ISSN 0005–1144
ATKAAF 43(3–4), 168/I–VI (2002)

Symbian Operating System for Mobile Phones

New mobile devices require powerful hardware and software to support customer's needs which grew rapidly in the last few years. Demanding user interface depends of the operating system, and Symbian created operating system that seems powerful enough to support forthcoming burst of data services in the mobile world.

Symbian OS is the common core of application programming interfaces (APIs) and technology that is shared by all Symbian OS phones. Symbian OS includes a multi-tasking kernel, middleware for communications, data management and graphics, the lower levels of the GUI framework, and application engines.

Key words: Symbian, operating system, mobile phones, multi-tasking, Bluetooth, Java MIDP, GPRS, ARM RISC processor

1 INTRODUCTION

Mobile phone networks are beginning to offer affordable and reliable data services in addition to their more traditional voice services. Supporting these data services is challenging for the mobile phone handset vendors. They have to provide timely production of feature-rich, innovative, and fashionable handsets to the mass market at reasonable prices. Symbian OS is the proven advanced data-enabled operating system for mobile phones and is structured to ease the integration of hardware and software.

Unlike PC design, mobile phone design puts constraints on a suitable operating system similar to those of advanced PDAs and more. The operating system has to have low memory footprint and low dynamic memory usage, an efficient power management framework, and real-time support for communication and telephony protocols. Furthermore, users often have a more cavalier attitude to mobile phones than to PCs. For instance, when removing the battery while the phone is still switched on a user still expects device and data integrity.

Symbian OS is designed for the mobile phone environment. It addresses constraints of mobile phones by providing a framework to handle low memory situations, a power management model, and a rich software layer implementing industry standards for communications, telephony and data rendering. Even with these abundant features, Symbian OS puts no constraints on the integration of other peripheral hardware. This flexibility allows handset manufacturers to pursue innovative and original designs.

This article focuses on several key aspects of Symbian OS that make it the optimal choice for a mobile phone platform. It covers:

- typical mobile phone platforms and the requirements of Symbian OS,
- Symbian OS software architecture,
- hardware integration methods,
- porting and customization for mobile phone handset manufacturers,
- an open architecture for third party developers.

2 MOBILE PHONE CORE PLATFORM

Central to data-enabled mobile phones is a fast, low power, low cost CPU core, which has compact code and can be highly integrated with peripherals. A system-on-chip (SoC) contains the CPU core and vital peripherals for the functioning of the phone operating system. The family of ARM architecture RISC processors, which have been incorporated with core peripherals into a standard package, is particularly suited for mobile phones. The system-on-chip is then placed on a PCB with the remaining peripherals to produce a phone.

Symbian OS is a 32-bit, little-endian operating system. It has been ported to many flavors of ARM architecture chips with V4 instruction set or higher. Further requirements of Symbian OS are for the CPU to have an integrated memory management unit (MMU) and a cache, to operate in various privileged access modes, and to handle interrupts and exceptions. The CPU, MMU and cache along with timers and hardware drivers, all reside on the system-on-chip. These SoCs are often commercially available and are sometimes custom built by handset manufactures.

Symbian OS has been ported to many ARM cored system-on-chips. These include the PrimeXSys platform from ARM, the StrongARM and XScale architectures from Intel, the OMAP platform from

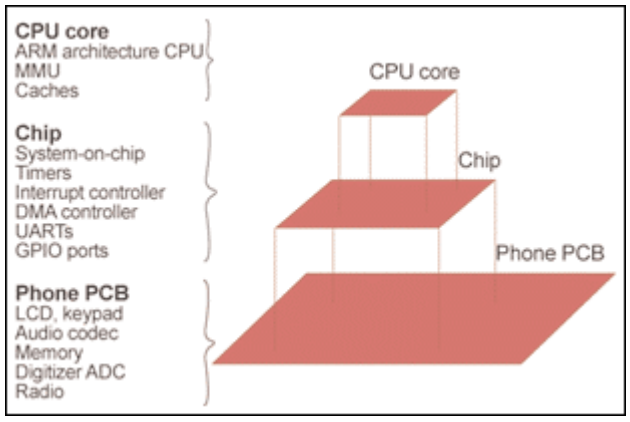

Fig. 1 Mobile phone hardware can be divided into three logical layers: the CPU core, the SoC and the PCB. Symbian OS also conforms to this layering. This enables easy porting of Symbian OS as the code for particular CPU core or SoC can be reused in many products

Texas Instruments and the Dragonball platform from Motorola.

The MMU is used for several purposes. It protects process data from access by other processes, enforces protection of application and kernel code, and isolates the hardware from application code. The MMU is a crucial component in the design of the protected mode system, which enhances both the security and stability of the platform. A standard two-level page tabled MMU allows small 4 KB pages for efficient memory usage, while fast re-mapping speeds can be achieved with large first-level pages of 1 MB. Both data and instruction caches are required to produce acceptable performance. On-chip timers provide the real-time clock for the system tick timer, and millisecond scale timers are needed for use with hardware drivers.

While some memory will be available on the chip most will be provided off-chip. This off-chip memory has three major functions: storage of the Symbian OS image; persistence of user data in a file system; and storage of processes' data at runtime. Speed of memory access, cost of the memory chips and persistence of the data must be considered when choosing the memory for each of these three functions.

One option is for the operating system image to be located in ROM. ROM is cheap but cannot be reprogrammed, any change to the software at a later date would have to be performed by providing a patch in another persistent storage and inserting this into the image by reorganizing the MMU. Flash is more versatile than ROM allowing the image to be changed, but the cost is greater. Images can also be placed in RAM and marked as read-only by the MMU, however this doubles the amount of memory needed and increases the time to boot, as a bootloader has to copy the image from ROM to RAM at boot time. Flash is the most commonly adopted solution for mobile phones.

3 SYMBIAN OS IMAGE

The Symbian OS image is a compact collection of executable code and various data files. The image consists mainly of dynamically linked libraries (DLLs) and other required data, including configuration files, bitmaps, fonts and other file-resident resources.

The image code is execute-in-place. However, where speed is critical libraries in the image can be marked to be executed from RAM and hence will be copied there before being used. As libraries are expected to be resident in non-writeable memory, they must contain no writeable static data. Symbian OS does, however, provide a mechanism for DLLs to store a small amount of data associated with a particular thread.

By having almost all the code as DLLs, there is only a single copy of each library required regardless of the number of applications linked to it. To keep the size of the libraries down, application code is linked to DLLs by the ordinal position of the functions within the DLL. It is key for library developers to maintain the order of functionality within the DLL to maintain the binary compatibility of other software linked to that DLL.

Symbian OS makes extensive use of a specific type of DLL, called a polymorphic DLL, which has a known exported function at the first ordinal position. This exported function must return a class of well-known type (or a derivation thereof). Polymorphic DLLs act very much like factory classes and have the benefit of providing an interface that file-based plug-in systems can use. This type of DLLs is used throughout the operating system. One examples of a polymorphic DLL is the application DLL, which exports a single function called NewApplication(), to create an instance of an application. In the Symbian OS device driver model, devices are created by calling the CreateLogicalDevice(), which in turn calls a polymorphic device driver DLL.

All executables within the image are assigned unique identifiers (UIDs) that serve as a form of identification to associate files with their owning application and to protect against loading an incorrect type or version of a DLL.

4 HARDWARE INTERFACES

The Symbian OS kernel is a compact pre-emptive multitasking operating system with very little

dependence on peripherals. The core kernel executable – of less than 200 KB – fully supports the multi-threaded operating system. Peripheral hardware integration is added to the kernel in several ways. Hardware support is usually implemented in separate DLLs associated with particular hardware to allow the easy insertion and removal of hardware and to facilitate code reuse.

The MMU is configured so that all hardware registers can only be accessed in privileged mode. The kernel always executes in privileged mode and hence has access to all the hardware registers. Applications interface to kernel services through an API provided by the User library. Because all applications run in unprivileged mode, operations that require hardware access must either switch momentarily into privileged mode while maintaining the context of the application or issue a request to the kernel server, which involves a switch in context to that of the kernel process.

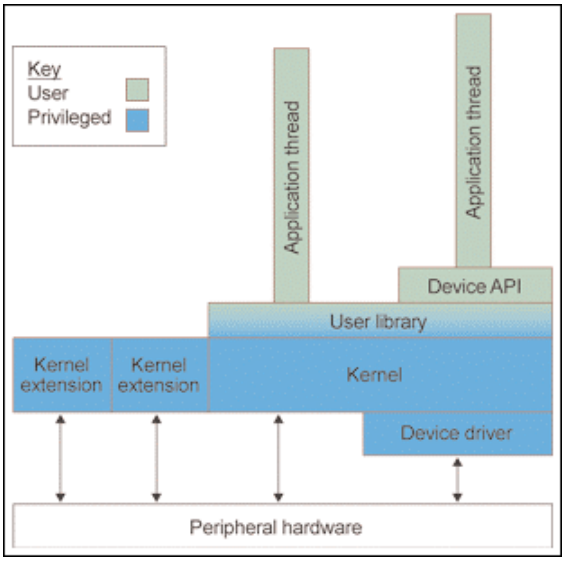

Fig. 2 All access to hardware occurs from, or through the kernel. There are several software frameworks that support hardware access, direct kernel access for vital hardware, kernel extensions for hardware associated with user input, and device drivers for further peripheral hardware exposed to applications and server

The kernel library includes support for all peripheral hardware that is resident on the chip (e.g., the ASIC or SoC) and that is essential to the operating system. The peripheral hardware includes such things as timers, DMA engines, interrupt controllers and UART serial ports. The kernel library is customized for a particular chip. Applications are not permitted to access peripheral hardware directly. Instead applications must link to the User library whose functions may invoke peripheral control through the kernel.

Peripherals associated with user input can be packaged as a separate DLL, called a kernel extension. User input simply provides events that are consumed by the kernel. Different kernel extensions can be written for keyboard, keypad, digitizer, and navigation button and wheels. The appropriate kernel extensions are added into the image, where the kernel detects their presence at boot time and initializes them. The kernel itself has no dependency on the extensions, and no kernel extension functionality is accessible to applications.

Device drivers expose an API to applications to allow control of hardware that is not essential to running the operating system. Device drivers can be loaded and unloaded at anytime. A device driver consists of two parts: a library providing the device's API to which applications can link; and one or two libraries, running in privileged mode, kernel-side, to access the hardware.

The kernel side library is often split into two libraries: a logical (LDD) and a physical (PDD) device driver DLL. The LDD encapsulates the logical functions of a device e.g., on and off, and read and write. The PDD carries out the functions on a specific device. The LDD contains all the complexity of typical device usage usually in the form of a state machine. If the choice of hardware part is changed in prototyping a phone or the progression of a product line, only the PDD needs to be replaced. For example, the media server uses a standard application-side API and a logical device driver so only a physical device driver has to be provided for a particular codec chosen for a phone.

Both logical and physical kernel-side device drivers are polymorphic DLLs. The device drivers must be loaded by the application-side before access to the hardware can be obtained. The application-side library uses a message passing mechanism to communicate to the hardware libraries through the kernel. The devices can be used synchronously or asynchronously, though asynchronous use is preferred, wherever possible, as it is more CPU efficient.

Support for file system media is also provided through a device driver. A file system drive consists of two components, a file system and a media driver. The file system is typically FAT but not necessarily if the system is internal to the phone and would benefit from a different format. The media driver is a physical device driver performing all the functions that the file server expects. By constructing drives with these two components, the file server conforms to a plug-in model where new media drivers and file systems can be plugged into the OS without affecting the core code in the file server. If the need arises for a ROM patch, device drivers

are used to implement the patch because they have access to the MMU, which has been protected by the kernel.

The only exception to this model of peripheral access controlled by the kernel is the screen buffer, which is copied via DMA to the LCD display. The screen buffer is usually given read/write permissions to all threads, for applications to provide fast drawing routines through a graphics API. This increases speed as no switch to privileged mode and back is required.

5 SYMBIAN OS SYSTEM SOFTVARE

Symbian OS contains an extensive and rich collection of libraries to implement many industry standards. This layer of system software, in Version 6, includes support for networking (TCP/IP, PPP, TSL, SSL, IPSec, FTP), communications (Bluetooth, IrDA, Obex), Security (DES, RSA, DSA, DH), messaging (POP3, IMAP4, SMTP, SMS, BIO), browsing (HTML, HTTPS, WAP, WML), telephony (GSM, GPRS, fax), graphics, multimedia (WAV, AU, WVE, JPEG, BMP, MBM, GIF) and many more.

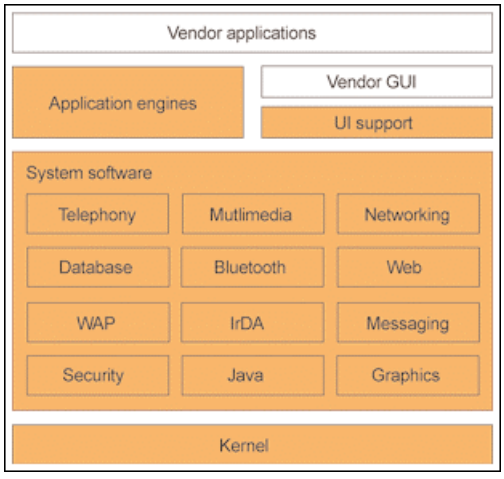

Fig. 3 The rich layer of system software for industry standard support is part of Symbian OS

Access to these services and resources is coordinated through a standard client-server framework. Servers run as unprivileged threads. Any application thread can be a client connecting to a server by name and passing messages through a standard interface imposed by the kernel. The framework is constructed through inheritance from server and session classes. The kernel support for the client--server framework is optimized for low memory use and speed; it also keeps a record of objects in the system such that any thread death results in all the memory being recovered.

The uses of the client-server architecture in Symbian OS include the file server, media server, telephony server and many more. The media server is a good example of a hardware resource, accessed through a device driver, which can be synchronized through the client-server framework.

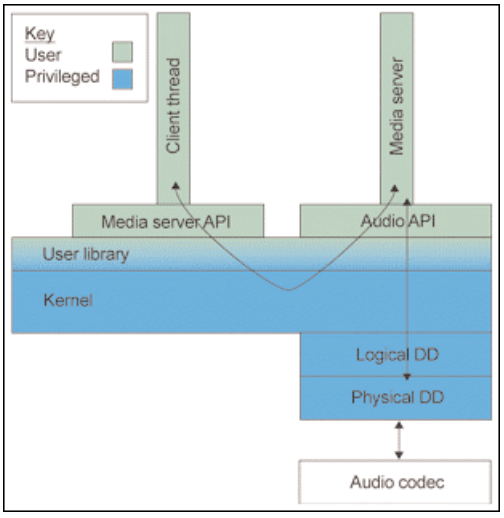

Fig. 4 Most standards are implemented as system servers. The servers can make use of device drivers to access the hardware required for particular services. For example the multimedia server can be used to play standard audio clips

The top layer of Symbian OS provides support for applications. This includes application engines for common Symbian OS phone applications: contacts (an address book), agenda (a diary application), and jotter (a document producing application). This layer also includes skeleton support for graphical user interface (GUI) components, however handset manufacturers provide the actual user interface. This allows the phones to maintain the unique look, feel and branding of each manufacturer.

6 APPLICATION SUPPORT

Symbian OS is fully multi-tasking. To ensure the system to run in an efficient and secure manner, some properties are imposed on applications. All applications run in a virtual machine (VM) environment. One benefit is that software can be fixed--up (built with predefined run-time memory configuration) such that the environment is exactly as expected. This saves memory, because the library does not need to have any relocation information associated with it. Also if there are two copies of the application running they can execute the same code.

The VM is made possible by use of the MMU to move data around in the virtual address space. An application consists of a single process, the unit of memory protection, in which one or more threads

are running. When an application is loaded from its polymorphic DLL, the application is given pages for the process data, and the thread data in the outer page table of the two-level MMU. When a context switch occurs to this process, the kernel adjusts the MMU configuration by moving all the pages to a pre-defined location in the virtual memory map. Execution continues in the appropriate thread.

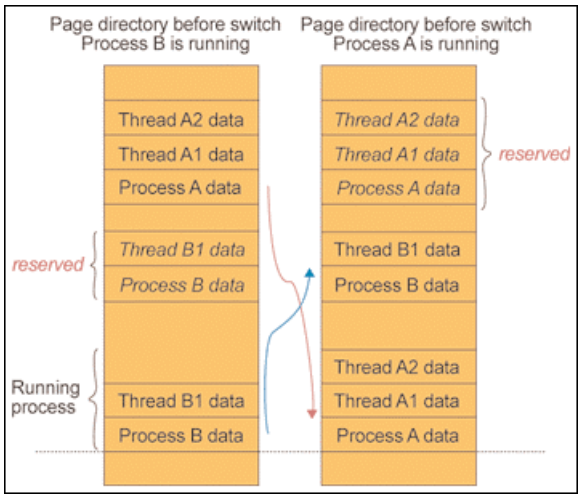

Fig. 5 The MMU is used for context switching allowing all processes to assume their data is resident at the same place. Here a context switch is made between process B and process A, the memory is moved by modifying the outer page table of the two-stage MMU

If the data cache is virtually tagged, then the cache must be flushed every time a context switch takes place. This introduces an inefficiency for applications, however for intensively used system server processes a unique data area can be assigned to them if they are only single copy. The file server is such an example: if an application requests a service from the file server, the context switch to the file server does not involve any change in the MMU or a cache flush and if the original application is returned to the data in the cache is still valid. Physically tagged caches do not have this complication.

7 OPEN FOR INNOVATION

The ability for independent software vendors (ISVs) to develop applications for mobile phones is already bringing new uses to these small, connected, mobile platforms. Keeping the platform open enables ISVs to focus on the expanding market of mobile phone applications.

ISVs can develop Symbian OS software in C++ using a Windows emulator that runs on a PC and maps Symbian OS calls to Win32 APIs. The mobile phone manufacturers ship this emulator as part of a Symbian OS software development kit (SDK). Any application or library written with an SDK links to the User library for kernel services and to other core libraries for core system services. Symbian OS features a Java virtual machine and supports applications written in Java.

Symbian OS also provides a framework designed for programming when memory capacity is limited. This framework consists of a clean-up stack onto which any partially constructed objects are placed until their construction has been completed. If the phone runs out of memory it is possible to delete the objects on this clean-up stack avoiding a memory leak and enabling the process to continue, without potentially losing user data associated with this process. This paradigm is used throughout Symbian OS and exposed to third-party (in the SDKs) to enable safe low memory programming.

Third-party software is installed on Symbian OS phones with an installation file (SIS file). This file is a concatenation of all the libraries and resources that are required by the application, which is then made secure with a certification system. This enables the delivery of tamperproof files where the software vendor can be identified. The installer populates the file system with the files from the SIS file allowing the enclosed application to run.

8 SYMBIAN OS BLUETOOTH ARCHITECTURE

The Bluetooth profiles were the key initial drivers for Symbian OS Bluetooth implementation. Symbian OS Version 6.1 Bluetooth stack is designed around supporting the Generic Access Profile, the Serial Port Profile and the Generic Object Exchange Profile. Key design goals included:

- The ability to run multiple simultaneous logical connections to multiple remote devices.
- The separation of the protocol stack from the security policy as suggested by the Bluetooth Security White Paper.
- The ability to support any specification compliant hardware through the Host Controller Interface.
- From the outset use case analysis was used to map required Bluetooth profile support to system features.

In the Symbian OS Bluetooth architecture, core stack functionality is implemented by two components, HCI.DLL and the Bluetooth protocol module (BT.PRT). The Host Controller Interface module encapsulates the canonical set of HCI commands and events. Currently the serial UART flavor of HCI has been implemented using the Ericsson

Bluetooth Development Kit, the DigiAnswer PCMCIA cards and most recently the Cambridge Silicon Radio Casira modules as reference hardware. Symbian is, however, able to support any HCI-compliant Bluetooth hardware through a modular Host Controller Transport Layer architecture.

BT.PRT encapsulates the Bluetooth L2CAP and RFComm layers. As a Symbian OS protocol module, it provides a sockets API to these protocols. BT.PRT furthermore spawns distinct Bluetooth Manager and Service Discovery Protocol (SDP) server threads. The Bluetooth Manager abstracts all User Interface interactions and access to non-volatile storage. This allows for a future implementation of flexible access policies to the range of services supported by a Symbian OS phone. The SDP server handles SDP queries and responses. Serial port emulation is supported by the Bluetooth comm server module (BTCOMM.CSY) module which provides a number of thin virtual serial ports for different legacy services running over RFCOMM socket functionality.

Both Symbian OS Bluetooth protocol module and comm server module are based around the usage of the State pattern.

8.1 Development evolution

Symbian OS Bluetooth will evolve as follows:

- Pre-Qualification against the v1.1 Bluetooth Specification.
- Symbian OS OBEX server implementation to replace existing static library.
- Bluetooth 2.0 support through the addition of an ad-hoc networking interface (IP over Bluetooth Network Encapsulation Protocol (BNEP) over L2CAP).
- Additional profile support.

The existing framework components will be enhanced to implement this support but the basic stack APIs will not change. The addition of ad-hoc networking via Bluetooth is a particularly exciting area as it represents a genuine paradigm shift in mobile computing, ushering in a new range of services. Symbian OS is well-positioned to handle the transition to this world of Personal Area Networking and offers mobile phone maunufacturers unique advantages. Support for Bluetooth has been designed into the core platform from the outset.

9 CONCLUSION

Symbian OS is a robust multi-tasking operating system, designed specifically for real-world wireless environments and the constraints of mobile phones (including limited amount of memory). Symbian OS is natively IP-based, with fully integrated communications and messaging. It supports all the leading industry standards that will be essential for this generation of data-enabled mobile phones. Symbian OS enables a large community of developers. The open platform allows the installation of third party software to further enhance the platform.

REFERENCES

[1] ..., Forum Nokia white papers (http://www.forum.nokia.hr)

[2] ..., Symbian.com (http://www.symbian.com)

[3] ..., The Bluetooth Security White Paper on the Bluetooth SIG Members website (http://www.opengroup.org/bluetooth)

[4] ..., Design Patterns (1st Ed.); Gamma et al.; Addison-Wesley Pub Co; ISBN: 0201633612; 15 January 1995

[5] ..., Sun Microsystems (http://java.sun.com)

[6] ..., WAP Forum (http://www.wapforum.org/what/technical.htm)

[7] ..., Open Mobile Alliance (http://www.openmobilealliance.org/)

[8] ..., 3GPP official website (http://www.3gpp.org/)

Symbian operativni sustav za mobilne telefone. Novi mobilni uređaji moraju imati napredan hardver i softver da bi mogli ispuniti korisničke zahtjeve koji su znatno porasli u posljednjih nekoliko godina. Korisničko sučelje ovisi o operativnom sustavu, a Symbian je stvorio operativni sustav koji je dovoljno snažan da može podržati porast podatkovnih usluga u mobilnom svijetu.

Symbian OS je zajednička jezgra za aplikacijska sučelja (API – application programming interface) i tehnologiju koju dijele svi Symbian OS telefoni. Symbian OS uključuje multitasking kernel, komunikacijske i podatkovne protokole, grafičko sučelje i aplikacijske strojeve.

Ključne riječi: Symbian, operativni sustav, mobilni telefoni, multitasking, Bluetooth, Java MIDP, GPRS, ARM RISC procesor

AUTHORS ADDRESS:

Željko Pavlaković
Mobis electronic d.o.o.,
Heinzelova 96, 10000 Zagreb, Croatia
zeljko.pavlakovic@mobis.hr