is highly inefficient in our case. This is due to the fact that the intentional relationship between a pair of operations depends on not only their temporal relationship and semantics but also the intentional relationships between those operations contained in their respective generation contexts. This problem of recursion can be solved by taking into account the contextual intention preservation property.

From the definitions of intentional relations in Section 3, it is clear that we need to reconstruct O_{new} 's generation context in order to determine the intentional relationships between O_{new} and the operations contained in the target versions. We can make use of the temporal information of the operations contained in a version by storing them in a list data structure. That is to say each maximal compatible group is represented by a list, where operations are arranged in the causal order. We refer to a causally ordered maximal compatible group as *maximal compatible list* (however, for convenience, we still use *mcg* to represent it, and assume both the set and list operators apply). From the definition of causal ordering, a causally ordered list has the following property.

Lemma 5.1: For any causally ordered list L , if $i < j$, then $(L[i] \parallel L[j]) \vee (L[i] \rightarrow L[j])$, where $i, j \in \{1, 2, \dots, |L|\}$ (or denoted by $i, j \in [1, |L|]$).

5.1 Reconstruction of Generation Context

From the contextual intention preservation property, a causally ready operation O_{new} is only to be applied to versions belonging to $MCGS_{III}(O_{new})$ defined by VerLoc_{III} , and its generation context $mcg(O_{new})$ is contained in these versions. However, since the generation context is unknown when executing O_{new} , we need to use $MCGS_{II}(O_{new})$, which can be easily located. There exists at least one version from $MCGS_{II}(O_{new})$, whose *mcg* is a superset of $mcg(O_{new})$. Obviously, $mcg(O_{new}) \rightarrow \{O_{new}\}$. Therefore, for each version in $MCGS_{II}(O_{new})$, we need to differentiate between those operations preceding O_{new} and those operations concurrent with O_{new} .

A maximal compatible list can be divided into two sub-lists: a preceding operation list (PR) and a concurrent operation list (CC). All operations in PR precede O_{new} , whereas all operations in CC are concurrent with O_{new} . Temporal information associated with the operations is sufficient for this purpose. This can be completed by Algorithm 5.1.

Algorithm 5.1: $PR_CC(L, O): \langle PR, CC \rangle$

Input: L is a list of operations $[O_1, \dots, O_m]$, which are arranged in accordance with the causal order; O is a causally ready operation with respect to the operations in L .

Output: PR and CC are lists of operations (in the causal order) that are causally preceding and concurrent with O respectively, i.e., $PR \rightarrow \{O\}$ and $\{O\} \parallel CC$.

```

{
  PR := [];
  CC := [];
  For i = 1 to |L|
    If L[i] → O
      PR := PR + [L[i]];
    Else
      CC := CC + [L[i]];
  End for
  Return <PR, CC>;
}

```

The above algorithm is straightforward. We present it as an independent block mainly for the sake of the generality of our multi-versioning algorithm to be presented later, i.e., it may be overridden in order to be applicable to other application domains.

Based on the above discussion, a function ($GContext$) can be easily designed to obtain the set (list) of operations contained in an operation's targeted version as in Algorithm 5.2.

Algorithm 5.2: $GContext(MCGS, O): mcg(O)$

Input: $MCGS$: a maximal version set constructed from an operation set OS ; O : a causally ready operation with respect to the operations in OS .

Output: $mcg(O)$: O 's generation context.

```

{
  mcg(O) := [];
  MCGS_{II}(O) := {MCG | (Target(O) ⊆ Vid(MCG) ∧ MCG ∈ MCGS)};
  For all MCG_i ∈ MCGS_{II}(O) (i = 1, ..., |MCGS_{II}(O)|)
    <PR, CC> := PR_CC(Ocg(MCG_i), O);
    If PR ⊃ mcg(O), then mcg(O) := PR;
  End for
  Return mcg(O);
}

```

Now that $mcg(O_{new})$ is available, the set of versions which O_{new} needs to target can be restricted to $MCGS_{III}(O_{new})$.

5.2 Determination of Intentional Relationships

There are three possible results of applying O_{new} to a target version MCG (in $MCGS_{III}(O_{new})$) with a compatible group of mcg . If $\{O_{new}\}$ is compatible with mcg , MCG will be updated such that the new compatible group becomes $mcg \cup \{O_{new}\}$. If O_{new} conflicts with all the operations in the mcg , then a new version will be created, whose compatible group contains only one operation, i.e., $\{O_{new}\}$. If O_{new} is compatible with some of the operations in the mcg , then a new version will be created, whose compatible group contains $\{O_{new}\}$ and all those compatible operations.

The central issue of the version composition process is how to divide the operations in the concurrent operation list (CC) of a target version (contained in $MCGS_{III}(O_{new})$) into two sub-lists, i.e., a compatible list (CP) and a conflict list (CF), in order to compose new versions. All operations in the compatible list are intentionally compatible with O_{new} , whereas O_{new} is intentionally conflicting with any operations contained in the conflict list.

Based on the contextual intention preservation property, the generation context of $CC[1]$ is a subset of the one of O_{new} , i.e., $mcg(CC[1]) \subseteq mcg(O_{new})$. Since all the target versions based on VerLoc_{III} contain $mcg(O_{new})$, $CC[1]$ must be intentionally compatible with all the operations in $mcg(O_{new})$. Therefore, we have the following conclusions: if $CC[1] \boxtimes_0 O_{new}$, then $CC[1] \odot_C O_{new}$; if $CC[1] \boxtimes_0 O_{new}$, then $CC[1] \otimes_{DC} O_{new}$.

Generally, based on Lemma 5.1 and the contextual intention preservation property, for any $CC[m+1]$, $m \in [1, |CC|-1]$, its generation context is a subset of $mcg(O_{new}) \cup CC[1, m]$, i.e., $mcg(CC[m+1]) = \{O \mid O \in (mcg(O_{new}) \cup CC[i]) \wedge i \in [1, m] \wedge O \rightarrow CC[m+1]\}$. Since $CC[m+1]$ must be intentionally compatible with the operations in $mcg(O_{new})$, the intentional relationships between O_{new} and those operations in CC can be determined one by one from $CC[1]$ to $CC[|CC|]$. If O_{new} intentionally conflicts with any operation causally preceding $CC[m+1]$, which is contained in the list $[CC[1], CC[2], \dots, CC[m]]$, denoted by $CC[1, m]$, then O_{new} indirectly conflicts with $CC[m+1]$. If O_{new} is intentionally compatible with all the operations that are causally preceding $CC[m+1]$ and contained in the list $CC[1, m]$, then the intentional relationship between O_{new} and $CC[m+1]$ depends on their spatial relationship. If $CC[m+1] \boxtimes_0 O_{new}$, then $CC[m+1] \odot_C O_{new}$; if $CC[m+1] \boxtimes_0 O_{new}$, then $CC[m+1] \otimes_{DC} O_{new}$.

Based on the above discussion, an algorithm for dividing a concurrent list into compatible and conflict lists can be devised as shown in Algorithm 5.3.

Algorithm 5.3: $CP_CF(L, O)$: $\langle CP, CF, DCFSet, EO \rangle$

Input: O is a causally ready operation. $CC = Ocg(MCG) - mcg(O)$, where MCG is a target version of O located by VerLoc_{III}, i.e., $MCG \in MCGS_{III}(O)$. L is a list of operations resulting from a causal ordering of CC .

Output: CP is a causally ordered list of operations contained in the set: $\{O_x \mid O_x \in CC \wedge O_x \odot_C O\}$; CF is a causally ordered list of operations contained in the set: $\{O_x \mid O_x \in CC \wedge O_x \otimes_{DC} O\}$; $DCFSet = \{O_x \mid O_x \in CC \wedge O_x \otimes_{DC} O\}$; EO is the execution form of O .

```

{
  CP := []; CF := [];
  DCFSet := {};
  EO := O;
  If |L| = 0, then return  $\langle CP, CF, DCFSet, EO \rangle$ ;
  For i = 1 to |L|
    If  $(\exists O_x)(O_x \in CF \wedge O_x \rightarrow L[i])$ , then
      CF := CF + [L[i]];
    Else if  $O \boxtimes_0 L[i]$ , then
      CP := CP + [L[i]];
    Else
      CF := CF + [L[i]];
      DCFSet := DCFSet  $\cup$  [L[i]];
  End for
  Return  $\langle CP, CF, DCFSet, EO \rangle$ ;
}

```

The set of directly conflicting operations collected in $DCFSet$ is useful for identifying versions. With the multi-versioning of objects with independent attributes, the execution form of an operation is the same as its original form. We include EO in the algorithm specification for the generality of our version composition algorithm such that it is still applicable to other application domains, such as, text object multi-versioning, where an operation may need to be transformed in order to compensate the effect caused by concurrent operations (considering that an operation such as “insert a character” or “delete a string” will shift the positions of characters in a text object) and the algorithm needs to be overridden (Ellis and Gibbs 1989).

5.3 A Version Composition Algorithm

Now we are ready to present the general version COMPosition and IDentification algorithm (*COMPIDE*). To reduce the complexity of presentation, we omit the version identification process (which is rather simple if the set of directly conflicting operations is available) in the algorithm. Therefore the case of the algorithm’s name is changed to a lower one, i.e., *compide*, which is illustrated in Algorithm 5.4. Its technical details are explained as follows.

If O is a local operation, it can be applied to the targeted version directly. Therefore, we focus on the execution of a causally ready remote operation.

The first step is to locate the set of target versions ($MCGS_{II}(O)$) in terms of the VerLoc_{II} scheme. After the reconstruction of O ’s generation context as in step 2, we can obtain the set of target versions ($MCGS_{III}(O)$) in terms of the VerLoc_{III} scheme. The set of maximal compatible groups in $mcs_2 \cup mcs_3$ corresponds to the set of unintended versions in $MCGS - MCGS_{III}(O)$. Since they do not contain the whole generation context of O , their states (but not identifiers) will remain unchanged after the execution of O .

In step 3, O is applied to each target version in order to create potential new versions. The compatible groups of both the target versions and the new versions created are stored in mcs' in step 3. Since some of the new compatible groups composed in step 3 may not be a maximal one, it is necessary to maximize them, i.e., remove those compatible groups that are a subset of another. This process is completed in step 4, which is similar to the one in MOVIC. Step 5 simply returns the new set of maximal compatible groups.

For the sake of clarity of presentation, the algorithm contains some redundancy.

Algorithm 5.4: $compide_C(MCGS, O)$: mcs'

Input: $MCGS$ is the set of maximal versions created from a set OS of operations, and mcs is its corresponding set of maximal compatible groups; O is a causally ready operation with respect to OS .

Output: mcs' is the set of maximal compatible groups created from $OS \cup O$.

```

{
If  $O_{new}$  is a local operation, and  $(Target(O_{new}) = Vid(MCG)) \wedge (MCG \in MCGS)$ , then return  $(mcgs - \{Ocg(MCG)\}) \cup \{Ocg(MCG) \cup \{O_{new}\}\}$ ;
Else executing the following steps:
1. Differentiate  $O$ 's target versions (in terms of VerLocii) from unintended versions:
 $MCGS_{ii}(O) := \{MCG \mid Target(O) \subseteq Vid(MCG) \wedge MCG \in MCGS\}$ ;
 $m := |MCGS_{ii}(O)|$ ;
Let  $mcgs_1$  be an array containing the maximal compatible groups of  $MCGS_{ii}(O)$ , each of which is causally ordered;
/*  $mcgs_2$  is a set of unintended compatible groups. */
 $mcgs_2 := mcgs - \bigcup_{i=1}^m \{mcgs_1[i]\}$ ;
2. Reconstruct  $O$ 's generation context:
 $context := []$ ;
/*  $PRS$  ( $CCS$ ) is an array with  $m$  components, each of which contains a list of operations preceding (concurrent with)  $O$ . */
 $PRS, CCS$ : Array;
For  $i = 1$  to  $m$ 
 $\langle PRS[i], CCS[i] \rangle := PR\_CC(mcgs_1[i], O)$ ;
If  $PRS[i] \supset context$ , then  $context := PRS[i]$ ;
End for
3. Create new compatible groups:
/*  $mcgs_3$  is the set of unintended compatible groups contained in  $MCGS_{ii}(O) - MCGS_{iii}(O)$ , where  $MCGS_{iii}(O) := \{MCG \mid mcg(O) \subseteq Ocg(MCG) \wedge MCG \in MCGS\}$ ; */
 $mcgs_3 := \{\}$ ;
 $mcgs' := \{\}$ ;
For  $i = 1$  to  $m$ 
If  $PRS[i] \subset context$ , then
 $mcgs_3 := mcgs_3 \cup \{PRS[i] + CCS[i]\}$ ;
Else
 $\langle CP, CF, DCFSet, EO \rangle := CP\_CF(CCS[i], O)$ ;
 $mcgs' := mcgs' \cup \{PRS[i] + CP + [EO]\}$ ;
 $mcgs' := mcgs' \cup \{PRS[i] + CCS[i]\}$ ;
End for
4. Merge (maximise) the versions:
For any  $cg \in mcgs'$ , if there is another  $cg' \in mcgs'$  such that  $cg \subset cg'$ , then  $mcgs' := mcgs' - \{cg\}$ ;
5. Return the final result:
Return  $mcgs' \cup mcgs_2 \cup mcgs_3$ ;
}

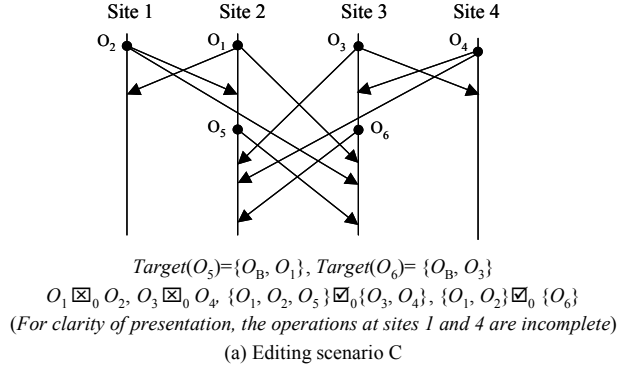
```

6 An Integrated Example

In this section, we present an integrated example to illustrate incremental version composition process of the contextual intention oriented multi-versioning scheme.

There are six operations generated by four users as in Figure 3(a). O_1 intentionally conflicts with O_2 , and O_5 is generated to target a version containing O_1 . Similarly, O_3 intentionally conflicts with O_4 , and O_6 is generated to target a version containing O_3 . Obviously, $Target(O_5) = \{Oid(O_B), Oid(O_1)\}$, $Target(O_6) = \{Oid(O_B), Oid(O_3)\}$, neither is a subset of the other. Since O_3 is intentionally compatible with O_1 and O_5 , when O_6 arrives at site 2, there exists a version containing O_1, O_3 , and O_5 , which is

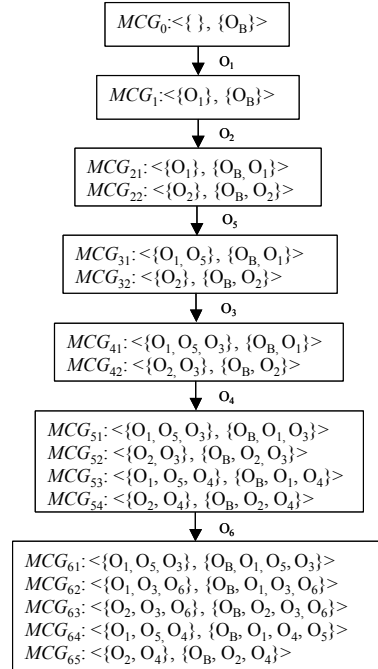
a sub-version of O_6 's targeted version. If O_5 and O_6 are spatially compatible, they can be combined in one version, otherwise a new version will be created to accommodate O_6 (Figures 3(b) and 3(c)). Therefore the definition of spatial relations for operations like O_5 and O_6 is necessary, that is, *distinct versions* (i.e., one version is neither a sub-version nor a super-version of another) cannot be treated as distinct independent objects, otherwise intention violation may occur (i.e., concurrent conflicting intentions may not be preserved).



(a) Editing scenario C

\mathfrak{R}	O_1	O_2	O_3	O_4	O_5	O_6
O_1	\odot_c	\otimes_{DC}	\odot_c	\odot_c	\odot_c	\odot_c
O_2		\odot_c	\odot_c	\odot_c	\otimes_{IC}	\odot_c
O_3			\odot_c	\otimes_{DC}	\odot_c	\odot_c
O_4				\odot_c	\odot_c	\otimes_{IC}
O_5					\odot_c	\otimes_{DC}
O_6						\odot_c

(b) Intentional relationships ($O_5 \boxtimes_0 O_6$)



(c) Incremental version composition and identification at Site 2 ($O_5 \boxtimes_0 O_6$)

Figure 3. An integrated example

The incremental version creation and identification process (for the editing scenario C) at site 2 is illustrated in Figure 3(c). The operations are executed in the following order: $O_1, O_2, O_5, O_3, O_4, O_6$. The contextual

intentions of O_5 and O_6 are preserved, i.e., it is not necessary to apply them beyond $MCGS_{III}(O_5)=\{MCG_{21}\}$ and $MCGS_{III}(O_6)=\{MCG_{51}, MCG_{52}\}$ respectively in order to compose the compatible groups. This is an evidence of the correctness of the integration of VerCom(\odot_C) and VerLoc_{III}. Nevertheless, it is worth noting that for consistent version identification VerLoc_{III} is not sufficient. For example, O_6 directly conflicts with O_5 , but O_5 has been applied to a version other than those in $MCGS_{III}(O_6)$ when O_6 is to be executed. This problem can be easily solved according the contextual intention preservation property. Operation O_5 would not have been applied to any version beyond $\{MCG \mid Target(O_5) \subseteq Vid(MCG) \wedge MCG \in MCGS\}$ (where $MCGS$ is the current set of maximal versions when executing O_5), that is, $\{MCG_{51}, MCG_{53}\}$. In Figure 3(c), O_5 is contained in version MCG_{53} ($\notin MCGS_{III}(O_6)$), which needs to be updated to MCG_{64} (with the identifier of O_5 being included) after the execution of O_6 .

7 Conclusion

Consistency maintenance is a fundamental issue in groupware systems. Intention preservation is an important aspect of consistency maintenance in real-time collaborative editing systems. The multi-version approach is able to preserve individual users' concurrent conflicting intentions in a consistent way. In this paper, after having reviewed our contextual intention oriented multi-versioning scheme that can preserve users' contextual intentions, we discussed how to incrementally compose versions. We have made use of the contextual intention preservation property in locating target versions, reconstructing generation contexts, and determining intentional relationships of operations. Consequently, an efficient version composition algorithm has been devised for the multi-versioning scheme.

Our future research is directed towards a formal evaluation of the time and space complexity of the *compide_C* algorithm.

8 References

- CHEN, D. and SUN, C. (1999): A distributed algorithm for graphic objects replication in real-time group editors. *Proc. of the ACM Conference on Supporting Group Work*, 121-130.
- DOURISH, P. (1996): Consistency guarantees: exploiting application semantics for consistency management in a collaboration toolkit. *Proc. of the ACM Conference on CSCW*, 268-277.
- ELLIS, C.A. and GIBBS, S.J. (1989): Concurrency control in groupware systems. *Proceedings of the ACM SIGMOD Conference on Management of Data*, 399-407.
- GREENBERG, S. and MARWOOD, D. (1994): Real time groupware as a distributed system: concurrency control and its effect on the interface. *Proc. of the ACM Conference on CSCW*, 207-217.
- KANAWATI, R. (1997): LICRA: A replicated-data management algorithm for distributed synchronous groupware application. *Parallel computing*, **22**:1733-1746.
- KARSENTY, A., TRONCHE, C. and BEAUDOUIN-LAFON, M. (1993): GroupDesign: shared editing in a heterogeneous environment. *Usenix Journal of Computing Systems*, **6**(2):167-195.
- LAMPORT, L. (1978): Time, clock, and the ordering of events in a distributed system. *CACM* **21**(7):558-565.
- MORAN, T. P., MCCALL, K., VAN MELLE, B., PEDERSEN, E. R. and HALASZ, F.G.H. (1995): Some design principles for sharing in Tivoli, a white-board meeting support tool. In *Groupware for Real-time Drawing: A Designer's guide*. 24-36. S. GREENBERG, S. HAYNE, and R. RADA (eds.). McGraw-Hill,
- MUNSON, J. P. and DEWAN, P. (1996): A concurrency control framework for collaborative systems. *Proc. of the ACM Conference on CSCW*, 278-287.
- MUNSON, J. P. and DEWAN, P. (1997): Sync: a Java framework for mobile collaborative applications. *IEEE Computer*, June 1997.
- PRAKASH, A. (1999): Group editors. In *Computer Supported Cooperative Work*. 103-133. M. BEAUDOUIN-LAFON (ed.). John Wiley & Sons.
- RAYNAL, M. and SINGHAL, M. (1996): Logical time: capturing causality in distributed systems. *IEEE Computer Magazine*, **29**(2):49-56.
- SUN, C. and CHEN, D. (2000): A multi-version approach to conflict resolution in distribute groupware systems. *Proceedings of International Conference on Distributed Computing Systems*. IEEE CS Press.
- SUN, C. and CHEN, D. (2002): Consistency Maintenance in Real-time Collaborative Graphics Editing Systems. *ACM Transactions on Computer-Human Interaction*, **9**(1):1-41.
- SUN, C., JIA, X., ZHANG, Y., YANG, Y. and CHEN, D. (1998): Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction*, **5**(1):63-108.
- XUE, L., ORGUN, M. and ZHANG, K. (2002): Editing any version at any time: an unconstrained consistency maintenance mechanism in Internet-based collaborative environments. To appear in *the Proc. of the International Conference on Parallel and Distributed Systems (ICPADS '2002)*, Taipei. IEEE CS Press.
- XUE, L., ZHANG, K., ORGUN, M. and SUN, C. (2001): Version composition and identification schemes in distributed real-time groupware systems. *Macquarie Computing Reports*, No.C/TR01-01, Macquarie University.
- XUE, L., ZHANG, K. and SUN, C. (2000): Conflict control locking in distributed cooperative graphics editors. *Proceedings of the 1st International Conference on Web Information Systems Engineering (WISE 2000)*, 401-408, Hong Kong, IEEE CS Press.
- YANG, Y., SUN, C., ZHANG, Y. and JIA, X. (2000): Real-time cooperative editing on the Internet. *IEEE Internet Computing*, May 2000, 18-25.