Eve: An Icon Monitor Coordinator *

Clinton L. Jeffery

August 20, 1992

Abstract

Eve is a program execution monitor coordinator. It coordinates one or more program execution monitors, providing them with requested events and various global services. Eve runs under MT Icon, an extension of Version 8.7 of the Icon programming language.

1 Introduction

Often a monitoring situation calls for not just one monitor but a set of monitors executing concurrently. In such a situation each program event must be *multicast* to the subset of the monitoring programs that are interested in events of the given type. Eve is an Icon program that performs this function for monitors running in the MT Icon [Jeff90] environment. Eve selects the minimal set of events from the monitored program required to satisfy the group of monitoring programs. Eve uses X-Icon for its user-interface facilities [Jeff91] and some of its features are tailored towards program monitors that are themselves X-Icon visualization tools.

This document provides information about Eve for the user and for the program monitor author. It also discusses the event demultiplixing problem and presents Eve's central algorithms.

*This work was supported in part by the National Science Foundation under Grant CCR-8713690 and a grant from the AT&T Research Foundation.

2 Running Eve

2.1 Starting up

Eve's command line syntax is

eve [-f *configfile*] [-all] *icon-command-line*

Eve takes as its arguments the command-line that invokes the monitored program. It reads a list of program monitors from a *configuration file* specified by the -f option, or else from $HOME/.eve if it is present.

2.2 Configuration files

An Eve configuration file is simply a list of command lines starting with the tool name and including command-line options in the standard form. Here is a sample Eve configuration file:

```
/usr/icon/ibin/colors
/usr/cjeffery/tools/algae
/usr/cjeffery/tools/mempie
/usr/cjeffery/tools/piano -P580,0
/usr/cjeffery/tools/gnames -P 0,696
/usr/cjeffery/tools/lstyle -P 580,696
```

Note that this configuration includes private monitors from /usr/cjeffery/tools and a monitor from the suite of publicly-available monitors in /usr/icon/ibin.

In addition to whatever tool-specific options a tool supports, Eve configuration file options frequently include options that specify window features such as size and position. These options follow a set of conventions, so that all tools use the same window options. See procs/optwindow.icn in the X-Icon section of the Icon program library.

After reading the list of available monitors, Eve pops up a window on the user's X Window display (see Figure 1).

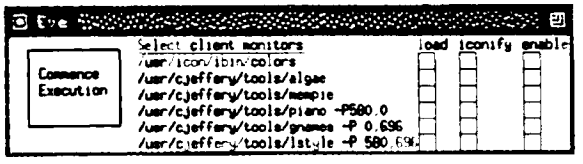

Figure 1: Eve displays a configuration on startup

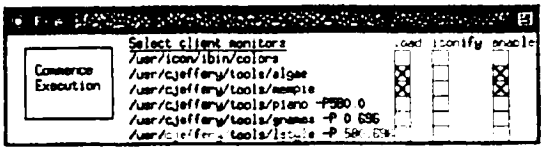

Figure 2: Tool selection

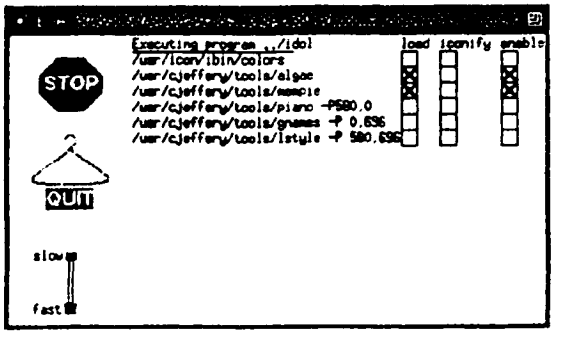

Figure 3: Execution mode

2.3 Tool selection

If the user ran Eve with the -all option, Eve commences executing all the listed monitors. Otherwise, the user selects one or more monitoring tools by clicking a mouse button in the load column check box (see Figure 2). The user completes tool selection by pressing the mouse button in the upper left corner area marked Commence Execution. Eve then loads each tool.

2.4 Executing the monitored program

After initializing the monitors, Eve redraws its window to indicate execution mode (see Figure 3), and then goes into an event processing loop. During each step of the loop, an event is obtained from the monitored program based on the union of the monitors' desired events and is transmitted to each monitor requesting events of that type.

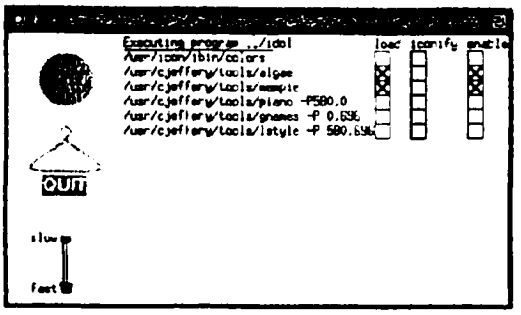

Figure 4: Execution is paused, press GO to continue

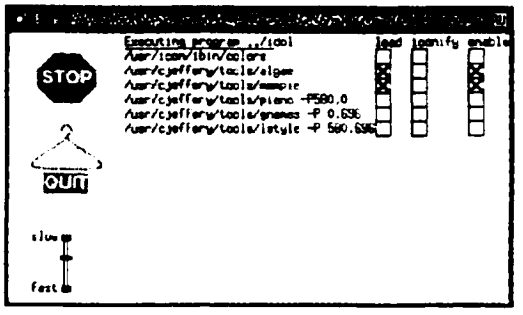

Figure 5: Changing the slider slows execution

2.5 Eve's execution controls

Eve provides a few execution controls that are universal to all tools in a column to the far left of its control window. Clicking on the stop sign pauses execution and the stop sign is changed into a go signal that the user clicks to continue (see Figure 4). Clicking on the QUIT button terminates Eve and all monitors immediately.

Sliding the slider towards the end of the scale marked SLOW inserts delays into the execution without stopping it (see Figure 5).

Eve also provides means by which one can iconify and/or disable individual tools if the tools follow certain conventions. In order for iconification to work, the tool must assign a global variable Visualization to be either a window or a list of windows that the tool has open on-screen. In order for disabling to work, the tool must handle certain non-character events described in the section below on general purpose virtual events.

Pressing a checkbox in the column marked iconify causes Eve to attempt to find and iconify that tools windows (see Figure 6). The box may be toggled off by clicking it again, reversing the effect.

Similarly, toggling off a checkbox in the column marked enable causes Eve to send the tool a special *disable event* (see Figure 7). An disable event may be interpreted as a request that the tool minimize its resource consumption; some tools are able to temporarily suspend themselves and miss events with no ill effects. Other tools may continue to request events in order to maintain their internal state, but may stop updating the screen continuously.

Checking the box again causes an *enable event* to be sent, requesting that the tool reactivate itself.

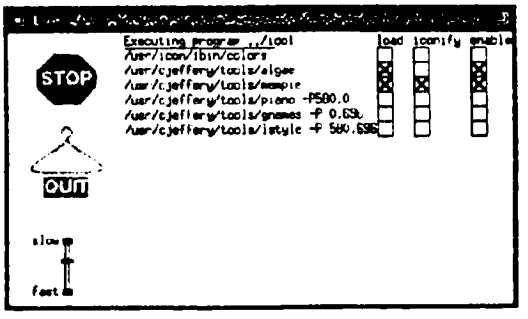

Figure 6: Mempie is iconified

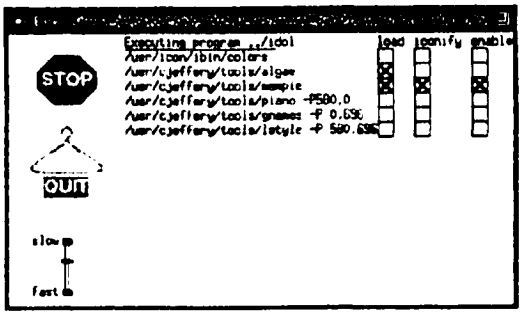

Figure 7: Algae is disabled

3 Writing monitors to run under Eve

Eve presents events to client monitors using the interface defined in [Gris90]. This section describes a few differences between the standalone interface and the Eve environment. Note that programs written for the Eve environment run without change or recompilation as standalone tools.

3.1 Client environment

After each monitor is loaded, Eve initializes it with references to its event source (the Eve program itself) and the monitored program. The former is necessary so that monitors yield control to Eve to obtain each event. The latter is provided so that the state of the monitored program may be examined or modified directly by all monitors. These references in the form of co-expression values are assigned to the keyword &eventsource and the global variable Monitored, respectively; the global variable Monitored is declared in each monitor when it links to the evinit event monitoring library.

Since under Eve &eventsource is *not* the program being monitored, execution monitors should always use Monitored to inspect program state; for example, to inspect the name of the current source in the executing program a monitor should call keyword(" file " , Monitored) rather than keyword(" file " ,&eventsource).

3.2 Eve is (nearly) implicit to programs

From a programmer's standpoint Eve's operation is implicit. The monitored program is never aware that monitoring is taking place, and monitoring does not inherently affect behavior within the monitored program (other than slowing execution). Similarly, within the various program monitors Eve's presence is not normally visible; the event stream appears as though it is sourced directly from the monitored program.

3.3 General-purpose virtual events

Tool communication in general is handled by the use of artificial events with event codes that are not one-character strings. In order to receive any of these special event codes, a tool must pass EvGet() a non-null second parameter, which serves as a flag to let these events through.

Eve itself sends certain codes when directed by the user (in the Eve control window) or by other tools. The most important are the disable and enable events discussed above, E_Disable and E_Enable. When an E_Disable event is received, a tool is requested to disable itself. Tools that do not maintain any state between events can simply shut off their event stream by calling EvGet(", 1). Tools that require events in order to maintain internal consistency can at least skip their window output operations while they are disabled. An E_Enable event informs the tool that it should resume operation, updating its display first if necessary.

4 Eve in operation

This section describes the primary algorithms responsible for Eve's performance. The key ideas are to filter events at the source and to precompute the multicast function by which events are distributed.

Eve's primary function is to obtain events from a monitored Icon program on behalf of its client monitors. The events are obtained from an *event stream* in the form of an executing program.

Different monitors require different kinds of events. After obtaining a list of clients to execute, Eve loads each client. It then activates each tool for the first time; when the tool completes its initialization, it calls EvGet(), passing it an event mask.

4.1 Computation of the minimal event set

Each time a monitor requests its next event from Eve, it transmits a cset event mask indicating what events it is interested in. Eve could simply request all events from the monitored program, and forward events to each tool based on its current mask.

The interpreter runtime system is instrumented with so many events that this brute-force approach is too slow in practice. In order to minimize the cost of monitoring, Eve asks the monitored program for the least set of events required to satisfy the monitors.

From the event masks of all monitors, Eve computes the union and uses this cset to specify events from the monitored program. The pseudo-code for this union calculation is

```
unioncset := ''
every monitor := !clients do {
```

```
if monitor.enabled === E_Enable then
    unioncset ++:= monitor.mask
}
```

Although every tool can potentially change its event mask every time it requests an event, constant recomputation of the union mask would be more expensive than simply requesting all events. Fortunately, most tools call EvGet() with the same event mask cset over and over again. Eve avoids recomputing the union event mask unless it has changed since the previous call.

4.2 The event code table

The minimal event set described above reduces the number of events actually generated from the monitored program. When an event is received from the program, Eve dispatches the event to those programs that requested the event. The larger the number of monitors running, and the more specialized the monitors are, the smaller the percentage of monitors that will be interested in any given event.

Eve could simply test the event code with each monitor's cset mask with a call any(mask, &eventcode). This test is fast, but performing the test for each monitor is inefficient when the number of monitors is large and the percentage of monitors interested in most events is small. Instead, the list of monitors interested in each event code is precomputed as the union mask is constructed. These lists are stored in a table indexed by the event code. Then, after each event is received, a single table lookup suffices to supply the list of interested monitors. [1] The code for union mask computation thus becomes:

```
unioncset := ''
EventCodeTable := table()
every monitor := !clients do {
    if monitor.enabled === E_Enable then {
        unioncset ++:= monitor.mask
        every c := !monitor.mask do {
            /EventCodeTable[c] := [ ]
            put(EventCodeTable[c], monitor)
            }
        }
    }
```

4.3 Eve's main loop

Eve's main loop activates the monitored program to obtain an event, and then dispatches the event to each monitor whose mask includes the event code. Since this loop is central to the performance of the overall system, it is coded carefully. Event dispatching costs one table lookup plus a number of operations performed for each monitor that is interested in the event – monitors for whom an event is of no interest do not add processing time for that event. The code for Eve's main loop looks like:

[1] One might assume that an array of 256 lists might serve better than a (hash) table. Since the event code is a string, however, it would have to be converted to an integer in order to be used as a list index; T["a"] is faster than L[ord("a")].

```
while @target do {
   every monitor := !EventCodeTable[&eventcode] do {
      C := event( , , monitor.prog)
      if C ~=== monitor.mask then {
         if type(C) ~== "cset" then {
            #
            # The client program has raised a signal; pass it on.
            #
            broadcast(C, monitor)
            }
         else {
            monitor.mask := C
            computeUnionMask()
            }
         }
      }
      # check for user activity in the control window
      # if the slider is not zero, insert delay time
   }
```

5 Summary

Eve is a simple monitor coordinator that enables multiple monitors to operate on a single subject program. In event monitoring performance considerations are serious, and event more so when running multiple monitors. Nevertheless, Eve is written in the same high-level interpreted language as the monitors and the subject program, and runs acceptably on contemporary hardware. This demonstrates the efficiency of MT Icon.

References

[Gris90] Griswold, R. E. Processing Icon Event Streams. Technical Report IPD152, Department of Computer Science, University of Arizona, December 1990.

[Jeff90] Jeffery, C. L. The MT Icon Interpreter. Technical Report IPD169c, Department of Computer Science, University of Arizona, July 1990.

[Jeff91] Jeffery, C. L. X-Icon: An Experimental Icon Windows Interface. Technical Report 91-1, Department of Computer Science, University of Arizona, January 1991.