WindView: A Tool for Understanding Real-time Embedded Software Through System Vizualization

David Wilner
Wind River Systems

Introduction

There is enormous pressure today on software development teams creating real-time embedded systems. The software content of embedded systems is increasing, the applications are becoming more complex, the time-to-market pressure is increasing, and more often than not, the success of the entire project or product hinges on the software. Thus these software developers find that their job is becoming more difficult and more critical, but that they have less time to do it.

Modern embedded software applications are marvelously complex machines. Real-time operating systems provide a rich set of facilities for decomposing an application into multiple concurrent tasks interacting through mechanisms for synchronization, mutual exclusion, communication, timing, interrupt handling, exception and error handling, and so on. Many applications use a sophisticated client-server model of interaction. Many use multiple processors, both loosely and tightly coupled. A typical embedded application will use hundreds or even thousands of system objects including tasks, semaphores, message queues, timers, I/O devices, network connections, and more.

The problem is that while advances in hardware, operating systems, and application frameworks have provided the facilities for building such complex machines, the development environments have not provided the tools to manage such complexity during development, testing, and debugging. The tool described in this paper, WindView®, is designed to fill this need.

WindView's Goals

WindView is based on three principles that are essential for the next generation of development tools: system-level development, system visualization, and system dynamics.

System-Level Development: Raising Developers' Sights

WindView's first goal is to raise the level at which application developers are working while keeping pace with the increasing productivity demands - moving from source-level debugging to system-level debugging.

With software applications becoming more complex, development methodologies improving, and code reuse increasing, developers find themselves spending less time writing and debugging sequential lines of code in a single source module, and more time working on systems integration, debugging interactions between modules, tasks, and entire subsystems. Developers need tools that understand system level constructs of tasks and processes, interprocess synchronization and communication, interrupts, timers, and exceptions, and their interactions. WindView allows developers to monitor and control these interactions directly, raising the developers' vantage point and thus their productivity.

System Visualization: A Picture is Worth a Thousand Words

Considering that a complex software application may consist of dozens of processes, tasks and interrupts, and hundreds of system objects such as semaphores, message queues, and timers, it is easy to understand why developing such software is so difficult. How can information about the system be presented in a way

that helps rather than overwhelms the developer? The answer is in pictures.

Graphical displays can make complex systems understandable by presenting the mass of information in a form that developers can more easily digest. Today's graphical user interfaces (GUIs) make it possible to create tools that do this.

Unfortunately the first generation of GUI-based tools has been little more than window dressing on the old command line interfaces, providing buttons or menus to access the same old utilities. In contrast, the WindView is a completely new GUI-based tool that displays system information intuitively, shows relationships between system components, and helps the developer visualize the application's construction and behavior.

System Dynamics: Seeing Software in Action

In the last few years tools for exploring the static construction of software has become commonplace, such as source browsers and cross-referencers. But the difficult problems in real-time embedded systems are dynamic problems of interactions among system components. Software diagnostic tools have been limited to "snapshots" of the momentary state of various system objects (e.g., tasks, semaphores, message queues) and profilers that give broad system statistics such as average CPU utilization by task or by subroutine.

Developers have remained in the dark about the detailed inner-workings of their application and operating system. They have had no way to examine in detail the sequencing and timing of software state transitions, synchronizations, communications, context switches, interrupts, and significant application events of all kinds. Yet this is precisely the information that developers need to solve common software problems such as missed deadlines, inadequate response times, performance bottlenecks, race conditions, client-server deadlocks, and hardware/software interaction errors.

To meet this need, WindView captures a detailed history of the actions and interactions of application and system software components over time and then interprets and displays this history in the ways most meaningful to the software developer.

WindView: Taking Off the Blindfold

By capturing and graphically displaying the precise sequence and timing of events in an embedded application, WindView gives developers an unprecedented view of their software in action.

In various modes WindView allows the developer to see periodic patterns in the application, discern unused CPU bandwidth, help debug race conditions and deadlocks, identify performance bottlenecks, time segments of application code, assist in post-mortem analysis of failed systems, and debug client-server applications. Bugs that in the past have delayed product completion by days, weeks or even months can in many cases be located and solved in minutes.

The basis of WindView's capabilities is a variety of instrumentation capabilities built into the latest version of the VxWorks® real-time operating system and several other commercially available non-real-time operating systems. This instrumentation is what allows WindView to capture information and monitor the detailed activities of the system. For instance, WindView logs all task state transitions (i.e., the points at which tasks are started, interrupted, blocked and unblocked, and completed). The instrumentation likewise tracks changes in system objects such as the sending and receiving of messages, or giving and taking of semaphores. As elaborated below, considerable attention has been paid to keeping the overhead of the WindView instrumentation

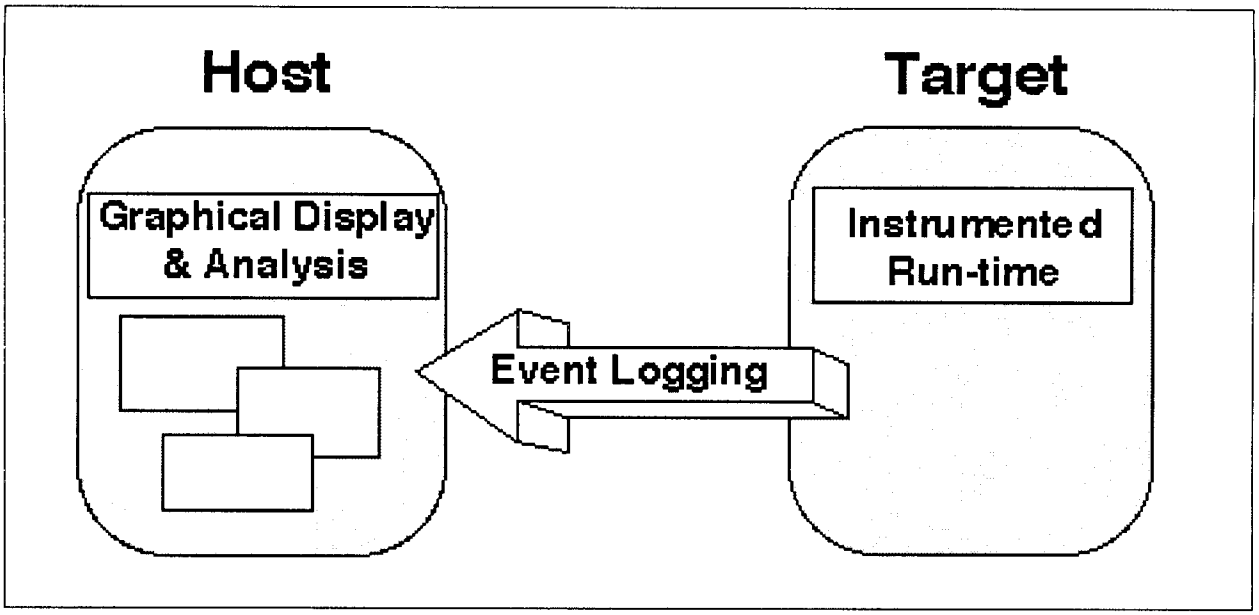

Figure 1

minimal.

WindView components on the host system analyze the event stream from the instrumented target and display them on an oscilloscope-like graph of system activity (see figure 2). The background is a trace of the thread of execution as the operating system context switches between various processes, tasks, and interrupts. Overlaid on this trace are symbols representing task state transitions and other system events on semaphores and message queues, interrupts, timer expirations, and so on. The display can be scrolled forward and backward in time and zoomed in and out to different time scales. Detailed information on each event is available by clicking on the corresponding symbol on the display. The precise high-resolution timestamp (better than 1 microsecond resolution, in most configurations) is available for each event, and the elapsed time between any two events can be displayed.

Using WindView

WindView is a rich tool that has many possible uses in the development of software applications. Because it is a new kind of tool, developers are still finding unexpected ways to use WindView.

With the display "zoomed out", developers can see the overall contour of execution of the system. In particular, cyclic and periodic patterns are easily discernible. These patterns can provide a kind of signature for the system that changes when the system changes modes. The "duty cycle", including the amount of unused processor bandwidth in a cyclic application, can be easily measured on the display.

On the other hand, "zooming in" shows a detailed event-by-event trace of the system. For the first time, developers have the information they need to find subtle problems such as race conditions, missed deadlines, deadlocks, and "performance bugs" (in which an application is generating correct results but at significantly slower speed than intended). For example, a task failing to complete its processing in time to handle a particular external interrupt would be immediately visible in WindView. Excessive competition for a particular mutual exclusion semaphore would be visible as a series of rapid context switches

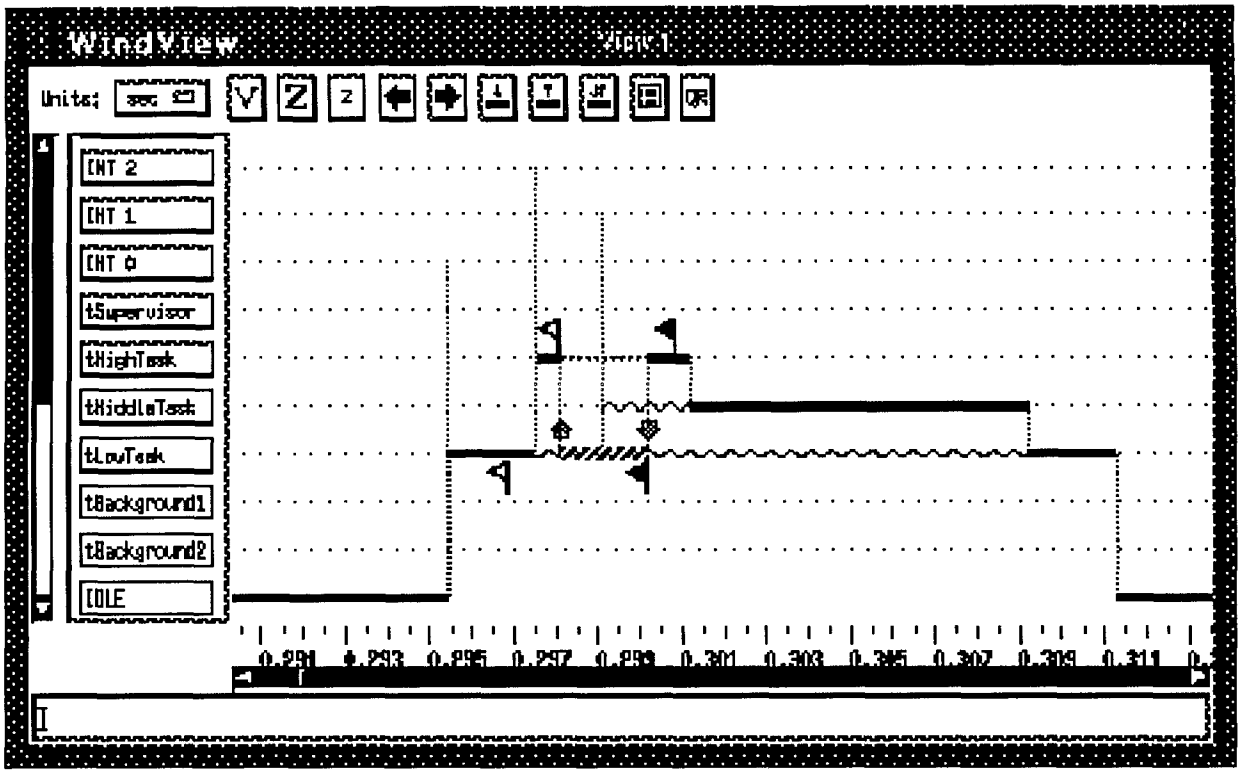

Figure 2

among the contending tasks. The cause of a deadlock (where, for example, each of two tasks blocked waiting for a resource owned by the other) is easy to track down by scrolling backwards to see the events leading up to the deadlock.

For clarity, the amount of information on the WindView display can be reduced by specifying filters such that only selected types of events are displayed and/or only events caused by selected tasks. For example, client-server based architectures are easy to examine by restricting WindView to display only events on the relevant message queues.

One of the most powerful capabilities of WindView is the ability to do post-mortem analysis, historically one of the most vexing of debugging jobs. With traditional tools, when a program under development crashes, no record is kept of what the system was doing when the failure occurred. The engineer typically has to figure out the cause by trial and error, running various pieces of code until the failure is duplicated and then tracking down the flaw. WindView, by contrast, keeps an event log that remains intact after program failures, making it possible to examine the sequence of events leading up to the failure.

Customizing WindView

WindView is not limited to displaying only operating system events: application developers can instrument their applications as well. Application code can make calls directly to the event logging facilities. These application events can then be displayed on the WindView display with their own user-defined icons along with the built-in events. This allows developers to obtain precise timing information about application internals. For example, the time spent in successive layers of a net-

work protocol stack can be easily determined by inserting logging calls at the entrance to each layer. Then each layer transition will be displayed on the WindView display and the elapsed time between each can be measured precisely.

More dynamically, developers can instrument their application interactively via the source debugger. By setting an "eventpoint" on a given line of source code, an event will be logged whenever that line of code is executed. These eventpoints will also be displayed on the timeline. Clicking on an eventpoint symbol on the timeline will bring up the corresponding source line in the source display window. Conversely, it is possible to have the event stream searched for the occurrence of a particular eventpoint and the WindView display positioned at that event.

WindView allows developers to time application algorithms with unprecedented ease and precision. Traditional methods of timing a piece of code often relied on invoking the code repeatedly until the elapsed time could be measured by a low-resolution clock. However, this gives only a statistical average of the execution time and can be very inaccurate if memory caches are involved. By installing eventpoints before and after the code to be measured, a very precise and accurate timing of a single invocation of the code is easily obtained.

WindView is also fully programmable. Using the popular TCL (Tool Control Language) users can create scripts to do anything that it is possible to do from the graphical user interface. TCL scripts can also be used to customize the display of user defined application events.

Finally, WindView has several layers of open APIs that allow users to access the underlying primitives. APIs are provided for timestamping, logging of events, inserting eventpoints, and manipulating log buffers. This has allowed WindView to be connected to a wide variety of environments including traditional desktop systems, embedded hardware/software systems, and simulators.

More About the WindView Instrumentation

A key issue in a dynamic monitoring and analysis tool like WindView is the overhead that it places on the target system and how that overhead might perturb the functioning of the system under test. Considerable attention has been paid to keeping the overhead of the WindView instrumentation minimal, so that even with all the instrumentation enabled on a very complex application, the total overhead is typically a fraction of a percent of the total available CPU cycles. Most developers find that the few extra microseconds incurred on certain system functions is far outweighed by the months of development time that can be saved with WindView.

WindView addresses the intrusion issue in several ways. First the timestamping and logging itself is highly optimized so that making a log entry takes only few microseconds in a typical configuration.

Secondly, the amount of logging is kept to an absolute minimum by a technique called "incremental annotation". The traditional brute-force approach to logging is to capture a complete description of the relevant system state and event parameters at each event. WindView's "incremental annotation" technique puts the event logging at carefully selected points in the kernel's internal state transitions, capturing only the minimum incremental information necessary to reconstruct the sequence of events in the application. This requires a much more sophisticated analysis tool to parse and recreate the system state transitions as they originally occurred. Thus WindView trades off higher post processing for

much reduced logging during program execution, thereby eliminating up to 50% of the instrumentation overhead at run-time.

Finally, WindView instrumentation is very flexibly run-time selectable. The instrumentation can be turned on to one of three instrumentation levels, giving successively more information about the behavior of the system at the expense of more logging overhead. The lowest level is the context switch events. With this level WindView can display the sequence of execution of all tasks and interrupts. The next level is task state transition events, such as a task becoming unblocked or a delay timer expiring. With this level, WindView can display the state of each task at all times. The highest level is the system object events, such as giving or taking a semaphore, or send or receiving a message. With this level WindView can display the actions that caused the context switches and state transitions. These levels of instrumentation can be selected interactively, at run-time.

Developers can also select classes of objects to be instrumented or not instrumented and even turn on and off instrumentation on a per object basis. For example, the developer can turn on instrumentation for semaphores but turn of the instrumentation of message queues. At a finer level of granularity the developer can select certain semaphores of interest to be instrumented while leaving all other semaphores

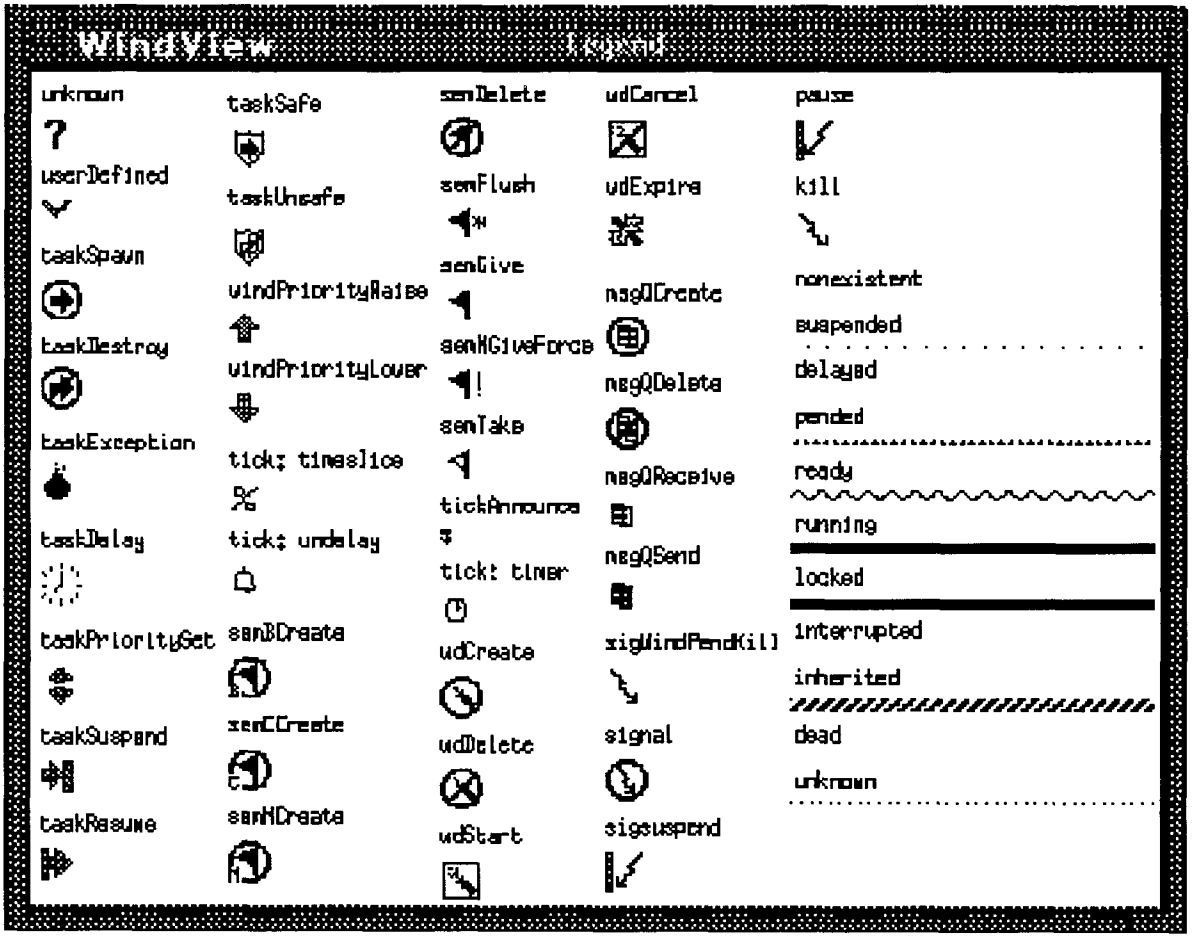

Figure 3

uninstrumented. Again, this selection of instrumentation options is done at run-time with no recompilation or rebuilding of the application.

More About the WindView Display and Analysis Tools

WindView's user interface is extremely sophisticated and yet is so simple and intuitive to use that developers can get started without reading the manual. The WindView display starts with traditional "zoom", "pan", and "scroll". The analysis tools include "filters", "searches", and "triggers" which allow the user to select and find the events of interest in a complex application by combinations of criteria. A "delta time" field displays the elapsed time between two user controlled cursors, allowing the time between events to be measured to the microsecond. "Bookmarks" allow the developer to mark and return to those locations with a single click. With WindView, the developer can bring up multiple display windows that can be scrolled and panned separately to allow comparison of different execution sequences.

The graphical user interface uses a consistent time-saving drag and drop paradigm. For example, to get the detailed information pertaining to a particular event, the user drags the event icon into an "Event Inspectors" window.

Finally, WindView contains an extensive online hypertext help system.

WindView is a versatile tool that has many possible uses in the development of an embedded system—from verifying high-level design to debugging common real-time problems to assisting quality assurance efforts in regression testing. Developers with imagination are finding many more uses for a tool of WindView's power.

About the author...

David Wilner, Vice President of Engineering for Wind River Systems, co-founded the company in 1981 and is a recognized expert in the field of real-time software engineering. Prior to Wind River, Wilner spent several years as a Senior Staff Scientist at the Lawrence Berkeley Laboratory, where he was chiefly responsible for the implementation of real-time control systems for several nuclear accelerators and high-speed computer networks. Wilner has a B.S. in computer science from the University of California, Berkeley. He is a member of several national standards committees, including the POSIX standards committee.