Graphics in embedded systems: Add HTML to Java- and C-based user-interface options

Ian Smith - February 18, 1999
Embedded-system developers looking for a way to add graphics to their systems have a range of methods to choose from; no absolute prescription for success exists. The traditional C-based graphical-user-interface (GUI) tool kit and associated development tools are strong contenders. Java is gaining acceptance, and smaller memory requirements make it a serious player. These two paradigms present attractive options for compelling solutions of next-generation embedded systems.

An HTML approach breaks away from these programmatic paradigms and builds on the huge presence HTML has in the World Wide Web. This viable approach solves many of the problems of traditional approaches and has the benefit of clearly separating the user interface from the application. Traditional real-time operating systems (RTOSs), designed for mission-critical applications, are powerful tools for building the devices becoming prevalent today. Combining an RTOS with HTML allows you to add powerful, appealing, and visually distinctive GUIs to embedded devices without compromising reliability or performance.

Graphics meets embedded systems

Until recently, graphics design has not been much of an issue for traditional embedded and real-time systems. Medical equipment, printers, and phones, for example, have either no local user interface or an exceedingly primitive one, such as a simple alphanumeric LCD or even just two or three LEDs.

However, this situation is quickly changing. New markets are rapidly emerging, particularly in hand-held and consumer devices, in which ever-cheaper displays with serious graphics capabilities are starting to appear on a range of products. Personal digital assistants (PDAs), car navigation systems, Web phones, set-top systems (using TVs as displays), video-disk players with "true GUI" setup menus—to name a few—are getting smarter. Silicon and display prices are falling. New display technologies, such as light-emitting polymers, may make it possible to have a display on just about any device, even without flat space to attach the display to.

The traditional embedded markets are also changing. Printers, copiers, and test instruments have traditionally had few or no local GUIs. Copiers have relied on a simple LCD panel, and a workstation often remotely manages networked devices. As display prices fall, these systems are acquiring increasingly sophisticated local user interfaces, either as alternatives to remote management or as facilitators of initial setup. High-end imaging devices offer high-resolution displays with GUIs that feature tabbed boxes, buttons, menus, and pop-up dialogs. This activity is all happening in the context of strong growth in the smart-embedded-system market—systems with 32-bit CPUs and firmware with a distinct homegrown or off-the-shelf OS layer.

Consider your GUI choices

As embedded products evolve, developers need to choose a paradigm to use for embedded-GUI development. Historically, designers have based GUIs on C or C++ application-programming interfaces (APIs), which is the case with the X Windows System (available on all Unix systems), Microsoft Windows (Microsoft Corp, www.microsoft.com), and various portable GUI libraries. A few interfaces use more advanced languages, such as Smalltalk and TCL/Tk. In the embedded-system world, however, such standardization is rare. The market is younger, and standardization with the implicit lock to a large application base is not the issue it is in the desktop world.

As a result, a number of distinct but coexisting technologies may serve embedded-systems graphics development. You can place these technologies into three camps. The first camp, traditional 3GL methods based on C and C++, will continue to have a strong presence for compelling reasons, including high performance and reliability. Java represents a second new-application-environment camp, for which a completely new programming environment provides a secure base for GUI development. And finally, new GUI paradigms that break away from the programmatic approach to GUI development are beginning to emerge.

The third generation

Traditional 3GL methods are still going strong. X Windows has gone embedded; Windows CE offers a subset of Win32; and a range of proprietary approaches exists from RTOS vendors, such as Photon (QNX Software Systems Ltd, www.qnx.com), Maui (Microware Systems Corp, www.microware.com), and Poptic (Integrated Systems Inc, www.isi.com). Zinc Software Inc offers a comprehensive, portable make utility, hypertext-based online documentation, and numerous examples and tutorials. However, some developers may not have a GUI environment that supports Java, which is the engine that propels the second camp. As a result, a three-tier approach is common. The first tier is a Java GUI running on a Java host and networking to a remote system that runs the actual application. The second tier is Java running in a Java interpreter. The third tier is a Java GUI running on a Java component that is hosted by the OS. The first two tiers are built around the Java Virtual Machine (JVM).
Embedded-system developers looking for a way to add graphics to their systems have a range of methods to choose from; no absolute prescription for success exists. The traditional C-based graphical-user-interface (GUI) tool kit and associated development tools are strong contenders. Java is gaining acceptance, and smaller memory requirements make it a serious player. These two paradigms present attractive options for compelling solutions of next-generation embedded systems.

An HTML approach breaks away from these programmatic paradigms and builds on the huge presence HTML has in the World Wide Web. This viable approach solves many of the problems of traditional approaches and has the benefit of clearly separating the user interface from the application. Traditional real-time operating systems (RTOSs), designed for mission-critical applications, are powerful tools for building the devices becoming prevalent today. Combining an RTOS with HTML allows you to add powerful, appealing, and visually distinctive GUIs to embedded devices without compromising reliability or performance.

Graphics meets embedded systems
Until recently, graphics design has not been much of an issue for traditional embedded and real-time systems. Medical equipment, printers, and phones, for example, have either no local user interface or an exceedingly primitive one, such as a simple alphanumerical LCD or even just two or three LEDs.

However, this situation is quickly changing. New markets are rapidly emerging, particularly in hand-held and consumer devices, in which ever-cheaper displays with serious graphics capabilities are starting to appear on a range of products. Personal digital assistants (PDAs), car navigation systems, Web phones, set-top systems (using TVs as displays), video-disk players with "true GUI" setup menus—to name a few—are getting smarter. Silicon and display prices are falling. New display technologies, such as light-emitting polymers, may make it possible to have a display on just about any device, even without flat space to attach the display to.

The traditional embedded markets are also changing. Printers, copiers, and test instruments have traditionally had few or no local GUIs. Copiers have relied on a simple LCD panel, and a workstation often remotely manages networked devices. As display prices fall, these systems are acquiring increasingly sophisticated local user interfaces, either as alternatives to remote management or as facilitators of initial setup. High-end imaging devices offer high-resolution displays with GUIs that feature tabbed boxes, buttons, menus, and pop-up dialogs. This activity is all happening in the context of strong growth in the smart-embedded-system market—systems with 32-bit CPUs and firmware with a distinct homegrown or off-the-shelf OS layer.

Consider your GUI choices
As embedded products evolve, developers need to choose a paradigm to use for embedded-GUI development. Historically, designers have based GUIs on C or C++ application-programming interfaces (APIs), which is the case with the X Windows System (available on all Unix systems), Microsoft Windows (Microsoft Corp, www.microsoft.com), and various portable GUI libraries. A few interfaces use more advanced languages, such as Smalltalk and TCL/Tk. In the embedded-system world, however, such standardization is rare. The market is younger, and standardization with the implicit lock to a large application base is not the issue it is in the desktop world.

As a result, a number of distinct but coexisting technologies may serve embedded-systems graphics development. You can place these technologies into three camps. The first camp, traditional 3GL methods based on C and C++, will continue to have a strong presence for compelling reasons, including high performance and reliability. Java represents a second new-application-environment camp, for which a completely new programming environment provides a secure base for GUI development. And finally, new GUI paradigms that break away from the programmatic approach to GUI development are beginning to emerge.

The third generation
Traditional 3GL methods are still going strong. X Windows has gone embedded; Windows CE offers a subset of Win32; and a range of proprietary approaches exists from RTOS vendors, such as Photon (QNX Software Systems Ltd, www.qnx.com), Maui (Microware Systems Corp, www.microware.com), and Poptic (Integrated Systems Inc, www.isi.com). Zinc Software Inc
Embedded-system developers looking for a way to add graphics to their systems have a range of methods to choose from; no absolute prescription for success exists. The traditional C-based graphical-user-interface (GUI) tool kit and associated development tools are strong contenders. Java is gaining acceptance, and smaller memory requirements make it a serious player. These two paradigms present attractive options for compelling solutions of next-generation embedded systems.

An HTML approach breaks away from these programmatic paradigms and builds on the huge presence HTML has in the World Wide Web. This viable approach solves many of the problems of traditional approaches and has the benefit of clearly separating the user interface from the application. Traditional real-time operating systems (RTOSs), designed for mission-critical applications, are powerful tools for building the devices becoming prevalent today. Combining an RTOS with HTML allows you to add powerful, appealing, and visually distinctive GUIs to embedded devices without compromising reliability or performance.

Graphics meets embedded systems
Until recently, graphics design has not been much of an issue for traditional embedded and real-time systems. Medical equipment, printers, and phones, for example, have either no local user interface or an exceedingly primitive one, such as a simple alphanumeric LCD or even just two or three LEDs.

However, this situation is quickly changing. New markets are rapidly emerging, particularly in hand-held and consumer devices, in which ever-cheaper displays with serious graphics capabilities are starting to appear on a range of products. Personal digital assistants (PDAs), car navigation systems, Web phones, set-top systems (using TVs as displays), video-disk players with "true GUI" setup menus—to name a few—are getting smarter. Silicon and display prices are falling. New display technologies, such as light-emitting polymers, may make it possible to have a display on just about any device, even without flat space to attach the display to.

The traditional embedded markets are also changing. Printers, copiers, and test instruments have traditionally had few or no local GUIs. Copiers have relied on a simple LCD panel, and a workstation often remotely manages networked devices. As display prices fall, these systems are acquiring increasingly sophisticated local user interfaces, either as alternatives to remote management or as facilitators of initial setup. High-end imaging devices offer high-resolution displays with GUIs that feature tabbed boxes, buttons, menus, and pop-up dialogs. This activity is all happening in the context of strong growth in the smart-embedded-system market—systems with 32-bit CPUs and firmware with a distinct homegrown or off-the-shelf OS layer.

Consider your GUI choices
As embedded products evolve, developers need to choose a paradigm to use for embedded-GUI development. Historically, designers have based GUIs on C or C++ application-programming interfaces (APIs), which is the case with the X Windows System (available on all Unix systems), Microsoft Windows (Microsoft Corp, www.microsoft.com), and various portable GUI libraries. A few interfaces use more advanced languages, such as Smalltalk and TCL/Tk. In the embedded-system world, however, such standardization is rare. The market is younger, and standardization with the implicit lock to a large application base is not the issue it is in the desktop world.

As a result, a number of distinct but coexisting technologies may serve embedded-systems graphics development. You can place these technologies into three camps. The first camp, traditional 3GL methods based on C and C++, will continue to have a strong presence for compelling reasons, including high performance and reliability. Java represents a second new-application-environment camp, for which a completely new programming environment provides a secure base for GUI development. And finally, new GUI paradigms that break away from the programmatic approach to GUI development are beginning to emerge.

The third generation
Traditional 3GL methods are still going strong. X Windows has gone embedded; Windows CE offers a subset of Win32; and a range of proprietary approaches exists from RTOS vendors, such as Photon (QNX Software Systems Ltd, www.qnx.com), Maui (Microware Systems Corp, www.microware.com), and Poptic (Integrated Systems Inc, www.isi.com). Zinc Software Inc