DOpE — a Window Server for Real-Time and Embedded Systems — Extended Abstract —

Norman Feske and Hermann Härtig
Technische Universität Dresden
Department of Computer Science
{nf2,haertig}@os.tu-dresden.de

Abstract

A window server used in real-time applications should be able to assure previously agreed-upon redrawing rates for a subset of windows while providing best-effort services to the remaining windows and operations such as moving windows. A window server used in embedded systems should be small and require only minimal operating system support, for example just threads and address spaces as provided by microkernels.

In this paper, we present the design and an implementation of the DOpE window server. The key techniques used are to move redrawing responsibility from client applications to the window server and to devise a simple scheduling discipline for the redrawing subtasks.

1. Introduction

Applications that need graphical representations can roughly be characterized as belonging to one of two domains:

- Interactive applications basically execute a loop in which the application waits for user input, changes its internal state, and updates its graphical representation accordingly. Typical representatives of this class are word processors, spreadsheet programs, web browsers, editors, and other dialog-based applications. The time requirements of these applications are imposed by the user's ability to supply input events fast enough. Thus, they spend most of the time idling. The only requirement with respect to user responsiveness is that the delay between a state change and update of the representation is less than approximately 100 ms.
- Isochronous and multimedia applications are driven not by user input events but by time. The output of such applications, for example video frames, must be available on the user interface in a periodic manner. Even small delays are perceivable for the user and compromise the quality of service that is expected.

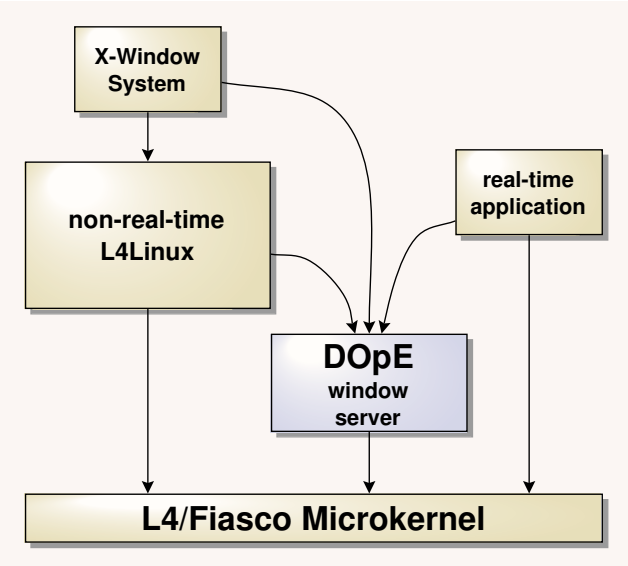

Figure 1. Real-time and non-real-time applications running in one environment

We refer to multimedia applications as real-time (RT) and to other applications as non-real-time (NRT) load.

The task of a window server is to multiplex graphical representations of concurrently running client applications onto the screen, which is one single physical resource. The objective is to guarantee the quality of service of real-time client applications even in overload situations. Such overload situations can be induced by massive output of non-

real-time applications or by the user who interactively rearranges windows. Window servers that are used in scenarios as described above often exhibit undesirable behaviour. Unawareness of real-time versus non-real-time requirements leads to uniform performance degradation in overload situations, even if there are enough resources available to serve the real-time client applications properly.

DOpE (Desktop Operating Environment), the window server presented in this paper, is implemented on top of the Dresden Real-Time Operating System (DROPS) [3]. DROPS enables the sharing of higher-level resources such as disk and network bandwidth via dedicated resource managers [4]. DOpE is such a resource manager that maps CPU cycles and main memory to window-redrawing bandwidth. In a general DROPS setting, DOpE supports real-time clients as well as L^4Linux and XFree86 as non-real-time clients.

With Artifact [7], there is only one real-time window server about which we have non-trivial architectural information. Artifact dynamically creates real-time models for client and server and then makes global admission decisions. In contrast, DOpE is based on a localized, simple, essentially time-driven scheduling approach. A further discussion about how DOpE relates to Artifact can be found in the full version of this paper [2].

In this paper, we present the overall architecture of DOpE (Section 2) and then concentrate on the real-time operation. We give some details about the implementation in Section 3. In Section 4 we rate the application of DOpE in the context of secure system architectures. We conclude this paper with a summary of the most important facts about DOpE and an outlook at our future work.

2. DOpE's architecture

The two key properties of DOpE's client-server architecture in regard to its real-time capability are:

- A client application and the window server share a compact description of the client application's graphical representation in a language with a semantics known to the window server. Thus, the window server can redraw the graphical representation of any client application without the active cooperation of the affected client application. This way no assumptions about the response time of client applications are needed.
- A client application updates the shared representation and then triggers a corresponding redraw operation in the window server. Figure 2 illustrates this architecture. In contrast, the classical approach is based on a fine-grained interface to submit graphical primitives to the server that — in turn — immediately performs the needed graphical operations.

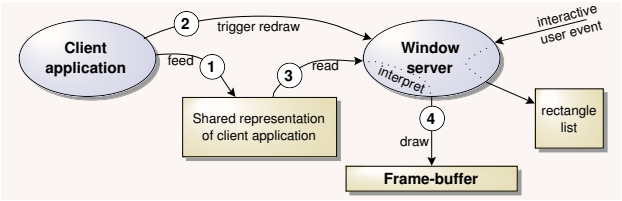

Figure 2. The client application and the window server share a compact description of the client's representation

With regard to real-time, our design allows local scheduling of redraw operations within the window server. The redraw operations are designed such that their execution time is known in advance.

For the output of real-time client applications we introduce a periodic activity in the window server that triggers redraw operations in a defined (and predictable) way.

To establish a periodical real-time output, a client application has to request an admission at the server by specifying the size and update frequency of the desired output area (widget) on the screen. Figure 3 shows an exemplary admission scenario.

After a successful admission, the server refreshes the negotiated widget periodically, based on the known representation of the client application's data, and optionally informs the client about a completed redraw using synchronization messages. It is up to the real-time client application to timely feed its current state to the graphical representation which, in turn, is shared with the window server.

We successfully address overload situations induced by NRT load with separating the cause and execution of NRT redraw requests. Incoming NRT requests are queued in a redraw queue on which we apply operations such as splitting large requests and merging requests that affect the same screen area. This way, all entries of the redraw queue refer to distinct screen areas. Thus, the maximum number of queued pixels is limited and we can guarantee a maximum latency for any output on the screen. These techniques are covered in more detail in the full version of this paper [2].

3. Implementation

We implemented the conceptual ideas discussed in Section 2 in the window server DOpE.

The DOpE window server runs on the top of the L4/Fiasco [6, 8] microkernel which provides threads, ad-

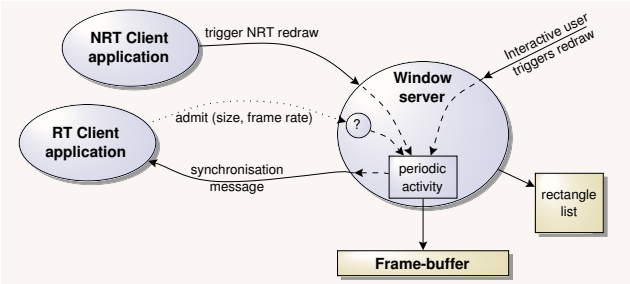

Figure 3. Example scenario showing real-time-admission, synchronization feedback and the triggering of non-real-time redraw requests

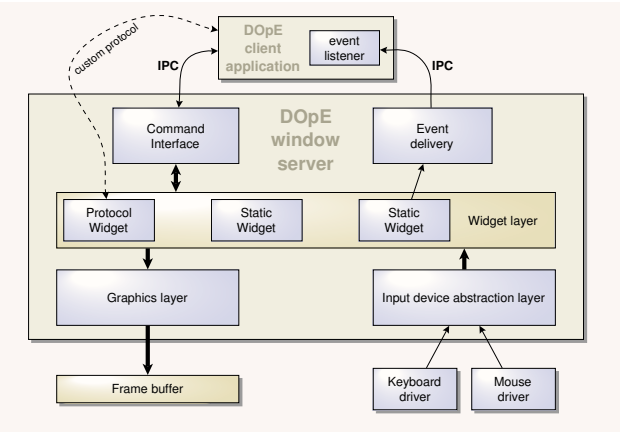

Figure 4. Schematic overview of the DOpE window server

dress spaces and inter-process communication including means to establish shared memory.

3.1. DOpE server structure

Figure 4 illustrates the structure of the DOpE window server. Currently, we use software rendering routines (Graphics layer) to access a VESA frame buffer. It manages clipping and contains functions for drawing graphical primitives such as scaled images and text.

The mouse and keyboard drivers are ported from Linux to the L4/Fiasco platform. On top of the input device drivers there is an input abstraction layer with support for different keymaps. DOpE handles user input events in a non blocking way. Thus, the interactive user does not impact the continuous output of real-time client applications when interacting with the window server — for example by moving a window.

The representation of client applications on the DOpE window server is based on widgets and their relationship to each other (topology). A widget defines the graphical representation of a dedicated type of data or protocol and the response to user interactivity. As illustrated in Figure 4, protocol widgets can establish a dedicated communication channel to the client application, for example via shared memory. This way the representation of a client application can be shared with the window server using custom protocols.

The widget layer is built on top of the graphics and input abstraction layers and contains the implementation of the following widget types:

- Window, button, scrollbar, scale, and frame as the basic primitives of the window server
- Grid allowing the arrangement of multiple child widgets to be aligned in a rectangular grid
- Terminal for textual output with support for a subset of VT100 escape sequences

With VScreen (virtual screen) we implemented a protocol widget with a separate communication interface to the associated client application. VScreen uses raw pixel data as shared representation of the client application. It can be used as a periodically updated real-time widget. The client application and the interactive user can vary the size of each VScreen widget resulting in a scaled output of the represented pixel data.

Similar to the approach of Tcl/Tk, user interface widgets are created and configured by using a text-command-based communication interface to the client application.

Input events are primarily handled by the affected widgets. In turn, widgets can forward events to the client application via the event delivery component (see Figure 4).

3.2. L4Linux, XFree86 and other non-real-time client applications

L^4Linux [1] is a user level implementation of the Linux kernel on top of the L4/Fiasco microkernel.

We implemented a driver module for the XFree86 X Window System which forwards the graphical output of the X Window System to a DOpE widget. Thus, we are able to run the broad range of non-real-time X11 applications together with native real-time applications in one environment.

3.3. Real-time clients

With VScrTest we implemented a real-time client application that makes use of the VScreen widget to display a continuous stream of pixel data. It calculates four different graphical effects: a 3D particle effect, a bumpmapping effect, a voxel landscape and a feedback effect. The effects are rendered into a shared memory buffer which, in turn, is displayed by the DOpE window server at a guaranteed frame rate of 25 frames per second. As VScrTest makes use of the real-time features of DOpE, the rearrangement of windows and the graphical output of concurrently running client applications have no impact on the constant update rate of the real-time client.

4. Secure systems and source-code complexity

In secure system architectures (as proposed in [5]) the graphical user interface is part of the trusted computing base. Thus, it is important to keep the source-code complexity of the window server low. Also, when window servers are used in embedded systems with limited resources, they should be small and they should not rely on large operating systems.

Altogether, the current implementation of DOpE consists of about 10,000 lines of code. This code includes a basic set of widgets (window, scrollbar, grid-layout, vscreen, terminal), all graphical rendering routines and abstractions for shared memory, screen, timers and input devices.

Internally, DOpE is structured in a component-based way, which allows a high degree of customization for special applications. By leaving out higher-level widgets such as grid-layout, DOpE can be scaled down to a minimalistic but fully working window server with about 7,000 lines of code. DOpE's extremly low source-code complexity enables an exhaustive verification of the window server and makes it viable for secure platforms.

The size of the executable binary of DOpE including input device drivers, graphics routines, graphical data (four bitmap fonts), and all widget types mentioned above is 250 KByte. The core functionality (without input drivers and the L4 Environment) of DOpE enfolds a binary size of only 150 Kbyte.

5. Conclusion

In this paper we presented DOpE — a window server that is capable of serving real-time clients and non-real-time clients together in one environment. After a successful admission, DOpE guarantees fixed update rates to its real-time clients while offering best-effort service to non-real-time clients. We devised a method for preventing overload situations by handling redraw operations predictably. We achieved this with only 10,000 lines of code and still, require only minimal support by the underlying operating system. This facts makes DOpE viable for embedded systems and secure architectures in which the window server belongs to the trusted computing base. A more detailed description of our approach is covered in the full version of this paper [2].

Currently, there is work in progress to create a driver infrastructure for the support of graphics-acceleration hardware to boost the overall performance of DOpE's graphical output. While doing this, we want to preserve the predictability of these operations by introducing models for the time response of these operations on specific hardware.

References

[1] Martin Borriss, Michael Hohmuth, Jean Wolter, and Hermann Härtig. Portierung von Linux auf den μ-Kern L4. In *Int. wiss. Kolloquium*, Ilmenau, September 1997.

[2] Norman Feske and Hermann Härtig. DOpE — a Window Server for Real-Time and Embedded Systems. Technical Report TUD-FI03-10-September-2003, TU Dresden, 2003.

[3] H. Härtig, R. Baumgartl, M. Borriss, Cl.-J. Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schönberg, and J. Wolter. DROPS: OS support for distributed multimedia applications. In *Proceedings of the Eighth ACM SIGOPS European Workshop*, Sintra, Portugal, September 1998.

[4] H. Härtig, L. Reuther, J. Wolter, M. Borriss, and T. Paul. Cooperating resource managers. In *Fifth IEEE Real-Time Technology and Applications Symposium (RTAS)*, Vancouver, Canada, June 1999.

[5] Hermann Härtig. Security Architectures Revisited. In *Proceedings of the Tenth ACM SIGOPS European Workshop*, Saint-Emilion, France, September 2002.

[6] Michael Hohmuth. The Fiasco kernel: System architecture. Technical Report TUD-FI02-06-Juli-2002, TU Dresden, 2002.

[7] Jay K. Strosnider John E. Sasinowski. Artifact: An experimental real-time window system. 1995.

[8] J. Liedtke. On μ-kernel construction. In *Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP)*, pages 237–250, Copper Mountain Resort, CO, December 1995.