MMUSIC
Internet-Draft
Expires: March 27, 2005

H. Schulzrinne
J. Lennox
J. Nieh
R. Baratto
Columbia U.

September 26, 2004

Sharing and Remote Access to Applications
draft-schulzrinne-mmusic-sharing-00

Status of this Memo

Copyright Notice

Abstract

We describe requirements for accessing general graphical user interface (GUI) applications remotely, either by a single remote user or embedded into a multiparty conference.

Table of Contents

1. Introduction

While two-party and multi-party conferencing using standards-based protocols is now common and well-developed, protocols for sharing applications are largely proprietary or based on the aging T.120 suite of protocols. In this draft, we summarize some requirements that a protocol or set of protocols for application sharing should satisfy.

We note that there are large similarities between remote access to an application ("remote desktop") and by multiple users sharing an application within a collaboration setting such as a multimedia call or multiparty conference.

Remote access differs from the transmission of screen video pixels in the type of encoding needed. In particular, screen encoding may need to be lossless and typically operates on artificial rather than natural (photographic) video input. The video input is characterized by large areas of the screen that remain unchanged for long periods of time, while others change rapidly. (However, rendering the output of a modern computer-generated animation application such as video games blurs the distinction between traditional motion video output and screen sharing.)

Unlike earlier systems, such as T.120, we believe that application sharing should be integrated into the existing IETF session model, encompassing session descriptions using SDP or successors and the Session Initiation Protocol (SIP). Application sharing needs many of the same control functions as other multimedia sessions, such as address binding and session feature and media negotiation. We believe that use of the session model is also beneficial for the remote desktop case, as it allows to re-use many of the well-developed session components and easily supports hybrid models, such as the delivery of desktop audio to the remote user.

We distinguish between local and remote users. The local users employs normal operating system mechanisms to interact with the running application. Remote users interact via the delivery protocol whose requirements are outlined here.

The application sharing problem can be divided into four components: (1) setting up a session to the node running the application, (2) transporting user input events from the remote viewers such as conference participants to the application, (3) delivering screen output from the application to the participants, (4) moderating access to shared human interface devices such as pointing devices (e.g., mice, joystick, trackball) and text input (keyboard). We refer to components (3) and (4) as the "remoting protocol". It is

the focus of this document.

Session negotiation and description can be provided by existing session setup protocols; user input access can be moderated by a floor control protocol. Thus, these two components are beyond the scope of this document, although they are important for an acceptable overall user experience.

Applications are more than just windows; they are likely a stack of related windows which serve the same task and are usually associated with the same process on the server. Some applications impose special constraints on the user input, e.g., through modal dialogs and floating (always-on-top) windows.

We believe that filtering what a remote user of a desktop is allowed to do with their control of an application or desktop keyboard and mouse input is out of scope; it's the same as the local user. However, it may be necessary to sandbox applications in some cases, with sandbox control outside the view of the remoting protocol.

2. Requirements

2.1 Overall Functional and Architectural Requirements

F-1 There are two use cases, namely application sharing and remote desktop. For application sharing, a user wants to show another user an application, e.g., to work within the application collaboratively. For remote desktop sharing, the user wants to manipulate a desktop or GUI-based application from somewhere other than where the applications are running.

F-2 Both whole desktops (screens) and applications must be shareable. Applications may display one or more windows, with the number and position of these windows possibly changing during the session.

F-3 Any numbers of viewers should be able to watch the same application or desktop. Evolving floor control mechanisms determine determine who gets to send input to them and, thus, negotiation of user input is beyond the scope of the protocol.

F-4 In most cases, applications cannot be modified to work with the remoting protocol. Thus, application sharing tools intercept windowing system APIs at some level. Modifications would most likely be required to applications to allow sending full-motion video separately.

F-5 Any protocol negotiation occurs at the session setup level, and is capability-based, not version-based.

F-6 Modular architecture: the components (session negotiation, floor control, graphical output delivery, user input conveyance) should be independent so that they can be replaced separately.

2.2 Input

I-1 A desktop or applications might or might not have an actual monitor, keyboard or mouse attached.
I-2 Input needs to be private, authenticated, integrity-protected, and access-controlled.
I-3 In most cases, at most one remote user at a time can provide input; however, remote and local input might occur simultaneously. The protocol must not restrict the number of simultaneous input sources.
I-4 The remoting protocol needs to be able to convey user input hints, such as modal input. (In model input, input focus is forced into a specific window within a set of windows belonging to an application.)
I-5 The application should be able to indicate which window has focus, to allow appropriate rendering of the window decoration at the remote host.
I-6 Relative timing information for user input may be needed for some applications, such as video games.
I-7 Internationalization: Two end systems may not have the same kind of keyboard.
I-8 Open issue: Can copy-and-paste be supported between the application and the remote user?

2.3 Video Output

V-1 The receiving end system may have a different color depth or screen resolution than the application host.
V-2 The application video output may consist of different types of video, some of which may be able to tolerate lossy encoding. For example, a video streaming application consists of a set of user interface elements that need to be rendered without distortion, while the motion video itself may well tolerate some encoding loss.
V-3 The remoting protocol must be able to convey window layering hints, such as forcing a window to be always on top.
V-4 Some windowing systems allow semi-transparent windows, with varying alpha.
V-5 Relative timing information between updates may be needed to render smooth motion.
V-6 Absolute timing information may be needed to synchronize video output with other remote-source output, such as audio.
V-7 Since different encoding algorithms have different efficiency-complexity trade-offs, the remoting protocol must allow a variety of encoding techniques.

V-8 Two end systems may not have the same set of fonts installed or available.

2.4 Audio and Full-Motion Video

A-1 Applications that are being shared also must be able to share audio streams, time-synchronized with the visual aspects of the application.
A-2 Some applications may also want to send full-motion video directly from the application, time-synchronized with the GUI and the audio stream.
A-3 Due to bandwidth constraint, the receiver may choose not to receive all screen updates. (Example: an area of the screen may contain full-motion video.)

2.5 Transport Usage and Requirements

T-1 Some remoting applications require perfect reliability, flow- and congestion control, for both input and output.
T-2 Some applications need to deliver output to a large number of receivers, possibly relaxing reliability. Thus, support for a variety of transport protocols, including reliable multicast, is needed.
T-3 Applications may be remoted across networks with different bandwidth characteristics.
T-4 Latency, for both input and output, must be minimized to maintain interactivity.
T-5 The protocol must support both high-latency and low-latency environments.

3. Security Considerations

Both input and output data may be highly sensitive. For example, input data may contain user passwords. Thus, encryption of all user input is likely to be required. For some applications, such as sharing slides during a public lecture, confidentiality for user output may not be required. Given the broad set of applications, the remoting protocol MUST support or be able to leverage end-to-end confidentiality and integrity protection mechanism.

Application sharing inherently exposes the shared applications to risks by malicious participants. They may, for example, access resources beyond the application itself, e.g., by installing or running scripts. It may be difficult to constrain access to specific user data, e.g., a specific set of slides, unless the user application can be run in some kind of "jail".

4. IANA Considerations

There are no IANA considerations for this document.

5. References

5.1 Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

5.2 Informative References

Authors' Addresses

Henning Schulzrinne
Columbia University
Department of Computer Science
450 Computer Science Building
New York, NY 10027
US

Phone: +1 212 939 7004
EMail: hgs+mmusic@cs.columbia.edu
URI: http://www.cs.columbia.edu

Jonathan Lennox
Columbia University
Department of Computer Science
450 Computer Science Building
New York, NY 10027
US

Phone: +1 212 939 7000
EMail: lennox@cs.columbia.edu
URI: http://www.cs.columbia.edu

Jason Nieh
Columbia University
Department of Computer Science
450 Computer Science Building
New York, NY 10027
US

Phone: +1 212 939 7000
EMail: nieh@cs.columbia.edu
URI: http://www.cs.columbia.edu

Ricardo Baratto
Columbia University
Department of Computer Science
450 Computer Science Building
New York, NY 10027
US

Phone: +1 212 939 7000
EMail: ricardo@cs.columbia.edu
URI: http://www.cs.columbia.edu

Appendix A. Acknowledgements

TBD

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.