Implementing the Shared Event Paradigm

Dirk Trossen
Nokia Research Center, USA

Erik Molenaar
Technical University of Aachen, Germany

INTRODUCTION

For collaboration among users, sharing audio-visual, textual, graphical, or even interface-related information is the essence of *computer-supported collaborative work* (CSCW). Since most applications that are being used in private and work life these days are merely usable on the computer on which they are executed, collaboratively working with a single application is the most challenging part of CSCW. This is not only true because these applications are unaware that they are executed in a distributed environment, but, in particular, because of the numerous possibilities of data to be shared among the distributed users. Thus, the distribution of the application's functionality over the network must be added transparently and, more important, subsequently without changing the application's semantic. 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. This problem is referred to as *application sharing* in the remainder of this article.

BACKGROUND

The realization of application sharing faces several challenges to be solved, as explained in more detail in Trossen (2001):

- *Amount of transferred data* is part of the indicator for the generated network load.
- Each technique adds certain *interception points* to the local system to gather required information to be distributed among the session members. The number of these points serves as an indicator of the generated network and processor load (Trossen, 2001).
- *Heterogeneity*: Sharing applications independent from each member's operating system is crucial for wide applicability of the technique.
- *Latecomer's support*: Joining the session later should be supported without leading to inconsistencies of the 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.
- *Synchronization*: The shared instances of the application have to be synchronized to ensure consistency of the workspace due to the different processing speed of the sites and the different delays of the transmission lines.

Two different paradigms can be distinguished to tackle the earlier mentioned challenges, namely, *Output Sharing* and *Event Sharing*. In Trossen (2001), a qualitative comparison and detailed presentation of both paradigms is available, including the different application scenarios for both paradigms.

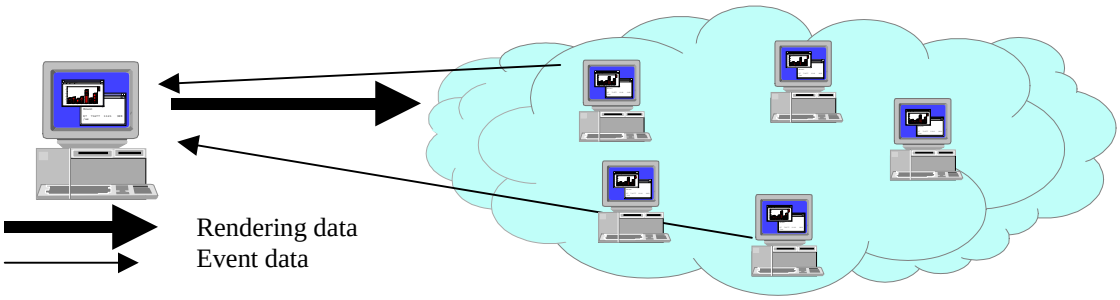
GUI Sharing Technique

The first technique is to share the server's application's output. 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, subject to the current floor control policy. Rendering data is transferred from the server to the receiver group, preferably using a reliable multicast protocol.

For that, as indicated 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 application. Usually, transferring event data to the server is controlled by means of *floor control* (Hao & Sventek, 1996), that is, the appropriate host is selected based on a *social protocol* (Dommel & Garcia-Luna-Aceves, 1995) with an associated floor, representing the right to control the 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,

Figure 1. GUI sharing approach



assuming appropriate rendering engines on the client's side. As shown earlier, 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 deal 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.

Event Sharing Technique

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 following statement is valid (Proof in Trossen, 2001):

- Theorem 1: A set of application instances that behaves deterministically can be held in a stable state if the starting state and all events can be captured.

In contrast to the shared GUI approach, there is no central server. The initiator of the session is merely used for defining the application's start state. Any input data is transferred from the current floor holder to *all* group members. There is no central entity to which the data is sent first to determine the new output. In this, *homogeneity* of the environment is crucial due to the requirement of having a local application instance.

Applicability of the Techniques

Each approach for tackling the application sharing problem has its specific advantages and weaknesses. The GUI sharing approach is well suited for heterogeneous environments and when using input data which cannot be shared among the other participants.

However, the event sharing 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's 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.

Table 1 shows typical scenarios for shared applications and the applicability of both paradigms in these scenarios. It is worth mentioning that the list is only meant to outline sample scenarios; thus, the list is neither exhaustive nor exclusive.

It can be seen that the event sharing 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 as the shared event approach promises to provide a higher responsiveness of the system and, therefore, an improved user's perception. Specifically, 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 event sharing technique is better suited for local environments and high demands on the responsiveness of the shared application, while