

A Constraint Maintenance Strategy and Applications in Real-Time Collaborative Environments

Kai Lin¹, David Chen², Chengzheng Sun² and Geoff Dromey²

School of Information and Communication Technology,
Griffith University, Brisbane, QLD 4111, Australia

¹ Kai.Lin@student.griffith.edu.au

² {D.Chen, C.Sun, G.Dromey}@griffith.edu.au

Abstract. A constraint expresses a relationship among one or more variables. Constraints are very useful in the development of collaborative applications, such as collaborative CAD and CASE systems, but satisfying constraints in the presence of concurrency in collaborative systems is difficult. In this article, we discuss the issues and techniques in maintaining constraints in collaborative systems. In particular, we also proposed a novel priority strategy that is able to maintain both constraints and system consistency in the face of concurrent operations. The strategy is able to resolve constraint violations in multi-constraint systems and is independent of the execution orders of concurrent operations. To illustrate the applicability of the proposed priority strategy, the applications of the approach in various collaborative systems are discussed in detail.

1 Introduction

A constraint expresses a relationship among one or more variables. Interactive applications, such as CAD and CASE systems, may specify many constraints to confine the relationships or states of constrained objects.

Maintaining constraints automatically is advantageous in collaborative systems. A task demanding people to work collaboratively is often complex and may contain many constraints. Thus, it is very practical and powerful for collaborative systems to maintain constraints automatically on behalf of users. For example, when people work collaboratively to design a project using Java Class notation, many conflicts may arise if a system only relies on individuals to maintain Java single inheritance constraint.

Constraints are very useful in handling complex tasks in collaboration scenarios, but satisfying constraints in the presence of concurrency in collaborative systems is difficult. Concurrent operations may result in some constraints becoming difficult to satisfy even though they may be maintained easily in single user environments. For example, it is hard for us to satisfy the constraint defining “ $X=Y+Z$ ”, when three users concurrently change the values of X , Y and Z . In addition, interferences among constraints may be very intricate and difficult to coordinate in collaborative systems.

Much work has been done on the maintenance of constraints in single user interactive applications [2], [3], [5], [13]. However, maintenance of constraints in concurrent environments has many new features which cannot be handled by single user strategies, such as ensuring both constraint satisfaction and system consistency, handling the constraint violations generated by concurrent operations, etc.

In this paper, we discuss the issue of maintaining constraints in collaborative environments and propose a novel strategy to achieve both constraint satisfaction and system consistency in real-time collaborative systems. The proposed strategy is independent of the execution orders of concurrent operations. In addition, the applications of the proposed strategy in various collaborative systems are discussed in detail.

The rest of this article is organized as follows. In section 2 we discuss problems of constraint maintenance in real-time collaborative systems and propose a novel strategy that is able to handle constraint violations and maintain system consistency in collaborative environments. Section 3 introduces the applications of the proposed strategy. Comparison with related work is introduced in section 4 and the major contributions and future work of our research are summarized in the last section.

2 A Priority Strategy of Maintaining Constraints in Collaborative Systems

Collaborative systems are groupware applications to support people working together in groups, such as electronic conferencing/meeting, collaborative CAD and CASE [10]. Constraints are very useful in collaborative systems, which can confine and coordinate concurrent operations.

To meet the requirement of high responsiveness in the Internet environment, replicated architecture is widely adopted in collaborative systems. Shared documents are replicated at the local storage of each collaborating site, so that operations can be performed at local sites immediately and then propagated to remote sites [1], [10]. However, maintaining both consistency and constraints in collaborative systems adopting replicated architecture is difficult, which is illustrated in the following two scenarios:

Scenario 1. A horizontal-line constraint, C_1 , restricts $left\text{-}endpoint.y=right\text{-}endpoint.y$ of any horizontal-line. Two users concurrently move both endpoints of a horizontal-line to different vertical positions.

Scenario 2. There is a constraint C_2 that confines objects A and B should not overlap with each other. On the initial document state, A is at position P_a and B is at P_b . Two users concurrently move A and B to the same position P_c from different sites.

There is a contradiction between satisfying constraints and retaining operations' display effects in both scenarios. The problem is caused by concurrent operations competing to satisfy the same constraint in different ways. To characterize these operations, we define competing operations group of a constraint.

Definition 1. A Competing Operations Group of constraint C , denoted by COG_C , is a set of user operations, $\{O_1, O_2, \dots, O_m\}$, such that:

- (1) For any $O_i \in COG_C$, there is $O_j \in COG_C$ while O_i and O_j are concurrent,

- (2) The executions of all the operations in COG_C will result in a constraint violation of C , which cannot be restored if all these operations retain their display effects,
- (3) For any $O_i \in COG_C$, the executions of all the operations in $COG_C - O_i$ will not generate the condition described in (2).

In multi-constraint systems, concurrent operations may violate unspecific constraints, as illustrated in the following scenario:

Scenario 3. A collaborative system enforces two constraints, C_1 and C_2 , which define “ $X.color=Y.color$ ” and “ $Y.color=Z.color$ ” (Here, notation $X.color$ represents the color of object X). Two users concurrently generate operations O_1 and O_2 that change X and Z to different colors respectively.

In scenario 3, the concurrent executions of O_1 and O_2 form competing operations group of neither C_1 nor C_2 , because C_1 can be satisfied while both O_1 and O_2 retaining their display effects, if we do not enforce C_2 . For the same reason, C_2 can be satisfied by un-enforcing C_1 . However, if both users’ operations retain their display effects, C_1 and C_2 cannot be satisfied at the same time. In this scenario, the concurrent executions of the two operations form a competing operations group of an unspecific constraint of the system. To characterize these operations, we define competing operations group of a system.

Definition 2. A Competing Operations Group of system S , denoted by COG_S , is a set of user operations, $\{O_1, O_2, \dots, O_m\}$, such that:

- (1) For any $O_i \in COG_S$, there is $O_j \in COG_S$ while O_i and O_j are concurrent,
- (2) The executions of all the operations in COG_S will result in that all the constraints enforced in system S previously cannot be satisfied at the same time if all these operations retain their display effects,
- (3) For any $O_i \in COG_S$, the executions of all the operations in $COG_S - O_i$ will not generate the condition described in (2).

According to the definitions 1 and 2, if constraint C is enforced in system S , a competing operations group of C , COG_C , must be a competing operations group of S , COG_S .

If a competing operations group of system S , COG_S , is generated, to maintain all the constraints enforced in S , one operation in COG_S will lose its display effect. If different operations lose their display effects at different sites, divergence occurs. To solve this problem, a priority strategy is adopted. If there is a conflict in retaining all operations’ display effects in collaborative systems with constraints, the strategy masks operations’ effects according to their priorities. For instance, in scenario 1, if the operation changing *left-endpoint.y* of a horizontal-line has a higher priority than the other operation, its display effect will be retained while the display effect of the other operation will be masked at each site.

In this paper, the timestamp of an operation denotes its priority, the bigger the timestamp the higher the priority. For any two operations O_a and O_b , the timestamp of O_a is bigger than the timestamp of O_b , if and only if O_b total ordering precedes O_a [10].

If the execution of operation O generates a set of competing operations groups of system S , $COGS_S = \{COG_1, COG_2, \dots, COG_n\}$, $1 \leq n$, O must be contained in each $COG_i \in COGS_S$, $1 \leq i \leq n$. If O is the operation with the lowest priority in a $COG_i \in COGS_S$, $1 \leq i \leq n$, O will be masked. As a result, no other operation should be masked in

any $COG_j \in COGS_S$, $1 \leq j \leq n$, $j \neq i$, because O is also an operation in COG_j and is masked.

Masking an operation, O , may cause some masked operations that have lower priorities than O restore their display effect (i.e. be unmasked). For instance, in scenario 2, two users concurrently execute O_1 and O_2 that move objects A and B to P_c respectively. If O_2 is masked to satisfy the overlapping constraint, when O_1 is masked by other operations, O_2 may recover its display effect. Therefore, if an operation, O , is masked to satisfy the constraints of a system, all the masked operations that have lower priorities than O should be checked. If their executions on the new document state can ensure the satisfactions of these constraints, they should be unmasked. On the other hand, unmasking an operation O_j may cause other operations that have lower priorities than O_j be masked.

Based on the above discussion, we propose a priority strategy as follows:

For any operation O that is ready for execution in a multi-constraint system S :

- (1) If the execution of O will not generate competing operations group of S , O can be executed directly at the site,
- (2) If the execution of O will generate a set of competing operations groups of S , $COGS_S = \{COG_1, COG_2, \dots, COG_n\}$, $1 \leq n$, and O is the operation with the lowest priority in a $COG_i \in COGS_S$, $1 \leq i \leq n$, O will be masked at the site,
- (3) If the execution of O will generate a set of competing operations groups of S , $COGS_S = \{COG_1, COG_2, \dots, COG_n\}$, $1 \leq n$, and O is not the operation that has the lowest priority in any $COG_i \in COGS_S$, $1 \leq i \leq n$, to satisfy all the constraints in S , O should be executed after the operations with the lowest priorities in each $COG_i \in COGS_S$, $1 \leq i \leq n$, are masked.
- (4) Suppose O_i is the operation that has the highest priority amongst all the masked operations in each $COG_i \in COGS_S$, $1 \leq i \leq n$. After the execution of O , all the masked operations that have lower priorities than O_i should be checked in descending order of their priorities. If their executions on the new document state can ensure the satisfactions of the constraints enforced in S , they will be unmasked.

The above priority strategy can maintain both constraints and consistency in collaborative systems, which is independent of the execution orders of concurrent operations. It can be adopted in many collaborative applications, including CAD, CASE, spreadsheets, graphical interface toolkits, simulation systems, etc.

3 The Applications of the Priority Strategy

To illustrate the applicability of the proposed priority strategy, in this section we discuss the applications of the approach in various collaborative systems.

3.1 Maintaining Constraints in Real-Time Collaborative CASE Systems

Collaborative CASE systems support people working together on large projects. For this many users need to work on shared documents to get the maximum benefits in

terms of productivity gains. Real-time collaborative CASE systems allow a group of users to view and edit the same documents at the same time from different sites to analyze requirements and create designs. They provide several benefits, such as easy interaction among designers, sharing document easily and collaboratively integrating requirements into a design, etc.

In Object-oriented CASE systems, constraints are widely adopted to confine the relations of different Classes. For example, each Java Class should satisfy single inheritance constraint and Class hierarchy is always acyclic. Therefore, the proposed priority strategy can be applied to maintain constraints in collaborative CASE systems, as illustrated in the following scenario:

Scenario 4. A collaborative CASE contains three Java Classes, *A*, *B* and *D*, on the initial document state. Three users concurrently generate operations. O_1 denotes that *B* extends *A*. O_2 requires that *D* inherits *B* and O_3 represents *B* extends *D*, as shown in Fig. 1 (a). We use a notation $O.priority$ to represent the priority of operation O . In this scenario, $O_1.priority > O_3.priority > O_2.priority$.

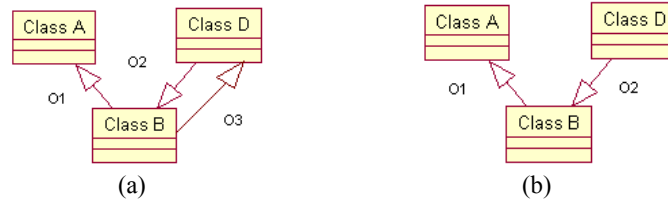


Fig. 1. Three concurrent operations violate Java single inheritance constraint and form cyclic Class hierarchy in (a). After O_3 is masked, the Class hierarchy of the three Java Classes is shown in (b)

In scenario 4, the executions of the three concurrent operations violate Java single inheritance constraint and form cyclic Class hierarchy, because the executions of O_1 and O_3 cause *B* extend two Classes, and O_2 , O_3 form a circular inheritance. In this scenario, the maintenances of the two constraints interfere with each other. According to the relative priorities of the three operations, O_2 should be masked to satisfy acyclic Class hierarchy constraint and O_3 should be masked to maintain Java single inheritance constraint. However, if O_3 is masked to enforce Java single inheritance constraint, there is not cyclic Class hierarchy anymore, so that O_2 can retain its display effect, as shown in Fig. 1 (b).

The proposed priority strategy can handle the interferences among constraints' maintenances in collaborative environments, which is independent of the execution orders of concurrent operations. For example, in scenario 4, if O_2 is masked previously to ensure acyclic Class hierarchy, when O_3 is masked to maintain Java single inheritance constraint, O_2 will be unmasked. On the other hand, if O_3 is masked before the execution of O_2 , the executions of O_1 and O_2 will not generate any competing operations group of the system, so that both of them can retain their display effects. Under both conditions, the final results are identical with the one shown in Fig.1 (b). The proposed strategy can be adopted in collaborative CASE systems to maintain constraints.

3.2 Maintaining Constraints in Collaborative CAD Systems

Constraints have been used in a number of ways in CAD systems, such as to denote the special requirements of geometric design, to maintain consistency among multiple views on data, etc.

The proposed priority strategy can be adopted to maintain constraints in collaborative CAD systems, which is demonstrated by the following example:

In CAD system S , the radiuses of three circles, A , B , D , representing the wheels of a tricycle, must be the same. Thus, two constraints, C_1 and C_2 , should be maintained, which define " $A.radius=B.radius$ " and " $B.radius=C.radius$ " respectively. Three users concurrently generate operations O_1 , O_2 and O_3 from different collaborating sites, which change $A.radius$, $B.radius$ and $C.radius$ respectively. $O_1.priority > O_2.priority > O_3.priority$.

Suppose at a site, the three operations are executed in the order O_1, O_2, O_3 :

- (1) O_1 will not generate COG_S of the system on the initial document, so that it can be executed,
- (2) When O_2 arrives, it will be masked, because O_1 and O_2 form a competing operations group of C_1 and $O_1.priority > O_2.priority$,
- (3) When O_3 is ready for execution, it will be masked too, because O_1 and O_3 generate a competing operations group of S and $O_1.priority > O_3.priority$.

Thus, after the executions of the three operations, both O_2 and O_3 are masked while O_1 has display effect.

At another site, the three operations are executed in the order O_3, O_2, O_1 :

- (1) The execution of O_3 on the initial document state will not form competing operations group of S . Thus, it can be executed,
- (2) When O_2 is ready for execution at the site, O_3 will be masked, because O_2 and O_3 form a competing operations group of C_2 and $O_2.priority > O_3.priority$,
- (3) When O_1 is ready for execution, O_2 will be masked, because O_1 and O_2 form a competing operations group of C_1 and $O_1.priority > O_2.priority$,
- (4) After O_2 is masked, O_3 will be checked. Because O_1 and O_3 form a competing operations group of S and $O_1.priority > O_3.priority$, O_3 cannot recover its effect.

After the three operations are executed in different orders at different sites, each site achieves the same final document state. The proposed priority strategy can be adopted to resolve constraint violations in multi-constraint collaborative CAD systems.

3.3 Maintaining Constraints in Other Collaborative Applications

The proposed priority strategy is a generic solution to handle constraint violations in multi-constraint collaborative systems. It can be applied in many kinds of collaborative applications. For instance, it may be used by a spreadsheet system to maintain the constraints confining the relationship between different cells, a simulation system representing current and voltage relationship of a complex circuit to confine that concurrent operations always satisfy Ohm's law, a graphic editing system to coordinate the concurrent operations that update graphic objects, such as coordinating the operations that concurrently change the *left*, *right* and *width* of a rectangle, etc.

The proposed strategy masks operations' display effects according to their priorities in case there is a conflict in retaining all operations' display effects in collaborative systems with constraints. The approach is independent of systems and constraints. However, how to detect competing operations group and mask operations is constraint and application dependent.

4 Related Work

There is a large body of research related to constraint maintenance in single user environments [2], [3], [5], [13]. However, maintaining constraints in concurrent environments has many new features that cannot be handled by the strategies adopted in single user environments.

CAB [7] and SAMS [9] are related to constraint control in collaborative environments. CAB presents an active rule based approach to modeling user-defined semantic relationships in collaborative applications and explores a demonstrational approach for end-user customization of collaboration tools to support the definition of those relationships. However, just as its author stated that many complications of maintaining constraints in collaborative environments, such as how to handle constraint violations and coordinate interferences among constraints, are not investigated in CAB.

The intention of SAMS is to achieve semantic consistency by integrating semantic constraints to the operational transformation approach. SAMS is based on XML resources. Its application in other environments has yet to be investigated. Moreover, SAMS uses loose constraint satisfaction strategy that allows constraints be violated.

By comparing with the above approaches, we proposed a priority strategy that is able to maintain both constraints and system consistency in the face of concurrent operations in collaborative environments. The strategy is independent of the execution orders of concurrent operations and can resolve constraint violations in many kinds of multi-constraint collaborative applications.

5 Conclusions and Future Work

Constraints are very useful in collaborative systems, which can confine and coordinate concurrent operations. However, maintaining constraints in real-time collaborative systems is a challenging task. The difficulties are caused by concurrent operations that violate constraints. Being able to solve this problem is crucial in the development of collaborative CAD and CASE applications.

In this paper, we proposed a novel priority strategy to solve this problem. This solution ensures constraint satisfaction while maintaining consistency. Our solution allows concurrent operations be executed in any order. It can be adopted to handle constraint violations in many kinds of multi-constraint collaborative applications. The applicability of the proposed strategy is discussed in detail.

We are currently investigating the scenario that user operations manipulating the constrained variables are concurrent with constraints' additions and deletions. If constraints can be added to or deleted from a system when users are manipulating the constrained variables, maintaining consistency will become very difficult. How to solve this problem is currently being investigated and will be reported in our subsequent publications.

References

1. Begole James et al.: Resource Sharing for Replicated Synchronous Groupware, IEEE/ACM Transactions on Networking, Vol. 9, No.6, Dec. (2001) 833-843.
2. Borning, A. and Duisberg, R.: Constraint-Based Tools for Building User Interfaces, ACM Transactions on Graphics, Vol.5, No.4, Oct. (1986) 345-374
3. Borning, A. et al.: Constraint Hierarchies, Lisp and Symbolic Computation, Vol. 5 No. 3, Sept. (1992) 223-270.
4. Dourish, P.: Consistency Guarantees: Exploiting Application Semantics for Consistency Management in a Collaborative Toolkit, In Proceedings of the ACM Conference on Computer Supported Cooperative Work, ACM, New York (1996) 268-277
5. Freeman-Benson, B. et al.: An Incremental Constraint Solver, Communications of the ACM, Jan. (1990) 54-63
6. Ignat, C.-L., Norrie, M.C.: Grouping in Collaborative Graphical Editors, ACM Conference on Computer-Supported Cooperative Work, Chicago, USA, Nov. 6-10, (2004) 447-456
7. Li, D. and Patrao, J.: Demonstrational Customization of a Shared Whiteboard to Support User-Defined Semantic Relationships among Objects, ACM GROUP'01, Boulder, Colorado, USA, Sept. 30-Oct. 3, 2 (2001) 97-106
8. Sannella, M. et al.: Multi-way versus One-way Constraints in User Interfaces: Experience with the DeltaBlue Algorithm, Software-Practice and Experience, Vol. 23(5) (1993) 529-566
9. Skaf-Molli Hala, Molli Pascal and Ostér Gerald: Semantic Consistency for Collaborative Systems, the Fifth International Workshop on Collaborative Editing Systems Hosted by the 8th European Conference of Computer-Supported Cooperative Work, Helsinki, Sept.15, (2003)
10. Sun, C. et al.: Achieving Convergence, Causality-Preservation, and Intention-Preservation in Real-Time Cooperative Editing Systems, ACM Transactions on Computer-Human Interaction, 5(1), Mar. (1998) 63-108
11. Sun, C. and Chen, D.: Consistency Maintenance in Real-Time Collaborative Graphics Editing Systems, ACM Transactions on Computer-Human Interaction, Vol. 9, No.1, Mar. (2002) 1-41
12. Sun, D. et al.: Operational Transformation for Collaborative Word Processing, ACM Conference on CSCW, Chicago, USA, Nov. 6-10, (2004)
13. Zanden, B.: An Incremental Algorithm for Satisfying Hierarchies of Multi-way Dataflow Constraints, ACM Transaction on Programming Languages and Systems, Vol.18, No.1, Jan. (1996) 30-72