SWPY002 – January 2002
White Paper

Programming Considerations for Developing Next-Generation Wireless Embedded Applications

By Justin Helmig, Senior Technical Staff - Wireless Software Applications,
Texas Instruments Wireless Terminals Business Unit

Summary

Today's advanced-generation and tomorrow's third-generation (2.5G and 3G) wireless systems are adding not only multimedia capabilities but also applications that are common today in personal digital assistants (PDAs). Software developers who are targeting the new wireless systems often come from a PC background and are not familiar with embedded platforms. These developers need to be aware of the differences they will find in the new hardware and software environment. Important among these are the media transmission rates, memory resources, user interface, power requirements, processing core architectures and development environments. Developers who are aware of these differences can plan and execute their software more effectively as they create multimedia and PDA-like applications for next-generation wireless systems.

The Growth of Wireless Applications

The technology of wireless communications is evolving quickly, and with it the demand for advanced wireless services continues to grow. While the first two generations of wireless handsets provided exclusively voice services, new standards will enable third-generation (3G) wireless networks to transmit audio-video and data as well as voice. Today's advanced wireless networks are already adding data services to voice as an intermediate (2.5G) step toward the full-featured multimedia services of the future. Along with new types of communications, wireless handsets are beginning to offer a greater range of personal applications, such as those currently found in PDAs. The result is a whole new array of mobile communications equipments, ranging from smart handsets to wireless PDAs, notebooks and Web appliances.

Realizing the business potential of the growing wireless market, many software developers are beginning to create applications that can be used on the new types of mobile equipment. To date, though, relatively few applications developers have much experience with wireless systems, or even with non-wireless handheld systems such as PDAs. By far, the largest group of applications programmers has been trained to work with PCs, and they are comfortable with PC hardware, operating systems (OSs) and development environments, as well as with the wired networks that offer nearly ubiquitous communication among PCs. It is from this large pool of experienced PC programmers that development teams for many new wireless applications will be drawn.

Programmers who are accustomed to working only with the relatively mature development technology of PCs and wired networks need to be aware of differences that exist in wireless systems and networks. Wireless transmission rates are slower than wired rates, and portable systems have more stringent constraints in power consumption, available memory and the user interface than PCs do. Wireless systems are built on digital signal processors (DSPs) as well as RISC processors, so developers need to recognize the advantages of each type of processor as they build their programs. Since many of the new types of wireless systems are in their infancy, the applications development environment is less sophisticated, resembling the PC environment of earlier years in many respects. Developers who are aware of these differences can plan and execute their software more effectively as they create next-generation wireless applications.

Narrow Bandwidth

The first big difference is the narrower transmission bandwidth available in the wireless medium today. PC developers are accustomed to relying on wired Ethernet networks with 10 megabits per second (Mbps) of data transfer. Wired telecommunications tend to be slower, since they rely on transfer rates as low as those of analog modems, with 56 kilobits per second (kbps) upstream and 33 kbps downstream. This picture is changing, though, as virtual private networks (VPNs) bring Ethernet speeds to interoffice networks, and as digital subscriber line (DSL) and cable modems open up megabit broadband transmission rates to private users.

By contrast, the digital wireless networks that are in place today, which were designed for voice services, offer about 15 to 20 kbps per channel, with the rate varying somewhat among transmission standards. Newer 2.5G schemes multiply this rate a few times, so that the end result is comparable to an analog modem or faster in bandwidth. When 3G wireless networks are eventually deployed, transmission speeds under optimum conditions will be comparable to broadband. The ITU/UMTS definition of 3G is 384 Kbps for mobile communications, with 2 Mbps for low mobility and fixed wireless. These are best-case numbers, however, since actual transfer rates will vary from connection to connection, depending on factors such as signal noise and strength, environmental interference, and overall saturation of the waveband by callers in a given area. 3G networks will also support 2G and 2.5G rates for lower-speed communications.

The wireless transfer rate factors into applications development because developers need to think in terms of which type of network to target. In the long run, 3G networks will offer the most potential for video, imaging and other high-bandwidth transfers. However, 2G infrastructure is in place today, and 2G speeds will continue to reach the largest group of subscribers for some time to come. 2.5G falls in the middle of this calculation in terms of transmission rates and time of availability.

In addition, developers need to think in terms of whether their applications can operate with different feature sets and image quality, depending on the bandwidth available. Bandwidth will vary not only on the type of transmission available (2G, 2.5G or 3G), but also on the network conditions at a given time. If the application can continue to operate under varying transmission conditions, it will be more widely useful than if it always requires a high-bandwidth connection.

System Resources

Wireless bandwidths must be considered together with system resources in making decisions about client-server applications. With all networks, developers must determine where the processing will take place--in the local system or in the network server. Factors that enter into this determination not only include the transfer rate, but also the amount of performance available in the local system. In the case of mobile, battery-operated systems, the power consumption required by a program is an issue, and so is the available memory.

All of these factors make a difference in determining how much of the processing should be performed in the handset and how much can be offloaded to the network. Generally speaking, if the application produces a lot of data, especially in real time, the goal will be to reduce the need for data transfer through compression and decompression. On the other hand, if the application is computation-intensive but involves relatively little data, the goal will be to offload as much processing to the network as possible.

Different types of operations lend themselves to each approach. For instance, searching a large data base for information would produce relatively little data for the amount of processing involved. This application, which is limited by processing rather than by bandwidth, is clearly better performed on a server. On the other hand, decoding an MP3 file is more appropriate for handset processing. Bandwidth is the constraint in this case, and the goal is to minimize the amount of data that has to be transmitted.

Memory Constraints

Within the system itself, an important constraint for programmers is the amount of available memory. Whereas PCs today have gigabytes of program storage and virtual swapped memory, wireless handsets typically have 16 to 32 Megabytes (MB) that is shared between stored and active program memory. This memory is not readily expanded by upgrading, and it cannot be virtually enlarged, since hard disk drives are not a component of most hand-held wireless systems.

Memory constraints make it essential that applications programmers minimize program and data space requirements by optimizing software and removing any unnecessary features. Optimization may require a more granular approach than PC developers are accustomed to, with a line-by-line analysis of the source code to examine how compact it can be made, as well as how efficiently it executes. Programs may need to be designed modularly, so that more routines operate at the server end than with PCs, or so that individual program features can be downloaded through the network to the handset only when they are required during the course of a session.

Programmers who are accustomed to the transparency of memory utilization in PCs and workstations should realize that memory management in embedded OSes is not as sophisticated. Since there is no virtual swap space on hard disk drives, dynamic memory allocations must be monitored carefully to avoid running out of memory. In addition, some

embedded OSes will not clean up all allocated memory when exiting processes. Therefore, the application should not only eliminate unnecessary memory allocations, but it should also free all memory allocations on exiting processes to prevent memory leaks.

Among programming techniques, developers should beware of recursive functions and other procedures that push stacks to large sizes. When applications must use a function that calls itself, the function should not be deeply nested. Similarly, copying large objects or passing them by value should be avoided whenever possible in favor of using pointers or passing the objects by reference.

Display Limitations

Anyone who has looked even briefly at a handheld system realizes that the display is smaller and has a lower resolution than that of a PC. While 1024 x 768 pixels is a common resolution on PCs, mobile devices typically have screens with resolutions of 240 x 320 pixels or less. Within this small space, wireless OSs do not normally support multiple windows, though dialog boxes for input, messages and so forth may be available.

The limitations of handheld displays seem obvious, yet they have profound implications when it comes to designing the "look and feel" of the application. Developers must take care to eliminate unnecessary data from the screen in order to present a simple, intuitive interface that best uses the available space. Often the appeal of an application in a larger system lies in taking advantage of the extensive capabilities of the display and system graphics. In a handheld system, with its small, low-resolution screen and simple graphics, the application will have to be more limited in its video output. Here the challenge for the software developer is to take less and make the most of it to create a satisfactory visual experience for the user.

Power Conservation

In mobile systems, power consumption is an overriding concern, so developers should be aware of and use low-power system features that are available to them. Wireless OSs typically provide power management features that allow for the partial shutdown of the system when there are idle cycles. Therefore, it is important for the application to return control to the OS when it is waiting for a system resource. For example, if the application needs input from a button on the keyboard, it should create an event, then wait for the OS to inform it that the event has occurred. Doing so eliminates so-called "busy waiting," when the application does not return control to the OS while it is idling, thus saving power and enabling longer use of the system between battery charges.

Other power-saving measures that programmers should employ include using memory efficiently and eliminating unnecessary processing steps and data transmission. Even when the handset has plenty of performance available, it may still help conserve power to offload some tasks to the network instead of performing them locally.

Single—Versus Dual-processor Platforms

The wireless system may be based on a single-processor or dual-processor platform, with the better solution usually determined by the dominant system application. A single microcontroller provides the performance needed for applications normally found in PDAs, but it is inadequate by itself for handling streaming video and other multimedia applications. Adding a DSP for handling the mathematically intensive algorithms involved in multimedia not only increases real-time performance and response, but it also saves power and allows the microcontroller to operate more efficiently on system-level tasks.

System developers must consider a variety of issues in choosing a platform. Since a PDA-only design may need to grow to include multimedia functionality, a single-core platform and its software architecture should support rescaling for dual-core development. The software architecture should be designed to simplify code partitioning between the cores for greater performance and power efficiency, and it should make the underlying hardware as transparent as possible for applications programmers. With well-designed software architecture, appropriate tools and a wide selection of off-the-shelf multimedia modules available, wireless developers can enjoy the performance and power advantages of a dual-processor platform with the straightforward development approach of a single-processor platform.

For application programmers, the key concern is optimizing the software to take advantage of the architecture. In a dual-core platform, the developer needs to decide which portions of the application run better on the DSP, and which run better on the microprocessor. Well-balanced software architecture enables the most efficient use of the system, and the most satisfying experience for the user.

Development Environment

The wireless development environment is different from that of PCs. Embedded OSs offer fewer application programming interfaces (APIs) than PCs do, and the APIs in some OSs will feel unfamiliar to those who have programmed only PCs. OSs that use a subset of PC APIs can reduce the learning curve and make it easier to port software, but programmers need to be aware that not all PC functionality is supported in mobile systems. The various OSs available each have different advantages, but in all cases the functionality of mobile systems is more limited than that of PCs.

Although development tools for mobile systems continue to improve, they typically do not have all the features of tools designed for a PC host. As a result, applications developers have to be prepared to improvise, and they need to be aware of the impact of their programs on the system as a whole. They are also faced with a choice that is unfamiliar to many PC programmers: whether to debug the application on a PC simulator, or remotely on the target embedded system itself. The support situation is changing rapidly as the wireless embedded systems and applications evolve, but for now PC developers are faced with a challenging environment as they turn their efforts to creating wireless applications.

Considerations In Developing Wireless Applications

The growing market for wireless multimedia and PDA-type systems makes it inevitable that applications developers will cross over from PCs to the new types of embedded systems. The following list summarizes many of the issues that PC programmers need to consider as they turn for the first time to applications development for wireless systems.

1. Remember that data bandwidth is narrow compared with wired networks.
2. Determine how much of the processing will be performed on the handset and how much can be offloaded to the network.
3. Keep code and program data requirements small, since memory resources are limited.
4. Carefully track dynamic memory allocation for the best use of resources and to avoid memory leaks.
5. Beware of recursive functions and other procedures that push stacks to large sizes.
6. Design for a small, low-resolution screen with a single window.
7. Return control to the OS to keep power consumption low.
8. Partition code to gain maximum performance from dual-processor hardware.
9. Make do with fewer APIs in wireless OSs.
10. Be prepared to improvise, since there are fewer support tools for embedded systems than PCs.