# From Predefined Consistency to User-Centered Emergent Consistency in Real-Time Collaborative Editing Systems

Mehmet A. Orgun, *Senior Member, IEEE*, and Liyin Xue

***Abstract*—In this paper, we introduce a basic conceptual model of real-time collaborative editing systems which differentiates the coordination aspect of actions (which do not exist in single-user systems) from the application-dependent semantic aspect of actions (i.e., those actions that happen in single-user systems). Based on this model, a new taxonomy is provided to analyze the existing consistency-maintenance mechanisms and highlight potential new mechanisms. Furthermore, a new user-centered consistency model is proposed, which focuses on dynamic consistency that is negotiable among end users rather than static consistency that is automatically determined by the system. In the mean time, some challenging research issues in real-time collaborative editing systems are outlined.**

***Index Terms*—Collaborative work, conflict resolution, consistency maintenance, multiuser negotiation.**

## I. Introduction

DISTRIBUTED real-time collaborative editing systems (or real-time group editors) support multiple users to view and edit the same document concurrently from geographically dispersed sites connected by a communication network [1], [2]. They reduce the time required to complete a task by avoiding the overhead of merging different versions of the same document into one.

Consistency maintenance is a major issue in the design of group editors, which have inherited many consistency-maintenance mechanisms from traditional database systems, where the basic principle is conflict prevention [1]–[3]. Generally, this is in contradiction to the spirit of collaboration. Theoretically, conflicts are inevitable in collaborative work. Conflicts may not necessarily be destructive to the collaboration process. Instead, they may contribute to the emergence of group intentions. With collaborative authoring on a document, discussion and negotiation play a major role in the formation of the document. Therefore, conflict control and negotiation support are important issues in the design of groupware systems in general and group editors in particular [4].

There have been some efforts in proposing techniques for consistency maintenance in real-time group editors which take into account human users' intentions and are particularly suitable for wide area network environments like the Internet [4]–[10]. However, it is not clear how these techniques are related to the traditional ones invented for database systems. It is necessary first to have an analytical framework to systematically examine the existing consistency-maintenance mechanisms, which may shed light on new research directions.

In this paper, we advocate a shift of consistency-maintenance strategies from conflict prevention to conflict control (cure or management), and a shift of consistency-maintenance mechanisms from system-based automatic conflict resolution to user-centered negotiation support. Our major concern is the intentions of human users rather than the predetermined behavior of the systems. We observe that the existing research work has not sufficiently taken into account human users' involvement in the evolution of consistent documents. Therefore, it is necessary to investigate how users perceive consistency and how systems can support the users. This calls for a new user-centered consistency model.

The rest of this paper is organized as follows. Section II proposes a conceptual model of real-time collaborative editing systems. Section III presents a taxonomy to classify existing consistency-maintenance mechanisms. Section IV discusses our view in a user-centered consistency model. Some new research issues in consistency maintenance in real-time collaborative editing systems are outlined in Sections III and IV. Section V concludes the paper with a summary.

## II. Conceptual Model of Real-Time Collaborative Editing Systems

In this section, we propose a conceptual model to capture the characteristics of user actions and interactions in order to understand how users collaborate to reach a common goal.

### A. Generalized Single-User Editing

There are many interaction functions, such as browsing, editing, tailoring, and annotating functions, provided in single-user editors. By activating browsing functions, a user can view various parts and structure of the document under editing. Editing functions are used to change the document, such as "insert a character into a particular position." Tailoring functions are used to modify the setting of browsing and editing functions,

Manuscript received October 3, 2005; revised April 30, 2006 and July 10, 2006. This paper was recommended by Guest Editor G. Cabri.

M. A. Orgun is with the Department of Computing, Macquarie University, Sydney, NSW 2109, Australia (e-mail: mehmet@comp.mq.edu.au).

L. Xue is with the Analytics, Change Program, The Australian Taxation Office, Civic, Canberra, ACT 2601, Australia.

Digital Object Identifier 10.1109/TSMCA.2006.883181

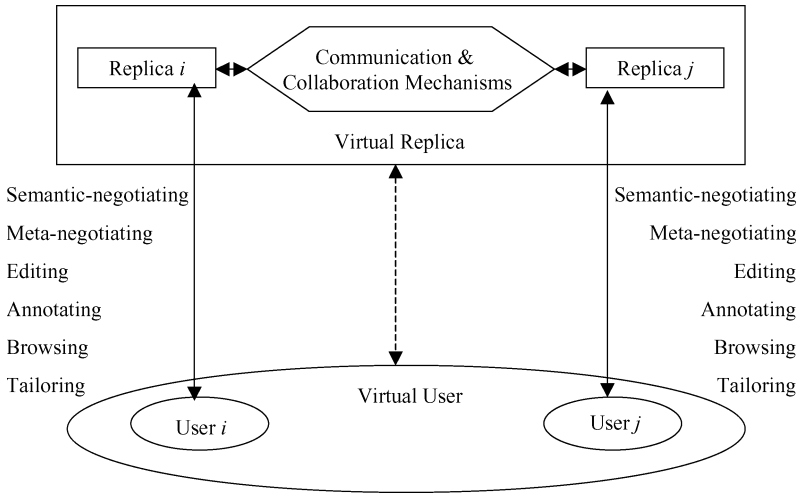


Fig. 1. Conceptual model of real-time groupware systems.

i.e., to configure the application. By reflecting on the current state and potential changes to the data items in a document, a user may annotate them in order to explain their modification or provide alternative modifications.

Each of the interaction functions has a corresponding category of user actions. Therefore, an application can be modeled by its data, view, and a set of interaction functions, i.e., browsing, editing, tailoring, and annotating functions.

### B. Generalized Multiuser Editing

We can extend the generalized single-user editing model into a multiuser editing model by providing communication and collaboration supports among individual applications. A conceptual model of real-time groupware systems is illustrated in Fig. 1. Each user directly interacts with their own replica (copy) of the application and indirectly with others' copies, i.e., we assume a fully replicated architecture such that the responsiveness of user's operations can be guaranteed [11].

Communication and collaboration supports consist of mechanisms for supporting flexible communications and for maintaining consistency. Flexible communications are used to model various aspects of group communications, i.e., who communicates what to whom, when, and how. The mechanism for maintaining consistency depends on the configured communication mechanism. Similarly, whether user's actions affect remote applications depends on the current configuration of communication and collaboration supports.

After being generated, an editing operation can be sent immediately to facilitate the mutual awareness of multiple users, or delayed for a period of time in order to be batched with other operations for the sake of efficiency of channel utilization or atomicity of multiple operations [12]. In many group editors, only the editing operation itself instead of its execution effect is propagated, which is different from that in most replicated database systems [13].

If browsing actions are not propagated (i.e., they are localized), one user's view is independent of another's. Users can browse a document asynchronously. Being propagated to remote sites, browsing actions can be used in two different ways. If they are treated as awareness information and not executed at remote sites, the browsing is still nonsynchronous. However, if they are executed immediately at remote sites, the system is tightly coupled, and the browsing or playing is synchronized. If the browsing actions are not properly coordinated among the users, "window wars" and "scroll wars" will occur [14].

Tailoring (or configuring) is similar to browsing with respect to synchronization. Since tailoring is a metafunction, it would be more conceivable to assign the access right to a particular role in order to avoid tailoring wars caused by users concurrently configuring the same function.

In group settings, annotating can be used as a mechanism for commenting on other users' work or for providing alternative suggestions. It is widely used in asynchronous groupware systems [15], [16]. However, its applications in synchronous groupware systems are largely unexplored.

The concepts of virtual user and virtual replica in the model represent the collective aspect of collaboration. An individual user's operation will be applied to the virtual replica only when it commits, i.e., it has been executed at all participating replicas. At this time, it can be considered as executed (accepted) by the group, i.e., the virtual user. The introduction of these two concepts has an important implication for group intention preservation [10].

### C. Metanegotiation and Semantic Negotiation

In addition to the above browsing, editing, tailoring, and annotating actions, we introduce two extra categories of actions, i.e., metanegotiation and semantic negotiation. Metanegotiation is necessary for a user to have the right to activate browsing, tailoring, editing, and annotating functions. Semantic negotiation is about the content of the document under editing.

*1) Metanegotiation:* Metanegotiation can be system determined or user centered. Access control is an example of system-determined metanegotiation. Whether a user has the right to access a data item is predetermined by system administrators or its owner. The metanegotiation functions are defined and

implemented in advance, and can be configured. Negotiability is a user-centered metanegotiation mechanism that supports the regulation of conflicts between an activator (who is to activate a function) and a user affected by offering a technical channel of communication and by allowing the affected user to intervene against the decision of the activator [17].

Metanegotiation can be implicit or explicit. With respect to explicit metanegotiation, before activating a nonmetanegotiation function (e.g., editing), users need to explicitly activate its corresponding metanegotiation function (e.g., requesting a lock). Many systems avoid the requesting overhead placed on human users by introducing implicit metanegotiation mechanisms, i.e., as soon as users activate a nonmetanegotiation function, the system will automatically generate a request or invoke a testing function to check their access rights or to provide a negotiation support. We advocate a design principle that wherever system-predetermined metanegotiation mechanisms are appropriate, an implicit mechanism should be integrated, and explicit mechanisms are necessary only when user-centered metanegotiation mechanisms are implemented.

*2) Semantic Negotiation:* Semantic negotiation is an act of message exchanges by (various combinations of) user actions in order to reach some agreements on the data items among users. For example, a user may edit or annotate the data items such that their intention is known to the other participants (the process of asking for permission to do this is metanegotiation). Negotiation about the data items may happen as users browse, modify, or annotate them.

*3) Negotiation About the Mechanisms of Semantic Negotiation and Metanegotiation:* A groupware system may be used by different groups of users to support various tasks with dynamically changing requirements. There is no one mechanism or policy that will suffice for all groups in all situations [18]. Therefore, the mechanisms of metanegotiation and semantic negotiation should be configurable based on the needs of the users.

### D. Comparison and Implication

Dewan *et al.* claim that generalized single-user editing can model a variety of contemporary interactive applications, and that the design space of collaboration can be characterized by using the notion of generalized multiuser editing [19]. However, in contrast to our concern with the characteristics of individual users' actions, the model of Dewan *et al.* focuses on the design considerations of collaboration functions, which include coupling, concurrency control, access control, and multiuser undo. There are other models that focus on other aspects of system functionality [20], [21].

Our model has two major implications. First, it can be used as a task analysis framework for usability evaluation of group editors. By specifying the user actions with a set of lower level operations, we can assess the support of various communication and collaboration mechanisms for collaborative editing activities [22]. Further work on this aspect is necessary but beyond the scope of this paper. Second, the concepts of metanegotiation and semantic negotiation lead to the differentiation of conflict prevention and conflict control, which is our major concern and will be addressed in the next section.

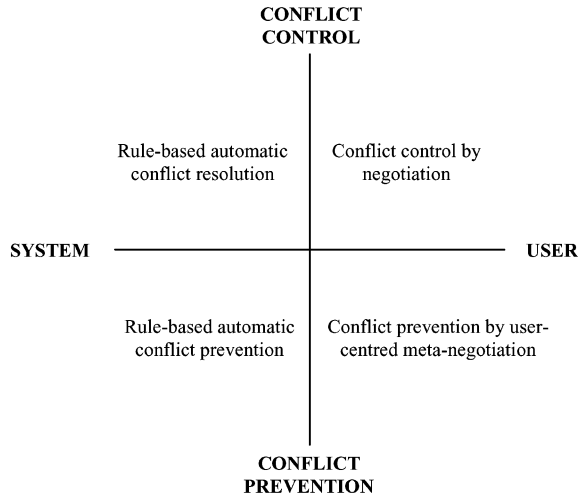


Fig. 2. Taxonomy of consistency-maintenance mechanisms.

## III. Taxonomy of Consistency-Maintenance Mechanisms in Groupware Systems

### A. Analytical Framework

Metanegotiation and semantic negotiation are two categories of actions that are group oriented. Consistency can be maintained by properly supporting them.

Concurrency control is generally used to prevent inconsistent concurrent updates in database and distributed systems [23], [24]. It focuses on conflict prevention instead of conflict control. Being related to the coordination of users' access to the data or the functionality of the application in the course of cooperative working, metanegotiation can be supported by access control and concurrency control mechanisms.

Since conflict-prevention approaches are inherently restrictive, some cooperative editing systems dispense with concurrence control altogether [14], [25]. These systems rely upon social protocols and users' awareness of others' actions to prevent conflicts. However, empirical evidence shows that conflicts are inevitable in the absence of conflict prevention [26]. Therefore, supporting mechanisms are needed to help users resolve them. Conflict regulation, conflict resolution, conflict control, and conflict management have been widely used to mean consistency maintenance in groupware literature, particularly those on asynchronous groupware systems. We confine them to mechanisms supporting human users in resolving conflicts due to concurrent activation of some functions of an application by different users, or due to alternative intentions of multiple users to act upon the same part of data.

The above discussion leads to our first dichotomy for the classification of consistency-maintenance mechanisms, i.e., conflict prevention and conflict control. By emphasizing the involvement of human users, we introduce the second dichotomy, i.e., user's involvement in or detachment from the process of conflict prevention or conflict control. Therefore, we can classify consistency-maintenance mechanisms into four categories, as illustrated in Fig. 2.

*1) Rule-Based Automatic Conflict Prevention:* With rule-based automatic-conflict-prevention approaches, whether a user is allowed to act is determined by a set of rules implemented in the system. Access to a data item is a result of negotiation between the user and the system. Only one user at a time is able to update it such that no conflict will occur. Access control is a well-known example. Other examples are as follows.

*Locking:* Locking is a widely used concurrency control approach in both database systems and cooperative applications [23], [27], [28]. Conflicts can be prevented by locking, but locking is undesirable in collaborative editing because it interrupts users in their work and causes unnecessary overhead. The overhead of requesting locks will increase response time in distributed environments.

*Transaction by locking:* Transactions have been used in non-real-time groupware systems, such as CES [29], [30] and Quilt [31], [32]. For real-time groupware systems, it will incur a cost to response time. The COAST system fully replicates all the objects that are part of a shared document, and supports transactions with ACID properties [33]. Generally long transactions are not well suited to interactive use, because changes made during a transaction are not visible to other users until the transaction commits [1].

*2) Conflict Prevention by User-Centered Metanegotiation:* With the conflict prevention by user-centered metanegotiation approaches, there is no more than one user editing a shared data item at any time, and access to the data item is a result of negotiation among participants. The central point of user-centered metanegotiation is how the process of negotiation is supported. Some researchers focused on the kind of communication channels built up to support the negotiation and the messages transmitted via them, which may be structured, semi-structured, and unstructured [34], [35]. The simplest example is *negotiable turn-taking* where a user needs to negotiate with the coordinator or current token holder to obtain the access right. This approach may overly inhibit the free and natural flow of information among participants [3]. However, it is suitable for and widely used in audio and video conferencing systems. Another example is the aforementioned negotiability approach.

*3) Rule-Based Automatic Conflict Resolution:* With rule-based automatic conflict-resolution approaches, users are allowed to edit the shared data freely, and any conflict will be automatically resolved by the system based on prespecified rules. This category of approaches is rather popular in real-time group editors.

*Ordering approaches:* These approaches guarantee that all operations are executed in the same global order at all participating sites. It can be pessimistic serialization, where the global order is imposed by a central server, and an operation is executed only after it is sent to and received from the server rather than when it is issued [13]. It can also be optimistic serialization where the global order is enforced by using logical clocks (time stamps) and an operation is immediately executed at the local site despite that it may be undone and reexecuted in a correct order if there exist concurrent operations [36]. GroupDesign [37] and LICRA [38] implemented this mechanism. If the concurrent operations are commutable, undoing and redoing the out-of-order operations are not necessary [39]. Some systems such as DECAF simply roll back the execution effect of one of the two concurrent operations [40].

*Operational-transformation approach:* Ellis and Gibbs pioneered this approach in their Grove real-time text editor [3]. In Grove, local operations are executed immediately as they are. When an operation is received from a remote site, the local editor needs to check whether there are concurrent operations executed, if so, the operation needs to be transformed (or compensated) such that the requestor's original intention is preserved. Major enhancements to the original work in this area include the CCU formalism [41], the Jupiter system [42], the adOPTed algorithm [43], the SOCT algorithms [44], [45], the REDUCE system [6], the GOTO algorithm [7], the landmark-based approach [46], and the POLO system [47].

*Merge-matrix-based approach:* This mechanism is implemented in SYNC [48]. A merge matrix consists of rows and columns labeled 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, e.g., accepting one operation or the other.

The rule-based automatic conflict-resolution approaches have the potential problem of intention violation, i.e., only one user's intention is preserved when multiple users' operations concurrently target the same data item.

*4) Conflict Control by Negotiation:* Automatic conflict resolution has the advantage of being efficient. Nevertheless, it is generally not feasible for the system to have the knowledge to properly resolve conflicts among users. Conflicts are best resolved by cooperative users, with the system providing appropriate negotiation support mechanisms. We discuss them in the following section.

### B. Major Stages of a Negotiation Process

There are too many parameters that can affect the process of conflict resolution. Here, we present different mechanisms of negotiation support in terms of their roles at different stages of a negotiation process, i.e., from conflict notification, mediation support, coordination support, to final decision support. A negotiation support system may employ a combination of these mechanisms.

*1) Conflict Notification and Visualization:* Supporting responsiveness, lazy (optimistic) consistency approaches contain inherent race conditions. So the first thing a system can do is to detect them after they occur and make the users aware of them. These are the basic functionalities of conflict awareness mechanisms, which differ in what to report to and how to notify the involved users.

*Temporal conflict and simple notification:* Stefik *et al.* use time stamps of operations to detect conflicts, which are then resolved manually [14]. The conflict is defined at time stamp level without operation semantics being taken into account. For a simple notification, a message (e.g., contained in a popup window) can be delivered on the screens of the involved users, which describes the place of the conflict in the document. The drawback is that the users have to locate the conflict and identify its nature.

*Syntactic or semantic conflict and conflict visualization:* If the system can use operation semantics to identify the nature of conflicts and present them visually, the involved users can then focus on how to resolve them. The diffing approaches have been widely used in asynchronous document merging [49]. With interactive diffing, the system can point out the differences between two versions of a document and ask the user to make decisions. The PREP system's flexible diff can visually report conflicts in various granularities such that users can detect the conflicts without being distracted from more appropriate tasks [50]. Any system supporting negotiation must support some conflict notification and/or visualization mechanism. For example, the real-time graphics editors supporting multiversioning such as Tivoli, GRACE, and POLO visualize conflicts in multiple versions [51]–[53].

*2) Mediation Support:* Upon being aware of the conflict, the involved users will begin a negotiation process to resolve it. Negotiation can be mediated in different ways in terms of communication channels and the organization of messages. However, we will focus only on the visible referential artifacts of the negotiation process, which are shared by the involved users. A referential artifact could be the base document or any annotation to it. With the former, the users may be allowed to either edit the document during the process of negotiation or browse it first and then update it only when they reach a consensus via invisible channels (e.g., voice). With the latter, the content of the referential artifact could be either a new version or simply some comments on the base document. We categorize three alternative mechanisms of mediation support as follows:

*Commenting annotation:* Annotation is widely used in asynchronous group work. For example, PREP [54] and Quilt [32] are well-known research prototypes supporting annotation. Virtually, all commercial document-processing packages, such as Microsoft Word and Lotus Notes, support some form of annotation. Recently, there has been much research on web annotation for supporting group work [15]. In many real-time groupware systems, the commenting-annotation mechanism is implemented as a voice channel.

*Intrusive negotiation:* When discussing about the conflict, coauthors sometimes can directly modify the document. The drawback of such an intrusive approach is that it may make the document become messy due to the unpredictable nature of negotiation dynamics. There are some real-time group editors supporting intrusive negotiation by embedding multiple versions of the same object in conflict into the document, such as Tivoli [51] and GRACE [52]. The versions created are mixed up with other objects in the base document. Its positive aspect is all individuals' intentions are visualized, thus facilitating the negotiation process. However, the embedded versions change the context of the object in conflict. This may confuse the users.

*Alternating annotation:* If multiple writers disagree with each other on a particular chunk of the document, each of them can propose their own version but none of them will be included into the final document before a consensus is reached. They can discuss about the appropriateness of different versions by temporally replacing the original version with a new version. Each user can edit their own version while the negotiation is in progress. Therefore, the document is annotated with versions rather than embedded with them. The Quilt system supports both commenting annotation and alternating (called revision) annotation in an asynchronous way [32]. With respect to real-time group editors, POLO implements such a mechanism [10], [53].

*3) Coordination Support:* Concurrency control is used to prevent potential conflicts. With unconstrained and responsive editing, if conflicts occur, coordination mechanisms may be needed to control them such that they do not become unmanageable. Therefore, concurrency control (or coordination support for conflict prevention) and coordination support for negotiation are rather similar. They differ in that the former prevents any conflict from occurring before users make any change to the data, whereas the latter prevents any further conflict from occurring or controls the complexity of further conflicts after the users have been involved in conflicts.

*Unconstrained multiversioning:* Multiversioning mechanisms provide basic coordination that guarantees a convergent document state for all users and preserves all individuals' intentions [9], [51], [52]. They can be considered as extensions of conventional version-control mechanisms in real-time and multireplica environments. The Tivoli system's versioning model extends the basic checkout/checkin model, which has been implemented in tools like Source Code Control System (SCCS) [55] and Revision Control System (RCS) [56]. In both models, once a version is created, it becomes immutable, that is to say, it cannot be modified. They differ in several ways. The granularity of versioning is at the level of object contained in a document (file) in Tivoli's model, but at file level in the checkout/checkin model. All latest versions are available to all participants in Tivoli, whereas each user has an exclusive one in RCS or SCCS. SCCS and RCS use file locking to control concurrent updates to files. Locking is not necessary with Tivoli. In Tivoli, whenever a user modifies a version, the old version available to her is deleted, which may be available to others until the modification operation is synchronized globally. This means that other users may concurrently target the same version. In contrast, all the versions created persist in SCCS and RCS.

Similarly, the approaches of GRACE and POLO can be seen as extensions of the copy–modify–merge model of the Sun Network Software Environment (NSE) tempered with an automatic merging support [57]. NSE supports concurrent access to the same data object. It combines the checkout/checkin model with the classical optimistic concurrency control approach [58]. Unlike checkout/checkin model, NSE provides some assistance to users in merging different versions of the same data item. Both GRACE and POLO provide synchronous and automatic merging of the effects of concurrent modifications. If all modifications are reconcilable, then no new versions will be created. Only when they are not, new versions will be created.

*Postlocking:* Postlocking is a class of mechanisms proposed in POLO for coordinating users in resolving the conflicts, which result from unconstrained accesses to shared data [4], [53]. Only when a conflict occurs will the system automatically lock the object in conflict. It is different from the previously mentioned conflict-prevention locking, which we call prelocking. Postlocking can be made dynamic or negotiable.

For example, if there are two users involved in the conflict, the system may first give the right of modifying the object to one of the users. Later on, the two users may negotiate with each other on the access right to the object. If the multiversion approach is employed, users may transfer lock ownerships on different versions dynamically such that group discussion and negotiation are smoothly facilitated. Therefore, postlocking opens up a new research direction for applying the conventional conflict-prevention mechanisms in a new context.

*4) Decision Support:* We restrict decision support to mean activities involved in the final step of negotiation, when several alternatives for the resolution of the conflict may have been proposed for a final decision after rounds of discussion and negotiation. A user may represent the group to resolve the conflict, or to realize the agreed-upon group intention. For example, in NSE, users are allowed to simultaneously access their respective private copies of the same object. Before making their copies visible to others, they have to reconcile the changes they made with the changes that other users have concurrently made on the same object [57]. The NSE is an asynchronous groupware system. Another example, GINA can support synchronous group editing. When conflicts occur, it provides a selective undo/redo mechanism for one of the involved users to resolve the conflicts [59].

The final decision may also be made by participatory decision or voting as is in POLO.

To end this section, it is worth mentioning Coda, which is an optimistically replicated file system [60]. Conflicts can arise if replicas of an object are modified simultaneously in different network partitions. Generally, Coda uses application-specific knowledge for automatic conflict resolution. Only if resolution fails does the user have to resort to manual repair. No negotiation is supported in the system.

### C. Comparison and Implication

Many reviews on consistency-maintenance mechanisms simply gave a list of some of the mechanisms with their respective characteristics, but without pointing out the relationships between them [1], [3], [6], [24], [39]. Greenberg and Marwood provided a taxonomy for various locking and serialization approaches [27]. Wulf classified conflict-regulation mechanisms for asynchronous groupware systems [8]. Sun and Ellis systematically analyzed existing operational-transformation schemes [7]. Bhola and Ahamad presented a taxonomy for a set of ordering mechanisms [13].

We systematically classify consistency-maintenance mechanisms in terms of the conceptual model presented in the previous section. Therefore, our taxonomy is more comprehensive.

Our taxonomy reveals a new category of consistency-maintenance mechanisms, i.e., conflict control by negotiation, which is yet largely unexplored. We focus on analyzing this category and illustrate that the traditional conflict-prevention mechanisms can be innovatively employed in various stages of the negotiation (or conflict resolution) process in a real-time environment. Multiversioning and postlocking are typical examples. Designing an appropriate interface for each stage of the negotiation process will be a major challenge.

## IV. User-Centered Consistency Model

The shift from conflict prevention to conflict control necessitates the involvement of human users in the process of consistency maintenance. Therefore, it is necessary to investigate how users perceive consistency and how groupware systems can support the users in realizing their expectation. This calls for a new user-centered consistency model [61]. First, we need to differentiate between externalized consistency and emergent consistency. The former is enforced statically by the system, and the latter is agreed upon dynamically among the users.

### A. Externalized Consistency and Emergent Consistency

In database systems, consistency is enforced by access control, concurrency control, and integrity-constraint mechanisms. In groupware systems, these mechanisms are generally either too restrictive or inflexible, although they are also useful. For example, in the previous section, we have mentioned that conflict control is more flexible than conflict prevention in groupware systems. Moreover, not all constraints can be formalized in advance in multiuser environments. In some circumstances, consistency is emergent, i.e., it is contingent on the negotiation process involving multiple users.

The designers of groupware systems must anticipate the mutually acceptable perspectives (intersubjectivity) of multiple users, which can be predefined in the systems. This static intersubjectivity reflects only a rather limited aspect of consistency. A dynamic version of intersubjectivity can emerge only after a negotiation process during the real application of groupware systems. Therefore, we need to discuss the two aspects of consistency below, namely, externalized consistency (or *a priori* consistency) and emergent consistency (or *a posteriori* consistency).

*1) Externalized Dimension of Consistency:* In a replicated environment, since each user can concurrently view and modify their respective copy of the document under editing, final documents would not be identical among participating sites if no proper collaboration mechanisms are provided. It is generally required that copies of the shared document be identical at all sites at quiescence, i.e., all operations generated by the users have been executed at all sites [3]. This consistency criterion is called a convergence property [3], [6]. In most systems, operations are required to execute in their causal order at all sites. This is the causality-preservation property presented in the consistency model of Sun *et al.* [6]. These two properties can be enforced by the systems automatically. In some collaborative applications, the property of What You See Is What I See (WYSIWIS) is necessary. View synchronization is important when multiple users are involved in a real-time discussion and negotiation process. Unfortunately, it is not easy or even feasible to provide such support in a distributed wide-area network environment with nonnegligible communication latency.

The convergence property reflects the relationships among multiple replicas of the same document. It is about intercopy consistency. Besides this, the data of each copy may be required to be consistent (intracopy consistency) in various linguistic

levels in many applications. For example, a multiuser software development environment may support multiple programming languages; it can be configured such that it provides a check of the correctness of user inputs in terms of lexical constructs of a language. This is the issue of lexical consistency. A programming language can be defined by describing what its program looks like (the syntax of the language) and what its programs mean (the semantics of the language). A correct program must satisfy the syntax specified, i.e., syntactic consistency must be maintained. Similarly, a document may be required to be formatted and organized in a specific way, i.e., structural consistency property must be maintained.

Externalized consistency can be enforced automatically by the system through either fixed implementation or run-time configuration. Although an intelligent system may be able to capture some part of the emergent consistency, conceptually, it becomes externalized.

*2) Emergent Dimension of Consistency:* The extent of externalization of group perspectives on consistency is rather limited in the sense that it reflects only a part of the rich connotation of subjective consistency. For example, document semantics is generally difficult to specify in natural language environments. Consistency is not only application and user dependent, but also context dependent, where multiple users are involved in a situated conversation process. It is generally not possible for the system alone to automatically maintain the consistency required by human users. Nevertheless, emergent consistency must be consistent with externalized consistency in a given system.

A correctly spelled English word may not be suitable for a particular semantic context. Coauthors can negotiate with each other to find a proper synonym. This is a simple example of emergent consistency.

Suppose a shared document contains the sentence:

A: "John forgot lock the door."

There is an English grammar error in it, which may be detected by a well-designed system. However, the best thing the system can do is to notify the users and present the following two alternatives:

B: "John forgot to lock the door."

C: "John forgot locking the door."

Both B and C are syntactically correct but with different meanings. The involved users need to discuss about which one is appropriate for the semantic context. This is an example of emergent semantic consistency.

Generally, a system does not have sufficient knowledge of what consensus the users are to reach in advance; hence, the maintenance of emergent consistency can only be supported rather than enforced by groupware systems.

From the above discussion, we can conclude that users are the final judges of consistency. Consistency is an agreement among the involved users. If it is agreed upon in advance, then it is externalized consistency, otherwise, emergent consistency.

### B. Analytical Framework for Intentions

Consistency embodies in the document state that the users consider satisfying some correctness criteria. A document state is the result of applying a series of operations. An operation issued by a user reflects their perspective on the current document state and their intention to change it to a new state. In groupware systems, since the context may be changed by concurrent operations issued by other users, the result of the execution of the original operation on the new context may not be the user's original intention. Therefore, intention preservation becomes an important issue of consistency maintenance in collaboration systems [6], [7].

Most of the earlier prototypes of group editors employ various locking techniques as consistency-maintenance mechanisms, which are similar to the database transaction approach; thus, no intention-preservation problem needs to be considered. For systems without conflict-prevention support, intention violation may occur. In real-time group editors, a particular intention-violation problem was first tackled by the operational-transformation approach implemented in Grove [3]. There have been many publications devoted to operational transformation since then [6], [7], [41]–[47]. However, the nature and significance of intention preservation have not yet been fully investigated.

Considering that some consistent states of the document may not satisfy all of the involved users, i.e., the perspectives of individual users and the group on consistency are different, we differentiate group intention from individual intention. An individual user's intention is usually represented by one or more operations generated by the user (we will use user's intention and intention of an operation interchangeably). Therefore, individuals' intentions are a set of intentions of operations generated by a group of individual users. The intention of one user may conflict with some other users' intentions. A group intention represents the one that is agreed upon among the group either *a priori* (externalized group intention) or *a posteriori* (emergent group intention). In other words, a group intention is the one resulting from merging individuals' intentions via a synergistic coordination or negotiation process among the involved users. Before looking at the synergistic dimension, we first investigate the characteristics of individual intentions.

Individual intentions can be categorized according to the user actions discussed in the conceptual model of real-time groupware systems. Therefore, we have editing intention, annotating intention, and negotiating intention, etc. Here, we focus on the analysis of editing intention.

*1) Contextual Dimension of Intention:* Whenever a user is to issue an action, they must have been aware of the current context or document state, and a decision or intention is in their mind. Therefore, individual users' intentions are generally context dependent or context sensitive. Any context-sensitive intention must be executed on its original context (called generation context) or compensated (transformed) such that it can be executed on a new context; otherwise, an intention violation may occur. In fact, the pessimistic locking approach implemented in collaborative editors assumes that any intention is absolutely context sensitive, and concurrent modifications of the same part of a document are not allowed. However, the scope of a context is generally rather limited. Modifications beyond this scope may not be interrelated with it. The granularity of context, for example, paragraph or section in

a text document, is an important issue in the specification of consistency-maintenance policies.

*2) Connotative Dimension of Intention:* In database systems, any user or application can only see the consistent states of data, i.e., a database can only move from one consistent state to another from the viewpoint of users or applications. Nevertheless, during the editing process, users are allowed to see inconsistent states of the document in collaborative systems. They may perform some actions on the document to make it consistent, while inconsistency may persist for some time during the editing process. In other words, whenever issued, an action is connoted with rich indirect intentions of its issuer, in addition to its direct intention.

For example (consider the example in Section IV-A), a user may issue an operation $O_1$ to insert "to" at the position between "forgot" and "lock," thus changing sentence A to B. The system can mechanically interpret a user's explicit intention reflected in the operation. However, it does not comprehend their implicit syntactic intention (e.g., correcting a syntactic error) and semantic intentions (e.g., the meaning of sentence B instead of C) connoted with the operation.

Connotative dimension of intention is important when users need to explicitly negotiate about the document content to which an operation has been applied. Neglecting such a factor may cause intention violation. An extra communication channel (such as a voice channel) may be necessary for communicating the implicit syntactic and semantic intentions.

*3) Relational Dimension of Intentions:* An operation is defined on a particular context, which is the result of executing multiple operations and may be concurrently modified by other operations. Therefore, the discussion of contextuality always resorts to the relationships of operations. Two operations may have multiple relationships in terms of their execution (or generation) order, the objects or attributes that they are generated to target, and some mathematical properties.

*Temporally and intentionally dependent relations:* If operation $O_1$ is causally preceding operation $O_2$, $O_2$ is said to be temporally dependent on $O_1$. However, temporal dependence may not necessarily mean intentional dependence. For example, if $O_1$ is not in the scope of $O_2$'s generation context, $O_2$ is intentionally independent of $O_1$. Therefore, given two operations, if one is temporally dependent on but intentionally independent of another, they can be executed in any order, i.e., they are commutative. In this case, it is not necessary to maintain the causality-preservation property. In other words, causality preservation can be removed from the externalized consistency. However, this case is too special. It is hardly possible for the system to know all the relationships automatically. However, operations targeting semantically related data may be controlled by integrity constraints specified in advance.

*Transactional relation:* The above dependent relations are related to operations generated by different users. A user may issue a group of interdependent operations like those contained in a transaction in database systems. Other users may be allowed to read the intermediate execution results but not to issue any operation on the same objects targeted by those operations, such that the user's intention is not interfered with. The maintenance of atomicity of intention consisting of multiple subintentions, which we call transactional intention, is an important issue in real-time collaborative editing systems [62].

*Semantically compatible and conflict relations:* If two operations target syntactically and semantically unrelated regions of a document, they are said to be semantically compatible, otherwise, they are semantically conflicting. For example, two operations targeting the same object and changing the same attribute to different values are semantically conflicting with each other. It is application and user dependent whether two operations targeting the same region or object of a document are conflicting or compatible. Semantic relationships are important in configuring the consistency-maintenance policies.

*Intentionally compatible and conflict relations:* Any two operations that are both concurrent and semantically conflicting are said to be intentionally conflicting. Otherwise, they may be intentionally compatible. Obviously, if intentionally conflicting operations are applied to the document together, intention violations will occur, whereas intentionally compatible operations can be applied to the document without interfering with each other [5], [9].

*4) Synergistic Dimension of Intention:* Similar to consistency, a group intention can also be classified into either externalized group intention or emergent group intention.

*Externalized dimension of group intention:* It would be tedious for users to negotiate with others about every intention. Only when the users feel that it is necessary to have a discussion or when the system automatically detects that there is a (potential) conflict, should the users be brought together to participate in a negotiation process. An individual intention committed without negotiation is in fact an externalized group intention, which had been agreed upon even before the users started editing the document, i.e., an operation that does not intentionally conflict with other operations can be applied to the document directly without negotiation by default. An externalized group intention can be an individual intention or a combination of multiple individual intentions based on predefined rules. All involved users must be aware of these rules. If they disagree with each other on them, the system must be reconfigured to adapt to the new consensus.

Since a group intention generally involves multiple individual intentions, it can be embodied in the aforementioned relational dimension of intentions. Particularly, compatible and conflict relationships of operations are important aspects of externalized group intention. They can be defined in various ways depending on the shared perspectives of users. For example, two operations targeting the same paragraph may be defined as conflicting, and explicit negotiation for a group intention is necessary. They may also be treated as compatible, and a group intention is reached automatically via a proper transformation of them [3].

*Emergent dimension of group intention:* Operations targeting completely different parts of a document are in fact not synergistic, although all contribute to the formation of the document. Only when they target the same part or semantically related parts, synergy may happen. It is generally not feasible for the system to know the negotiation result, i.e., emergent group intention, in advance.

Consider again the example presented in Section IV-A. Assume user $i$ issues an operation $O_1$ to insert "to" at the position between "forgot" and "lock," thus changing sentence A to B. User $j$ concurrently issues an operation $O_2$ to insert "ing" at the ending position of "lock," thus changing A to C.

The existing operational-transformation schemes (in fact, externalized rules for group intention) can preserve the direct intentions of the two operations such that the final document contains sentence D.

D: "John forgot to locking the door."

Apparently, this result is still grammatically incorrect, although both B and C are grammatically correct. Neither of the users' connotative intentions as in sentences B and C is preserved. In addition, the meanings of sentences B and C are different. It is not possible for the system to accommodate semantically conflicting intentions into one correct sentence.

From this example, we see that an externalized prespecified rule for group intention should be adaptable to various situations. The rule should be specified such that two operations targeting the same sentence or paragraph are conflicting with each other. A specification language may be required to specify application-specific conflict-resolution rules as is in Coda [60]. However, the only way to preserve the above connotative intentions of the users is by annotation or multiversioning. After negotiation, both of the users may agree on one of the versions, i.e., either sentence B or C.

For more complicated cases, users may negotiate with each other over multiple rounds by modifying their respective versions and merging them such that a group intention will eventually emerge.

In summary, a system needs to preserve various individual intentions, externalized group intentions, and emergent group intentions. It maintains the externalized consistency and preserves externalized group intentions automatically without the involvement of cooperating users.

### C. Comparison and Implication

*1) Comparison to Related Work on the Consistency Model:* Consistency maintenance is a well-studied topic in parallel and distributed systems, mobile systems, conventional database systems, and groupware systems. Various consistency models have been proposed. They are primarily system oriented and focus on objective and static aspects of consistency without taking into account the involvement of human users. For example, in distributed shared memory systems, a consistency model is essentially a contract between the software and the memory. If the software agrees to obey certain rules, the memory promises to work correctly. If the software violates these rules, the correctness of memory operations is no longer guaranteed [63]. Many consistency models trade consistency for better performance or ease of programming [64]. A parallel view on consistency in groupware systems has been proposed by Dourish [65]. He introduced the notion of consistency guarantees as a technique to increase the effectiveness of the explicit semantics approach. Consistency guarantees regard various locks as guarantees of some level of achievable consistency, i.e., if a client promises to follow the restrictive rules determined by the server, it receives some guarantee of future consistency. In other words, consistency is a contract between the client and the server. However, a client can break a promise, in which case, the server is no longer held to its guarantee. This has a significant implication for the design of a groupware toolkit to provide support for opportunistic working styles.

We address consistency from the point of view of multiuser interaction and argue that consistency is an agreement (or intersubjectivity) among human users in groupware systems. Such a user-centered interpretation results in the introduction and differentiation of externalized consistency and emergent consistency. The latter catches the fundamental characteristic of collaborative work, where human users' situated intentions and negotiation play a major role.

Many group-editing research prototypes are based on existing models available from distributed and database systems. For example, those systems implementing transaction (based on locking mechanisms) are based on the serializable model from database systems, which reduces the consistency problem to that of testing for serializable schedules [24], [27]. The serialization approach implemented in GroupDesign [37] and LICRA [38] is based on the sequential consistency model from distributed shared memory systems, which guarantees that operations are executed in a global order at all participating sites [64]. Generally, they are either too restrictive or unable to provide support for intention preservation and the situated negotiation among users.

The only consistency model that takes into account intentions of users was proposed by Sun *et al.* [6], [7]. However, it is still system oriented, and intention preservation is simply juxtaposed with convergence and causality preservation in the model. The problem here is that the preservation of multiple users' concurrent conflicting intentions may make a document convergent but semantically inconsistent. For example, the embedded multiple versions (the aforementioned intrusive negotiation) for the preservation of conflicting intentions will interfere with the other objects in the document, resulting in semantic inconsistency. In contrast, our model can accommodate the issues of intention preservation and negotiation support more systematically.

*2) Some Research Issues:* Intention preservation plays a crucial role in supporting the maintenance of consistency. If users are not informed about each other's real intentions, they simply cannot negotiate. Furthermore, facilitation mechanisms are necessary to support users in obtaining emergent group intentions. Therefore, the focus of consistency-maintenance shifts from traditional conflict prevention to intention preservation and negotiation support in unconstrained real-time distributed collaborative editing environments. This raises new research issues and calls for new mechanisms.

*Preservation of individuals' intentions:* The challenge of preserving individuals' intentions arises from the existence of interference of one intention with another. The preservation of any one of the contextual, connotative, and transactional intentions usually resorts to either isolation or compensation of the interference. Currently, there are two approaches to the preservation of contextual intention in unconstrained real-time collaborative editing environments, namely, operational

transformation and multiversioning [3], [5], [7], [9], [51]. Further work is necessary to generalize and specialize the current approaches. The preservation of transactional and connotative intentions needs to be investigated as well.

*Facilitation of negotiation processes:* Some major issues in supporting negotiation in real-time group editors are as follows. First, it is necessary to control the complexity caused by unconstrained concurrent editing. For example, if the multiversion approach is employed, novel mechanisms such as postlocking implemented in POLO are necessary to control the proliferation of versions and facilitate the negotiation process among multiple users. Second, coordination mechanisms and multimodal communication modes for negotiation processes should be investigated. Third, designing a multiuser interface for negotiation is a challenging issue. Finally, in order to evaluate the current techniques and guide future research, we need empirical evidence on how people collaborate to resolve conflicts in real-time collaborative environments.

*Preservation of emergent group intentions:* When a group intention is reached after negotiation, it needs to be incorporated into the document. Technical mechanisms are necessary for the representation and preservation of such an emergent group intention [10].

## V. Conclusion

The collaboration model of Dewan *et al.* was intended to improve the understanding, design, and automation of collaborative applications [19]. Our conceptual model of real-time groupware systems emphasizes user actions or intentions and the interactions among users (metanegotiation and negotiation). Based on this model, a systematic analysis of consistency-maintenance mechanisms in groupware systems has been provided in terms of the following two distinct shifts of focus: from conflict prevention to conflict control and from system-based conflict resolution to user-centered negotiation support. Our taxonomy reveals a category of consistency-maintenance mechanisms, i.e., conflict control by negotiation, which are largely unexplored in real-time collaborative editing environments. We have examined the existing techniques and raised new research issues in the four major stages of a negotiation process.

In contrast to the existing system-centered consistency model, we have proposed a new user-centered consistency model that emphasizes the importance of subjective or negotiating aspect of consistency and human user intentions. We examine the constituent parts of the model, particularly, various aspects of user intentions in details. Based on the model, a number of new research issues with respect to intention preservation in real-time collaborative editing systems are outlined.

In addition to other systems, we have used our POLO prototype to illustrate some techniques for negotiation support and intention preservation.

To conclude our paper, it is necessary to mention that the purpose of this paper is to propose models for systematically and critically examining the existing techniques and identifying new research issues. It is not the purpose of this paper to evaluate the usability of the consistency-maintenance mechanisms. This nevertheless requires further investigation.

## References


[1] C. A. Ellis, S. J. Gibbs, and G. L. Rein, "Groupware: Some issues and experiences," *Commun. ACM*, vol. 34, no. 1, pp. 38–57, Jan. 1991.

[2] A. Prakash, *Computer Supported Cooperative Work*, M. Beaudouin-Lafon, Ed. Hoboken, NJ: Wiley, 1999, pp. 103–133.

[3] C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems," in *Proc. ACM SIGMOD Conf. Manage. Data*, May 1989, pp. 399–407.

[4] L. Xue, K. Zhang, and C. Sun, "Conflict control locking in distributed cooperative graphics editors," in *Proc. 1st Int. Conf. WISE*, Hong Kong, Jun. 2000, pp. 401–408.

[5] D. Chen and C. Sun, "A distributed algorithm for graphic objects replication in real-time group editors," in *Proc. ACM Conf. Supporting Group Work*, Nov. 1999, pp. 121–130.

[6] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems," *ACM Trans. Comput.-Hum. Interact.*, vol. 5, no. 1, pp. 63–108, Mar. 1998.

[7] C. Sun and C. A. Ellis, "Operational transformation in real-time group editors: Issues, algorithms, and achievements," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, Nov. 1998, pp. 59–68.

[8] V. Wulf, "On conflicts and negotiation in multiuser applications," in *Encyclopedia of Microcomputers*, vol. 23, A. Kent and J. G. Williams, Eds. New York: Marcel Dekker, 1999, pp. 63–88. Suppl. 2, New Basel.

[9] L. Xue, M. A. Orgun, and K. Zhang, "Editing any version at any time: An unconstrained consistency maintenance mechanism in Internet-based collaborative environments," in *Proc. Int. Conf. Parallel and Distributed Syst.*, Dec. 2002, pp. 69–74.

[10] L. Xue, M. Orgun, and K. Zhang, "Group-based time-stamping scheme for the preservation of group intention," in *Distributed Communities on the Web DCW 2002*, vol. 2468. Berlin, Germany: Springer-Verlag, 2002, pp. 125–137.

[11] P. Dewan, "Architectures for collaborative applications," in *Computer Supported Cooperative Work*, M. Beaudouin-Lafon, Ed. Hoboken, NJ: Wiley, 1999, pp. 169–193.

[12] D. Li, C. Sun, L. Zhou, and R. Muntz, "Operation propagation in real-time group editors," *IEEE Multimedia*, vol. 7, no. 4, pp. 55–61, Oct.–Dec. 2000.

[13] S. Bhola and M. Ahamad, "The design space for data replication algorithms in interactive groupware," Georgia Inst. Technol., Atlanta, Tech. Rep. GIT-CC-98-15, 1998.

[14] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, and L. Suchman, "Beyond the chalkboard: Computer support for collaboration and problem solving in meetings," *Commun. ACM*, vol. 30, no. 1, pp. 32–47, Jan. 1987.

[15] J. J. Cadiz, A. Gupta, and J. Grudin, "Using web annotation for asynchronous collaboration around documents," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, Dec. 2000, pp. 309–318.

[16] E. F. Churchill, J. Trevor, S. Bly, L. Nelson, and D. Cubranic, "Anchored conversations: Chatting in the context of a document," in *Proc. ACM CHI*, The Hague, The Netherlands, 2000, pp. 454–461.

[17] A. Pfeifer and V. Wulf, "Negotiating conflicts in active databases," in *Proc. 4th Int. Conf. Concurrent Eng.*. Rochester, MI: Oakland Univ., Aug. 20–22, 1997, pp. 443–450.

[18] C. Lauwers and K. A. Lantz, "Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems," in *Proc. ACM Conf. Human Factors Comput.*, 1990, pp. 303–311.

[19] P. Dewan, R. Choudhary, and H. Shen, "An editing-based characterisation of the design space of collaborative applications," *J. Organ. Comput.*, vol. 4, no. 3, pp. 219–240, 1994.

[20] A. Karsenty and M. Beaudouin-Lafon, "SLICE: A logical model for shared editors," in *Groupware for Real-time Drawing: A Designer's Guide*, S. Greenberg, S. Hayne, and R. Rada, Eds. New York: McGraw-Hill, 1995, pp. 157–173.

[21] C. Ellis, "Team automata for groupware systems," in *Proc. ACM SIGGROUP Conf. Supporting Group Work*, Nov. 1997, pp. 415–424.

[22] D. Pinelle, C. Gutwin, and S. Greenberg, "Task analysis for groupware usability evaluation: Modeling shared-workspace tasks with the mechanics of collaboration," *ACM Trans. Comput.-Hum. Interact.*, vol. 10, no. 4, pp. 281–311, Dec. 2003.

[23] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Reading, MA: Addison-Wesley, 1987.

[24] J. P. Munson and P. Dewan, "A concurrency control framework for collaborative systems," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, 1996, pp. 278–287.

[25] S. Greenberg, M. Roseman, D. Webster, and R. Bohnet, "Issues and experiences designing and implementing two group drawing tools," in *Proc. IEEE 25th Annu. Hawaii Int. Conf. Syst. Sci.*, Jan. 1992, pp. 139–250.
[26] J. D. Campbell, "Usability and interference for collaborative diagram development," in *Proc. Collaborative Editing Workshop at ACM Comput. Supported Cooperative Work 2000*, Philadelphia, PA, 2000. Workshop website, (last accessed 10 July 2006). [Online]. Available: http://www.csdl.tamu.edu/~lidu/iwces2/papers/campbell.pdf
[27] S. Greenberg and D. Marwood, "Real time groupware as a distributed system: Concurrency control and its effect on the interface," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, Oct. 1994, pp. 207–217.
[28] A. Michailidis and R. Rada, "A review of collaborative authoring tools," in *Groupware and Authoring*, R. Rada, Ed. New York: Academic, 1996, pp. 9–44.
[29] I. Greif and S. Sarin, "Data sharing in group work," in *Proc. 1st Conf. Comput. Supported Cooperative Work*, 1986, pp. 175–183.
[30] I. Greif, R. Seliger, and W. Weihl, "Atomic data abstractions in a distributed collaborative editing system," in *Proc. 13th Annu. ACM Symp. Principles Programming Languages*, 1986, pp. 160–172.
[31] R. Fish, R. Kraut, M. Leland, and M. Cohen, "Quilt: A collaborative tool for cooperative writing," in *Proc. ACM Conf. Office Inf. Syst.*, Mar. 1988, pp. 30–37.
[32] M. D. P. Leland, R. S. Fish, and R. E. Kraut, "Collaborative document production using Quilt," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, 1988, pp. 206–215.
[33] C. Schuckmann, L. Kirchner, J. Schuemmer, and J. M. Haake, "Designing object-oriented synchronous groupware with COAST," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, Nov. 1996, pp. 30–38.
[34] T. W. Malone, K. R. Grant, K.-Y. Lai, R. Rao, and D. Rosenblitt, "Semistructured messages are surprisingly useful for computer-supported coordination," in *CSCW: A Book of Readings*, I. Greif, Ed. San Mateo, CA: Morgan Kaufmann, 1988, pp. 311–334.
[35] T. Winograd, "A language/action perspective on the design of cooperative work," in *CSCW: A Book of Readings*, I. Greif, Ed. San Mateo, CA: Morgan Kaufmann, 1988, pp. 623–653.
[36] M. Raynal and M. Singhal, "Logical time: Capturing causality in distributed systems," *IEEE Comput. Mag.*, vol. 29, no. 2, pp. 49–56, Feb. 1996.
[37] A. Karsenty, C. Tronche, and M. Beaudouin-Lafon, "GroupDesign: Shared editing in a heterogeneous environment," *Usenix J. Comput. Syst.*, vol. 6, no. 2, pp. 167–195, 1993.
[38] R. Kanawati, "LICRA: A replicated-data management algorithm for distributed synchronous groupware application," *Parallel Comput.*, vol. 22, no. 13, pp. 1733–1746, Feb. 1997.
[39] A. Karsenty and M. Beaudouin-Lafon, "An algorithm for distributed groupware applications," in *Proc. 13th Int. Conf. Distributed Comput. Syst.*, May 1993, pp. 195–202.
[40] R. Strom, G. Banavar, K. Miller, A. Prakash, and M. Ward, "Concurrency control and view notification algorithms for collaborative replicated objects," *Computers*, vol. 47, no. 4, pp. 458–470, Apr. 1998.
[41] C. V. Cormack, "A calculus for concurrent update," Dept. Comput. Sci., Univ. Waterloo, Waterloo, ON, Canada, Res. Rep. CS-95-06, 1995.
[42] D. Nichols, P. Curtis, M. Dixon, and J. Lamping, "High-latency, low-bandwidth windowing in the Jupiter collaboration system," in *Proc. ACM Symp. User Interface Software and Technol.*, Nov. 1995, pp. 111–120.
[43] M. Ressel, D. Nitshce-Ruhland, and R. Gunzenbaeuser, "An integrating, transformation-oriented approach to concurrency control and undo in group editors," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, Nov. 1996, pp. 288–297.
[44] M. Suleiman, M. Cart, and J. Ferrie, "Serialization of concurrent operations in distributed collaborative environment," in *Proc. ACM Conf. GROUP*, Phoenix, AZ, 1997, pp. 435–445.
[45] N. Vidot, M. Cart, J. Ferrie, and M. Suleiman, "Copies convergence in a distributed real-time collaborative environment," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, Dec. 2000, pp. 171–180.
[46] R. Li and D. Li, "A landmark based transformation approach to concurrency control in group editors," in *Proc. GROUP*, Sanibel Island, FL, Nov. 6–9, 2005, pp. 284–293.
[47] L. Xue, M. Orgun, and K. Zhang, "Intention preservation by multiversioning in distributed real-time group editors," in *Proc. Int. Conf. EDCIS*, Beijing, China, 2002, vol. 2480, pp. 510–524.
[48] J. P. Munson and P. Dewan, "Sync: A Java framework for mobile collaborative applications," *Computer*, vol. 30, no. 6, pp. 59–66, Jun. 1997.
[49] ——, "A flexible object merging framework," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, Oct. 1994, pp. 231–242.
[50] C. M. Neuwirth, R. Chandhok, D. S. Kaufer, P. Erion, J. Morris, and D. Miller, "Flexible diff-ing in a collaborative writing system," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, Nov. 1992, pp. 147–154.
[51] T. P. Moran, K. McCall, B. van Melle, E. R. Pedersen, and F. G. H. Halasz, "Some design principles for sharing in Tivoli, a white-board meeting support tool," in *Groupware for Real-time Drawing: A Designer's Guide*, S. Greenberg, S. Hayne, and R. Rada, Eds. New York: McGraw-Hill, 1995, pp. 24–36.
[52] C. Sun and D. Chen, "Consistency maintenance in real-time collaborative graphics editing systems," *ACM Trans. Comput.-Hum. Interact.*, vol. 9, no. 1, pp. 1–41, Mar. 2002.
[53] M. Orgun, L. Xue, and Z. Han, "Supporting distributed collaborative work with multi-versioning," in *Proc. 10th Int. Conf. CSCWD,* W. Shen em *et al.,* Eds., May 3–5, 2006, vol. 1, pp. 193–198.
[54] C. M. Neuwirth, D. S. Kaufer, R. Chandhok, and J. H. Morris, "Issues in the design of computer support for co-authoring and commenting," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, Nov. 1990, pp. 183–193.
[55] M. J. Rockind, "The source code control system," *IEEE Trans. Softw. Eng.*, vol. SE-1, no. 4, pp. 364–370, Dec. 1975.
[56] W. F. Tichy, "RCS—A system for version control," *Softw.—Pract. Exper.*, vol. 15, no. 7, pp. 637–654, Jul. 1985.
[57] E. W. Adams, M. Honda, and T. C. Miller, "Object management in a CASE environment," in *Proc. 11th ACM Int. Conf. Softw. Eng.*, May 1989, pp. 154–163.
[58] H. Kung and J. Robinson, "On optimistic methods for concurrency control," *ACM Trans. Database Syst.*, vol. 6, no. 2, pp. 213–226, Jun. 1981.
[59] T. Berlage, "A selective undo mechanism for graphical user interfaces based on command objects," *ACM Trans. Comput.-Hum. Interact.*, vol. 1, no. 3, pp. 269–294, Sep. 1994.
[60] P. Kumar and M. Satyanarayanan, "Flexible and safe resolution of file conflicts," in *Proc. USENIX Winter Tech. Conf.*, New Orleans, LA, Jan. 1995, pp. 95–106.
[61] L. Xue, M. Orgun, and K. Zhang, "A user-centred consistency model in real-time collaborative editing systems," in *Proc. DCW*, 2002, vol. 2468, pp. 138–150.
[62] L. Xue and M. A. Orgun, "Locking without requesting a lock: A consistency maintenance mechanism in Internet-based real-time group editors," *J. Parallel Distrib. Comput.*, vol. 65, no. 7, pp. 801–814, Jul. 2005.
[63] S. Adve and M. Hill, "Weak ordering: A new definition," in *Proc. 17th Annu. Int. Symp. Comput. Archit.*, 1990, pp. 2–14.
[64] A. S. Tanenbaum, *Distributed Operating Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
[65] P. Dourish, "Consistency guarantees: Exploiting application semantics for consistency management in a collaboration toolkit," in *Proc. ACM Conf. Comput.-Supported Cooperative Work*, Nov. 1996, pp. 268–277.

**Mehmet A. Orgun** (SM'96) received the B.Sc. and M.Sc. degrees in computer science and engineering from Hacettepe University, Ankara, Turkey, in 1982 and 1985, respectively, and the Ph.D. degree in computer science from the University of Victoria, Victoria, BC, Canada, in 1991.

He joined Macquarie University, Sydney, Australia, in 1992, where he is currently an Associate Professor. His current research interests include temporal reasoning, data mining, temporal and modal logics, and reactive and distributed systems. He has received many research grants from Macquarie University and the Australian Research Council in the aforementioned areas.

**Liyin Xue** received the B.E. degree in communication engineering from the Harbin Institute of Technology, Harbin, China, in 1983, the M.E. degree in electronic engineering from Shanghai Jiao Tong University, Shanghai, China, in 1986, and the Ph.D. degree in computer science from Macquarie University, Sydney, Australia, in 2003.

His current research interests include data mining and computer-supported cooperative work. He is currently with the Australian Taxation Office, Canberra.