IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITP4453

Application Sharing Technology: Sharing the Application or its GUI ?

Dirk Trossen
Nokia Research Center Boston, 5 Wayside Road, Burlington, MA 01803
Dirk.Trossen@nokia.com, Phone: +1 (617) 794 7041

ABSTRACT

Several toolkits have been developed in the past for the provision of audiovisual conferencing or collaborative working applications. Audiovisual information exchange can be seen as the minimal functionality of such tools. The exchange of commonly shared data, transfer of accompanying slides, or sharing a common whiteboard content is provided in many collaboration applications. Additionally, sharing an arbitrary local application in a conference is becoming more and more important, because this functionality is aiming at sharing a local, non-distribution-aware, application among all members, e.g., for demonstration purpose. This application sharing problem can be seen as the most challenging one in the development of conferencing tools. This paper compares two fundamental paradigms to solve the application sharing problem, namely sharing the application's state or sharing the GUI output. For that, the basics of both techniques are presented, and their advantages, disadvantages, and implementation problems are outlined. Furthermore, an architecture is proposed for the event sharing approach which allows the local execution of the application with a commonly shared state. The paper concludes with outlining the expected problems of this architecture when being implemented.

1 INTRODUCTION

Recent development of the Internet infrastructure and progress in protocol functionality led to an increasing usage of services for collaborative scenarios. In addition to transferring plain audiovisual information, using more sophisticated shared workspace facilities is crucial for many scenarios in the tele-learning and tele-education area. This shared workspace functionality reaches from transferring accompanying slides or working with a shared whiteboard to sharing arbitrary applications over the network.

The latter functionality is the most challenging one since most applications nowadays are still unaware of any distribution among networks. Thus, the distribution of the application's functionality has to be added to the application subsequently without changing the application's semantic. Hence, the effect has to be created at each remote site that the application is running locally and therefore can also be controlled by any remote user with a more or less immediate effect to the application.

1.1 Challenges to be Solved

Sharing a local application among distributed users involves the synchronized transfer of application-specific data among different, possibly heterogeneous, users. Several challenging issues have to be addressed for a proper solution being outlined in the following. Note that this list does not claim to be exhaustive:

- *Amount of transferred data*: The amount of data per packet to be transferred is part of the indicator for the generated network load.
- *Number of interception points*: each technique adds certain points to the local system to intercept the required information to be distributed among the session members. First, the information has to be extracted for building the appropriate packet to be sent. Second, the packet has to be transferred through the protocol stack degrading the overall system performance. Additionally, together with the amount of transferred data (see above), the resulting number of bytes to be sent over the network can be used as an indicator of the generated network load.
- *Heterogeneity*: sharing applications independent from the member's operating system is crucial for a wide applicability of the technique. This requires appropriate software at each site.
- *Latecomer's support*: joining the session later should be supported without leading to inconsistencies of the shared application's state.
- *Shared data problem*: using any kind of input data within the shared application should not lead to inconsistencies of the distributed copies of the application. For instance, no inconsistencies should occur when copying local input data into a shared spreadsheet.
- *Synchronization*: the shared instances of the application have to be synchronized to ensure consistency of the workspace among all users due to the different processing speed of the sites and the different delays of the transmission lines.

Two different paradigms can be distinguished for providing shared application services, namely *sharing the application's state* or *sharing the application's GUI output*. In this paper, both paradigms are presented on a functional level, which allows a comparison of the applicability of both approaches based on the challenging problems listed above. Furthermore, an architecture is presented applying the event sharing approach under specific assumptions. With this architecture, the application is executed locally on each host, while the state of the application is distributed by sharing and synchronizing the occurring events among all members. The paper concludes by outlining expected problems using this approach when being implemented in real systems.

The remainder of the paper is organized as follows. Section 2 presents and compares both sharing application paradigms and outlines the applicability of the presented techniques. An architecture is proposed in Section 3 for an application sharing system using the event sharing approach. In Section 4, related work in the area of application sharing is depicted, before concluding in Section 5.

2 SHARING GUI VS. SHARING STATE

As stated in the introduction, two paradigms are distinguished for realizing remote application sharing, namely sharing the application's GUI output or sharing the application's state.

In the following, both paradigms are introduced by depicting the main characteristics of the approaches. Furthermore, a comparison is presented based on the challenging issues presented in section 1.1. Finally, the applicability of both approaches is outlined.

2.1 Sharing GUI Technique

The first technique is to share the application's GUI output to the set of users. For feedback from the receivers, any input data like mouse or keyboard events is transferred back to the sender and fed into its local event loop for control. Figure 1 illustrates this technique. The server host runs the local application. *Rendering data* is transferred from the server to the receiver group using a specific protocol, e.g., a reliable multicast transport protocol. Obtaining the rendering data can be realized on different system levels. Interception of the GUI rendering on a high level, such as on *windows engine level*, results in a high number of interception points [12]. Each user interface routine, which results in a graphical output at server's side, has to be intercepted. This adds computational overhead to the server due to the additional transfer of the routine's input parameters to the receivers. However, the resulting data packets might be rather small depending on the number of parameters and the size of the parameter data.

Intercepting the GUI output on lower level, such as on *graphics engine level*, reduces the number of interception points. However, the amount of data to be transferred might be higher compared to the last approach, especially for high level routines like creating a window.

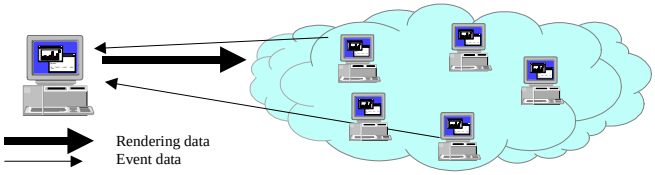
As it can be seen in figure 1, *event data* is sent back to the server to be fed into its local event loop for a remote control of the local application. Usually, transferring event data to the server is controlled by means of *floor control* [2], i.e., the appropriate host is selected based on a *social protocol* with an associated floor representing the right to control the shared application.

Latecomer's support is provided by invoking a full refresh of the local GUI resulting in a transfer of the entire GUI content to the receiver group. Furthermore, the shared GUI approach allows a heterogeneous receiver group assuming appropriate rendering engines on client's side. As shown above, the input event data is the only data to be synchronized with the local application, which is realized by means of floor control. Any additional data, like files or local device data, is held locally with the server's host. Hence, there is no *shared data* problem to be dealt with. However, the different processing speeds of the client rendering engines have to be considered for synchronization of the workspace. For that, *synchronization points* can be used which have to be acknowledged by each member.

2.2 Sharing Event Technique

The second technique to solve the shared application problem is the *sharing event* approach. The assumption being made is that if a set of identical applications is executed with the same start state and evolves using the same sequence of events, its timeline evolution is identical on each site. Hence, the basic approach of this technique can be outlined as follows:

Figure 1 : Sharing GUI Approach



- define the start state to be distributed among all group members
- start local copies of the application to be shared on each host
- distribute input events of the current controlling group member to evolve the current application's state

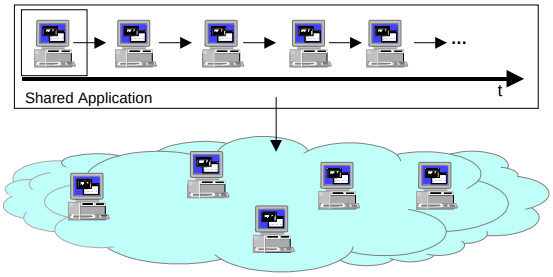
This technique is demonstrated in figure 2. In contrast to the shared GUI approach, there is no central server after starting the shared application. The initiator of the shared application session is merely used for defining the start state of the application. Any input data is transferred from the current floor holder to *all* group members. There is no central entity to which the input data is sent first to determine the new output.

Additionally, the *number of interception points* can be drastically reduced since only the main input event handling loop of the current floor holder has to be intercepted and the (relatively) small packets are to be distributed to the other participants. It can be seen that there is no GUI information to be shared. Hence, the local applications are executed at full speed as a local copy when not having any input data to be fed into the system. Thus, it is expected to perceive much lower overhead degradation caused by the sharing protocol.

However, it can easily be seen that the *homogeneity* of the environment is crucial due to the requirement having a local application instance. Hence, heterogeneous environments are not supported. *Latecomers* can be supported by maintaining a history of the event evolution which is distributed to the recently joined participant.

A special treatment is necessary handling the *shared data problem*. If there is any local data to be fed into the local instance, this case has to be handled to ensure consistency among the different copies. The architecture, presented in section 3, introduces a specific component to deal with this problem.

Figure 2 : Shared Event Technique



Similar to the shared GUI technique, the different processor speeds have to be considered for synchronization. Inserting *synchronization events* is a common solution for this problem.

2.3 Comparison of the Techniques

This section shows a comparison of the application sharing paradigms depicted in the previous sections with respect to the challenging problems of section 1.1. For the shared GUI approach, two interception techniques are distinguished, namely on high, i.e., windows engine level, and on low, i.e., graphics driver level. Table 1 shows the comparison of the three approaches.

It can be seen from this table that the main advantage of both shared GUI variants is the heterogeneity (assuming appropriate rendering engines on client's side) and the absence of the shared data problem. However, the additional load at the server for intercepting and transferring the data to the clients is the major weakness of this approach due to the large amount of data to be transferred for a consistent view of the shared application.

	High Level Shared GUI	Low Level Shared GUI	Shared Event
Application Location	Server Host	Server Host	Each Host
No. of Interception Points	High	Small	Small
Amount of Transferred Data	Small	Large	Small
Additional Load	High	High	Small
Heterogeneity	Yes	Yes	No
Latecomers Support	Full Refresh	Full Refresh	History
Shared Data Problem	No	No	Restricts Applicability

Table 1 : Comparison of Shared Application Techniques

The additional overhead for the shared event approach is much smaller due to the local application running on each host. Relatively small packets are exchanged among the participants resulting in a low end system and network load. Hence, the shared event technique is well suited for graphics intensive applications (like multimedia or 3D tools) without adding significant overhead to the system.

However, the lack of heterogeneity is an inherent problem due to the usage of a local copy of the application. Hence, this technique requires homogenous systems. Furthermore, the shared data problem restricts the applicability of the approach.

2.4 Applicability of the Techniques

It can be seen from the comparison in section 2.3 that each approach has its specific advantages and weaknesses. The first technique, either sharing the GUI on higher or lower level, is well suited for heterogeneous environments and when using input data which cannot be shared among the other participants.

However, the shared event approach has also specific advantages which makes this technique attractive for specific scenarios. Due to the local copy of the application, the additional load on each host is expected to be much lower which increases the responsiveness of the system and thus improves the user perception of the system. However, the problem of ensuring the consistency of each user's view when using shared data restricts the applicability of the approach either to not using shared input data or to use the technique in local environments where data sharing is feasible to some extent. Furthermore, this technique is not applicable in heterogeneous scenarios.

The following table shows typical scenarios for shared applications and the applicability of both paradigms in the these scenarios. It is worth mentioning that the list is only meant to outline sample scenarios. Thus, the list is neither exhaustive nor exclusive.

Table 2 : Scenario Examples and Paradigms Applicability

Scenario Description	Shared GUI	Shared Event
Multimedia presentation in a local environment	--	++
Programming environment in a lecture	-	++
Development environment in a closed user group	-	+
Spreadsheet in an heterogeneous Internet environment	++	-
Accompanying presentation in an Internet lecture	++	-

It can be seen that the shared event technique is not applicable to the last two scenarios due to the heterogeneous character of these situations, while the first three scenarios are fairly good examples where the shared event approach promises to provide a higher responsiveness of the system and therefore an improved user perception. Especially the multimedia presentation is hardly conceivable using the shared GUI approach due to the large amount of data to be transferred which is avoided by the local copy of the application when using the shared event technique. Furthermore, due to the local character of the scenarios, the shared data problem can be handled much easier.

It can be summarized that the shared event technique is better suited for local environments and high demands on the responsiveness of the shared application, while the shared GUI approach is to be preferred in heterogeneous environments and when having problems with data to be shared.

3 PROPOSED ARCHITECTURE

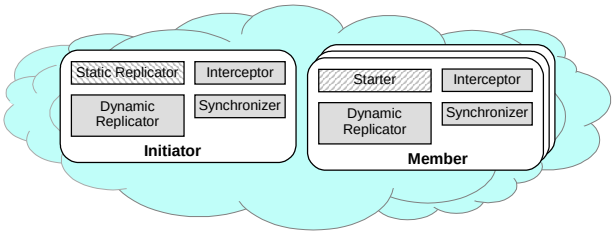
This section proposes a shared application architecture based on the shared event technique outlining the main components and depicting expected difficulties in their realization.

Figure 3 shows the proposed architecture and their components. It can be seen that the system is entirely distributed, i.e., there is no central entity for any control or synchronization purposes. The following components are realized within each host:

- *interceptor*: This component realizes the interception of events to be shared among all session members. For that, the component subscribes for the relevant events at the local operating system, generates appropriate messages to be distributed, and injects the intercepted messages to the local system. The intercepted events are forwarded to the synchronizer if a synchronization operation is necessary. This component is enabled when the appropriate host has become the current floor holder, i.e., the floor has been assigned to the participant.
- *synchronizer*: Synchronization of the different copies is crucial due to the different processing speeds at the local hosts. However, not all events are required to be synchronized assuming that certain operations are realized almost in parallel, e.g., all graphics-related functions like opening a menu or a dialog box. If the event invokes an application-based computational action, a special event is used as a synchronization point for all applications. Furthermore, memory swapping operations make synchronization necessary. These special events are sent to the session's synchronizer entities to be acknowledged after the specific action.
- *dynamic replicator*: After starting the session, input data like files might be given to the application. Usually, this information is defined at the beginning of the session and replicated at startup (see below). However, there might be some input data which is not known at the beginning, e.g., clipboard content containing graphical information of another local application. This information has to be replicated during runtime of the system, if possible, or the system has to generate a runtime warning that data is added to the system which is not available at the other sites. These tasks are realized by the dynamic replicator component.

In addition to these common components, there are initiator- and member-specific components. At the session initiator, the start state has to be determined. This is realized by the *static replicator*, which gathers all application-relevant information including program settings, data to share, and environmental infor-

Figure 3 : Event Sharing Architecture



mation such as screen resolution or system font settings. The entire set of start information is distributed to the session members to configure their local copies to be started.

At each session member, the *starter* component is responsible for setting a consistent start state of the local copy, starting the local application, and restoring the environment after closing the session. The latter point is crucial for not destroying any personal local information of the application.

For the support of latecomers, each distributed component stores its information in an *evolution history* which is delivered to the latecomer for correct setup of its local copy.

It is worth mentioning that the distribution of the information like start state and events is not within the scope of the architecture. However, it is assumed to use reliable multicast protocols for the transfer of such information to increase the effectiveness of the system.

The presented architecture is kept very general to be realized on specific systems like X Windows or Microsoft Windows. However, several system-specific problems like the description of an event and mapping onto system specifics, the realization of the event interception, or the definition of the start state have to be realized in a system-dependent manner. Especially the latter point is crucial for the applicability of the system in real-life scenarios due to the complex configuration and user setting capabilities of software and operating systems nowadays regarding user interface configuration, option settings, and the different storage facilities for these settings. However, the architecture presented in figure 3 depicts a framework for this realization.

4 RELATED WORK

Many toolkits and systems were proposed in the area of shared applications to enable cooperative working with arbitrary applications which were not originally designed for distribution.

Many of recently proposed systems are based on the *X-Windows* system [5]. This system comprises a central server being responsible for the application functionality. The output of the application can be redirected to be rendered at *X Windows clients*. Hence, this system provides a common base for distributing application functionality among the network. Extending this system to a multipoint scenario, as done in [1][8][9][10][12], enables very easily a shared application system for cooperative working. However, floor control facilities have to be added for coordinated control, which was done for instance in [1][9]. Thus, the X Windows system offers a natural base for application sharing since this platform was designed as a client-server-based rendering approach of application's GUI output.

Despite the wide deployment of X Windows systems, its applicability is mainly restricted to Unix systems. Although X Windows client software is also available for platforms like Microsoft's Windows and Apple's operating system, the problem remains to share for instance Windows software on other platforms. Hence, the heterogeneity problem is only partially solved using an extended X Windows system.

For this, the ITU proposed a shared application protocol for multipoint scenarios [6] which is intended to be platform-independent. Rendering and interception functionality is defined without taking any specific platform features into account. The main disadvantage of the ITU approach is not only its shared GUI approach and therefore the overhead on the server system, but also the usage of the ineffective transport system. For the latter, the T.120 recommendation for multipoint transfer is used, which is shown to be very inefficient even in real multicast environments [11].

In [4], a shared event approach was proposed which enables to run local applications with a shared event processing. In this proposal, the entire data workspace is replicated before starting the application copies. Hence, a dynamic sensing of the shared data problem is not supported. Synchronization among the different copies is ensured for every next incoming event leading to a significant overhead. Specific synchronization events are not used for overhead reduction. Moreover, the event mapping and distribution is realized in a central server. Hence, the proposal follows a distributed application, but a centralized control approach.

5 CONCLUSIONS AND FUTURE WORK

This paper presented and compared two paradigms for solving the *shared application problem*, namely the sharing GUI and the sharing event technique. For both approaches, the main features and characteristics were presented. This led to a comparison regarding challenging issues, such as the amount of transferred data, the number of interception points, support of heterogeneous environments, latecomers support, handling of the shared data problem, and synchronization due to different processing speeds.

The outcome of this comparison was that the shared event technique promises to add lower overhead to the system than the shared GUI approach due to its local copy approach. Hence, the responsiveness of the system is expected to be increased, which also improves the user perception of the shared application. The major drawbacks of the shared event technique are the lack of heterogeneity support and the reduced applicability due to the shared data problem. However, when considering local scenarios like school lectures, the shared data problem is much easier to avoid with an increased usability of the shared application due to the higher responsiveness of the system.

In addition to the comparison, an architecture was outlined for a shared application system based on the shared event paradigm. The core components were presented, and the main problems for the realization of this architecture were figured out.

For the future work, a prototype system of the architecture is planned to demonstrate the feasibility of the shared event technique in general. For that, several system-dependent problems, especially the definition of the start state, have to be addressed. Moreover, shortcomings in specific scenarios are to be investigated.

6 REFERENCES

[1] M. Altenhofen, J. Dittrich, R. Hammerschmidt, T. Käppner, C. Kruschel, A. Kückes, T. Steinig: *The BERKOM Multimedia Collaboration Service*, Proceedings of ACM Multimedia, 1993

[2] H.-P. Dommel, J.J. Garcia-Luna-Aceves: *Floor Control for Activity Coordination in Networked Multimedia Applications*, Proceedings of 2nd Asian-Pacific Conference on Communications, 1995

[3] P. F. Fitzgerald, N.Y. Rosson, L. Uljon: *Evaluating Alternative Display Sharing Systems Architectures*, Proceedings of TriComm, 1991

[4] M. C. Hao, J. S. Sventek: *Collaborative Design Using Your Favorite 3D Application*, Proceedings of IEEE Conference on Concurrent Engineering, 1996

[5] E. Israel, E. Fortune: *The X Window System Server*, Digital Press, 1992

[6] ITU-T: *Multipoint Application Sharing*, ITU-T Recommendation T.128, 1998

[7] ITU-T: *Data Protocols for Multimedia Conferencing*, ITU-T Recommendation T.120, 1998

[8] O. Jones: *Multi-User Application Software using Xt*, The X Resource Issue 3, pp. 55-75, 1992

[9] W. Minenko, J. Schweitzer: *An Advanced Application Sharing System for Synchronous Collaboration in Heterogeneous Environment*, SIGOIS Bulletin, vol.15 no.2, pp. 40-44, 1994

[10] Schmidt, J. Schweitzer, M. Weber: *A Framework for Synchronous Tele-Cooperation*, Proceedings of International Workshop on Advanced Communications and High Speed Networks, 1994

[11] D. Trossen, T. Helbig: *The ITU T.120 Standard Family as Basis for Conferencing Applications*, Proceedings of SPIE International Symposium Voice, Video, & Data Communications, 1997

[12] K. H. Wolf, K. Froitzheim, P. Schulthess: *Multimedia Application Sharing in a Heterogeneous Environment*, Proceedings of ACM Multimedia, 1995