Maintaining Constraints Expressed as Formulas in Collaborative Systems

Maintaining Constraints Expressed as Formulas in Collaborative Systems

Kai Lin, David Chen, Geoff Dromey
School of Information and Communication Technology
Griffith University
Brisbane, QLD 4111, Australia
{K.Lin, D.Chen, G.Dromey}@griffith.edu.au

Chengzheng Sun
School of Computer Engineering
Nanyang Technological University
Singapore, 639798
CZSun@ntu.edu.sg

***Abstract*—Constraints allow users to declare relationships among objects and let the constraint systems maintain and satisfy these relationships. Formulas have been adopted to express constraints in a wide variety of single-user applications, because of their simplicity, efficiency and manageability. The needs and benefits of supporting formula-defined constraints in collaborative environments have long been recognized. However, maintaining both constraints and consistency in the presence of concurrency in collaborative systems is a challenge. In these systems, users may concurrently define formulas, which could result in that different formulas are defined to express the same object-attribute at different sites. In this article, we discuss the issues and techniques in maintaining formula-defined constraints in collaborative systems. In particular, we also proposed a method that is able to maintain both constraints and system consistency in concurrent environments based on the existing consistency maintenance approaches. This method extends the application of these approaches from collaborative systems without constraint to systems that support formulas. The proposed method has been applied to implement a collaborative Visio system, called CoVisio, which leverages single-user Microsoft Visio for multi-user collaboration. Specific issues related to CoVisio are also discussed in detail.**

Keywords-constraint satisfaction; consistency maintenance; formulas; collaborative systems; CoVisio

I. INTRODUCTION

Constraints specify semantic level conditions that must be satisfied, and will automatically be maintained by a constraint system. Constraint-based applications simplify users' jobs by allowing users to concentrate on saying what should be true, leaving it to the constraint systems to worry about when and how to make these things true [11].

Formulas have been adopted to express constraints in a wide variety of single user interactive systems, including spreadsheet [19], user interface [2], [3], graphical editing [17], etc., because of their simplicity, efficiency and manageability. A formula is an expression that may contain constants, operators, and object-attribute references. Formulas are often used to define the relationship between objects in a system. For example, in a graphic editing system, a line can be constrained to glue to another graphic object by a formula, so that when an end-user moves the object sideways, the line will be moved with it automatically. In a spreadsheet application, formulas can be applied to let two cells always be equal, or let one cell be the sum of several other cells, etc.

Constraints have been shown to be useful in simplifying and automating user tasks in single-user environments. Constraints in multi-user environments not only inherit these benefits, but there are additional benefits. Constraints can be used to maintain semantic consistency. Such consistency is much harder for multiple users to manually maintain, than for a single-user, as it requires extra communication and coordination between users.

On the other hand, maintaining constraints expressed as formulas in collaborative systems is a challenge. First of all, the use of formulas makes the editable objects related to one another, so that an operation, which updates an object-attribute, may result in a chain of reactions. For instance, once a user moves an object, all the lines glued to that object will be moved as well. Therefore, one operation often has multiple effects in a constraint system. In contrast, in a system without constraint, where objects are independent of each other, an update operation often only has effect on the object-attribute it directly targets. Moreover, in concurrent environments, users may concurrently update formulas, which could result in that different formulas are defined to express the same object-attribute at different collaborating sites of a collaborative application. It is obvious that maintaining system consistency in collaborative systems supporting formulas is more complex than in collaborative systems without constraint, as not only the same document states should be shown on user interfaces at all collaborating sites, but also the underlying object-relationships must be identical at all sites.

In this paper, we discuss the issues of maintaining constraints expressed as formulas in concurrent environments. We analyze the relationship between consistency and constraint maintenances in collaborative systems, and proposed an approach to satisfy constraints expressed as formulas in collaborative systems. Our approach ensures both constraint satisfaction and system consistency.

The proposed approach has been applied to leverage single-user Microsoft Visio system for multi-user collaboration. One feature that distinguishes Visio from other graphic editing

systems is that formulas are defined in Visio to express the attributes of each graphic object, and the relationship between different Visio graphic objects. The ability to describe shapes with formulas opens many possibilities for making shapes behave in complex and sophisticated ways. The collaborative Visio system, called CoVisio, enables a group of users to view and edit the same Visio documents at the same time from different sites.

The rest of this article is organized as follows. The next section introduces formula-defined constraints. Convergence maintenance in collaborative systems supporting formulas is discussed in section three. We discuss the problems of consistency maintenance in formula-based collaborative systems, and analyze the relationship among value, formula, and document convergences. Moreover, we propose an approach that is able to maintain both constraints and consistency in formula-based collaborative systems. In the fourth section, we describe the application of the proposed approach in CoVisio system. Comparison with related work is introduced in the fifth section and the major contributions and future work of our research are summarized in the last section.

II. Constraints Expressed As Formulas

A constraint specifies a relation or condition that must be maintained in a system [3], [20]. For example, resistors must obey Ohm's law. Constraints can be expressed in different ways. The simplest constraint can be expressed as a constant. However, a sophisticated program may be used to describe a complicated constraint. Formulas are adopted to express constraints in a wide variety of object-oriented applications, including spreadsheet [19], user interface [2], [3], graphical editing [17], etc.

In a formula-based application, an object-attribute is expressed by a formula. A formula is an expression that may contain constants, operators, and object-attribute references. Formulas are often used to define the relationship between object-attributes in a system. For example, in an object-oriented graphic system, a rectangle is a graphic object, which is associated with a lot of graphic attributes, such as *X* and *Y* coordinates of each vertex, *width*, *height*, *color*, etc. The *X* and *Y* coordinates of the top-right vertex of the rectangle could be defined by two formulas, *width×1* and *height×1*, respectively. Therefore, each time a user resizes the rectangle (i.e. updates the *width* or *height* attribute of the object), the formulas defining the coordinates of the vertex will be reevaluated, so that the position of the vertex will be changed accordingly by the constraint system.

In a formula-based system, an object-attribute is associated with both a formula and a value. For instance, the *Y* coordinate of the vertex is expressed by the formula, *height×1*, and its value could be *y*. The values of object-attributes determine the behavior and appearance of the objects on user interfaces. As the value of the *Y* coordinate of the vertex is *y*, the vertex will be located at position *y* in Y-axis. On the other hand, a value is always determined by a formula. The constraint system evaluates a formula to a result and then converts the result to the appropriate units for the attribute that contains the formula. Some formulas consist of a single constant, but all formulas go through this evaluation and conversion process [17]. In the above example, as *Y* is constrained by formula *height×1*, the value of *Y* is evaluated from this formula.

It is worth to notice that users could not change the value of an attribute directly in a formula-based system, as a value is always evaluated from a formula. In a formula-based interactive system, updating object-attributes can be achieved by updating formulas associated with these attributes. For example, if a user wants to change the color of an object to *red*, he/she could associate a new formula with the color attribute of the object (the new formula will replace the formula previously associated with the color attribute, as each attribute is expressed by one formula). The new formula may only consist of a single constant to express the color *red*. Then the constraint system will evaluate the value of the attribute according to the formula it is currently associated with, so that the value of the color attribute will be changed to *red*.

Updating a formula associated with an attribute may result in a chain of value-changes in a constraint system. For example, once the formula expressing attribute *C* is updated to *A+B*, the constraint system will evaluate the value of *C* according to the values of both *A* and *B*. Moreover, if there are other attributes defined by formulas where *C* is a referenced parameter, such as *C+D*, then the values of these attributes will be reevaluated according to the new value of *C*, and so on. This process is known as constraint propagation [4], [15], [27]. In this paper, we also use the term value-propagation to express the above process to emphasize that it is the value-change rather than formula-change that is propagated.

III. Maintaining Convergence In Formula-based Collaborative Systems

Constraints specify semantic level conditions that will automatically be maintained by the constraint systems. The needs and benefits of supporting constraints in collaborative systems have long been recognized [2], [14], [15], [16]. However, maintaining both constraints and consistency in the face of concurrency in collaborative systems is a challenge.

A. Consistency Maintenance Problems

Collaborative systems are groupware applications to support people working together in groups, such as electronic conferencing/meeting, collaborative CAD and CASE [22], [23]. 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 [22], [23]. However, as concurrent operations may be executed in different orders at different collaborating sites, maintaining consistency among replicas is more complex than sharing a single copy of centralized data, especially in collaborative systems supporting formulas, as illustrated by the following scenario:

Scenario 1: There are three object-attributes, *A*, *B*, and *C*, which are initially expressed by three constant formulas, *20*, *50*, and *60*, respectively. Three users concurrently edit formulas from different sites. User-1 hopes that the value of *C* is always

bigger than the value of *A* by *10*, so that user-1 expresses *C* with a new formula f_1: *A+10*. On the other hand, user-2 hopes that *C* could be always smaller than *B* by *10*, so that user-2 expresses *C* with another formula f_2: *B-10*. User-3 wants to change the value of *B* from *50* to *80*. Therefore, user-3 updates the formula expressing *B* to f_3: *80*.

Suppose at the site of user-1, user-1's operation is executed first, and then user-2 and user-3's operations are executed. The execution of user-1's operation will result in *C* be expressed by *A+10*. Once this formula is enforced, the constraint system will reevaluate the value of *C*, so that the value of *C* is changed from *60* to *30*. When user-2's operation is executed at the site of user-1, the formula associated with *C* is changed to f_2: *B-10,* so that the value of *C* will be changed from *30* to *40*. After user-3's operation is executed, *B* is defined by a constant formula, *80*, so that the value of *B* is changed to *80*. As *C* is constrained as *B-10,* the value-change of *B* will be propagated to the value of *C*, so that the value of *C* is changed to *70*. Therefore, after the execution of the three operations, *A*, *B*, and *C* are expressed by formulas, *20*, *80*, and *B-10*, respectively. Their values are *20*, *80*, and *70* respectively.

At the site of user-2, the three concurrent operations may be executed in the order of user-2's operation, user-3's operation, and user-1's operation. For the same reason, after the execution of the three operations, *A*, *B*, and *C* are expressed by formulas, *20*, *80*, and *A+10*, respectively. Their values are *20*, *80*, and *30* respectively. In scenario 1, both the values and formulas of *C* are different at the two sites.

B. Value, Formula, and Document Convergence

In scenario 1, different document states are maintained at different sites after the execution of the three operations at all sites, so that divergence occurs. Document convergence should be maintained in any replicated collaborative system, so that when the same set of operations have been executed at all sites, all copies of the shared document are identical [22], [23]. As an object-attribute is associated with both a formula and a value, both value convergence and formula convergence must be maintained to achieve document convergence in a formula-based collaborative system.

In an object-oriented collaborative application, a copy of a shared document consists of a group of objects that users can manipulate. Accordingly, we can define value convergence as follows:

Definition 1 (Value convergence). When the same set of operations have been executed at all sites, all the copies of the same document maintain the same set of objects with identical attribute-values at all sites.

Value convergence must be maintained in a collaborative system, as the values of object-attributes determine the behavior and appearance of objects on user interfaces. For instance, if the value of the *Y* coordinate of the top-right vertex of a rectangle is y_1 at one site, but it is y_2 at another site, where $y_1 \neq y_2$, then the vertex must be shown at different positions at different sites. Hence, divergence occurs. In a collaborative system without constraint, where object-attributes are independent of each other, value convergence guarantees document convergence. However, as value convergence does not ensure the same attribute be expressed by the same formula at all sites, it cannot guarantee document convergence in a formula-based collaborative system. For instance, suppose that two documents consist of the same set of graphic objects, G_a and G_b, and all attributes of G_a and G_b have the same values in the two documents (i.e. value convergence is maintained). In one document, the color attributes of the two objects are defined by the same constant formula, "*green"*. However, in the other document, the color attribute of G_a is defined as "*green*", but the color attribute of G_b is defined as $G_a.color$, which means G_b should have the same color as G_a. Under this condition, we cannot say that the two documents have the same state. If their states are identical, after the same set of operations have been applied to them, we must obtain the same new document states. In the above example, if an operation, which updates the color of G_a to *white*, is applied to the second document, G_b will change color to *white* as well, because G_b is defined to have the same color as G_a. However, when this operation is applied to the first document, the color of G_a will be changed to *white*, but the color of G_b will still be *green.* Therefore, we obtain different new document states.

Another type of convergence in a formula-based collaborative system is known as formula convergence that is defined as follows:

Definition 2 (Formula convergence). When the same set of operations have been executed at all sites, all the copies of the same document must maintain the same set of objects, and the formulas expressing the same object-attribute are identical at all sites.

Formula convergence must be maintained in a formula-based concurrent system, as it ensures that the same relationship is defined between the same set of object-attributes at all sites. However, formula convergence does not guarantee value convergence. For example, suppose formula convergence is maintained in a collaborative system, so that the formulas defining the same object-attribute are the same at all sites. There are two color attributes, *A* and *B*. *A* is defined by formula, *B*, and *B* is constrained by formula, *A*, at all the collaborating sites of the application. Formula convergence ensures that the same relationship between *A* and *B* has been maintained at all sites. However, it is still possible that at one site, the values of both *A* and *B* are *red*, but their values are *green* at another site. Therefore, document convergence is not maintained.

Neither value convergence nor formula convergence can guarantee document convergence. To achieve document convergence in a formula-based collaborative system, both value and formula convergences should be maintained. Accordingly, document convergence in collaborative systems that support formulas can be defined as follows.

Definition 3 (Document convergence). When the same set of operations have been executed at all sites, 1) all the copies of the same document maintain the same set of objects, 2) the values of the same object-attribute are identical at all sites, and 3) the formulas defining the same object-attribute are identical at all sites.

According to definition 3, document convergence is achieved if and only if both value convergence and formula convergence are maintained. It is obvious that maintaining document convergence in formula-based collaborative systems is more complicated than in concurrent systems without constraint, as both value convergence and formula convergence must be maintained.

C. Formula Convergence vs. Value Convergence

Before discuss strategies for document convergence maintenance in formula-based collaborative systems, it is worth to analyze the relationship between formula and value convergences.

A constraint graph can be used to express the relationship between different object-attributes in a document [4], [10], [27], which can be served as a vehicle for the investigation of the relationship among formula, value and document convergences. In a constraint graph, a circle represents an object-attribute. A triangle represents a constant, and a square expresses a formula. For any attribute, *A*, if it is expressed by formula *f* in a document, there is a directed edge from *f* to *A* in the constraint graph that expresses relationship between different object-attributes in the document. Moreover, for any object-attribute reference or constant which *f* is consisted of, there is a directed edge from the referenced attribute or constant to *f* in the graph. As an object-attribute can be expressed by exactly one formula, each attribute is pointed to by one directed edge. For example, suppose that attribute *S* is expressed by formula *f*: *(width+height)/2*, and *width* and *height* are defined by two constant formulas, f_1: *50*, and f_2: *60*, respectively. The constraint graph representing the relationship of the three object-attributes, *S*, *width* and *height*, is shown in figure 1.

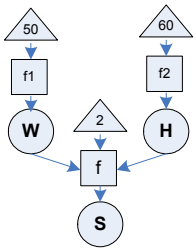

Figure 1. A constraint graph, W and H represent *Width* and *Height*, respectively

The directed edges in a constraint graph indicate the value-dependency between different object-attributes. If an attribute is defined by a formula consisting of only constant(s), such as *width* and *height* in figure 1, the constraint system can calculate the value of the attribute directly. On the other hand, if an attribute is constrained by a formula consisting of some other attribute-references, such as *S* in figure 1, the constraint system has to calculate the value of the attribute indirectly according to the values of the referenced attributes. Therefore, the value of *S* should be calculated according to the values of both *width* and *height*. Moreover, if some of these referenced attributes are constrained by formulas which consist of other attribute-references, the constraint system should calculate their values according to the attributes referenced by their formulas, and so on. This recursive calculation process can be illustrated by the following procedure:

```
valueCalculation(attribute A)
{//suppose that A is expressed by formula F
 for any attribute-reference B which F contains
      valueCalculation(B)
  calculate the value of A
}
```

In the above procedure, suppose that *A* is expressed by formula *f*. The value of *A* is calculated only if *f* is consisted of only constant(s), or the values of all the attributes referenced by *f* have been calculated. It is obvious that the above recursive procedure invocation is always guaranteed to terminate if and only if the formulas defined in the document will not form a cyclic value-propagation path, such as *A* be expressed by a formula that contains attribute-reference of *B*, and *B* be defined by a formula that contains attribute-reference of *A*. The directed edges in a constraint graph indicate the value-propagation path. If formulas defined in a document do not form a cyclic value-propagation path, then there is no cycle in the constraint graph that expresses the relationship between different object-attributes in the document.

In a replicated collaborative system, a shared document is replicated at all collaborating sites, so that each site is associated with a constraint graph that describes the relationship between different object-attributes in the document copy maintained at the site. Each time a user updates a copy of the shared document, the constraint graph corresponding to the document copy will be changed accordingly to reflect the effects of the user operation. For example, once the formula constraining *S* is changed from *f* to *f'*, this change should be reflected in the graph, so that all the directed edges from and to *f*, and *f* itself should be deleted from the graph. Moreover, *f'* and new directed edges from and to *f'* should be added into the graph.

If all collaborating sites of an application have identical acyclic constraint graphs, then value convergence must be maintained. Here, two constraint graphs are regarded as identical if and only if 1) the same set of attributes is contained in each graph, and 2) the formulas defining the same attribute are identical at both constraint graphs. According to the definition of value convergence, after the same set of operations have been executed at all sites, if all the copies of the same document maintain the same set of objects and the values of the same object-attribute are identical at all sites, then value convergence is maintained. A constraint graph expresses a complete document state, so that any attribute of any object in the document at that state must be shown in the graph. Hence, if all sites maintain identical constraint graphs, the document copies replicated at all sites must maintain the same set of objects. Moreover, for any object-attribute *A* in a document copy replicated at a site, the value of *A* is calculated according to the recursive procedure introduced above. The recursive procedure invocation is guaranteed to terminate, as all sites maintain identical acyclic constraint graphs. Furthermore, the directed edges in a constraint graph indicate the value-dependency between different object-attributes, so that the value of *A* is determined by the values of its upstream attributes

(for any two attributes, *A* and *B*, in a constraint graph, if there is a directed path from *A* to *B*, we say *A* is an upstream attribute of *B* or *B* is a downstream attribute of *A*). As any upstream attribute of *A* is defined identically at all sites, the same procedure-invocation results must be obtained at all sites. Therefore, the values of an object-attribute, *A*, must be the same at all sites. Accordingly, value convergence is maintained.

On the other hand, formula convergence ensures identical constraint graphs be maintained at all collaborating sites of a collaborative application. According to the definition of formula convergence, if formula convergence is maintained, all sites of a collaborative application maintain the same set of objects, and the formulas defining the same object-attribute are identical at all sites. Obviously, if formula convergence is maintained, the constraint graphs expressing the document states maintained at all sites must be the same.

In conclusion, formula convergence ensures identical constraint graphs at all sites. As long as the identical graphs are free from cycle, value convergence is guaranteed. If both value convergence and formula convergence are maintained, then document convergence must be achieved. Hence, our document convergence maintenance approach consists of two components: One is responsible for formula convergence maintenance. The other deals with the problem of cyclic value-propagation.

D. Formula Convergence Maintenance

Consistency maintenance in replicated collaborative systems has been investigated for decades, and many approaches have been proposed [1], [12], [22], [23], [25]. However, these approaches are originally designed for collaborative systems without constraint. Their applicability in formula-based collaborative systems has yet to be addressed.

Existing consistency maintenance approaches maintain document convergence according to the effects of user operations. User operations in an interactive application can be categorized into three abstract operations: *Create*, also known as *Insert*, which is to create an editable object, *Delete*, which is to erase an editable object, and *Update*, which changes an attribute of an object. There is almost no difference between *Create/Delete* operations executed in collaborative systems without constraint and in formula-based applications. Even if object-attributes are dependent on one another in formula-based applications, *Create/Delete* operations are still independent of each other. The creation or deletion of one object will not result in another object be created or deleted. However, *Updates* in the two types of collaborative applications are quite different. In a collaborative system without constraint, updating an object-attribute is to directly change the value of the attribute. Therefore, *U(object.key, new-value, old-value)* can be used to denote an *Update* operation, which updates the value of attribute *key* of *object* from o*ld-value* to *new-value* [25]. On the other hand, updating object-attributes is achieved by updating formulas associated with these attributes in a formula-based application. Moreover, once a formula is updated, a chain of value-changes may be propagated.

As an *Update* is expressed according to its attribute-value-change effect, existing consistency maintenance approaches ensure document convergence by achieving value convergence. In collaborative systems without constraint, where object-attributes are independent of each other, value convergence guarantees document convergence. These approaches ensures that after the same set of operations have been executed at all sites, 1) all the copies of the same document maintain the same set of objects, which is the execution effect of *Create/Delete* operations, and 2) the values of the same object-attribute are identical at all sites. The value of an object-attribute in the final document state is decided by *Update* operations that target the attribute. If no *Update* targets the attribute, it retains its initial value. Otherwise, if there are some *Updates* targeting the attribute, the final value of the attribute is determined by the last executed *Update* that targets the attribute in serialization undo/redo approach. On the other hand, an attribute-value is decided by the *Update* with the highest priority among all the *Updates* targeting the attribute in Operational Transformation (OT) approach.

Existing consistency maintenance approaches are not applicable for document convergence maintenance in formula-based applications. First of all, these approaches maintain document convergence by achieving value convergence. However, as discussed previously, value convergence cannot guarantee document convergence in formula-based collaborative applications. Moreover, most of existing approaches maintain value convergence based on the condition that object-attributes are independent of each other. It is possible that they even cannot maintain value convergence in formula-based collaborative applications, where object-attributes are related to one another.

On the other hand, existing approaches can be adopted for formula convergence maintenance in formula-based applications. The use of formulas makes object-attributes relate to each other, so that the modification of one attribute-value may be propagated to other attribute-values. However, even if attribute-values are dependent on one another, attribute-formulas are independent of each other. The change of one formula will not affect any other formulas. From the point of view of formula convergence maintenance, the value-change-effect of an *Update* operation is irrelevant. An *Update* should be expressed according to its formula-change-effect. Therefore, *U(object.key, new-formula, old-formula)* defines an *Update* which changes the formula that expresses the attribute *key* of *object* from o*ld-formula* to *new-formula*. Obviously, if an *Update* is expressed by its formula-change-effect rather than value-change-effect, applying existing consistency maintenance strategies, we can ensure that after the same set of operations have been executed at all sites, 1) all the copies of the same document maintain the same set of objects, which is the execution effect of *Create/Delete* operations, and 2) the formula expressing the same object-attribute are identical at all sites (The formula expressing an object-attribute on the final document state is decided by *Update* operations that target the attribute). Therefore, formula convergence is maintained. In the following paragraph, we use Operational Transformation (OT) approach as an example to explain how to apply existing

consistency maintenance approaches for formula convergence maintenance.

OT is an innovative and well-known consistency maintenance technique. The basic idea of OT is to transform (or adjust) the parameters of operations according to the effects of previously executed concurrent operations so that the transformed operations can achieve the correct effects and maintain document consistency [22], [23], [25]. Compared with other consistency maintenance strategies, OT is more favorable for formula convergence maintenance as it has two advantages: 1) OT is a generic strategy, which can be applied to a wide variety of collaborative applications, and 2) OT ensures document consistency independent of the execution orders of concurrent operations, which makes OT an efficient method as it will not undo/redo operations to ensure the same execution order of concurrent operations at different collaborating sites. Undoing/redoing an operation may result in a chain of value-change propagations in formula-based collaborative applications, which consumes extra system-time.

If an *Update* is expressed according to its formula-change-effect, two *Update* operations O_a and O_b are regarded as conflict with each other, expressed as $O_a \otimes O_b$, if and only if they are concurrent and they update the formula associated to the same attribute of the same object. In contrast, O_a and O_b are compatible, if and only if they do not conflict with each other [23]. According to OT, if editable objects are identified independently, transforming any operation against its compatible operations will not change any parameter of the operation. On the other hand, revised function conflictResolution() is used to transform an operation, O_a, against its conflicting operation O_b, as sketched below.

```
conflictResolution(Oa, Ob)
{ //O.priority expresses the priority of operation O
if(Oa.priority<Ob.priority)
    Oa.new-formula=Ob.new-formula
Oa.old-formula=Ob.new-formula
return Oa
}
```

The application of OT for formula convergence maintenance can be illustrated by using scenario 1 as an example:

In scenario 1, object-attributes, *A*, *B*, and *C*, are initially expressed by three constant formulas, *20*, *50*, and *60* respectively. Three users concurrently edit formulas from different sites. User-1 associates formula f_1: *A+10* with *C*, and user-2 constrains *C* as f_2: *B-10.* User-3 changes the formula expressing *B* to f_3: *80*. Three OT *Update* operations can be used to express the three user operations respectively: O_1=*U(C, new-formula=A+10, old-formula=60)*, O_2=*U(C, new-formula=B-10, old-formula=60)*, and O_3=*U(B, new-formula=80, old-formula=50).* Here, suppose that O_1 has the highest priority, and O_3 has the lowest priority among the three operations.

At the site of user-1, the three operations are executed in order: O_1, O_2, and O_3. After the execution of O_1, the formula expressing *C* is changed to *A+10*, and the value of *C* is changed to *30* accordingly. Once O_2 arrives at the site of user-1, it will be transformed against O_1, as both operations target the same attribute-formula. As O_1 has a higher priority than O_2, after the function conflictResolution(O_2, O_1) is invoked, O_2 is transformed to *U(C, new-formula=A+10, old-formula=A+10)*. Therefore, after the execution of the transformed O_2, *C* is still expressed by formula *A+10*, and its value is still *30.* Once O_3 is executed, the formula expressing *B* is changed to *80* and *B*'s value is *80*. After the execution of the three operations at user-1's site, *A.value=20*, *A.formula=20*; *B.value=80*, *B.formula=80*; *C.value=30*, *C.formula=A+10*.

At the site of user-2, the three operations are executed in order: O_2, O_3, and O_1. After the execution of O_2, the formula expressing *C* is changed to *B-10*, and the value of *C* is changed to *40*. Once O_3 is executed, the formula expressing *B* is changed to *80* and *B*'s value is *80*. As *C* is constrained as *B-10*, the value-change of *B* will be propagated to *C*, so that the value of *C* is changed to *70*. When O_1 arrives at the site of user-2, it will be transformed against O_2, as they are conflict. O_1 has a higher priority than O_2, so that O_1 is transformed to *U(C, new-formula=A+10, old-formula=B-10)*. Therefore, after the execution of the transformed O_1, *C* is expressed by formula *A+10*, and its value is changed to *30.* After the execution of the three operations at user-2's site, *A.value=20*, *A.formula=20*; *B.value=80*, *B.formula=80*; *C.value=30*, *C.formula=A+10*.

In the above example, even if the three operations are executed in different orders at different sites, formula convergence is maintained. Moreover, as no cycle is formed in any value-propagation path, both value and document convergences are maintained.

E. Preventing Cyclic Propagation Path

Cyclic propagation path must be prevented. Otherwise, value-propagation cannot stop without outside interference. For instance, suppose that attribute *A* is defined by *B-C*, while *B* is expressed as *A+C*, so that a cyclic propagation path exists between *A* and *B*. Once the value of *C* is changed, this change must be propagated to the values of both *A* and *B*, as *C* is a referenced attribute in both formulas expressing *A* and *B*. Because *A* is a referenced attribute in the formula defining *B*, the value-change of *A* will result in the value-change of *B*. For the same reason, the value-change of *B* will result in the value-change of *A*, so that a looped propagation occurs.

In a single user system that supports formulas, such as Microsoft Excel and Visio, if a user enforces a new formula that results in a cyclic value-propagation path, the user will get a warning and the operation will have no effect (i.e. not executed by the system). This strategy can be adopted by collaborative systems. If a group of concurrent operations form a cyclic value-propagation path, one operation in the group will be masked. Masking an operation is to temporarily eliminate the operation's effects from the current document state to break a cyclic propagation path.

Masking operations in concurrent environments may result in divergence. For instance, two users concurrently associate formulas to *A* and *B* respectively, so that *A* is expressed by a formula that contains attribute-reference of *B*, and *B* is defined by a formula that contains attribute-reference of *A*. Suppose at one site, user-1's operation is executed first. The execution of

user-2's operation at the site will result in a cyclic propagation path. Therefore, user-2's operation is masked at the site. At another site, user-2's operation is executed first. For the same reason, user-1's operation is masked there. As different operations are masked at different sites, divergence occurs.

The key point to design a masking approach in collaborative systems is to ensure that the masking effect will not interfere with consistency maintenance result, so that all sites of a collaborative application maintain the identical acyclic constraint graphs. A method to achieve this aim is described below.

A Cycle Prevention (CP) component is maintained at each site of a collaborative application. A CP maintains a constraint graph that represents the relationship between different object-attributes in the document copy replicated at the site where the CP is running. Each time a user updates the document copy replicated at a site, the CP running at the site will change the constraint graph corresponding to the document copy to reflect the effects of the user operation. It is noteworthy that any operation performed by CP has effect on neither user-interface nor the shared documents. CPs' actions only have effects on constraint graphs. Therefore, executing/undoing/redoing an operation by CP is only to change the constraint graph according to the effect of the operation. For example, suppose O_i changes formula constraining S from f to f', undoing O_i by CP will result in that all the directed edges from and to f,' and f' itself are deleted from the graph. Moreover, f and directed edges from and to f are added into the graph.

The functionality of CP components is to ensure that all sites maintain the identical acyclic constraint graphs. This can be achieved based on serialization undo/redo strategy. Once an operation, O, is ready for execution at a site, it will be sent to the CP component running at the site. Accordingly, the CP component will undo all the operations, which have been executed at the site and have higher timestamp-values than O, in timestamp-value descending order. After all the operations that have higher timestamp-values than O have been undone, CP will execute O (i.e. update the constraint graph according to O). If the execution of O will result in cyclic propagation path(s), O is masked. Then all the operations that are undone for executing O will be redone in timestamp-value ascending order. Otherwise, if the execution of O will not result in any cyclic propagation path, all the masked and undone operations that have higher timestamp-values than O will be checked in timestamp-value ascending order. If unmasking/redoing a masked/undone operation will not result in any cycle in the current constraint graph, the masked/undone operation is unmasked/redone. On the other hand, if redoing an undone operation will result in cycle(s), the undone operation will change its state from *undone* to *masked*. Serialization undo/redo strategy ensures that operations are executed in the same order at all sites, so that the same set of operations will be masked at all sites. Therefore, each site must maintain identical acyclic constraint graph.

Once CP determines which operations should be masked or unmasked, these masking/unmasking effects must be applied to the shared documents. After the ready-for-execution operation, O, is processed by CP, CP will send three groups of operations to Formula Convergence Maintenance (FCM) component. The first group consists of only operation O, which is the newly arrived operation that should be executed at the site. The second group consists of all the operations that should be masked, and the third contains all the operations that should be unmasked, after the execution of O. Suppose that FCM implements OT strategy. Then, OT will transform O against its concurrent operations and execute the transformed O. For each operation in the second group, OT will undo the operation to mask the operation on user interface. For any operation in the third group, the operation must have been undone by OT previously. Accordingly, OT will redo it to recover its effect. It is worth to notice that value-propagation must be delayed until OT finishes executing O and undoing/redoing all the operations that should be masked/unmasked, because before OT finishes these actions, it is still possible that cycles exist in the constraint graph.

Delaying value-propagation can improve system responsiveness, as performing value-propagation immediately each time a formula is associated with an attribute is unnecessary. For instance, suppose that S is expressed by formula, *(width+height)/2*. Two concurrent operations, updating the formulas defining *width* and *height* respectively, arrive at a site at the same time. Under this situation, it is desirable to delay value-propagation until both of the two operations have been executed, so that the value-changes of both *width* and *height* can be propagated to S by one value-propagation action. It is better than performing value-propagation twice, one for propagating the value-change of *width* to S, the other for *height.*

IV. COLLABORATIVE VISIO (COVISIO)

The document convergence maintenance approach introduced in the above section has been applied to leverage single-user Microsoft Visio system for multi-user collaboration. The collaborative Visio system, called CoVisio, enables a group of users to view and edit the same Microsoft Visio documents at the same time from different collaborating sites. CoVisio adopts replicated architecture and is implemented in the programming language C# based on Visio API without knowing or modifying Visio source code. The interface of CoVisio is shown in figure 2.

Figure 2. The CoVisio interface

A. Visio Formula

Microsoft Visio is one of the most prevalent commercial single-user graphic editing systems, which can be used to create a wide variety of business and technical drawings. One feature that distinguishes Visio from other graphic editing systems is that formulas are defined in Visio to express the attributes of each graphic object, and the relationship between different Visio graphic objects. The ability to describe shapes with formulas opens many possibilities for making shapes behave in complex and sophisticated ways.

An attribute of a graphic object, called a *cell* in Visio, is expressed by a formula. A Visio formula may contain constants, operators, functions, and object-attribute references. Microsoft Visio evaluates a formula to a result and then converts the result to the appropriate units for the attribute that contains the formula [17]. In a Visio *ShapeSheet* window, a user can display cell-contents as either values or formulas by clicking the appropriate command on the *View* menu.

The same as other formula-based applications users could not change the value of an attribute directly in Visio, as a value is always evaluated from a formula. Each time a user updates an attribute of a graphic object, he/she directly changes the formula expressing the attribute. There are three ways to change formulas in Visio: 1) Through Visio drawing pages by mouse/keyboard operations**.** For example, when a user moves a shape with the Pointer tool, Visio automatically changes and then reevaluates the formulas that define the shape's center of rotation, or pin, on the drawing page, because those formulas determine the shape's location on the page. 2) Through Visio *ShapeSheet* window where users can edit formulas directly. A *ShapeSheet* window gives users more precise control over the appearance and behavior of an object. 3) Through Visio API, where developers can modify formulas by program.

B. CoVisio Components

CoVisio is built by extending single-user Microsoft Visio into a multi-user collaborative application. The method, adopted to leverage commercial single-user Microsoft Visio for multi-user real-time collaborations, is known as Transparent Adaptation (TA) approach, which was first proposed by CoWord and CoPowerPoint projects [24], [26]. TA is based on the use of the single-user applications' APIs to intercept and replay users' operations, so it requires no access or change to the applications' source codes (thus being transparent).

A TA based collaborative application is composed of three components. The first component is a Single-user Application (SA), i.e., MS Word/PowerPoint or Visio, which provides the conventional single-user functionalities and interface features. This component is completely collaboration unaware.

Another component is Generic Collaboration Engine (GCE), which provides application-independent collaboration capabilities. This component is fully collaboration-aware, but completely unaware of the single-user application. Two critical functions of GCE in CoVisio are consistency and constraint maintenance. Operational Transformation (OT) is implemented in GCE for consistency maintenance. Constraint maintenance is responsible for preventing generation of cyclic value-propagation paths.

GCE is generic, but SA is not. SAs may define different data and operation models. Therefore, the third component, Collaboration Adapter (CA), is implemented to adapt application-specific SA to generic GCE. CA provides application dependent collaboration capabilities and is aware of both the single-user and multi-user collaboration applications.

The interactions between the three components in processing an editing operation can be illustrated based on the following simple scenario in a CoVisio application, as shown in figure 3.

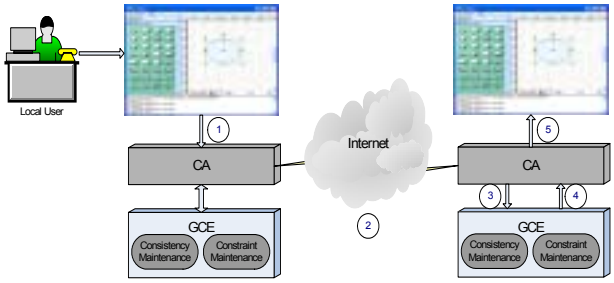

Figure 3. The interactions between CoVisio components

Suppose a user uses the keyboard and/or mouse to edit a graphic object in a shared Visio document, the following events shall occur at the local site:

(1) Once the operation is performed on the local document, the operation semantics is sent to CA via SA's API. Then it is translated into an OT recognizable operation by CA.

(2) The OT recognizable operation is propagated to remote sites by CA.

When the operation arrives at a remote site, the following shall happen:

(3) The received operation will be passed to GCE.

(4) The operation is processed by GCE for consistency maintenance and constraint satisfaction. After that, the processed operation is passed to CA.

(5) A suitable SA's API function is invoked by CA to replay the remote operation at the site.

C. Operation Interception and Replaying

In CoVisio, user mouse/keyboard operations are directly inputted into single-user Visio application (i.e. SA). To interpret the effects of user operations, Collaboration Adapter (CA) component of CoVisio registers some event-handlers in the single-user Visio application via Visio API. Therefore, when the events CoVisio interested arise, the single-user Visio application would automatically inform CA. For example, CA registers *ShapeAdded* event-handler on each Visio *page* object, so that each time a shape is added into a drawing page, CA will receive the detailed information of where and what a shape is created. In Visio, users update attributes of graphic objects by modifying formulas. Each time the formula associated with an attribute is updated a formula-change event and a chain of value-change events will be triggered. CA registers event-

handlers for formula-change events instead of value-change events, as value-changes are only side effects of the user operations, which are performed automatically by underlying Visio system.

CA abstracts user mouse/keyboards operations into three OT defined operations, *Insert*, *Delete* and *Update*. Once a user creates/deletes a Visio graphic object, the user operation will be abstracted to an OT *Insert/Delete* operation. If a user edits a formula associated with an attribute, CA will obtain a formula-change event. Then the detailed operation information, reported by the event, will be translated into an OT *Update* operation. Only the formula-change effect rather than value-change effect of the operation will be recorded in the *Update*. CA also associates timestamp and priority information with the OT defined operation and marshals the operation-information into a message sent to remote sites.

Once the message is received by CA component running at a remote site, it is passed to the constraint maintenance module of GCE component. The constraint maintenance module will check whether the execution of the operation will result in cyclic value-propagation and determine which operations should be masked and unmasked to ensure identical acyclic constraint graphs at all sites. Then, constraint maintenance module informs OT module in the same GCE which operations should be executed, masked, or unmasked. OT functions are performed to transform these operations against their concurrent operations that have been executed at the site, which is to achieve formula convergence, intention preservation and causality preservation [22], [23]. After that, these transformed operations are passed to CA, where suitable Visio API functions will be invoked to apply these operations to the Visio document replicated at the site.

V. Related Work

There is a large body of researches contributing to constraint maintenance in user interactive applications [3], [4], [10], [19], [27]. However, these researches focused on single-user applications. Consistency maintenance in collaborative systems supporting constraints is beyond their scope.

IceCube[13], Actions Constraints Framework [21], Doppler [2] and CAB [14] are related to constraint control in collaborative applications. IceCube explicitly captures constraints between actions. In Actions Constraints Framework, actions (operations accessing shared data, submitted by clients of a replicated system) are connected by binary constraints, which must be maintained by the systems. Compared with the above schemes, formulas are used to define the relationship between object-attributes, rather than constraints between actions.

Doppler supports distributed, concurrent, one-way constraints in user interface applications, which provides a high degree of concurrency and works in a asynchronous manner without the need for shared memory, shared or synchronized clocks, or centralized locking [2]. As shared documents are not replicated at different collaborating sites in a Doppler-based application, consistency maintenance is beyond Doppler's concern.

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. Constraints in CAB include those for coordination between distributed users such as awareness, access, and concurrency control, which are beyond the scopes of graphic objects [14]. However, just as its authors stated, 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 methods to maintain multi-way dataflow constraints in collaborative systems have been introduced in [15], [16]. A multi-way dataflow constraint can be expressed by an equation, such as $C=A+B$. Both multi-way dataflow constraints and formulas define relationship between object-attributes, the difference between them is that a formula expresses a specific attribute, but a multi-way dataflow constraint does not. For example, a multi-way dataflow constraint may define the relationship between three attributes: $C=A+B$. As the constraint is not to express a specific attribute, once a constrained attribute is updated, there are multiple options for performing constraint propagation. For instance, once a user changes A, the change may be propagated to either B or C to satisfy dataflow constraint $C=A+B$. On the other hand, as a formula is always associated with an attribute, the constraint propagation path is predefined. For instance, if C is expressed by formula $A+B$, the value-change of either A or B must be propagated to C. Accordingly, the main issue for multi-way dataflow constraint maintenance in collaborative systems is determining the propagation path according to the effects of concurrent operations. As described above, once C is updated, to satisfy constraint $C=A+B$, two constraint propagation paths are available: propagating the change to A or to B. If different propagation paths are adopted at different sites, divergence occurs. On the other hand, the constraint satisfaction problem in formula-based collaborative systems is that when users concurrently associate formulas with attributes, different formulas may be defined to express the same attribute at different sites. How to solve this problem is discussed in detail in this paper.

Generally speaking, multi-way dataflow constraints are more flexible and powerful, so that they are more complicated to maintain in collaborative systems. However, multi-way dataflow constraints have one drawback that impeded their acceptance. The multiple possibilities of propagation-paths often make constraint propagation results unpredictable. By contrast, because of their simplicity, efficiency and manageability, formulas have been adopted to express constraints in many types of interactive systems.

VI. Conclusion And Future Work

Formulas are adopted to express constraints in a wide variety of object-oriented applications, which can define relationship between object-attributes. The needs and benefits of supporting constraints in collaborative systems have long been recognized. However, maintaining constraints expressed as formulas in collaborative environments is a challenge. The

difficulties are caused by concurrent operations that result in value, formula and document divergences. Being able to solve this problem is crucial in the development of collaborative applications supporting formulas, such as collaborative spreadsheets, graphic editing systems, CAD, CASE, etc.

In this paper, we proposed a method to solve this problem. Our solution is generic. It consists of two components, one for formula convergence maintenance and the other for handling cyclic value-propagation. In our solution, formula convergence maintenance is achieved by existing consistency maintenance strategies, which are originally designed for collaborative systems without constraint. The application of these strategies in collaborative systems with constraints has never been addressed before. Our solution extends the application of these strategies, especially OT, from collaborative systems without constraint to systems supporting formulas. We applied our solution to CoVisio system to maintain both Visio formulas and consistency in concurrent environments. The constraint maintenance method implemented in CoVisio is generic and can be adopted by other collaborative systems that support formulas, such as collaborative spreadsheets, CASE, and CAD.

We are currently investigating how to efficiently propagate value-changes using multi-thread processes. Multi-thread can improve system performance and responsiveness, but the concurrently executing multi-thread may interfere with each other. How to efficiently coordinate the executions of multi-thread in performing constraint propagations will be reported in our future publications.

Over the last fifteen years, real-time collaborative systems have moved from being prototypes in laboratories to becoming usable commercial systems and also freeware. With the investigation of maintaining formula-defined constraints in collaborative systems, we hope to make real-time collaboration even much easier to build and use.

REFERENCES

[1] J. Begole, R.B. Smith, C.A. Struble, and C.A. Shaffer, "Resource sharing for replicated synchronous groupware", *IEEE/ACM Transactions on Networking*. Vol. 9, No. 6, pp.833-843, Dec. 2001.

[2] K. Bharat, and S.E. Hudson, "Supporting distributed, concurrent, one-way constraints in user interface applications", In *Proceedings of the ACM Symposium on User Interface Software and Technology*, ACM, New York, pp.121-132, 1995.

[3] A. Borning, and R. Duisberg, "Constraint-based tools for building user interfaces", *ACM Transactions on Graphics*, vol.5, no.4, pp.345-374, Oct. 1986.

[4] A. Borning, B. Freeman-Benson, and M. Wilson, "Constraint hierarchies", *Lisp and Symbolic Computation*, Vol. 5 No. 3, pp.223-270, Sept. 1992.

[5] J.D. Campbell, "Multi-user collaborative visual program development", *IEEE Symposia on Human Centric Computing Languages and Environments*, Arlington, VA, pp.122-130, 2002.

[6] J.D. Campbell, "Interaction in collaborative computer supported diagram development", *Computers in Human* Behavior Vol. 20, No.2, pp.289-310, 2004.

[7] P. Dourish, "Developing a reflective model of collaborative systems", *ACM Transactions on Computer-Human Interaction*, Vol. 2, No.1, 1995.

[8] P. Dourish, "Consistency guarantees: Exploiting application semantics for consistency management in a collaborative tookit", In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, ACM, New York, pp.268-277, 1996.

[9] W.K. Edwards, "Flexible conflict detection and management in collaborative applications", In *Proceedings of the ACM Symposium on User Interface Software and Technology*, ACM, New York, pp.139-148, 1997.

[10] B. Freeman-Benson, J. Maloney, and A. Borning, "An incremental constraint solver", *Communications of the ACM*, Vol. 33, No.1, pp.54-63, Jan. 1990.

[11] D.R. Hill, "The RENDEZVOUS constraint maintenance system", In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp.225-234, 1993.

[12] C.-L. Ignat, and M.C. Norrie, "Grouping in collaborative graphical editors", In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, Chicago, USA, pp.447-456, Nov. 2004.

[13] A.M. Kermarrec, A. Rowstron, M. Shapiro, "The IceCube approach to the reconciliation of divergent replicas", In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pp. 210-218, 2001

[14] D. Li, and J. Patrao, "Demonstrational customization of a shared whiteboard to support user-defined semantic relationships amongst objects", *ACM GROUP'01*, Boulder, Colorado, USA, pp.97-106, Sept. 2001.

[15] K. Lin, D. Chen, C. Sun, and R.G. Dromey, "Maintaining constraints in collaborative graphic systems: the CoGSE approach", In *Proceedings of the 9th European Conference on CSCW*, Paris, France, Sept. 2005.

[16] K. Lin, D. Chen, C. Sun, and R.G. Dromey, "Maintaining multi-way dataflow constraints in collaborative systems", In *Proceedings of IEEE 2005 International Conference in Collaborative Computing: Networking, Applications and Worksharing*, San Jose, CA, USA, Dec. 2005.

[17] Microsoft, Developing Microsoft Visio solutions, http://msdn2.microsoft.com/en-us/library/aa245244(office. 10).aspx.

[18] E. Monfroy, and C. Castro, "Basic components for constraint solver cooperations", *Proceedings of SAC*, 2003.

[19] B.A. Myers, "Graphical techniques in a spreadsheet for specifying user interfaces", In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, User Interface Management Systems, pp. 243-249, 1991.

[20] M. Sannella, J. Maloney, B. Freeman-Benson, and A. Borning, "Multi-way versus one-way constraints in user interfaces: experience with the DeltaBlue algorithm", *Software-Practice and Experience*, Vol. 23, No.5, pp.529-566, 1993.

[21] M.Shapiro, K.Bhargavan, "The actions-constraints approach to replication: definitions and proofs", Technical peport MSR-TR-2004-14. Microsoft Research, Mar. 2004.

[22] 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 Transactions on Computer-human Interaction*, Vol. 5, No.1, pp. 63-108, Mar. 1998.

[23] C. Sun, and D. Chen, "Consistency maintenance in real-time collaborative graphics editing systems", *ACM Transactions on Computer-Human Interaction*, Vol. 9, No.1, pp.1-41, Mar. 2002.

[24] C. Sun, Q. Xia, D. Sun, D. Chen, H. Shen, and W. Cai, "Transparent adaptation of single-user applications for multi-user real-time collaboration", A*CM Transactions on Computer-Human Interaction (TOCHI*), Vol. 13, No.4, pp.531-582, Dec. 2006.

[25] D. Sun, Q. Xia, C. Sun, and D. Chen, "Operational transformation for collaborative word processing", In *Proceedings of the ACM Conference on CSCW*, Chicago, USA, Nov. 2004.

[26] Q. Xia, D. Sun, C. Sun, D. Chen, and H. Shen, "Leveraging single-user applications for multi-user collaboration: the CoWord approach", In *Proceedings of the ACM Conference on CSCW*, Chicago, USA, 162-171, Nov. 2004.

[27] B. Zanden, "An incremental algorithm for satisfying hierarchies of multi-way dataflow constraints", *ACM Transaction on Programming Languages and Systems*, Vol.18, No.1, pp.30-72, Jan. 1996.