1996

Object Based Constraint Management for Collaborative Systems

Chandrajit Bajaj

Peinan Zhang

Report Number:
96-039

OBJECT BASED CONSTRAINT MANAGEMENT FOR COLLABORATIVE SYSTEMS

Chandrajit Bajaj
Peinan Zhang

Department of Computer Science
Purdue University
West Lafayette. IN 47907

CSD TR-96-039
June 1996

Object Based Constraint Management for Collaborative Systems

Chandrajit Bajaj Peinan Zhang
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Email: {bajaj,pnz}@cs.purdue.edu
Phone: (317)494-6531 Fax: (317)496-2567

June 28, 1996

1 Introduction

Although interest in distributed and collaborative applications is growing rapidly, construction of such applications is still difficult and time consuming. Flexible sharing and system consistency are fundamental problems. A typical example is a collaborative design system in which multiple users are involved. These users may be separated in geographically distributed sites, may share the same working space, and may be able to access and manipulate a shared object (data, view, etc) simultaneously through distributed interfaces. In a more complicated case, a shared object can be partitioned among users. A whole design task can be partitioned among users as well, and the same partial or final result can be viewed by all users. All of this requires a flexible sharing capability and introduces many consistency problems. More flexible and powerful sharing capabilities are required and more work in management and control is needed. When developing such application, there will be much more effort spent on sharing management and consistency maintenance than on the implementation of the application function itself.

If a general mechanism could be designed to handle sharing and system consistency, construction of multi-user applications would be greatly simplified.

Because of the diversity in motivations and interests among users and the variety of application domains, the types of relationships between objects, the meanings and the requirements for sharing and consistency could be very different. Thus, the maintenance mechanisms we need should be flexible, efficient and easily controlled.

Constraint systems have proved to be useful mechanisms to maintain relationships among values, such as the geometric or dependent relationship among graphical objects, or the consistency between underlying data and their graphical depiction. This technique has been widely used in many interactive graphical user interface systems [6, 22, 15]. So far, few system uses constraints to link the behaviors (i.e, functions, instead of variables) of objects.

We believe that a constraint system is a useful and ideal mechanism for specifying shared behaviors and maintaining relationships in multi-user systems in which there are a great many state variables that must be kept consistency. Furthermore, a multi-user system always allows independent controls to distributed views, and concurrent manipulation to internal data from distributed views. So consistency between data and views, or between distributed views are become complex, such that some time it is difficult to specify relationships only by variable value. The motivation of this work is to design a constraint management system to specify and maintain relationships among real object, that is not only the object variables values, but also object methods, so that those complex relationship in multi-user applications can be easily specified and maintained.

Prior Approaches

Constraint systems are widely used in many interactive graphical direct-manipulation systems, such as Coral [22, 23], Garnet [15], ThingLab II [14, 8, 9, 20], and Multi-Garnet [19]. We mainly investigate where the constraints are used and how they are used.

Geometric layout is a most natural application for constraints which are used for specified and maintained the geometry relations among entities or objects. Sketchpad [21] is the earliest system in this area, it used constraints to define the geometric relationships (parallel, attach, etc..) among geometric lines. The systems in this area include ThingLab [5], Juno [16] and Magritte [10].

Constraint-based user interface systems extends using constraints to specify the dependent relationship (e.g., parent and child relation) of interface widgets, as well as to make the consistency between application data and its graphical depiction. Grow [4], Coral, Garnet are some of these systems. Garnet [15], developed from Coral, is a user interface toolkit based on a prototype-based object systems. It relies on active value architecture, in which an active value is registered by application program with a set of callbacks procedures and is linked with a list of graphical objects, as shown in Figure 1. Once the active value is set, the constraints will propagate the change to linked graphical objects, and will call the procedure to inform the application about the changes. Garnet extends constraints to support the construction of complex compound of objects containing multiple graphics objects and constraints.

Rendezvous [18, 13] system provides high-level support for developing synchronous multi-user applications. It is based on Abstraction-Link-View architecture, in which applications are separated as abstract objects and view objects, and users from multiple views can interact with a central abstraction through constraint "Links", as shown in Figure 2. Rendezvous constraint system provides many features of Garnet in building its graphics system. And extensively, the constraint maintenance system in Rendezvous is used to link between the abstraction and views, to maintain the consistency between the multiple views, and to hide concurrent access from multiple users.

Although these systems have different capabilities and complexities, they are all based on common features. One is that the constraints are only associated with values, no matter graphical constraints or data constraints as defined in [22]. So if we think that the instance variables define object states, and methods describe object behaviors, then the existing systems only maintain the relationship among object states, but not object behaviors. The second is that value items are visible to all other values, this is contrast to object based

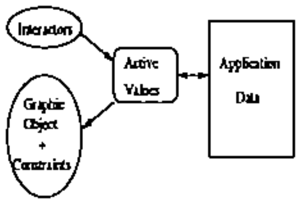


FIGURE 1: Application structure in Garnet

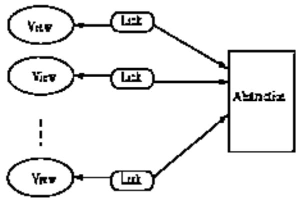


FIGURE 2: Application structure in Rendezvous

systems, and make them difficult to include heterogeneous objects. The third is that they assumed that all relationships between objects have the same requirements, so that they are represented and handled in the same way. Thus, programmers design more and more powerful methods for representing and solving constraint relationships (in order to increase their capabilities). Another assumption is that only one outside perturbation can apply to a set of objects at a time. Although Rendezvous allows multiple access to a shared object at once, the basic function of its access constraints is to hide concurrent access, using a turn-taking technique to serialize the multiple accesses to a sequence of cases.

Main Results

Because of our interests in the construction of multi-user applications, we have different emphases and different requirements than in single-user user interface systems. The above features and assumptions of existing systems are not always satisfied to a multi-user collaborative application. Here we mainly pay attention to the features unique to multi-user applications. These features include:

1. Internal data may have multiple views distributed in different sites. Each user from a distributed view can set up control to view objects independently. At sometime, it requires that these views be tightly constrained as WYSIWIS (What You See Is What I See), or while at the other time, it requires that these views be customized independently.

2. Multiple users can work in the same shared working space and are able to access a shared object simultaneously; at once, more than one user can manipulate the same set of constrained objects.

3. Working spaces can be partitioned among users so that each user can work independently on the interior of a partition. The boundary part will be subjected to access constraint. Any change in the boundary may affect all adjacent partitions. This implies that some constraint relationships may be created dynamically, and the constraint maintenance should force the changes to propagate into more than one subspace.

4. An application task can be partitioned among users, thus any action or result in one sub-task may affect the actions in other sub-tasks. The dependencies among tasks are difficult to be specified by values.

In general, a multi-user collaborative application will introduce many more types of objects and relationships than a single-user application does. These relationships are associated with object states as well as object behaviors. So it is desirable for a constraint system to maintain the relationship among variable value, as well as functions.

In a multi-user application, the complicity of these relationships are different widely.If a weak sharing and maintenance system is provided as an engine for construction of multi-user applications, an application programmer is forced to handle a lot of consistency controls. If an overly powerful maintenance system is provided, an application's performance will most likely be degraded by unnecessary overhead. We believe that a suitable constraint system for a multi-user application should provide many methods for specifying and maintaining relationships and allow different applications and different types of relationships to choose and link with the appropriate methods.

Lastly, we believe that simple turn-taking is unacceptable for collaborative applications in which human factors are involved.

Our work is motivated at the extension of constraints to the real object domain, and the investigation of a new way to flexibly organize a constraint management system based on existing technique rather that definition of a constraint-based graphical system or design new constraint solving mechanism.

This paper presents a flexible paradigm in which a constraint system is abstracted as objects and services, and provides multiple constraint maintenance services to multiple types of objects. This flexibility, especially constraints associated with object methods and concurrent modification, allows us to develop constrained based collaborative systems that are much more general than prior systems such as Coral, ThingLab, or Rendezvous. This paradigm is incorporated into Shastra, a distributed and collaborative geometric design and scientific manipulation environment, which is designed to ease the construction of multi-user distributed and collaborative applications.

The Shastra constraint system provides an effective way to implement our object service model. As a special consideration for multi-user applications, we designed a constraint maintenance service which allows multiple users to simultaneously manipulate the shared objects. A set of collaborative applications based on the constraint system were developed in the Shastra environment. Our experiences show that this paradigm greatly simplified the creation of collaborative applications.

In the rest of this paper, first we introduce our model and its features. Then we describe how we implemented the constraint system in Shastra. Next, we discuss a new constraint solving method which allows a set of constrained objects to be manipulated by multiple changes simultaneously. Several collaborative applications designed in the Shastra environment are presented to demonstrate some initial results. Finally, we draw conclusions and discuss future work.

2 Model

The main idea of our design is to provide a generic constraint maintenance service to a common object system. In this model, a constraint system consists of two parts: an object system and a maintenance system. The object system contains application-defined objects and relationships, whereas the maintenance system includes a set of services with different solving methods. This model can be described as following:

1. The object system contains a set of objects, which could be centralized or distributed.

2. The relationship can be defined not only among instance variables in objects, but also among methods in objects.

3. The relationships among objects can be different types (e.g., unidirection or bidirection. etc.) according to application requirements. Different types of relationships can be represented and maintained in different ways.

4. The maintenance system consists of a set of constraint services with different solving capability and complexity. Each service can be "connected" to objects to maintain the constraint relationships independently.

5. If one type of relationships in a set of objects is completely independent of others, it can be maintained iteratively or concurrently with other relationships.

The conceptual model is shown in Figure 3.

Based on this model, a constraint system is just as a set of object services with connection interfaces; thus, application programmers can specify the relationships among objects in multiple types according to application requirements and "link" them with the appropriate services.

2.1 Design and Implementation Issues

Usually, a constraint systems is defined as a set of variables, which store values, and relationships specifications. And a constraint relationship is specified as a set of source variables, one or more target variables, and one or more evaluation functions. Whenever the value of any of source variables changes, the functions is applied to the new value to create a value to targets.

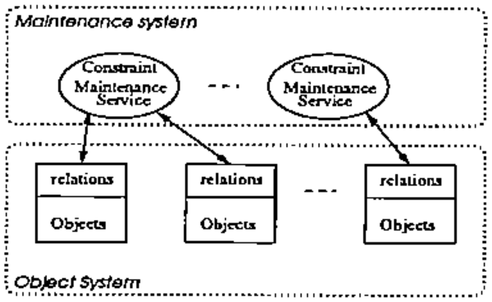


FIGURE 3: The conceptual model of a constraint system

In the object service model, the constraint system is defined by a set of objects, which is stored values, methods, and constraint relationships. And a constraint relation is specified as a set of source items, target items, and constraint functions. The source items and target items could be variable values in objects, or methods in objects. Whenever any source item is "write" accessed, the constraint function will be forced to create new value to the target variable or to trigger the target method.

The followings are important issues in the design and implementation of this model.

1. The object domain of the constraint system.
2. The definition or specification of the relationships among objects.
3. The constraint services provided by the constraint system, and the classification or organization of the services. This directly reflects the capability of a constraint system.

These issues reflect the features of our design and are unique in our model, we will discuss them in our design and implementation of the Shastra constraint manager.

2.2 Features

This model provides the following features:

- Since it provides multiple maintenance services, it is efficient in maintaining simple relationships, and provides the capability to represent and maintain complicated relationships as well.
- The object based model supports encapsulation, such that it is independent of object implementation and can easy to maintain the relationships among heterogeneous objects.
- It is based on a generic object system, so that it can specify relationships among object states, as well as object behaviors. In this sense, some complex, such as domains and tasks relations, can be specified easily and maintained uniquely.

- This object service model makes it possible to implement concurrent or distributed maintenance services. Once the sets of objects or the constraint relationships in different types are completely independent, they can be maintained by multiple processes simultaneously. This leads to much better performance when a complex system involves several thousands of constraints.
- We abstract constraint maintenance as an object service, which makes our constraint system easy to extend and integrate with any existing or special constraint services.

3 Shastra Constraint Manager

Shastra is a distributed and collaborative design and scientific manipulation environment, its system architecture and infrastructure for the construction of collaborative multimedia applications are described in detail in [1, 2, 3].

In this section, we mainly discuss the object domain of our constraint system, and how Shastra provides the interfaces to application objects, so that they can flexibly connect to the appropriate service. The next section will discuss how we organize our constraint maintenance services.

3.1 Object Domains

Shastra environment is used to construct multi-user applications for geometric modeling, collaborative design and collaborative visualization.

These applications usually contain complex structure data. Each structure data contain thousands of components with a small number of different types, such as a polyhedron structure data usually contain thousands of points, edges, and faces. Constraints are used to maintain relationships among components external to a structure data instead of internal to a structure data. Comparing with the amount of components in an objects, only a few components will have constraint relationships with others outside of the structure. Most of time, the components of a structure data will have same features, or states, and the constraints will be reevaluated whenever any of them changes. So, define constraints for each component of a structure data will be time and space consumed. Thus, we define the structure data as basic constrained data objects, such that the constraints are used to specify the relationships among their components. And an aggregate object type is defined for each type of components, which can be used to specify the relationships of this type of components hold. And the aggregate objects will have components pointed to components which are constrained with others.

Besides data objects, the constraint systems also includes view objects. The view objects in Shastra are 3D graphical objects, rather than 2D objects, so it will contain much more control information which is only related to 3D graphical views, such as camera, light, or some event functions. So constrain connection among multiple distributed view objects will be complex and different from other existing systems.

Constraint is defined as a separated object which links variables or methods among objects.

User interaction with view objects will changes the view objects, functions in application core will update the data objects, while constraint manager will evaluate the constraint objects, which will propagate the changes from view object to data object, vice visa, and, will propagate the change from view object to view objects.

We define the constraint relationship among variable values as value constraints, and the relationship among object method as method constraints. A typical constraint can be represented as $A <- f(B, C)$. For a value constraint, it can be interpreted as A's value is always equal to the value of B and C under the relation of f. If the value of B or C changes, the value of A is set to a new value. For a method constraint, it is interpreted as if method B and C occurs satisfying the condition of f, the A occurs. For example, the relation f is a OR relation, the above relationship means that either method B or C is applied, method A is applied.

3.2 Application Interface to Constraint System

The constraint system is added into Shastra as a service and it is attempted to reduce the additional works for application as far as possible.

We extend the idea of active data model, which is used in Apogee to store, derive and update data values on the basis of functional rules. The active data in our system is the active object in some sense. It not only can connect data values, but also can connect object method. And it supports the generation of message to invocate object method, as well as deriving object data values.

There are import and export interfaces between an application and a constraint system. When an object is created, application program uses its export interface to export the value items and method items, which can effect to or be effected by outside, to constraint systems. Relationships between items can be specified statically or added dynamically. Once there is a changes to those exported items, application will report the changes to the constraint system. The Constraint system uses import interface to import items and relationships specifications. It will maintain these relationships under indicated requirement. And it uses its export interface to export the derived value or method invocation to the application.

There are two main advantages. One is the constraint system can represent and maintain the relationships among heterogeneous objects. Another is it is possible for a constraint system to select different solving methods to satisfy different requirements.

3.3 Interface to Constraint Services

One of the critical problems in the implementation of our service based model is how a set of application objects can "connect" with the appropriate service.

Unfortunately, the system cannot know what type of relationship a set of objects will has. But application programmer do know the type of relations a set of objects will be. So programmer can declare the type of the relationships. The constraint relationships in Shastra are represented constraint graphs. When objects are created and the relationships are specified, the constraint graph is created based on the requirement of constraints indicated by the programmer. And the appropriate constraint solving method will be used to propagation the changes through the constraint graph. When a constraint is created, added or removed, or an object is created, added or removed, the system will reconstruct the constraint graph based on the type of the constraints.

According to our model, the constraint maintenance system has been implemented in two ways. One is as a set of library functions which can be linked into an application program and run by an application process. Another way is to implement it as a set of object servers; a different set of objects with different relationships can dynamically connect with different servers to request maintenance service.

In order to provide a flexible and transparent "connection" to different types of constraint maintenance servers, we extend the concept of an Object Request Broker (ORB) [17]. In our implementation, a broker works as a generic constraint maintenance service to all constraint objects, communicating with a set of constraint maintenance servers to do the actual work. According to the different constraint specifications, the broker routes the request to the appropriate constraint maintenance server. Figure 4 shows the conceptual paradigm.

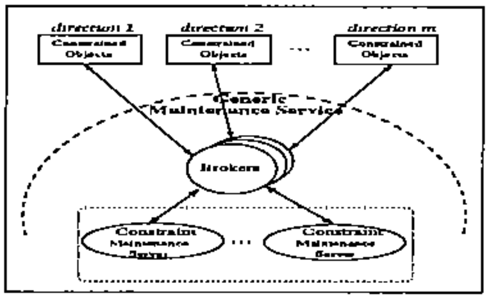


FIGURE 4: Brokered constraint maintenance system

3.4 Functions of Shastra Constraint System

Based on the Shastra application structure model [3], an application consists of a set of "Tools" and a "Core" which implements actual functionality. A tool consists of an "Interface" and a "Context" which is characterized by a "State" (see Figure 5). "Context" can be local or remote, and "State" can be private or shared. User actions generate "Events" to modify the "State" using functionality defined in the "Core". Collaborative setting in Shastra is based on a session model; that is, a session is a unit of collaborative activity. The session context is the setting of connected shared contexts in multiple tools and provides collaboration awareness to all users in the session (see Figure 5).

The Shastra constraint system is not used to construct its graphic systems, instead, it is mainly used to support requirements of its structure model and multi-user applications. That is:

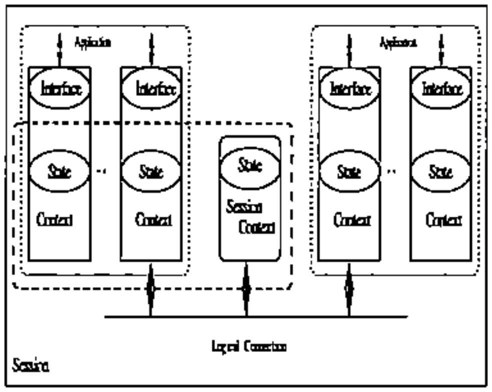


FIGURE 5: Session model based collaborative setting.

- It tightly connects interface with context state, so that the interface always depicts the current state.
- It connects multiple views of the same shared contexts.
- It connects multiple distributed instances of the shared context.
- It is used to connect multiple distributed view objects whenever necessary.
- It can specify and maintain the relationships among domain objects and task objects.
- It is used to handle multiple accesses to the shared objects.

Some of these functions are shown in Figure 6. The connector is just a set of constraints.

4 Constraint Maintenance Services

In our system, several constraint maintenance services are designed with different solution methods and various complexities, so that, for different types of relationships, we can use different types of constraints and link with different constraint maintenance services.

Currently, only one-way constraints are used in which a relationship can be uniquely represented by a single method, and all these solving methods are based on local propagation techniques. Conceptually, constraint relationships are represented as a graph (a directed

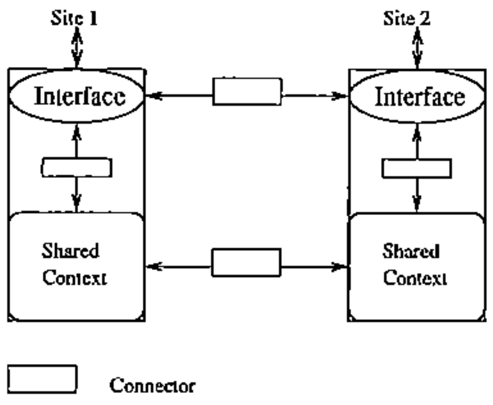


FIGURE 6: Constraint connector inside/between distributed tools of an application in Shastra

graph in one-way constraints). A node represents an object or a constraint, an edge represents that there is a relation between the object and the constraint. Figure 7 shows a simple constraint graph.

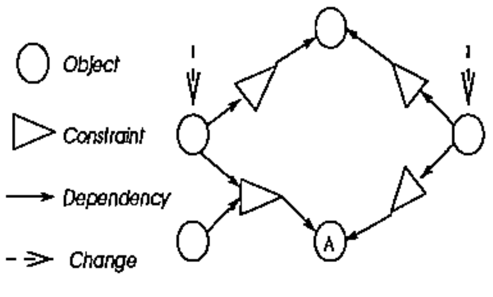


FIGURE 7: An one-way constraint graph with multiple changing sources

The types of constraint services are currently classified by two factors. One is the properties of the constraint graph: a "tree", a DAG (Directed Acyclic Graph), or an arbitrary directed graph with cycles. The second factor is the number of concurrent changes that are allowed.

Existing constraint systems have provided many efficient ways to solve constraints in a simple "tree" graph case, with multiple definitions, and sometimes in the presence of cycles as long as only one change occurs at a time [22, 12, 20].

Here, we present a new service for maintenance of constraint relationships under the occurrence of multiple and concurrent changes at a time. We believe this problem is a feature unique to multi-user applications, and deserves special consideration.

4.1 Problems

In multi-user applications, especially in a collaborative design applications, multiple users are expected to work in the same working space as far as possible. In this case, users access and manipulate shared objects from their own interface. If we specify the set of underlying objects with a constraint graph, then multiple changes are applied to different nodes in the graph simultaneously. Figure 7 shows such a constraint graph. These changes may propagate from different directions to a node, (e.g., node A in Figure 7), which is defined by multiple constraint methods, and cause a conflict or an inconsistent result in this node. Default turn-taking or serialization of actions is not satisfactory in the context of human factors [11]. We cannot take just one change and ignore the others either, because that will lead to an inconsistent result. Figure 8 shows an example of these cases: user A and user B try to paint a graph with four nodes w, x, y, z, and each node's color is constrained by the colors of two adjacent nodes, e.g., *y.color* $\leftarrow$ *x.color* and *y.color* $\leftarrow$ *z.color*. Now, user A and user B simultaneously begin to paint the graph with two different colors (e.g., red and green) from nodes x and z, respectively. User A paints the graph in the order of x, y, z, and w, and user B paints the graph in the order z, w, x, and y,

If we use a simple turn-taking method, then the color of this graph will first change to one user's color and then to the other user's color. This result is not the expected one in a collaborative application. If we simply take one change, whether the first or the last, and

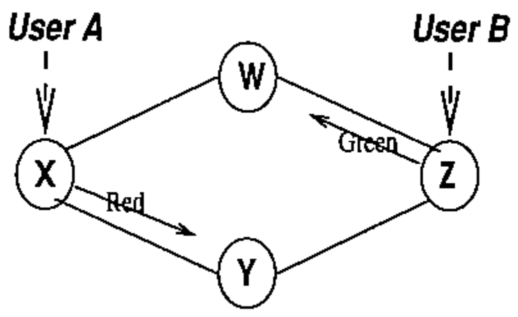


FIGURE 8: Two users are painting a graph simultaneously

ignore the other one, then if we allow some constraints to go unsatisfied, we will get the graph with partly red and partly green, otherwise the painting process will go on forever.

4.2 A Constraint Solving Method

We aim at designing a constraint solving method such that constraint relationships can be maintained under the situation of multiple change sources. In stead of using a simple locking mechanism, we integrate the concurrent control techniques in distributed systems.

Here we designed an algorithm to support concurrent modification. This algorithm allows multiple users simultaneously manipulate the same set of objects without explicit synchronization with each other or centralized control. In this method, if A and B, for example, change the node x and y in the graph at the same time, the final result should be the same as AB if there is no conflict, or A (reject the change of B), or B (reject the change of A).

The constraint solving process consists of three phases: generate a solution graph, order the constraints in the graph, and evaluate the constraints in the ordered list.

Once there is a change applied to a node, a solution graph (DAG) with this node as a source is generated according to the propagation of the change through the constraint graph. The consistency will be a problem only when the DAGs generated from each source have common nodes and if they exist at the same time. We add one more pass to mark nodes so that we can ensure that there is only one solution DAG if there is any conflict. In this case, the earlier action will have a higher priority.

Each node has a mark field with a pair (timestamp, color). Initially, each node is unmarked. A time stamp is assigned to each change and propagated with each change. From each changing source, each reachable node will be marked with (timestamp, Green) in the first pass and marked with (timestamp, Red) in the second pass. All nodes with the same mark (timestamp, Red) will be in the final solution graph.

Supposing a change arrives at time ts, there may be some nodes already marked with a timestamp tp, the algorithm marks nodes by DFS as followings:

- First Pass: For each reachable node do:

    1. If it is unmarked, then marks it with (ts, Green).

2. If it is marked with (ts, Green), then ignores it (it is a cycle, simply ignore it).
3. If it is marked with (tp, Green/Red), and tp<ts, then this change is rejected, stop (reject the later changes).
4. If it is marked with (tp, Green), and tp>ts, then re-marks the node with (ts, Green) (earlier action has a higher priority). Put tp into a record list.
5. If it is marked with (tp, Red), and tp>ts, then if tp is in the record list, re-marks the node with (ts, Green), and continue; otherwise, this change is rejected, stop.

- Second Pass: For each reachable node do:

  1. If it is marked with (ts, Green), then marks it with (ts, Red).
  2. If it is marked with (tp, Green/Red), and tp<ts, then this change is rejected, stop.
  3. Otherwise, error is reported, stop.

If these two passes are successful, a solution graph is generated and states of nodes can be ensured of no conflict or inconsistency. Otherwise, the user with the rejected change will be informed, so that the user in the collaborative activity is aware of the concurrent access.

After we get a solution DAG, we order the nodes in the DAG with a topological sorting algorithm [7]. Then the constraints in the ordered list are evaluated one by one.

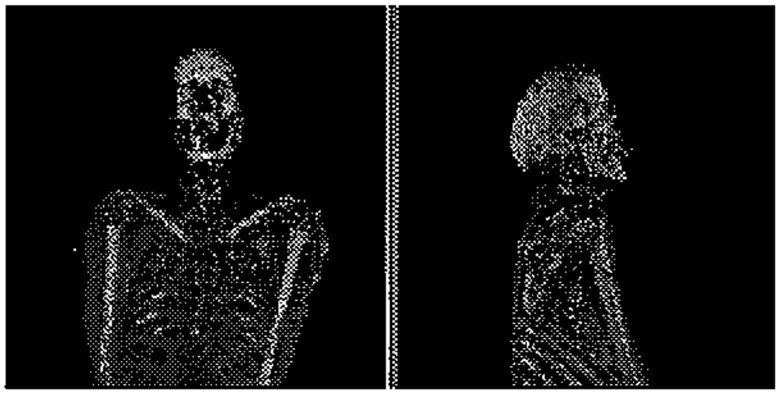
A more complicated service can be defined by allowing a solution DAG with cycles, or allowing nodes in the solution DAG to be defined by multiple methods. In this case, the above algorithm combines with previous common techniques, such as "once around the loop" or "constraint priority", to break cycles or choose one constraint definition.

5 Applications

We have developed several collaborative applications in the Shastra environment to test our design.

5.1 Collaborative Visualization

Collaborative VAIDAK is a medical imaging toolkit for collaborative visualization. One scenario is that two users simultaneously visualize the same CT data with different views while sharing output in a master-slave mode, that is, each user creates a master window (view) on his own site to control the visualization, and a slave window (view) is created on the other user's site to show what he is doing. Figure 9(left) shows two views in one user's site, the left is a master view which can be manipulated by this user, the right is a slave view to indicate the other user's result. Constraints in this example connect underlying data to image view, as well as slave views to master views, as shown in Figure 9(right). Both users can manipulate shared data through master views, if there is conflict between two different actions, one of them has to be rejected. When the shared data is changed, for example, some CT slices are added, removed, or edited, this change will be reflected to both master views as well as slave view sides by data-view constraints. If the user change some view parameters, like colors, or camera, in master view, then the changes will be "propagated" to slave views through view-view constraints. And both users can change his view controls independently, the slave view will be updated as well.



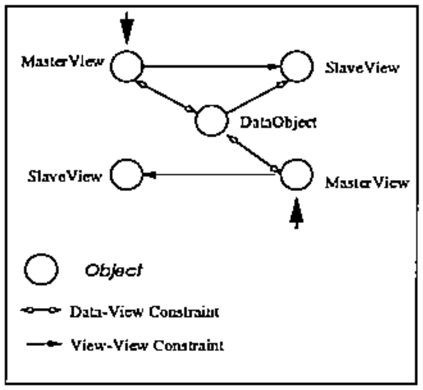


FIGURE 9: Collaborative Visualization in master-slave mode. Left is Master/Slave windows in one user's interface. Right is a constraint graph of data-view and view-view constraints.

5.2 Collaborative Modeling

Collaborative ISMA (interactive surface modeling and analysis) is a toolkit to interactively design and visualize geometric objects. This toolkit permits a group of users to collaboratively edit and smooth out a model by fitting certain continuous patches. Once the object is partitioned, the algebraic continuity requirement imposes constraints at the boundary of a partition. Users of adjacent parts must agree on the parameter setting for the boundary vertices and edges so that continuity requirements are not violated.

Figure 10 shows a scenario in which two users collaboratively smooth an object by partitioning the object into two parts: each user can set up control parameters and edit vertices, edges, and faces internal to his/her own part simultaneously. Constraints connect the shared data and views, and tightly link two distributed views to achieve WYSIWIS behavior. Each user can see the independent action of another user. When a user tries to edit an edge along the partition boundary, an information dialog box will pop up in the user's site to inform the user that this edge is subjected to an access constraint and that he/she needs to wait for agreements from the others. Request dialog boxes will pop up in the other users' sites, which allows the other users to indicate their agreement or disagreement (see Figure 11). Only when the other users agree to modify the boundary, can the modification or edit be applied. Once a boundary edge is updated, the adjacent patches in both parts

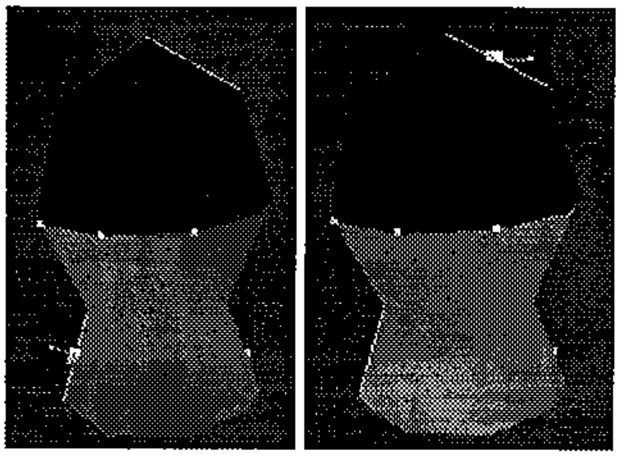


FIGURE 10: Two users collaboratively smooth an object, each user can modify a partition simultaneously

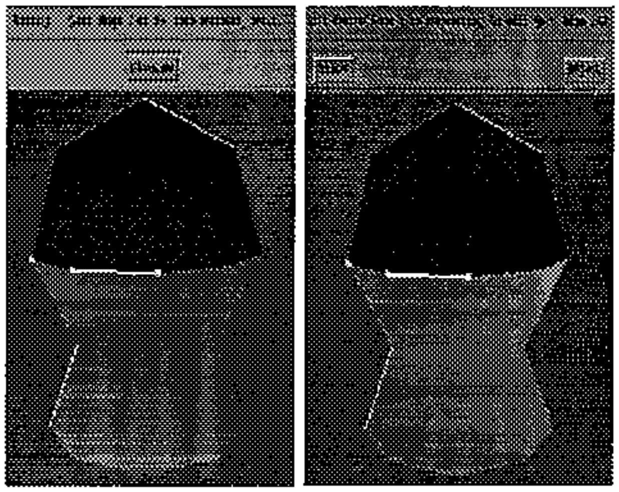


FIGURE 11: When a user tries to modify the boundary edge, access constraints will cause information/request dialog boxes to pop up in both users' sites

need to be recomputed in order to maintain the continuity. Figure 12 shows a constraint graph in this case.

5.3 Collaboration between Modeling And Analysis

Shastra hip prosthesis design is an example of collaboration between modeling and analysis. Figure 13(left) shows a scenario where three users are collaboratively work together. One user is using SHILP, a solid modeling toolkit, to model a femur. One user is using

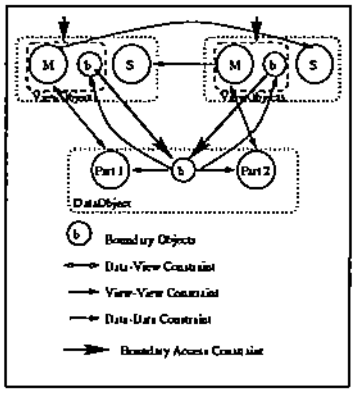


FIGURE 12: A constraint graph of a collaborative smoothing activity with an object partition.

SHILP to model a implant. The third user is using BHAUTIK, a physical analysis toolkit, to generate an external mesh of a femur for analysis. These three users can do modeling and analysis independently under some constraints, that is, the implant and femur model will be constrained by boundary requirements, and mesh generation function is constrained to the domain of the femur. Figure 13(right) shows a constraint graph between data objects and tasks (it does not include constraint relationships between views). When the geometric model of the implant changes, this change will propagate to the femur through data constraints ana cause the change of femur model, then the change in femur will trigger the mesh generation function through task constraints. The constraints between the femur and the implant will ensure that user A and B cannot modify the geometric models of these two objects simultaneously.

6 Conclusion

This paper presented our major considerations and preliminary experiences in designing a generic constraint system for multi-user collaborative systems. It is unique in the following ways:

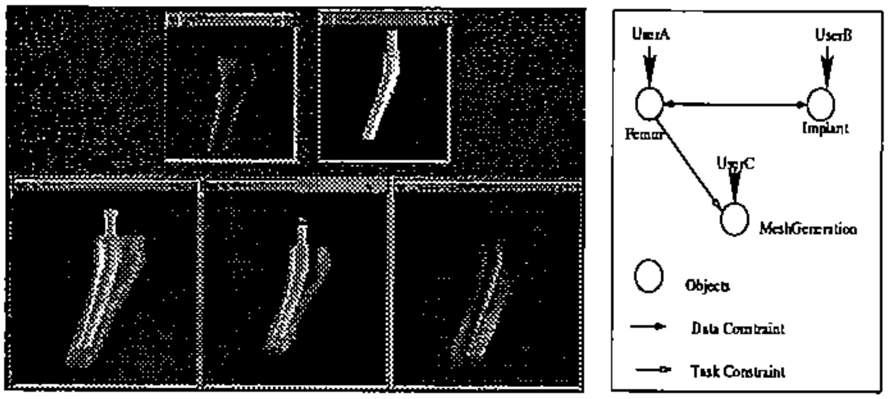


FIGURE 13: Collaborative modeling and analysis: User A is modeling a femur, user B is modeling a implant (left above), and user C is analyzing and visualizing (left below). Right is a constraint graph shown the relationships between data objects and tasks.

- It maintains the relationship among objects, including object states (values) and object behaviors (methods).
- The constraint system is built in object based approach, it can represent the relationships among heterogeneous objects.
- The constraint maintenance is defined as object services, so that applications can make different requirements. And this make it possible to implement distributed or concurrent management.
- We integrate the traditional concurrent control techniques into constraint solving, so that we can support concurrent modification.
- This model make our system very extensible.

Our experiences indicate that the Shastra environment with the constraint system greatly simplifies the development of multi-user collaborative applications.

Shastra, enhanced with the constraint maintenance system, is still under development; a formal and convenient interface for the system is being designed.

7 Acknowledgments

We thank Vinod Anupam, Steve Cutchin and Richard Kennell for all their assistance as fellow Shastra substrate team members. We also thank Susan Evans and Dan Schikore for developing the basic ISMA and VAIDAK toolkits in Shastra.

References

[1] V. Anupam and C. Bajaj. Collaborative Multimedia Scientific Design in SHASTRA. In *Proc. of the First ACM International Conference on Multimedia, ACM MULTIMEDIA 93*, pages 447–456. ACM Press, 1993.

[2] V. Anupam and C. Bajaj. SHASTRA - An Architecture for Development of Collaborative Applications. In *Proc. of the Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 155–166. IEEE Computer Society Press, 1993.

[3] Vinod Anupam. *Collaborative Multimedia Environments for Problem Solving.* PhD thesis, Purdue University, W. Lafayette, IN, August 1994.

[4] Paul S. Barth. An Object-Oriented Approach to graphical Interfaces. *ACM Transactions on Graphics*, 5(2), April 1986.

[5] A. H. Borning. The Programming Language Aspects of ThingLab, A Constraint-Oriented Simulation Laboratory. *ACM Transactions on Programming Languages and Systems*, 3(10), October 1981.

[6] Alan Borning and Robert Duisberg. Constraint-Based Tools for Building User Interface. *ACM Transactions on Graphics*, 5(4), October 1986.

[7] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, Canbridge, Massachusetts, 1990.

[8] Bjorn Freeman-Benson. A Module Mechanism for Constraints in Smalltalk. In *Proceedings of the 1989 Conference on Object-Oriented Programming: Systems, Languages and Applications*, October 1989.

[9] Bjorn N. Freeman-Benson, John Maloney, and Alan Borning. An Incremental Constraint Solver. *Communications of the ACM*, 33(1), January 1990.

[10] J. Gosling. *Algebraic Constraints.* PhD thesis, Pittsburgh, PA, 1983.

[11] Saul Greenberg and David Marwood. Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface. In *Proceedings of the CSCW'94 Conference on Computer-Support Cooperative Work*, pages 207–217, 1994.

[12] Ralph D. Hill. The Rendezvous Constraint Maintenance System. In *Proceedings of the Sixth Annual Symposium on User Interface Software and Technology*, 1993.

[13] Ralph D. Hill, Tom Brinck, Steven L. Rohall, John F. Patterson, and Wayne Wilner. The Rendezvous Architecture and Language for Constructing Multiuser Applications. *ACM Transactions on Computer-Human Interaction*, 1(2), June 1994.

[14] John Maloney, Alan Borning, and Bjorn Freeman-Benson. Constraint Technology for User-Interface Construction in ThingLab II. In *Proceedings of the 1989 Conference on Object-Oriented Programming: Systems, Languages and Applications*, October 1989.

[15] Brad A. Myers, Dario A. Giuse, Roger B. Dannenberg, Brad Vander Zanden, David S. Kosbie, Edward Pervin, Andrew Mickish, and Philippe Marchal. Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces. *IEEE Computer*, 23(11), November 1990.

[16] G. Nelson. Juno: A Constraint-Based Graphics System. In *(Proceedings of SIGGRAPH 85)*, pages 235–243, 1985.

[17] OMG. The Common Object Request Broker: Architecture and Specification. Technical report, OMG, verson 1.1, December 1991.

[18] John F. Patterson, Ralph D. Hill, and Steven L. Rohall. Rendezvous: An Architecture for Synchronous Multi-User Applications. In *Proceedings of the CSCW'90 Conference on Computer-Support Cooperative Work*, pages 317 – 328, 1990.

[19] Michael Sannella and Alan Borning. Multi-Garnet: Integrating Multi-way Constraints with Garnet. Technical report, University of Washington, 1992.

[20] Michael Sannella, John Maloney, Bjorn Freeman-Benson, and Alan Borning. Multi-way versus One-way Constraints in User Interfaces: Experience with the DeltaBlue Algorithm. *Software — Practice & Experience*, 23(5), May 1993.

[21] Ivan Sutherland. Sketchpad: A Man-Machine Graphical Communication System. In *Proceedings of the AFIPS Spring Joint Computer Conference*, pages 329–246, 1963.

[22] Pedro A. Szekely and Brad A. Myers. A User Interface Toolkit Based on Graphical Objects and Constraints. In *Proceedings of the 1988 Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 36–45, 1988.

[23] Brad Vander Zanden, Brad A. Myers, and Pedro Szekely. The Importance of Pointer Variables in Constraint Models. In *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology*, November 1991.