

# Collaborative Applications of Zoomable User Interface

Huahai Yang

CREW, School of Information, the University of Information

## 1 INTRODUCTION

Recent advancements in computer and communication technologies have revolutionized how computers are used. One of the application domains that are gaining increasing usage and visibility is *computer-supported cooperative work* or CSCW<sup>1</sup>. One of the goals of CSCW research is to allow geographically dispersed users to effectively collaborate on common goals using networked computers. The computer systems designed to support this are commonly known as *groupware* or *collaborative systems* [Ellis et al., 1991]. This type of systems usually provides an interface to a shared environment. Comparing to traditional multi-user systems such as distributed file systems and distributed databases, the focus of collaborative systems is to support human requirements in a group work scenario [Mandviwalla and Olfman, 1994]. In addition to system requirements such as data consistency, robustness, and scalability, collaborative systems deal with awareness, sharing paradigm, interactivity, and other human-centric requirements.

The entire shared work environment in a collaborative system usually contains enormous information that requires simultaneous access by a group of users. The amount of information that needs to be handled by user interface of collaborative systems is often larger than that of their counterpart single-user systems due to multiple information sources. This imposes severe challenges to the collaborative system developers. UARC (Upper Atmospheric Research Collaboratory) is one of the system development efforts that faces this kind of challenges [Olson et al., 1998]. Consider a group of space scientists try to work together, without traveling to a common place, on shared data gathered from various constantly running instruments located in different places. To support collaborative work like this, some reliable and scalable ways of supplying real-time data over heterogeneous network are required, and some responsive and flexible mechanisms of supporting real-time sharing and communication are essential. This work is an effort to explore the design space of the real-time collaborative systems.

With a large amount of data, finding effective ways to organize data into some tractable visual representations is a challenge for interface developers. Observation in UARC project shows that the users are demanding on the amount of information that is visually accessible from the user interface. However, the available visual presentation resource, the screen, has only limited size. This is known as small-screen problem. For instance, the space scientists like to bring up many graphs for comparison and analysis of data. In order to make these graphs simultaneously visible, some scientists have to work with several monitors.

In UARC system, based on room metaphor, the current solution to the small-screen problem is to organize graphs into categories, so they can be accessed via a nested menu

---

<sup>1</sup> See CSCW'98: <http://www.acm.org/sigchi/cscw98/>

system [Lee et al., 1996]. Each graph has its own window. When the number of graphs becomes very large, coping with the large nested menu and window system becomes a major usability problem due to excessive screen clutter and window overlap. This suggests that traditional window/icon/menu/pointer (WIMP) interface may not be a scalable solution. Exploring alternative user interface paradigm becomes an important research area. This project follows this research agenda and explores the design space of using alternative user interface paradigm in a real-time collaborative system.

There are not many ready-to-use alternatives to WIMP interface diagram. For more than a decade there have been efforts to devise alternative solutions that are sufficient to cope with a very large information world (see, for example, [Leung and Apperley, 1994]). The exploration of virtual 3D worlds is one alternative (for example, [Osborn and Agogino]; [Sutcliffe and Patel, 1996]). Although the 3D interface has rich representational power and great intuitiveness, it still follows traditional metaphor approach, in which the developers may devote scarce system resource to mimic physical worlds, therefore, under-utilize the power of the computer system. Meanwhile, there are many efforts that use 2D techniques to address the large information space problem (for example, [Furnas, 1986; Mackinlay, 1991; Robertson and Mackinlay, 1993; Bederson and Hollan, 1994;]). Central to most of these techniques is a notion of multi-scale viewing, whose interaction properties have been characterized by Furnas and Bederson [1995] in *space-scale diagram*. In interfaces employing this class of techniques, the information objects can be displayed at many different magnifications, or scales. By moving around and changing scale, the users can get integrated context and content of the large information space. Zooming technique is often used to change object scale over time, while distortion techniques like fisheye view [Furnas, 1986] are used to change object scale in a single view. Interfaces that use zooming as a primary interaction means are often called *Zoomable User Interface (ZUI)*. In this paper, an approach of using ZUI in collaborative systems will be described.

Specifically, the ZUI system discussed in this paper is a popular implementation called Pad++<sup>2</sup> [Bederson et al., 1995]. The system is designed as a widget for Tcl/Tk script language [Ousterhout, 1994], which provides a set of simple yet powerful programming abstractions for creating zooming-based applications. Although it was originally intended to be used as a general-purpose substrate for exploring visualizations of graphical data, Pad++ has now been used in a wild variety of situations, ranging from presentation to Web browsing. The Pad++ system consists of four types of software components: windows, views, surfaces, and objects. Surfaces represent the data space that objects reside on, where objects are graphical entities that have a visible representation and often support human interaction. Views represent a way to look at a particular part of the surface. Windows is the top-level operating system window that associated with a view and a surface. In addition, one special kind of object called a portal can have a view onto any surface.

Pad++ is considered by its creator as well suited to a collaborative work environment, since it places the user at a floating location in an information geography [Bederson et al., 1996]. Multi-user share one Pad surface is a very natural extension of Pad diagram. The detailed benefits of Pad++ collaborative application will be discussed in the next section. However, until now, there is neither publication that explicitly discusses the application

---

<sup>2</sup> Available from <http://www.cs.umd.edu/hcil/pad++>

of ZUI as the primary user interface of a collaborative system, nor discussion on ZUI-specific collaborative infrastructure. However, there has been ongoing work in this area. Jonathan Meyer, one of the implementers of Pad++, is currently developing a low-level network protocol for Pad++ at Media Research Laboratory of New York University<sup>3</sup>. The details of the implementation are not available yet. Gutwin and Greenberg [1998a] used Pad++ in a collaborative concept map authoring tool to demonstrate some of the collaborative awareness techniques. The purpose of Pad++'s being used here is mainly to ease the implementation of various awareness techniques, and no ZUI-specific collaborative features were discussed. As we will discuss at section 2.2, Pad++ is equipped with a rich set of graphic primitives that would facilitate the implementation of some of the collaborative awareness techniques. Another related work is called KidPad, which is a single display, multiply input device Pad++ application [Druin et al. 1997]. It runs on a single workstation and allows several users work on a single Pad surface at the same time, and each user has their own input device that is connected to the same machine. This study experiments a distributed version of Pad++ that works over the Internet and hopefully informs on-going work on Pad++ collaboration.

The structure of user interface and its relationship with the underlying share data is an important aspect of collaborative system architectures [Ellis, 1994]. When Pad++ is used in a collaborative environment, due to its unique interaction paradigm, some unique advantages and disadvantages will emerge. One of the goals of this paper is to identify these properties and to exploit the benefits while compensating the shortcomings.

Using Groupkit<sup>4</sup>, a groupware toolkit [Roseman and Greenberg, 1996], I implemented a collaborative layer for Pad++, which I have called GroupPad++. Based on this infrastructure, a sample drawing application included within Pad++ distribution called PadDraw has been transferred into a collaborative drawing program. This paper discussed the design principles of collaborative application of ZUI, using concrete example from GroupPad++ and the drawing program. This works was originally motivated by UARC project. To accomplish part of similar functionality of UARC system, a simple data dissemination mechanism has also been implemented within this infrastructure.

The rest of the paper is organized as follows. Section 2 details the design of a collaborative system using ZUI. The system architecture, data dissemination, sharing paradigm, concurrency and awareness features will be discussed in the context of previous related work. This paper also discussed the advantages and shortcomings of ZUI application in these areas. Section 3 describes the implementation of GroupPad++, and its limitations. Finally, the directions for future work are suggested.

## 2 DESIGN OF COLLABORATIVE APPLICATION OF ZOOMABLE INTERFACE

### 2.1 System Architecture

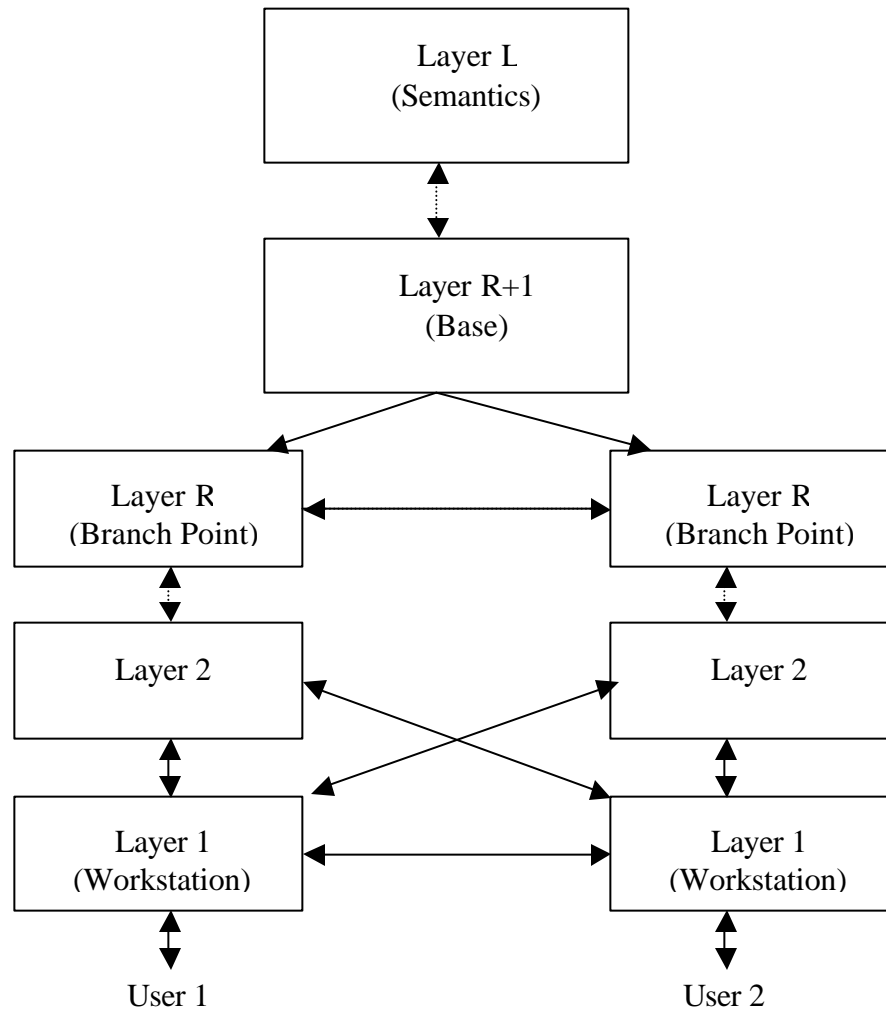
The architecture of a software system characterizes the components of the application, the function of each component, and the interaction among these components. It is an important issue in the design of the system since it influences the performance, ease of

---

<sup>3</sup> <http://www.mrl.nyu.edu/>

<sup>4</sup> Available from <http://www.cpsc.ualgary.ca/grouplab/groupkit/>

modification, and other properties desired by users and developers [Shaw and Garlan,



**Figure 0 Generic architecture of collaborative systems**

1996]. Most of the collaborative applications follow an editing-based collaboration model [Dewan et al., 1994]. In this model, the feedback users received in response to commands entered by themselves is determined by single-user semantics, while the feedback users receive in response to commands entered by others is determined by collaborative semantics. As shown in Figure 0, the generic architecture of this type of collaborative applications assumes that a user's input/output is processed by a hierarchy of layers. Some layers are *shared* while others are *replicated*. Events allow different layers work together.

Within this model, Dewan [1997] further presented a framework that defined the design space of the collaboration architectures. Collaborative systems differ in the way of how they deal with these important issues: 1) Single-user architecture: how application semantics are implemented and what kinds of user-interface layers are assumed? 2) Collaboration awareness: which layer implemented the collaborative semantics? 3) Replication: which components of application are replicated; 4) Concurrency: which of these components can execute concurrently and how to solve the conflicts; 5)

Distribution: which of these component can execute on separate hosts. To make Pad++ collaboration-aware, one must make all of these design decisions.

### 2.1.1 Collaboration Awareness

There are generally two approaches to transform a single-user application into a multi-user one. One is the sharing of legacy single-user applications, wherein multiple users simultaneously interact with a single-user application. This is called *collaboration transparency* [Begole et al., 1997]. The idea is to keep the same set of layers existed in a single-user application, and add collaborative functions to one or more of these layers by modifying them. Examples of this method include Suite [Dewan, 1992], Shared X [Garfinkel et al., 1994] and JAMM [Begole et al., 1997]. Microsoft NetMeeting also belongs to this category. The benefit of this approach is that when the runtime environment supports collaboration transparency, an application developer need not write any code to make an application collaborative. From user's point of view, they need not to learn a new interface of collaborative application. However, this approach requires modifications to the layers that are made collaborative. Sometimes it is difficult to modify a layer that implements both the single-user and the collaborative semantics. Finally, it is not viable if the source code of the layer to be changed is not available.

Another approach is to put a *pseudo-layer* between two existing single-user layers to support collaboration. To each of these two layers, the pseudo-layer provides an extension of the interface the other one provided. Depending on the nature of interface between the two existing layers, the addition of the pseudo-layer may require recompiling and/or relinking of the existing layers. However, this approach does not require changes to the existing layers. Moreover, the added layer will always be valid even if the implementations of other layers have been changed, as long as these implementations provide the same set of interfaces. Examples of this approach include XTV [Abdel-Wahab and Feit, 1991], COLA [Trevor et al., 1994] and DistView [Prakash and Shim, 1994]. Since Pad++ is a third-party program, and it is subject to implementation changes out of our control, this pseudo-layer approach is reasonable for this study. The GroupPad++ implemented here is built upon Pad++ and uses its APIs. Collaborative applications of Pad++ are in turn built upon GroupPad++. This way, GroupPad++ serves as a middle layer between Pad++ infrastructure and its applications. Due to this nature, it needs not to be changed when new implementation of Pad++ comes along, as long as the Pad++ APIs keep consistency. Similarly, applications built upon GroupPad++ need not to be changed when the GroupPad++ implementation has been modified.

Moreover, this approach can provide a more flexible coupling schema, because what have been shared are high level abstractions of visual representations, not visual representation themselves. Users can apply different transformations to the shared abstractions, resulting to different views of the shared objects. This is particular important for Pad++ since one of its key features is to have different views for a single object. In general, by adding collaboration support at a higher level, concurrency control, coupling, access control and other collaboration functions can operate on a more meaningful granularity, therefore it is easier to implement application semantics.

### 2.1.2 Replication

The replication architectural dimension determines the base and branch points in the generic architecture as shown in Figure 0. All layers below a base are replicated. Two extreme approaches to replication are called *centralized* and *replicated* approaches, the former has no replicated layers while the later has no base layer. Groupkit is an example of systems that support full replication. MMM [Bier, 1992] is an example of centralized approach. Most of other collaborative systems support the hybrid architecture, but the degree of replication varies dramatically. Examples are Weasel [Graham and Urnes, 1992] and Rendezvous [Hill et al., 1994]. A more replicated approach has some important advantages that are critical to collaborative user interface. Since every site keep a local copy of the shared states, replicated approach achieves better responsiveness and is less vulnerable to network bottleneck. Moreover, replicated approach allows more divergence in the users' states, resulting to more flexible views of the shared data. However, it is a challenge to keep these distributed replicas consistent. On the other hand, it is easier to achieve consistency in a centralized approach, since only one copy of shared state exists. But this also suggests a severe efficiency problem if the operations on the shared state are expensive, since every operation has to be queued on central site. An expensive operation can delay other operations' execution.

In ZUI systems, users zoom in, zoom out and pan around the large information space very quickly. If the required transformation must be executed at a centralized server and then sent back to the action originated site, it is very possible that the latency makes these ZUI-specific actions useless. Because of these considerations, GroupPad++ adopted the replicated approach. In GroupPad++, each site keeps a copy of the shared Pad, and users interact with the shared Pad locally. Handling of interactions and screen updates occurs in parallel at each replica. Users' activities are processed immediately. However, to fully exploit the unique feature of ZUI, a dynamic partial replicated approach might be better than fully replicated approach. Since the information space in Pad++ is theoretically infinite large, it is not efficient to keep all of the shared states in every sites. Moreover, users usually focus only on part of the information space at a time, it is unnecessary to always keep a full copy of the large collaborative data space. Aura, a concept originally developed in DIVE [Fahlen et al., 1993], a 3D virtual environment system, and later extended by MASSIVE [Geenhalgh and Benford, 1995], a collaborative virtual environment, might be instructive here. In these systems, each object has an aura in which it can interact. Interaction between two objects only becomes possible when their auras collide or overlap. This leads to better performance than a full interaction model. By analogy, every participant of Pad++ collaboration has an aura, only the objects that reside within the participant's aura should be copied to the site where the participant takes part in the collaboration. The participant's aura changes as her viewpoint changes, the underlying data structures should also be updated correspondingly. This way, unnecessary data transportation is saved.

Additional benefit of this approach is that the collaborative awareness now is tightly coupled with the sharing schema. Information of the two modules can be inferred from each other, resulting to more efficient implementation in both parts. This approach has more awareness and access control implication. In addition to the notion of focused attention of participants, there is a concept of visibility of participants' aura to other participants. Participant can either be granted control over the accessibility of objects

within her aura, or be granted control over the visibility of her aura. It is a new type of access control policy (see [Dewan and Shen, 1998] for a review of access control). The drawback of this dynamic partial replicated approach is increased complexity. In this study, this feature is not implemented due to the limitations of shared data structure.

### 2.1.3 Monitoring

So far, the general structure of the shared states of GroupPad++ has been covered, but the unique feature of the Pad++ collaboration lies in its graphic representation of these shared data structures. To exploit the unique set of visual components of the Pad++ system, such as surfaces, views and portals, a Pad++ specific collaborative workspace and its interaction model are devised. Following common user interface model originated from Smalltalk, a Model-View-Controller like (see [Hill et al., 1992]) approach is used to deal with the mapping between the shared data structure and the graphic view that the user interact with. Figure 1 shows the overall architecture of the GroupPad++. In this structure, two types of objects are defined. The view objects are native Pad++ objects. The model objects are underlying data structures of Pad++ objects that are implemented in GroupPad++ layer. A model object holds attributes of a Pad++ object, such as its ID number, position, or scale. According to the replicated sharing schema described above, the model objects are shared among participated sites and are eventually kept consistency among these sites.

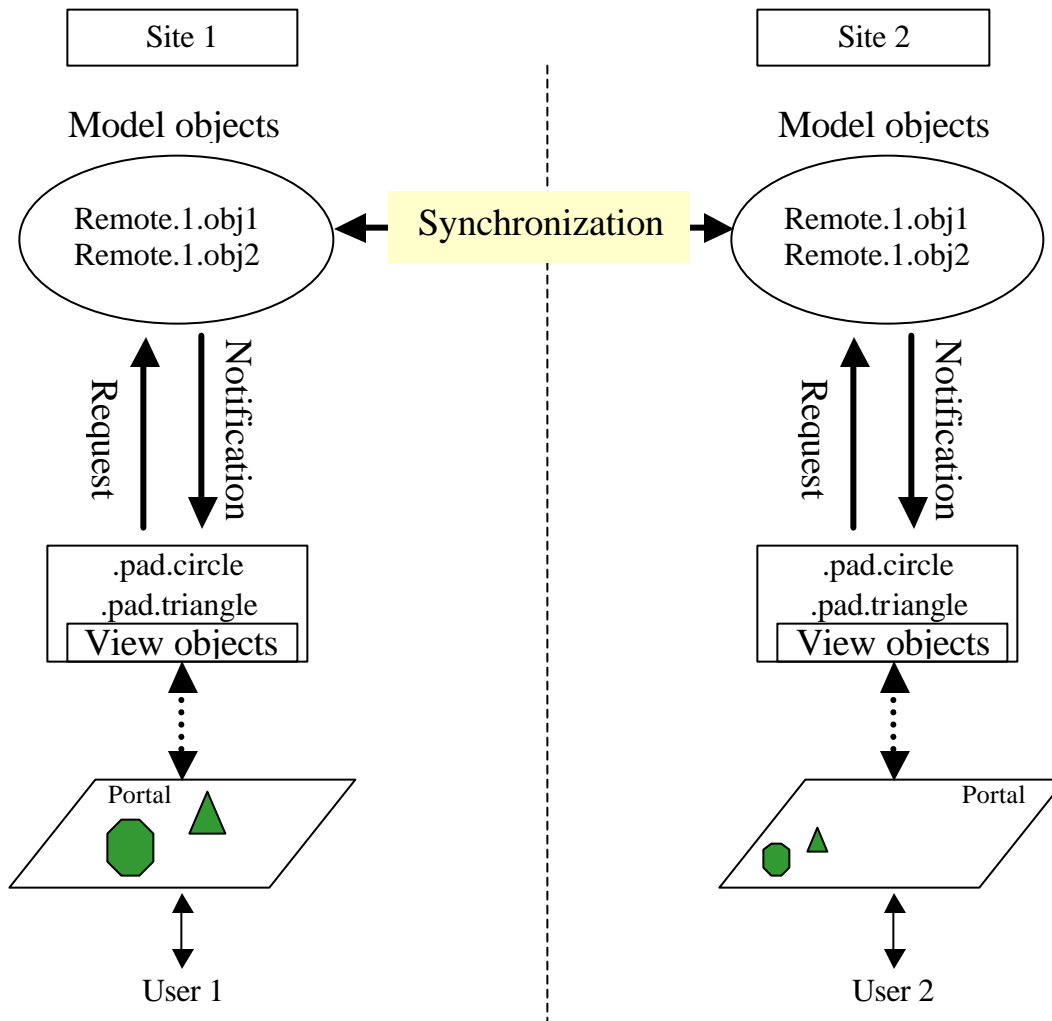


Figure 1 The architecture of GroupPad++

As can be seen, the centerpiece of this collaborative system is the model object being manipulated by users through the user interface. Depending on the semantics of the application, the model objects and their structures can be either very simple or very complex and containing a hierarchy of nested objects. Mapping between the view objects and the model objects are achieved through notification and request mechanisms. This is one instance of the *monitoring* issues that are pervasive in collaborative systems [Rodham and Olsen, 1997]. System modules such as an interface module or a concurrency module need to monitor the shared data structures for changes made by other system components so that they can respond to those changes. A monitoring system should be able to detect all changes in a monitored data structure and report these changes to interested watcher modules. When data structure becomes complicated, monitoring becomes a challenge, since it become hard to determine which part of the data structure has changed and which watcher module should be called upon to handle the change. On the other hand, the watcher module needs not response to irrelevant changes of the data structure. For example, the view module should not be called upon when non-visual attributes of an object have changed. As we will discuss in the GroupPad++ implementation section, the monitoring problem is largely determined by the constraint of the underlying data structures and its notification mechanism.

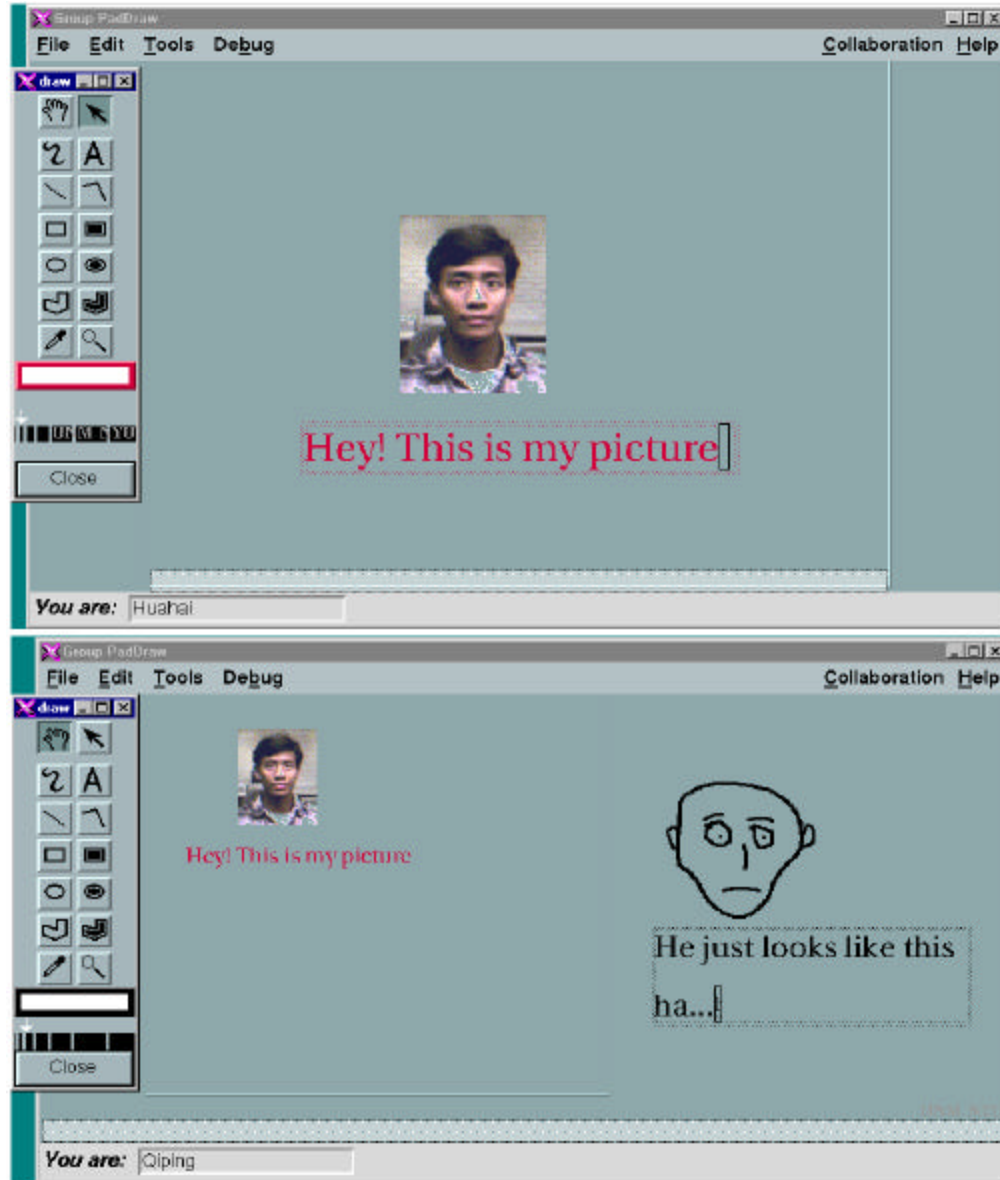
#### 2.1.4 Coupling and Access Control

A collaborative system creates multiple user interfaces allowing multiple users to interact with the program. The program must couple its user interfaces in some ways, i.e. allow these interfaces share some states, so that the users can collaborate with each other [Dewan and Choudhary, 1995]. For example, WYSIWIS is a coupling policy that shares entire interface among users; DistView toolkit supports a window-level sharing; and Rendezvous system support a flexible interface sharing schema. A good coupling model is one that is flexible, easy for user to specify, and easy for developer to implement.

Pad++ interface has some powerful graphic primitives that are suitable for the design of a flexible coupling schema. The basic idea of a GroupPad++ coupling schema is that users look onto some shared Pad surfaces through portals that are located on their private Pad surfaces. The shared Pad surfaces can be any Pad surface created by any participants and running on any machines, as long as the appropriate access control requirements are satisfied.

As shown in Figure 2, both of the users, “Huahai” and “Qiping”, have their own Pad surface, on which a portal looks onto a shared Pad surface. “Huahai” loads an image on to the shared Pad surface and it is sent to all participants of this session. Both users have control over their own views of the shared Pad surface. In figure 2, the same objects display different magnifications on the two users’ interface, but the scale properties of the objects are identical in these two sites, the different view is achieved by looking through different portals.

In principle, the users can not only view the same objects in different scales, but also in different graphic presentations by applying various lenses on the portals. Since object manipulation events can pass through these portals, the users can interact with shared Pad++ objects collaboratively. Moreover, the users can use their private Pad surfaces the same way as using the single-user Pad++ interfaces. In Figure 2, user “Qiping” drew a cartoon on her private surface, which is not currently visible for user “Huahai”.



**Figure 1 Group PadDraw Interface**

However, if she likes to share her private Pad surface with “Huahai”, she can set its access level to “Public” and tell him, so that he can create another portal on his Pad surface to look onto her private surface. GroupPad++ supports a simple access control schema, in which the Pad surface is identified as either “Public” or “Private”. A private Pad surface can only be accessed by its creator. Notice that Pad surface can be created but not be actually drawn on the GUI. In Figure 2, the shared Pad surface is hidden, even though the objects resided on the surface are visible through portals.

### 2.1.5 Concurrency

Replicated approach brings about the complexity of *concurrency control*. Achieving consistency among replicas is a major technical challenge for the collaborative system

development. Since the users interact with their own local copies of shared data simultaneously and then propagate the actions to remote sites, the final result of users' actions tends to be inconsistent among different sites.

Three independent major inconsistency problems were identified by [Sun et al., 1998]. They are divergence, causality violation, and intention violation problems. Divergence problem is resulted from different orders the operations arrive and be executed at different sites. This problem can be solved by serialization, which ensures operations are executed in the same total order at all sites. However, this total order can be out of the natural cause-effect order of these operations and leads to unexpected results, this is the causality violation problem. Based on timestamps methods, this problem can be solved by selectively delaying the executions of some operations to enforce correct causal order. The most challenging problem is the intention violation problem, which concerns with the effect difference between the executions of an operation at operation-generating time and at actually execution time.

In addition to the methods mentioned above, there are generally two approaches to ensure consistency among replicas [Prakash, 1997]. One approach requires the acquisition of a lock of the object before changing an object's shared data structure (*Pessimistic* approach). Another approach is to commit the operation locally first, then undo or transform operations that have been incorrectly done when appropriate operation arrives from remote sites (*Optimistic* approach). The optimistic approach is considered to be a better approach in text editing task domain because of its responsiveness benefit and its mathematical rigorous. For example, one recent optimistic approach demonstrated in REDUCE system can solve all of these three problems [Sun et al., 1997]. However, in graphic editing task domain, together with optimistic methods, various pessimistic methods were used. For example, Emsenble [Newman-Wolfe et al., 1992] uses implicitly placed write locks for concurrency control, with locks placed when an object is selected and removed when it is deselected; DistView uses a token-based locking algorithm. As Greenberg [1994] argued, there is no general applicable concurrency control methods that is effective in all situations.

Considering most of objects in Pad++ are graphic items, to achieve the efficiency of implementation, a hybrid approach is devised to achieve consistency in GroupPad++. This method applies different concurrency control methods for different operations. While creating an object, a serialized method is used because the users usually can tolerance some delay at this moment, and the causal order of object additions does not influence final results too much. When modifying an object, an optimistic locking method is used. Modification on the view object is committed first locally to give a quick feedback to users, a lock on the model object is then requested by asking all sites for the token of the object. The token of an object can only be held by one of the participated sites. If the site that currently holds the token need not to lock the object any more, the token will be transferred to the requesting site. Once the site gets the token, change on the model object will be committed and propagated to all of the participated sites, resulting to identical model objects in all sites. However, if the token is not released by current-holding site, the locally committed changes on view object may be undone to keep consistency with corresponding model object, resulting to unexpected disturbing for the users. Since the users rarely work on the same object at the same time due to social norms, this situation is rarely happened. The lock request is serialized to

preserve causal order. And the token is cached locally, if the site has already hold the token, any lock request to this object will be skipped for quick response. For deleting operation, a pessimistic locking method is used to prevent data lose. However, this approach is sometimes still problematic, especially after the users join or leave the session. Improvement on this hybrid concurrency approach should be pursued further.

## 2.2 Collaborative Awareness

Many kinds of collaborative work require participants maintain a sense of other's activities because they are working on the same shared workspace. The awareness of others in a collaborative setting is called *collaborative awareness*. It involves support users' knowledge of who is present, where are they working, and what they are doing. Collaborative systems with awareness support have better usability than those without [Gutwin and Greenberg, 1998b].

Several techniques have been developed to address the collaborative awareness requirement. For example, in Figure 3, one participant's view port is shown as a rectangle labeled with user name in another participant's workspace on GroupPad++, this give a sense of where others are working on. To achieve both users' focus on their own work and their awareness for others' work, Gutwin and Greenberg [1998a] implemented four awareness techniques in Pad++ and made comparisons among them. These techniques are radar view, fisheye view, dragmag view, and two-level view. Pad++ makes the implementation of these techniques much easier than the traditional interface. All of these techniques involve separated view of the workspace, this can be achieved using separated portals. For example, radar overviews shrink the entire workspace to fit within a single window, a portal with a view onto the entire workspace will achieve the effect. To put different view of the workspace together, layering is a useful technique, which is supported natively in Pad++. Layering is also useful for implementation of telepointer, a collaborative awareness feature used in a wild variety of collaborative

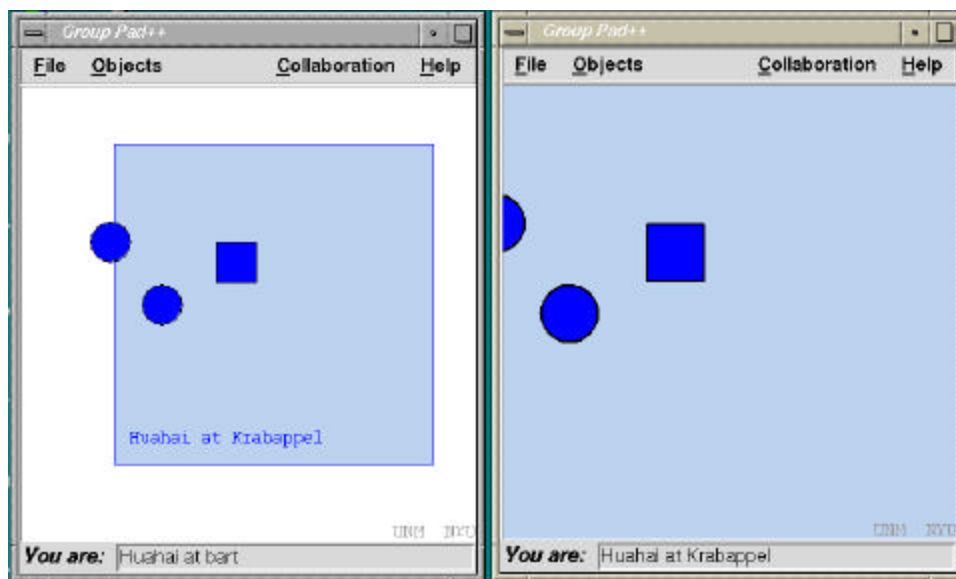


Figure 1 Viewpoint of others

systems. On the other hand, the use of collaborative awareness techniques in ZUI may help alleviate some of the ZUI-specific navigation problems such as “desert fog” problem [Jul and Furnas, 1998], since they provides additional cues for navigation.

### 3 GROUP PAD++ IMPLEMENTATION

#### 3.1 *Groupkit and its limitations*

Building collaborative system on top of some groupware toolkits is much easier than building everything from scratch. In this study, Groupkit is used in the development of GroupPad++. There are four reasons. First, Groupkit follows a replicated architecture, as we discussed before, this architecture is well suitable for the collaborative application of Pad++. Second, Groupkit has a set of programming abstracts built on top of Tcl/Tk language, which is the same language as of Pad++ programming interface. Using these programming primitives, one can build new groupware widgets or applications relatively easier. Third, Groupkit support separation of view objects and model objects by providing an underlying data structure and a set of notification mechanisms. Finally, along with several session managers supplied in the distribution, it also supports flexible mechanisms of building customized session manager. These can be utilized to help establish a collaborative system rather quickly.

The shared data structure in the Groupkit environment is called *environment*. Basically, environments are hierarchically structured dictionaries. Information in the environments is accessed via a key, which specifies where within the environment information is stored. Keys use a period as the hierarchy delimiter. For example, as shown in “Model objects” of Figure 1, “Remote.1.obj1” specify the location of a shared object’s information within the data structure, which has at least three depth levels. In GroupPad++, the identification number of an object on a shared Pad surface is accessible as “Remote.Pad1.obj1.padid”.

As can be seen, as object attributes become complex, the data structure becomes deeper and larger. A powerful monitoring mechanism is needed to keep other system modules aware of changes in the data structure. Groupkit supports monitoring by generating events in response to environments changes. The event type, (add, change, or delete) and the environment key are returned whenever events are generated. However, other information about the changes is missing. For example, it is useful to know from which site a change is originated. Currently, the programmers have to encode this additional information into environment themselves.

Another limitation is concerned with rather fixed granularity of the environment. For example, a set of points’ coordinates specifies the shape of a polygon. Developers have to decide which environment representation is better, either “poly.coordinates” or “poly.coords.point1”. The former packs the coordinates of all points in one environment, while the latter creates a separated environment for each point. The problem with the former is the unnecessary information transfer over the network if only a few points change occur. The later has divergent causal-order problem, because environment events are invoked separately while changes are made at the same time. A desired environment is one that has fine-grained notification mechanism, and allowing flexible monitoring.

In general, this restricted hierarchy data structure does not has sufficient flexibility to support large-scale collaborations that may be critical for ZUI collaboration. The

promise of ZUI collaboration is its potential to support large shared information space. As we discussed before, support for dynamic partial replication schema is critical for the scalability of the ZUI collaboration. This can not be easily achieved in current state of collaborative infrastructure.

There is also a general problem with groupware toolkits in that they are usually forced to impose fixed models of programming abstractions. This follows directly from the traditional structuring techniques in software development, which hide implementation details behind the abstraction, out of reach of application developer. The value of this approach is that it relieves developers from low lever details, so that they can focus on those areas specific to application. The cost is that the range of applications that are suitable for development within the toolkit is restricted, since the implementation decisions within the toolkit constrain the kinds of interactions that can be supported in applications. In the future, the ZUI specific collaborative layer should be built to best exploit its potential.

### 3.2 GroupPad++

As described in section 2.1.1, pseudo-layer approach of collaborative awareness needs recompile or relinking of existing system. Currently, GroupPad++ needs recompile of Tcl/Tk shell, because Pad++ and GroupKit use different Tcl/Tk shell programs. The former uses 'padwish', the later uses 'gwish'. An integrated shell enables Groupkit commands in Pad++ environment. When a ZUI specific collaborative layer is developed, dynamic linking method can be used.

A set of commands that used to turn a single user Pad++ application into a multi-user version is developed. These commands should be placed before appropriate Pad++ commands. Use some searches and replaces, an existing single user Pad++ application written in Tcl can be turn to a multi-user version. Below is a list of these commands.

#### **gp\_init**

Initiate GroupPad++.

#### **gp\_newobj <Pad++ create item command>**

Create new Pad++ object.

e.g. gp\_newobj .pad create line 10 10 10 10 -pen red

#### **gp\_chgobj <Pad++ item configure command>**

Change attribute of Pad++ object.

e.g. gp\_chgobj .pad coords line 10 10 110 310

#### **gp\_delobj <Pad++ delete item command>**

Delete Pad++ object

PadDraw, a Pad++ drawing program, has been turned into collaborative version using these commands. It proves to be a workable solution. A small number of PadDraw functions are not preserved because they do not fall into the command categories as shown above. However, the suitable commands for them can be developed under GroupPad++ as well.

## 4 FUTURE WORK

A lot of room left for improvements on the collaborative application of ZUI. Many have been mentioned before, such as a ZUI specific shared data structure, a more elaborated access control model, a better concurrency control algorithm, exploring novel

collaborative awareness techniques in ZUI and user specified interface coupling schema. One of the major shortcomings of this paper is that empirical study has not been carried out. Using ZUI in collaboration generate new possibilities such as very large-scale collaboration. However, new problems must exist that should be identified and solved through empirical studies. This should be one of the focuses of the future work. And the potentials of this approach can not be realized without being put into a real-world situation. In order to prove the design concepts described in this paper, a UARC like prototype may be developed later.

## 5 REFERENCE

- Abdel-Wahab, H., & Feit, M. (1991). XTV: A framework for sharing X window clients in remote synchronous collaboration, *Proceedings of IEEE TriComm* (pp. 159-167).
- Bederson, B. B., & Hollan, J. D. (1994). Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics, *Proceedings of the ACM Symposium on User Interface Software and Technology* (pp. 17-26).
- Bederson, B. B., & Hollan, J. D. (1995). Pad++: A Zooming Graphical Interface System, *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems* (Vol. 2, pp. 23-24).
- Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D., & Furnas, G. (1996). Advances in the Pad++ Zoomable Graphics Widget. *Journal of Visual Languages and Computing*, 7, 3-31.
- Begole, J., Strubble, C. A., Shaffer, C. A., & Smith, R. B. (1997). Transparent Sharing of Java Applets: A Replicated Approach, *Proceedings of the ACM Symposium on User Interface Software and Technology* (pp. 55-64).
- Bier, E. A., Freeman, S., & Pier, K. (1992). MMM: The Multi-Device Multi-User Multi-Editor, *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems* (pp. 645-646).
- Dewan, P. (1997). Architectures for Collaborative Applications. In M. Beaudouin-Lafon (Ed.), *Collaborative Systems*: John Wiley & Sons Ltd.
- Dewan, P., & Choudhary, R. (1992). A High-Level and Flexible Framework for Implementing Multiuser User Interfaces. *ACM Transactions on Information Systems*, 10(4), 345-380.
- Dewan, P., & Choudhary, R. (1995). Coupling the User Interfaces of a Multiuser Program. *ACM Transactions on Computer-Human Interaction*, 2(1), 1-39.
- Dewan, P., Choudhary, R., & Shen, H. (1994). An Editing-Based Characterization of the Design Space of Collaborative Applications. *Journal of Organizational Computing*, 4(3), 219-239.
- Dewan, P., & Shen, H. (1998). Controlling Access in Multiuser Interfaces. *ACM Transactions on Computer-Human Interaction*, 5(1), 34-62.
- Druin, A., Stewart, J., Proft, D., Bederson, B., & Hollan, J. (1997). KidPad: A Design Collaboration Between Children, Technologists, and Educators, *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems* (Vol. 1, pp. 463-470).
- Ellis, C. A., Gibbs, S. J., & Rein, G. L. (1991). Groupware: Some Issues and Experiences. , 9-28.

- Ellis, C. S., & Wainer, J. (1994). A Conceptual Model of Groupware, *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work* (pp. 79-88).
- Fahlen, L. E., Stahl, O., Brown, C. G., & Carlsson, C. (1993). A Space Based Model for User Interaction in Shared Synthetic Environments, *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems* (pp. 43-48).
- Furnas, G. W. (1986). Generalized Fisheye Views, *Proceedings of ACM CHI'86 Conference on Human Factors in Computing Systems* (pp. 16-23).
- Furnas, G. W., & Bederson, B. B. (1995). Space-Scale Diagrams: Understanding Multiscale Interfaces, *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems* (Vol. 1, pp. 234-241).
- Garfinkel, D., Welti, B., & Yip, T. (1994). Shared X: A tool for real-time collaboration. *Hewlett-Packard Journal*(April), 23-26.
- Graham, T. C. N., & Urnes, T. (1992). Relational Views as a Model for Automatic Distributed Implementation of Multi-User Applications, *Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work* (pp. 59-66).
- Greenberg, S., & Marwood, D. (1994). Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface, *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work* (pp. 207-217).
- Greenhalgh, C., & Benford, S. (1995). MASSIVE: A Collaborative Virtual Environment for Teleconferencing. *ACM Transactions on Computer-Human Interaction*, 2(3), 239-261.
- Gutwin, C., & Greenberg, S. (1998a). *Design for Individuals, Design for Groups: Tradeoffs Between Power and Workspace Awareness*. Paper presented at the Proceedings of ACM CSCW'98 Conference on Computer Supported Collaborative Work.
- Gutwin, C., & Greenberg, S. (1998b). Effects of Awareness Support on Groupware Usability, *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems* (Vol. 1, pp. 511-518).
- Hill, R. D. (1992). The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications, *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems* (pp. 335-342).
- Hill, R. D., Brinck, T., Rohall, S. L., Patterson, J. F., & Wilner, W. (1994). The Rendezvous Architecture and Language for Constructing Multiuser Applications. *ACM Transactions on Computer-Human Interaction*, 1(2), 81-125.
- Jul, S., & Furnas, G. (1998). Critical Zones in Desert Fog: Aids to Multiscale Navigation, *Proceedings of the ACM Symposium on User Interface Software and Technology* (pp. 97-106).
- Lee, J. H., Prakash, A., Jaeger, T., & Wu, G. (1996). Supporting Multi-User, Multi-Applet Workspaces in CBE, *Proceedings of ACM CSCW'96 Conference on Computer-Supported Cooperative Work* (pp. 344-353).

- Leung, Y. K., & Apperley, M. D. (1994). A Review and Taxonomy of Distortion-Oriented Presentation Techniques. *ACM Transactions on Computer-Human Interaction*, 1(2), 126-160.
- Lieberman, H. (1994). Powers of Ten Thousand: Navigating in Large Information Spaces, *Proceedings of the ACM Symposium on User Interface Software and Technology* (pp. 15-16).
- Mackinlay, J. D., Robertson, G. G., & Card, S. K. (1991). The Perspective Wall: Detail and Context Smoothly Integrated, *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems* (pp. 173-179).
- Mandviwalla, M., & Olfman, L. (1994). What Do Groups Need? A Proposed Set of Generic Groupware Requirements. *ACM Transactions on Computer-Human Interaction*, 1(3), 245-268.
- Olson, G., Atkins, D., Clauer, R., Finholt, T., Jahanian, F., Killeen, T., Prakash, A., & Weymouth, T. (1998). The Upper Atmospheric Research Collaboratory (UARC). *interactions*, 3(May), 48-54.
- Osborn, J. R., & Agogino, A. M. (1992). An Interface for Interactive Spatial Reasoning and Visualization, *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems* (pp. 75-82).
- Ousterhout, J. K. (1994). *Tcl and Tk Toolkit*. Addison Wesley.
- Prakash, A. (1997). Group Editors. In M. Beaudouin-Lafon (Ed.), *Treads in CSCW*.
- Prakash, A., & Shim, H. S. (1994). DistView: Support for Building Efficient Collaborative Applications using Replicated Active Objects, *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work* (pp. 153-164).
- Robertson, G. G., & Mackinlay, J. D. (1993). The Document Lens, *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology* (pp. 101-108).
- Rodham, K. J., & Dan R. Olsen, J. (1997). Nanites: An Approach to Structure-Based Monitoring. *ACM Transactions on Computer-Human Interaction*, 4(2), 103-136.
- Roseman, M., & Greenberg, S. (1996). Building Real-Time Groupware with GroupKit, a Groupware Toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1), 66-106.
- Shaw, M., & Garlan, D. (1996). *Software Architecture: Perspective on an Emerging Discipline*. New Jersey: Prentice Hall.
- Sun, C., Jia, X., Zhang, Y., Yang, Y., & Chen, D. (1998). Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction*, 5(1), 63-108.
- Sun, C., Zhang, Y., Jia, X., & Yang, Y. (1997). A Generic Operation Transformation Scheme for Consistency Maintenance in Real-Time Cooperative Editing Systems, *GROUP'97: International Conference on Supporting Group Work* (pp. 425-434).

- Sutcliffe, A., & Patel, U. (1996). 3D or not 3D: Is it Nobler in the Mind?, *Proceedings of the HCI'96 Conference on People and Computers XI* (pp. 79-94).
- Trevor, J., Rodden, T., & Mariani, J. (1994). The Use of Adapters to Support Cooperative Sharing, *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work* (pp. 219-230).