Platform Leadership through System Interfaces: A Study of Application Programming Interfaces for Mobile Operating Systems

by

Ashok Chakravarthy Mandala

Bachelor of Engineering, Computer Science
Birla Institute of Technology and Science, Pilani, India

Submitted to the System Design and Management Program
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

at the

Massachusetts Institute of Technology

February 2007

Signature of Author______________________________

Ashok Mandala
System Design and Management Program
February 2007

Certified by______________________________

Michael A. Cusumano
Thesis Supervisor
Sloan School of Management

Certified by______________________________

Patrick Hale
Director
System Design and Management Program

Platform Leadership through System Interfaces: A Study of Application Programming Interfaces for Mobile Operating Systems

By

Ashok C. Mandala

Submitted to the System Design and Management Program on January 19, 2007 in Partial Fulfillment of the Requirements for the Degree of Master of Science in Engineering and Management

ABSTRACT

The Smart Mobile device industry is witnessing rapid growth with the increased convergence of voice-centric mobile phones and data-centric personal digital assistant systems. Improving capabilities in device hardware have allowed development of complex user interfaces, multimedia and communication capabilities on these devices. Modern Mobile Operating Systems manage this complexity in the mobile device by administering hardware resources and providing a platform for development of new consumer and enterprise applications. This thesis studies the architecture, design goals and services offered by the three major mobile operating systems – Palm OS, Symbian OS and Windows Mobile.

The Mobile Operating Systems studied in this thesis differ in their architectures, services and programming interfaces offered to application software developers, independent hardware vendors and OEM licensees. Their design reflects the OS vendor's strategy toward the mobile platform which is decipherable based on a study of the OS architecture and application programming interface. Three conclusions are made based on this study – each of them suggests a strategy that the vendor has attempted to use to gain platform leadership through product architecture and degree of openness of interfaces.

Thesis Supervisor: Michael A. Cusumano
Title: Sloan Management Review Distinguished Professor of Management, Technological Innovation & Entrepreneurship.

Table of Contents

List of Figures

Chapter 1

1.1 Introduction

Many of today's products are complex – consisting of multiple independent pieces interfacing and interacting with each other to achieve the product function. The independence of the components allows suppliers of those components to innovate independently of the others. For participants of a product system wherein there is no single vertically integrated manufacturer, innovation cannot happen in isolation. Innovation in such systems requires an architectural vision and strategy to carry together innovations in each of the subsystems for a coherent evolution of the product platform.

Organizations strive to drive innovation in their industry in a direction that is more beneficial to them than their competition i.e. they aspire to become Platform leaders. A four-lever framework to design a strategy for achieving platform leadership has been proposed by Cusumano and Gawer[1]. The second lever in this framework is Product technology and it signifies the degree of openness of a product's architecture, interface and intellectual property that a Company is ready to reveal to other component makers of the Platform. Open interfaces spur the development of complementary innovative products, Open architectures allow designers of the complementary products to take advantage of the architecture to ensure optimal platform performance.

Mobile Operating Systems are interesting because of the huge growth of the mobile device market, primarily in smartphones and the variety of devices being innovated. The operating system dictates the user experience on a mobile device, its versatility in terms of availability of third party applications and stability and robustness of the device. This thesis will examine the architecture and programming interfaces to three major mobile

[1] Platform Leadership: How Intel, Microsoft and Cisco drive Industry Innovation, 2002, Cusumano and Gawer

operating systems – The Palm Operating System, Symbian Operating System and the Windows Mobile Operating System.

The Mobile Operating Systems studied in this thesis differ in their architecture, their core design goals, and services offered to users and device manufacturers. Their design reflects the OS vendor's strategy toward the mobile platform and is decipherable based on a study of its architecture and application programming interface

1.2 Organization of the Thesis

This Thesis will be organized on the central theme of the Operating System Architecture and Application Programming Interfaces to the three major handheld operating systems – Palm, Windows and Symbian.

Chapter 2 discusses the general architecture of each of the operating systems consisting of the major modules and their interactions.

Chapter 3 discusses the evolution of the three Operating Systems, tracing the services and features added in each major release. The Symbian and Windows Operating Systems show a gradual evolution built on an original architecture that provides the platform. The Palm Operating System underwent a major architectural change with the release of Palm OS Cobalt signifying its availability on ARM processors.

Chapter 4 analyzes each of the Operating Systems, comparing their architectures and programming interfaces in terms of evolving a strategy to build a mobile Platform.

Chapter 5 closes by making three conclusions that each suggests a strategy for organizations to gain platform leadership through product architecture and degree of openness of interfaces.

Chapter 2

2.1 Palm Operating System – High Level Architecture

This section discusses the architecture of the Palm Operating System based on a study of the developer API documentation in the "Palm OS 68K API Documentation, Volume I and Volume II". The architecture described here is a summarization of the Core Palm OS concepts found in the official documentation. The official Palm OS API documentation provides a more detailed explanation of this architecture. This section uses the same terminology and description as the official Palm OS documentation of the several subsystems and components that make up the Operating System. This documentation is available for download at http://www.access-company.com/developers/documents/palmos/palmos.html.

The Palm Operating system is a 32-bit operating system supporting 8, 16 and 32-bit internal data types and 32-bit addresses allowing for a large address space for storing code and data. It is a single-threaded, event driven operating system where only one application runs at a time[2.] Palm Operating Systems starting with Palm OS v1.0 and ending with Palm OS Garnet have been targeted to run on Motorola 68K processors. These versions of the Operating Systems are single-threaded and have retained the basic architecture since version 1.0. Palm OS Cobalt (version 6.x) is a complete rewrite of the original Palm OS and is designed to run on ARM processors. It comes with a new OS architecture providing multi-threading support. Palm OS Cobalt retains compatibility with existing Palm OS 68K applications by including a run-time environment called Palm Application Compatibility Environment (PACE).

This thesis will discuss the architecture of Palm OS Garnet family (version 5.x) since they have the most installed base and availability of third-party applications. The entry point of a Palm application is the PilotMain function which accepts a "launch code" as

[2] Palm OS Programmer's companion: Volume 1, Chapter 1

one of its parameters. This launch code is a hint to the application from the OS on how it should execute. Different launch codes are issued to wake up the application to respond to an alarm, to query for a search string from a global find or more typically to just start up and display its UI to the user, accept input and perform actions.

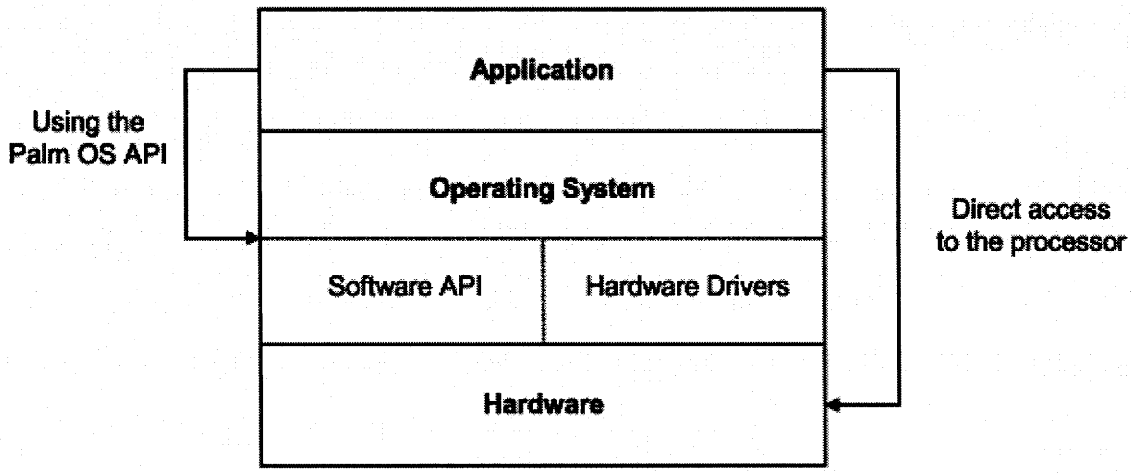


Fig. 1 High Level Architecture of Palm OS[3]

Palm specifies "Palm OS User Interface Guidelines" for application developers to conform to and develop well behaved Palm OS applications that provide a user interface consistent with the rest of the Palm OS built-in applications. The Application Programming Interface to the Operating System consists of multiple "managers" which are groups of functions that implement a particular feature – applications use these managers to perform their tasks. The following sections will explain the architecture of the Palm OS managers and their interactions.

User Interface

The Palm OS represents individual UI elements comprising the User Interface as a resource structure within the Operating System – the resource defines the element's

[3] http://www.usenix.org/events/sec01/full_papers/kingpin/kingpin_html/arch.gif (accessed Oct 2006)

appearance and location on screen[4]. The UI element is a compact C structure and applications can manipulate it programmatically though a reference to the structure. The following are the different Palm OS UI Managers.

Event Manager – Interface between the Operating System and Application executing tasks in response to user actions. The Event Manager is the higher level manager of event operations and uses the following lower level managers to generate events and send them to the application.

Graffiti Manager – handles user input in the form of letters, numbers and symbols from the input area and generates events.

Key Manager – handles user input in the form of hardware button presses and generates events.

Pen Manager – handles user input in the form of user taps on a control in the main display area and generates events.

Keyboard Dialog Manager – displays an on-screen keyboard that will allow user to input text. The dialog is automatically displayed in the context of a text field that requires text input. It can be displayed programmatically too.

Menu Manager – handles taps that display a menu and also taps that select a menu.

Form Manager – handles all drawing operations to render UI objects contained on a form to the screen, dispatches events to that are not already handled by the system and application to event handlers of active forms.

Window Manager – The Form Manager handles drawing of predefined UI controls on the screen, but if any custom drawing is needed e.g. for animation or creation of a custom gadget, the Window manager can be called to handle these drawing operations. The Window manager supports APIs to draw lines, rectangles, characters, bitmaps and pixels. The operations it supports are Draw, Fill, Erase, Invert and Paint. The Window manager provides comprehensive drawing operations including using bitmap patterns for filling shapes. Off-screen windows used for double buffering to smooth animations and reduce flickering are also created using the Window Manager functions.

Alert Manager – functions to display an alert dialog to screen.

[4] Palm OS Programmer's Companion, Volume I

Progress Manager – functions to display a progress dialog and respond to progress events.

System Manager – Collection of Palm OS functions to interact with the core operating system and invoke OS level functionality etc. Launching applications, sending notifications to other applications, displaying the system keyboard, default event handling etc.

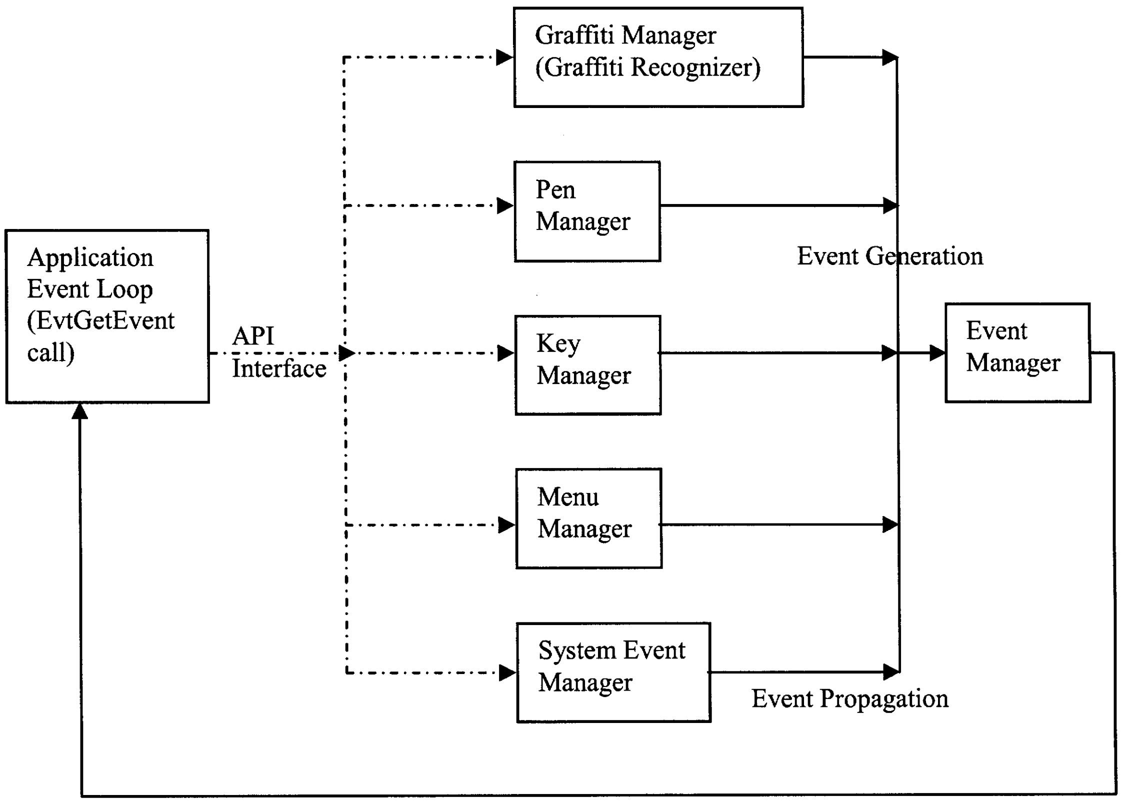


Fig. 2 Event Driven processing of a Palm OS application.

Memory Management

All memory on a Palm OS device resides on a memory module called a "card" – a card is a logical construct and is not the same as a storage/expansion card[5]. A Palm OS card can

[5] Pam OS Programmer's Companion, Volume I

store ROM, RAM or both. All available RAM on a card is divided into a storage heap which allows applications to store non-volatile data (memory region that is not erased on a soft-reset. On a hard-reset of the device, the storage and dynamic RAM regions are reset, but ROM storage is not) and a dynamic heap to store volatile data (RAM memory region that is erased on a soft-reset of the device) like application local & global variables, stack etc. Applications access and manipulate memory through the Memory Manager which allocates variable sized chunks of contiguous memory. The Palm OS memory manager relocates chunks to compact memory to avoid fragmentation. Applications use handles to refer to these re-locatable chunks – they lock a chunk to make it non-moveable and acquire a pointer to manipulate that chunk.

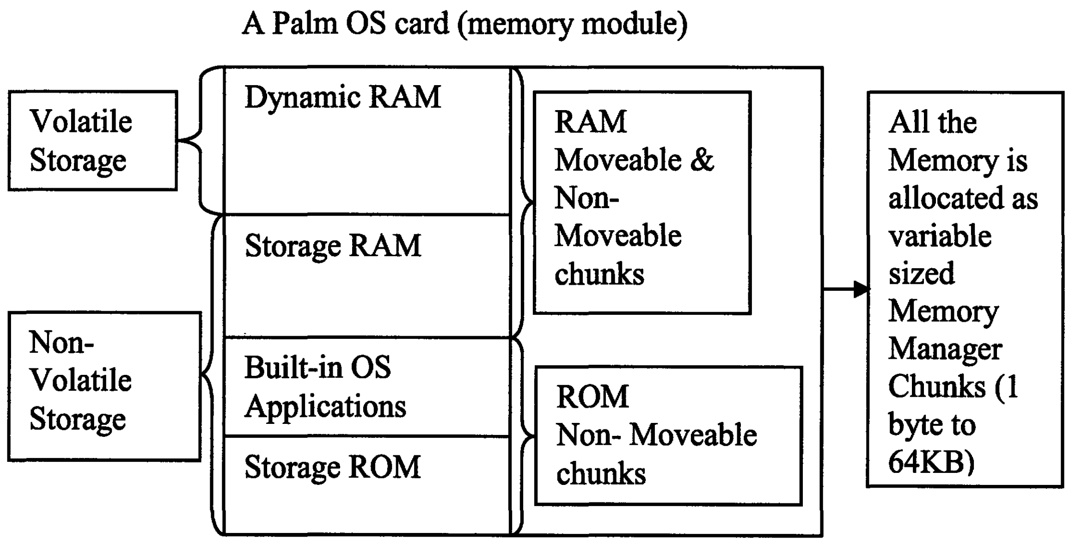

A Palm OS card (memory module)
Volatile Storage
Non-Volatile Storage
Dynamic RAM
Storage RAM
Built-in OS Applications
Storage ROM
RAM Moveable & Non-Moveable chunks
ROM Non- Moveable chunks
All the Memory is allocated as variable sized Memory Manager Chunks (1 byte to 64KB)

Fig. 3 Memory Architecture in Palm OS

The Application uses Memory Manager API for dynamic memory allocations and the Data Manager API for allocations in the storage memory. The Data Manager in turn uses the low-level API calls of the Memory Manager to manipulate storage memory. Unlike a desktop operating system that transfers data from disk to RAM, manipulates data in RAM and then writes back to disk, Palm OS manipulates memory in place. The OS also provides a standard C file streaming interface to the Data Manager which is meant for applications accessing or manipulating large amounts of data. Applications requiring access to secondary storage on expansion cards (SD or CF memory cards etc.) use the VFS (Virtual File System) manager to uniformly access the various file systems that could be present on the different expansion cards. The Expansion manager supports expansion cards by managing the slot drivers for individual slots on a Palm OS device and provides application access to these media by mounting card resident volumes using file system libraries.

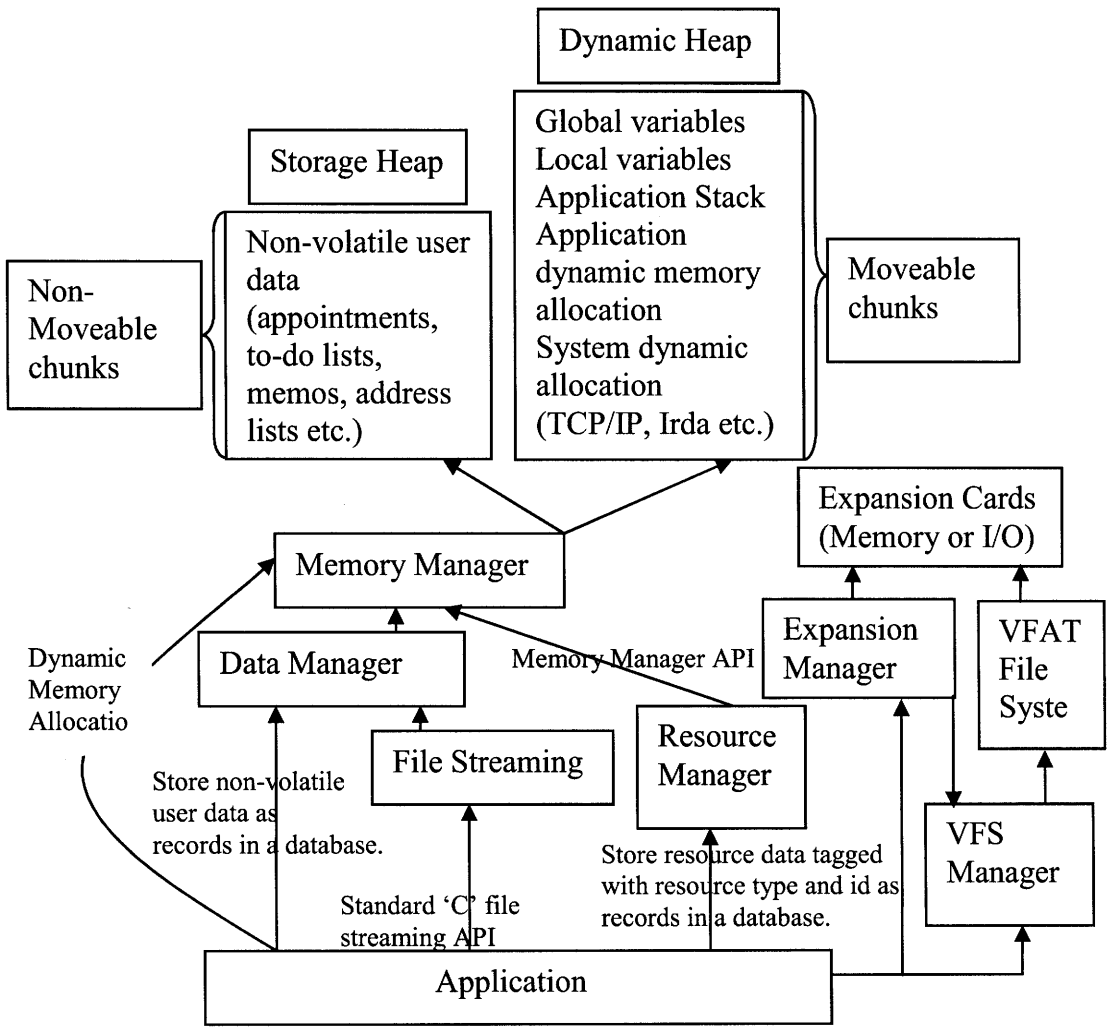

Dynamic Heap
Global variables
Local variables
Application Stack
Application dynamic memory allocation
System dynamic allocation (TCP/IP, Irda etc.)
Moveable chunks
Storage Heap
Non-volatile user data (appointments, to-do lists, memos, address lists etc.)
Non-Moveable chunks
Memory Manager
Expansion Cards (Memory or I/O)
Expansion Manager
VFAT File Syste
Dynamic Memory Allocatio
Data Manager
Memory Manager API
Resource Manager
File Streaming
Store non-volatile user data as records in a database.
Store resource data tagged with resource type and id as records in a database.
Standard 'C' file streaming API
VFS Manager
Application

Fig. 4 Palm OS Application Memory Interface

Object Exchange

The Exchange manager in Palm OS allows applications running on the same or different devices to exchange or communicate typed data with each other. Typed data is any arbitrary data along with header information indicating some information about the data[6]. Applications interact with the exchange manager through a standard API that is transport agnostic. The exchange manager uses different exchange libraries that implement the actual transport like IR, SMS, Bluetooth which the application can choose from.

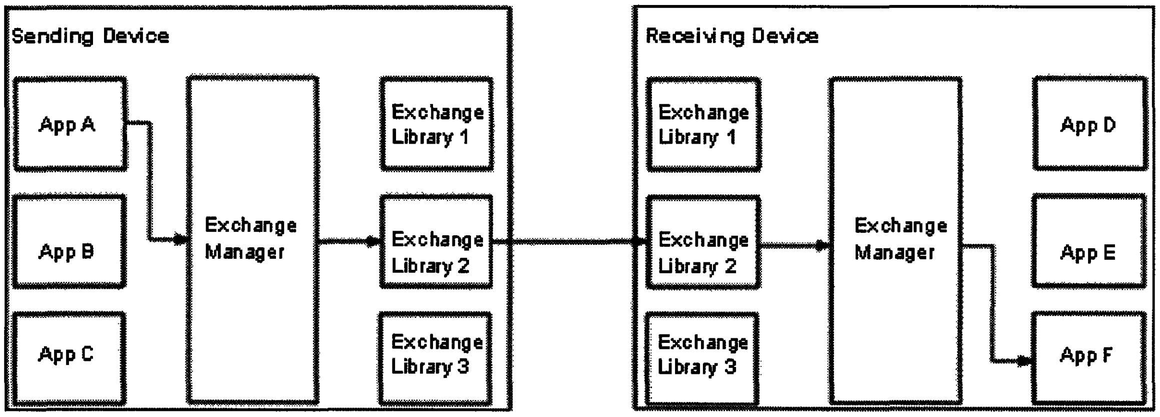


Fig. 5 Exchange Manager Communication Path[6]

Infrared (IR) Library

Palm OS applications use the Exchange Manager to beam and receive information from other devices through IR communications. Palm OS also provides a more direct interface to the IR library for applications that need to access IR directly instead of going through the exchange manager. Applications can also use the Serial Manager to access IR capabilities.

[6] Palm OS 68K API Documentation, Volume II

Serial Communications

The Serial Manager in Palm OS provides the high level API to interface with the serial communications hardware including byte-level serial I/O, packet based I/O with CRC 16, reliable transport with retries and acknowledgements and connection management. The Serial manager is responsible for control of RS-232, IR, Bluetooth and USB signals.

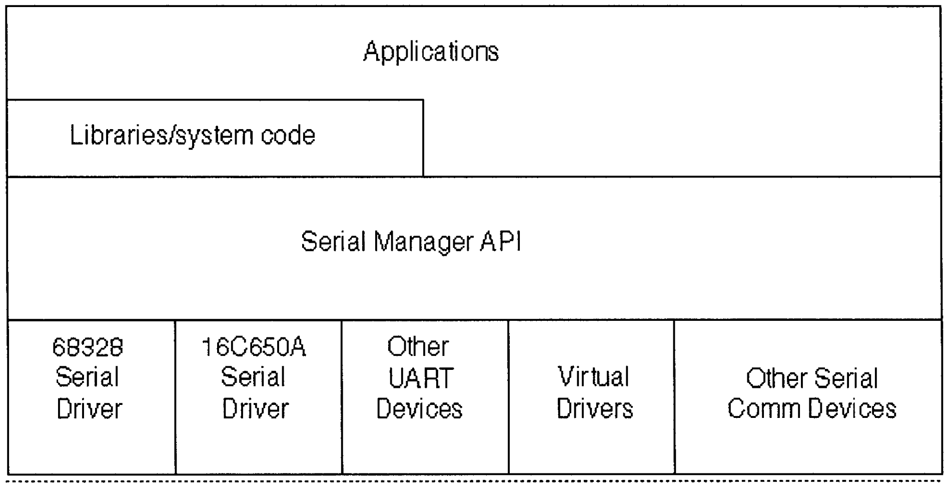


Fig. 6 Serial Communications Architecture[7]

The Connection Manager in Palm OS is used by application to access connection profiles that specify connection preferences for different connection types. A connection profile specifies information such as port, baud rate, flow control etc. to connect with devices such as mobile phones and Bluetooth devices.

Bluetooth

The Bluetooth API provides an interface for applications to use the Palm OS Bluetooth system. The Bluetooth system facilitates device discovery and authentication, serial port emulation and object exchange. Applications use Exchange Manager in conjunction with the Bluetooth exchange library to support Object Push and Generic Object Exchange profiles. They can also use the Serial Manager API in conjunction with the Bluetooth

[7] Palm OS 68K API Documentation, Vol II

Virtual Serial Driver to perform serial communications on a serial port abstraction. Applications can also use the Bluetooth API directly for more flexible, fine grained access to the Bluetooth functionality.

Network Library

The Palm OS provides two different network libraries for networked applications – the netLib library for low level TCP/UDP socket API calls and the Internet Library for socket-like API calls to higher level protocols like HTTP. The Internet library uses the netLib calls to implement its functionality. The network library consists of two components – the netlib interface for applications that runs in the application's task and the protocol stack implementing TCP/IP that runs as a separate task. The two components interact using an operating system mailbox queue. This scheme allows multiple applications to wait for data from network without blocking each other and doing other tasks while waiting for data to arrive[8].

The network library consists of two application programming interfaces to sockets – a Berkley sockets API and the network library's native API. The Berkley sockets API is a standard API to socket programming in UNIX systems and is well understood and familiar to network programmers. The Berkley sockets API implementation on Palm OS is actually a wrapper around the native API. The higher-level Internet library provides applications with an easy way to access internet web pages by encapsulating the internals of socket communication. It provides API calls to replace the typical User Interface Event Loop's EvtGetEvent calls to retrieve web data asynchronously – this allows application user interfaces to be responsive to user actions and not block waiting for data over slow internet connections.

Palm OS also provides a SSL (Secure Sockets Layer) Library for applications that need secure communications. The SSL library provides a SSL and non-SSL I/O interface to applications. It uses the network library for low level communication transport and

[8] Palm OS Programmer's Companion, Vol. II: Communications

implements SSL processing to encrypt and MAC data before passing it onto the network library.

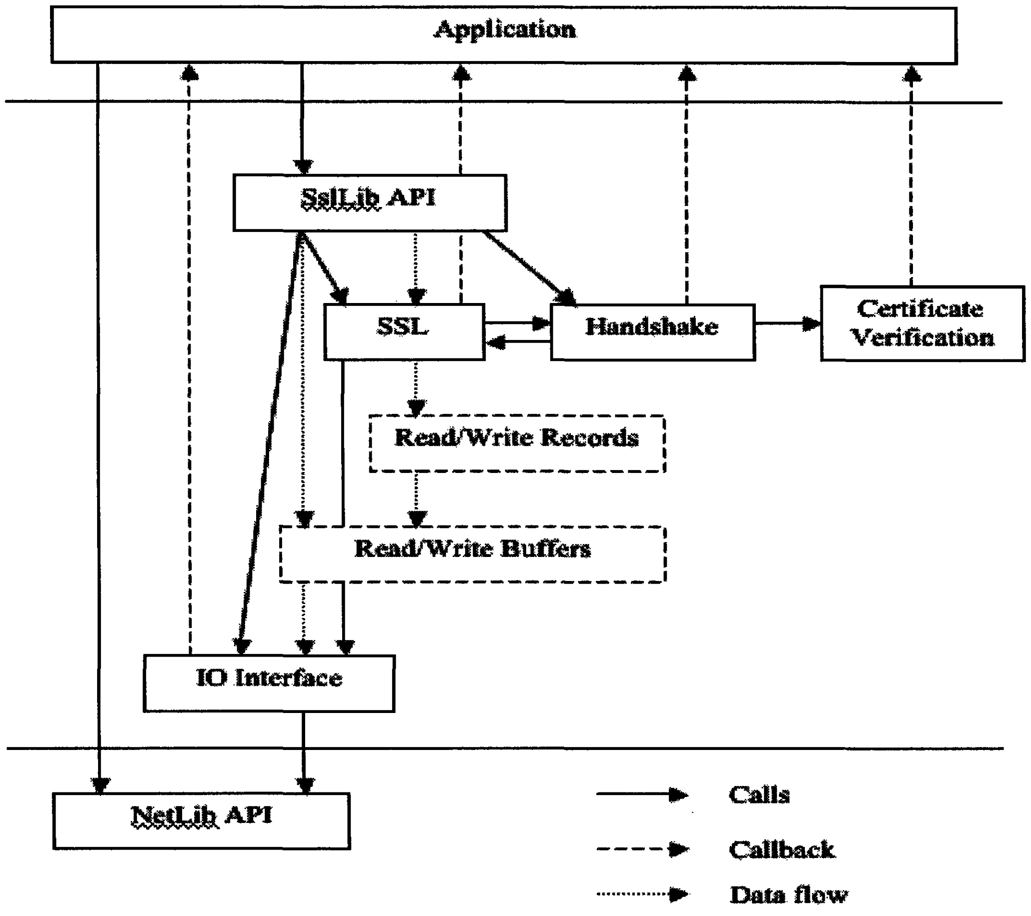


Fig. 7 SSL library architecture and Network Library interface[9]

Telephony

The Palm OS provides a set of APIs through the Telephony manager to access the telephony functionalities in a smartphone. These APIs are categorized into subsets called service sets where each service set represents a particular feature – not all features are present on all devices. The API provides macros to verify the presence of a feature on a device before an application can attempt to use it. The Telephony manager library is a

[9] Palm OS 68K API Documentation, Vol II

shared library that is loaded only when required by the application. A unique feature of Palm OS's telephony library is the ability to call any of the telephony functions synchronously or asynchronously. A Palm OS application can also register for telephony notifications like receiving incoming calls or SMS messages when it is not the currently running application. The OS will notify and run the application to handle such events when they occur through the normal notification message mechanism.

2.2 Symbian Operating System – High Level Architecture

This Section discusses the architecture of the Symbian Operating System based on a study of the Symbian OS guide at http://www.symbian.com/developer/techlib/v9.1docs and the book "Symbian OS C++ for Mobile Phones" by Harrison. The architecture described here is a summarization of the Core Symbian OS concepts found in the official documentation which provides a more detailed explanation. This section uses the same terminology and description for the several subsystems as the official Symbian guide.

The Symbian Operating System is a multi-threaded, event-based, object oriented and preemptive Operating system written in C++. The primary design goals of the operating system according to its designers have been flexibility, customizability, efficiency, robustness and communications-centric. In addition, modeling the Programming Interface using object oriented constructs provides application developers and OS licensees with a powerful tool to help customization and ensure rapid application development. Symbian OS has a micro-kernel based architecture where the OS kernel is light-weight consisting of core system functionality like scheduler and a base user library that execute in privileged-mode. This requires hardware supported privilege execution mode control. The file system and graphical windowing system and implemented by servers running as processes in user-mode.

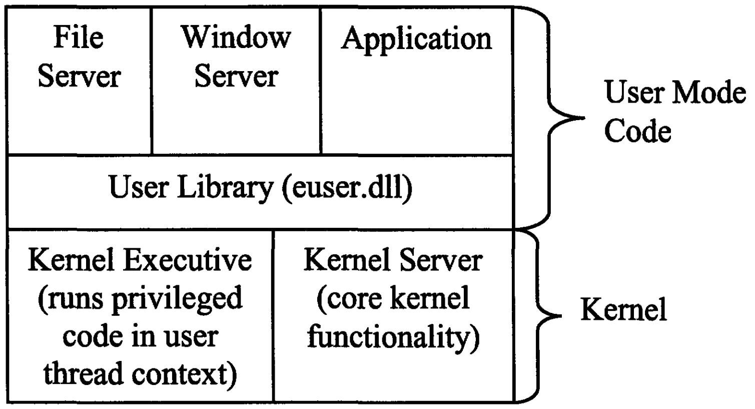
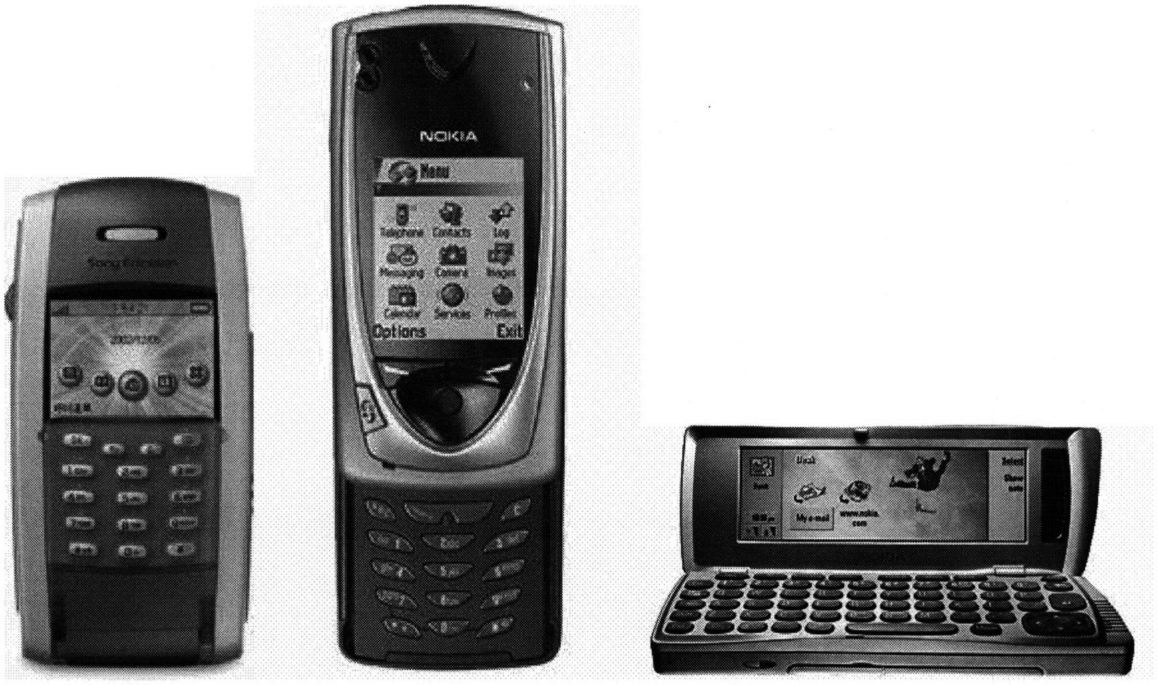

File Server
Window Server
Application
User Library (euser.dll)
Kernel Executive (runs privileged code in user thread context)
Kernel Server (core kernel functionality)
User Mode Code
Kernel

Fig. 8 Symbian OS Kernel and E32 boundary

Symbian OS provides the fundamental operating system features including graphical, communication and application frameworks. A framework is a collection of abstract and concrete classes – the abstract classes are designed for extension by being implemented and customized by the application. Depending on the features of the target machine – size / aspect ratio of the screen, keyboard or stylus based input, and voice/data centric features – an additional GUI layer to support device specific customization is supported. The base Symbian OS provided UI layer is called Uikon. The additional GUI layers available for UI customization are currently – UIQ (Sony Ericsson P800, P900, and Motorola A1000), Series 60 (Nokia N70, N90, Panasonic X800) , Series 80(Nokia Communicator 9300, 9500) and NTT DoCoMo Platform (NTT DoCoMo FOMA F205)[10]. These customizable user interfaces are implemented as a graphical framework layer on top of the base Uikon – for instance, the UIQ user interface is implemented by the Qikon interface on top of Uikon.

[10] http://www.symbian.com/phones/index.html



Sony Ericsson P800 running UIQ UI

Nokia 7650 running Series 60 UI

Nokia 9210 running Series 80 UI

Fig. 9 User Interface Layers based on Symbian OS[11]

The Phone specific UI layers also include their own APIs for application programs to use. Thus the UIQ library has its own SDK on top of the Symbian OS API that programmers targeting the UIQ platform can use. Application programs that are designed to work on multiple Symbian platforms have to be developed in such a way that they abstract the UI platform specific features into separate recognizable modules that can be easily ported to the various UI platforms – rest of the code that uses the common Symbian OS API will work without conversion. This thesis will not cover the UI platform specific APIs like UIQ – it will study the Symbian OS API common across all platforms.

Symbian OS consists of a large set of interface libraries consisting of over 600 components and 2500 header files[12]. The components can be grouped at a higher-level based on functionality into around 20 major subsystems comprising the Symbian system.

[11] http://www.i-symbian.com/forum/articles.php?action=viewarticle&artid=38 (accessed Nov. 2006)
[12] Symbian OS v9.1 Guide available at http://www.symbian.com/developer/techlib/v9.1docs

These subsystems can be represented as in the diagram below. The discussion on the subsystems further on is based on the latest stable Symbian Operating System as of this writing, Symbian OS v9.3.

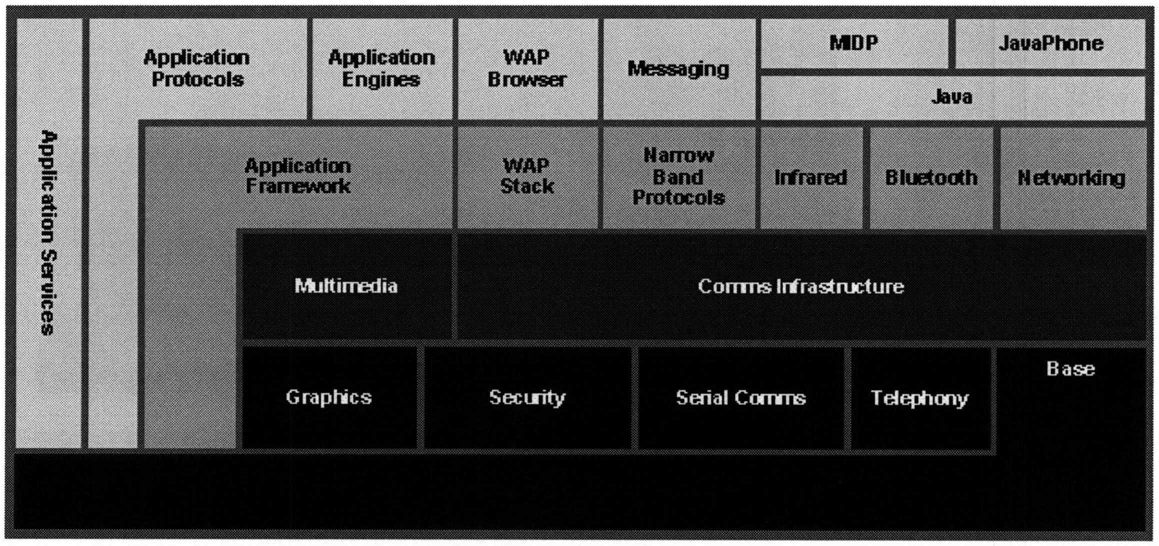


Fig. 10 Subsystems in Symbian OS[13]

Base API

The Base API provides basic programming types and interfaces over which all other subsystems and application programs are built. The Base API provides C++ types that are designed and optimized for mobile devices – definitions of basic data types like integer to guarantee their size and type on any Symbian platform, utility classes like strings, arrays and lists in place of the C++ Standard Template Library. It also includes a lightweight version of exception handling tailored to the Symbian platform. In addition the Base API provides access to Core OS resources like processes, threads and memory management.

[13]http://www.symbian.com/developer/techlib/v9.1docs/doc_source/guide/N1001A/SubsystemsAndAPIs.guide.html#devguides%2eSubsystems%2estart (accessed Nov, 2006)

Handles

Application programs use handles that encapsulate handle-numbers to refer to objects or resources owned and managed by a different process – The Symbian OS kernel mediates interactions between the holder of the handle and the object referred to by the handle. Handles can be thread or process level – a thread-relative handle is accessible only to the thread creating it and its lifetime is limited to the thread lifetime whereas a process-relative handle is accessible to all threads running in that process and its lifetime is the lifetime of the process. Handles can also be local or global – the thread that creates the handle determines its scope. A local handle is accessible only to the thread creating it whereas a global handle is accessible to all threads in all processes. A global handle could thus be used to share resources across processes.

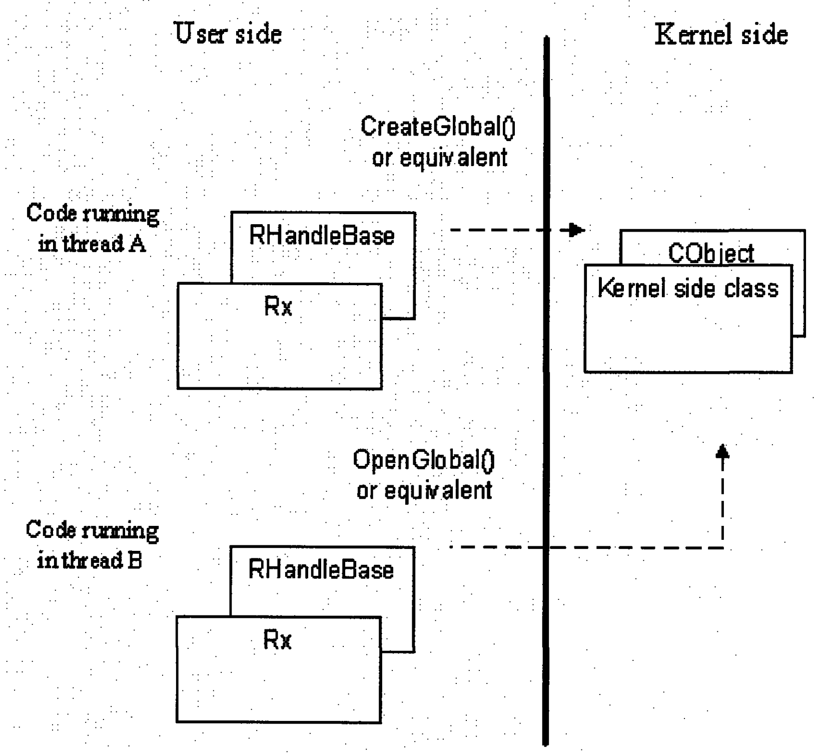


Fig. 11 Symbian OS Handles[14]

[14] http://www.symbian.com/developer/techlib/v9.1docs/doc_source/guide/Base-subsystem-guide/N10086/Handles/HandlesLocalGlobal.gif (accessed Nov. 2006)

Inter-Process Communication

Symbian OS API for Inter Process Communication is fundamental to Symbian programming and provides the encapsulation for the underlying active-object framework used by Symbian applications. It is the mechanism used by Client Applications to make service requests on other service providers like the windowing or telephony system servers. The Inter-Process Communication API calls are wrapped in the client-server architecture framework which in turn is used as the base framework for active-objects. The mode of communication is asynchronous – thus, a request made to a service provider returns immediately to indicate the request was received and completion of the service is signaled to the requestor asynchronously. Two methods of asynchronous service handling exist[15]:

- Low-level service handling APIs provide basic API calls using which the caller can request asynchronous service and be indicated of the service completion though a thread-request semaphore.
- High-level service handling is accomplished through use of the active-object framework. The framework consists of the active-object which implements the client-side interface used to make asynchronous service requests on the service provider and to handle completion of those requests and the active-scheduler that implements a wait-loop to wait on the service-requestor thread's request semaphore waiting for outstanding requests to complete.

[15] Symbian OS v9.1 Guide at http://www.symbian.com/developer/techlib/v9.1docs

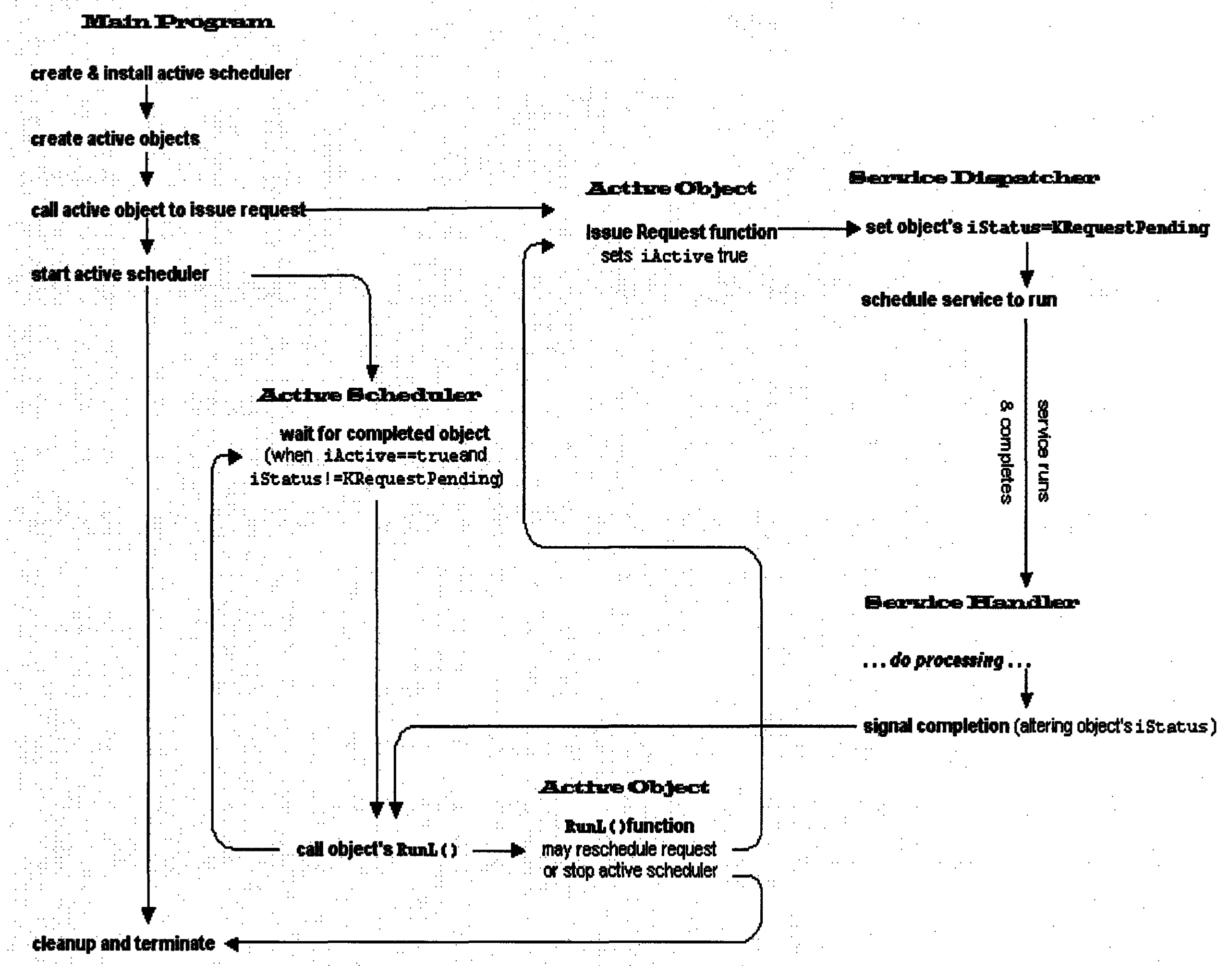


Fig. 12 Active-Object framework in Symbian[16]

A client program can issue asynchronous service requests to four kinds of service providers – kernel service provider, device driver service provider, user-side service provider and a service-provider in the same user thread. The last type is used to implement a long-running task by an application thread by dividing the task into multiple shorter tasks and scheduling them in sequence, thus allowing running longer tasks without needing to start new threads. Symbian also provides Message Queues to enable communication between different threads within a process or across processes. Another way for threads to communicate is through sharing a common heap that will be discussed further in Memory Management later on.

[16] http://www.symbian.com/developer/techlib/v9.1docs/doc_source/guide/Base-subsystem-guide/N10086/InterProcessCommunication/AsynchronousServicesGuide/AsynchronousServicesGuide3/LifeCycleModel.gif (accessed Nov, 2006)

Symbian Client-Server Framework

Symbian provides several system-level APIs to build a client-server framework to provide services to client programs. Using a client-server framework enables isolation of functionalities and prevents malfunctioning clients from corrupting system resources, enables asynchronous service requests and enables managing and sharing resources from a central service provider. System servers like the Windowing server, File server and messaging server use the client-server framework architecture to provide services to applications and other system programs. Servers provide a client API that clients can use to request services through a messaging protocol that flows through the Operating System Kernel. The Key concepts involved in the Symbian Client-Server framework are: server, session, sub-session and message[17]. The server handles connection requests from clients and establishes sessions representing a channel of communication between client and server. Multiple sub-sessions can be created to enable simultaneous uses of a server by the same client-thread. Messages are the unit of communication between client and server and convey information about the request including the request code and arguments to the request.

Memory Management

Symbian integrated memory management and cleanup APIs into its application programming interface architecture to provide programmers for the Symbian system with an efficient and effective tool to manage memory and handle error conditions. This support is built into the Core API through virtual destructors in the base class, CBase, and using a lightweight exception handling mechanism – User::Leave() to throw an exception, TRAP and TRAPD to setup a trap harness to catch the exception. Objects allocated on the heap after the trap harness has been setup and until the exception is raised are pushed onto a CleanupStack that tracks these memory allocations. If an exception is raised and the stack unwound, memory allocations that have been pushed onto the stack are automatically freed and memory leaks thus prevented. The Uikon

[17] Symbian OS v9.1 Guide at http://www.symbian.com/developer/techlib/v9.1docs

Interface framework makes effective use of the memory management and exception handling support from the Operating System to provide the basic infrastructure for well behaved memory handling to User Interface programs.

The Symbian platform has been designed to serve as the operating system for mobile phones and this is reflected in its memory architecture – ROM is the read-only memory area of the phone that supplies the operating system and built-in applications from the factory. RAM is the dynamic region of memory used to load programs (from the storage card) and allocate memory for program needs. Code for programs in ROM are not loaded into RAM for execution – RAM is only used for their execution stack and dynamic allocations.

Programs in Symbian run as individual processes, each containing one or more threads of execution, in user-privilege mode and have their own virtual private address space. The Kernel is the only process that runs in supervisor-privilege mode and contains two threads – one is the kernel server thread that implements memory allocation/deallocation on the kernel heap and the other is the null thread that places the processor in idle mode to conserve power and runs when there is no other process thread to run[18]. The address space of a process consists of several chunks – a chunk is a contiguous area of RAM. These chunks are one of three types – stack/heap chunk that contains the program stack and dynamic memory allocations, code stack that contains program code for programs loaded from the storage card and the data chunk that contains static program data. Each new thread gets its own stack/heap chunk. Processes can share data using Global chunks – Global chunks are named to identify the chunk to another process that wishes to use it. It is important to note that processes have a virtual address space that is linear – the operating system takes care of mapping the virtual address into a physical RAM address. This is not the same as the disk-backed virtual memory scheme used by desktop operating systems since mobile phones do not have large capacity hard disks in them.

[18] "Memory Management" section of Symbian OS v9.1 Guide at http://www.symbian.com/developer/techlib/v9.1docs

This is a key motivation for the memory cleanup and exception handling framework supported by Symbian for application programs.

Process Model

Application Programs run as individual processes in their own virtual private address space in Symbian OS. The operating system supporting multi-threading and a process can have multiple threads of execution though this is frequently not necessary – Symbian provides an active-object, event driven framework wherein Applications perform actions in response to specific events[19]. The basic unit of execution in Symbian OS is a thread. The OS kernel schedules threads preemptively – at any time the highest priority thread in running. If a thread with a higher priority than the currently running thread becomes ready, the kernel suspends the current one and schedules the higher priority thread. Threads having the same priority are scheduled on a round robin basis in fixed time slices. Context switching between threads is expensive because the current thread's execution state has to be saved and a different context loaded by the Kernel. To avoid these multi-tasking costs, Symbian provides users with the active object framework to run multiple tasks in a non pre-emptive manner. All threads running in the same process share the same address space of the process and thus can access each other's data. This gives rise to the classic resource sharing and protection issues – Symbian provides API support for Semaphores, Mutexes and Critical Sections to ensure mutually exclusive access to resources and ensure resource integrity. Critical Sections are used to serialize access to resources across threads running within the same process whereas Mutexes can serialize resource access across threads in different processes. In addition, the Operating system provides comprehensive APIs for Process and Thread management.

[19] "Thread and Process Management" section of Symbian OS v9.1 Guide at http://www.symbian.com/developer/techlib/v9.1docs

File System

The Symbian Operating System provides desktop compatible, VFAT file system through the File Server – a system server running in user-mode[20]. The File server manages input/output activity, resource contention, file sharing and locking. The File Server provides a comprehensive client API to access the file system by managing ROM, RAM and storage card memory regions and exposing a file interface to those memory areas. In addition, the File Server exposes an interface to allow dynamically installable file systems. The Stores API defines a Stream Stores interface layered over the File System for applications to use – this provides a more natural byte stream interface to file reading/writing. A Store is a collection of streams and implements persistence of objects. The ROM area which contains the operating system code is mapped to drive z:, RAM is dynamically allocated and mapped to drive c:, removable storage cards are mapped to remaining drive letters.

Symbian provides an interface to relational databases with transaction support and SQL views through the DBMS API for applications that require database functionality. Two DBMS implementations and API are provided – a client side implementation for applications that require a datastore for their internal usage that is lightweight and efficient, and a client-server implementation for applications that run as a server and provide DB access to their clients. The database implementation uses the File Stores API for the underlying data storage.

Bluetooth

Bluetooth is a short range radio communications technology to enable independent, different types of devices to communicate with each other using a common protocol. It is implemented as a hierarchy of functionally isolated components constituting a Bluetooth stack. Symbian provides access to the Bluetooth communication capabilities through a

[20] "File Server (F32) API" in Symbian OS v9.1 Guide at http://www.symbian.com/developer/techlib/v9.1docs

socket API which encapsulates access to the L2CAP (Logical Link Control and Adaptation Protocol – controls access of multiple users to a link, handles packet segmentation and reassembly) and RFCOMM (adapts the Bluetooth connection as a serial connection)[21].

The Symbian Operating System implements SDP (Service Discovery Protocol) using the Bluetooth Service Discovery Database to enable remote devices to discover device presence and services offered and the Bluetooth Service Discovery Agent to discover services available on remote devices and the attributes of those services. The Symbian Operating system also provides the Bluetooth Security Manager to set security requirements for incoming connections (authorization, authentication and encryption) and a User Interface for users to select devices and their services. Symbian provides a comprehensive Bluetooth client API to enable applications to programmatically determine services from remote devices and provide filtered selections for the user based on pre-configured user preferences and capabilities offered by the remote device. An API to perform low-level Bluetooth configuration using HCI commands, L2CAP options RFCOMM commands and options is also provided.

Communications Infrastructure

Symbian provides a Communications Database where all communications related settings are stored by the control panel applets. These settings are globally accessible to all programs requiring communications functionality like Networking, Telephony, and WAP etc. A series of tables are stored in a relational database managed by CommDb which provides the Client API to access these settings for other communications subsystems and hides the DBMS functionality underneath.

Symbian provides a generic sockets interface modeled after the Berkeley Sockets API for applications programs to use. A socket is a communication endpoint that a program uses to transmit and receive data. The socket infrastructure in Symbian follows the standard

[21] Symbian OS v9.1 Bluetooth API at http://www.symbian.com/developer/techlib/v9.1docs

Symbian Server framework – Socket services are implemented by a generic socket server to which protocol modules for different transports (TCP/IP, IrDA) can be plugged in as dynamically loaded libraries. The socket server exposes a Client API to list available protocols and for clients to access socket services. The socket client API provided by the Symbian library ESOCK allows applications to open connections, read/write data, close connections, hostname and service resolution. The socket client API communicates with the socket server asynchronously, which in turn communicates with the various protocol modules that provide the communication implementation. The Symbian ESOCK API also provides interface to network databases such as LM-IAS with IrDA (Infra-red). An API to set QoS (Quality of Service) on a communication channel to control service parameters is also provided.

Secure Communications over a public network is supported through the Secure Sockets API. It is provided as a plug-in architecture with a generic client interface to which various protocol modules can be added. Plug-ins for SSL (Secure Sockets Layer) v3.0 and TLS (Transport Layer Security) v1.0 are provided as part of the secure sockets library.

Infrared Communications

Symbian OS provides IR support through the implementation of the IrDA (Infrared Data Association) stack. It allows applications to search for other IR devices in range, query services offered and use a reliable/unreliable data protocol to transfer data[22]. Symbian provides access to IR facilities through a Sockets interface or by emulating the serial port over IR. The serial interface is provided through the Symbian OS Serial Communication Server API. The Serial Communications Server implements a plug-in architecture for modules providing serial communications over different transports – the Serial IR module (IrCOMM) plugs into this server. IR facilities can also be used through the Sockets Client Interface to the Socket Server. In this case, IrDA Sockets is the plug-in module to the Socket Server which clients access through the generic Sockets Client API.

[22] Symbian OS v9.1 Serial Comms (C32) API Guide

Client programs can use the sockets interface to access IR specific configurations – IrTinyTP for reliable transport, IrMUX for unreliable datagram service, IrDA discovery and for device discovery, and for access to IrDA IAS (Information Access Service).

Messaging

Symbian provides an extensible mechanism for messaging applications through a framework for supporting new messaging protocols that can be plugged into the architecture. A plug-in module is made from several components called Message Type Module (MTM) that interacts with the lower level communication protocols like TCP/IP. Messaging applications like SMTP, POP3, SMS and Fax are implemented as MTMs and use the Messaging architecture. The messaging architecture is designed to allow client applications to dynamically discover new messaging types from the installed MTM modules and allow users to select those messaging facilities. This open architecture and interface specification allows client applications to use new messaging types and offer them for user selection dynamically.

Graphics

Symbian abstracts the low level drawing operations to screens and printers to provide an abstract graphical device interface for higher level functionality like the window server. It also provides the font and bitmap server that manages device fonts and bitmaps and provides access to client threads[23]. By centralizing fonts and bitmaps in a server that provides a shared memory area accessible to the window server and its clients, the Symbian graphics architecture enables smooth and efficient rendering of graphics and animations. Drawing operations are buffered by the Window Server Client API in application program memory and flushed at specified times to the Window Server – this helps in reducing number of context switches between the different processes and enhances system performance.

[23] "Graphics", "Font and Bitmap Server (FBSERV)" sections of the Symbian OS v9.1 Guide

The central and most important component in the Symbian Operating System is the Window server which manages access to the device screen and keyboard by running applications. It runs as the highest priority user thread to enable quick response to client application requests and dispatching events. It allows applications to control their drawing area, request for events and handle them when they occur and draw to their display region using the windows graphics context provided by the graphics device. The windows server also allows dynamic extensibility by providing a plug-in architecture to load third-party animation dynamically loaded libraries.

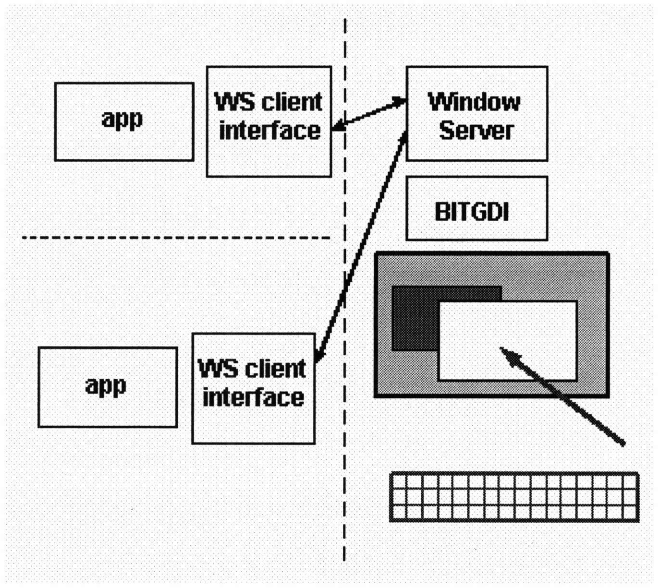


Fig. 13 Symbian Window Server[24]

The application user interface framework is built over three layers: Uikon and UI variants provide the concrete user interface classes constitute the top layer which applications use. The abstract middle layer constitutes the control framework which provides the framework to create user interface controls, handle user interface events and environment utilities to access windowing functionality. The lowest level consists of the window server providing basic windowing functionality using device independent graphics and servicing device interrupts to generate events. The active-object framework that

[24] http://www.symbian.com/developer/techlib/v9.1docs/doc_source/guide/Graphics-subsystem-guide/N100B6/WindowServerGuide1/wsintro-server.gif

application programs for the Symbian OS typically use for GUI functionality encapsulate the low-level asynchronous service API provided by the Window server. Using active objects, applications handle streams of events from the Window server.

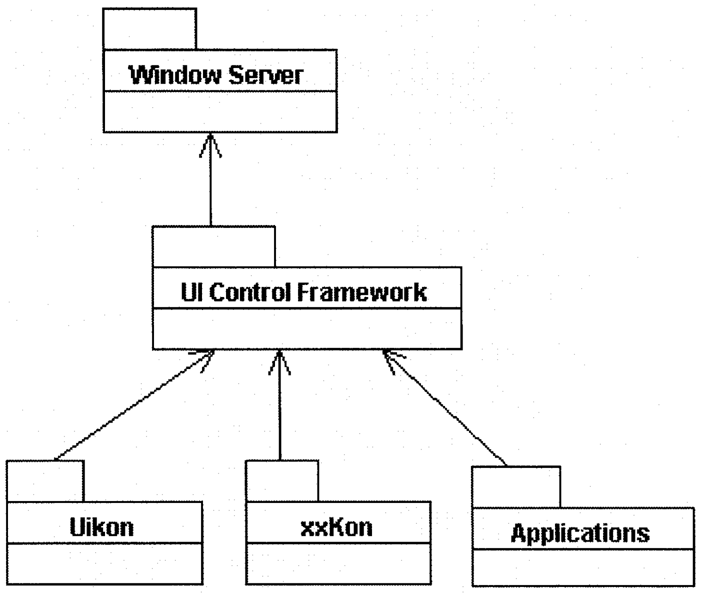


Fig. 14 Symbian UI Control Framework[25]

Abstraction of the User Interface functionality through creation of the UI Control Framework middle layer allows device manufacturers to develop different concrete User Interface control libraries at the top layer to customize and differentiate their devices based on device capabilities – pen/keyboard based input, display size/resolution etc. The UI Control Framework layer (also called CONE for Control Environment) simplifies access to the Window Server by implementing the common idioms to work with the Window server in the CCoeEnv class, as well as providing a reference framework for creation of new concrete UI libraries at the top layer.

The Uikon layer provides the top level and fundamental framework used by all Symbian applications. The Uikon library is common to all Symbian based phones. Other UI libraries like UIQ and Series 60 provide controls derived from Uikon that implement look

[25] http://www.symbian.com/developer/techlib/v9.1docs/doc_source/guide/Application-framework-subsystem-guide/N10046/UIControlFrameworkGuide1/cointro-apiarch.gif (accessed Nov, 2006)

and feel specific to those platforms. Uikon integrates the Application Architecture Framework and the middle level UI Control Framework to provide the framework for standard application design. Applications and UI variant libraries use the Uikon framework libraries as the basis for implementation of their functionality.

Multimedia

The Multi Media Framework client API provided by Symbian encapsulates a plug-in architecture to create, play and convert audio and video files, stream audio and play tones. The plug-in architecture enables installation of controller plug-ins that can be used by the audio, video recorder and player interfaces as well as the audio converter interface. The output from the plug-ins can be directed to a file, screen or low level audio device driver. Audio streaming and tone playing interfaces do not require any encoding or decoding and thus interface directly with the device driver instead of going through the framework. The multi media framework supports extensibility by support of the interfaces to add controller plug-ins capable of playing/recording new media formats, format encoder/decoder plug-ins to read/write new media data formats, codec plug-ins for media conversions between different encodings and source/sink plug-ins to handle reading from/writing to sources/sinks of media data.

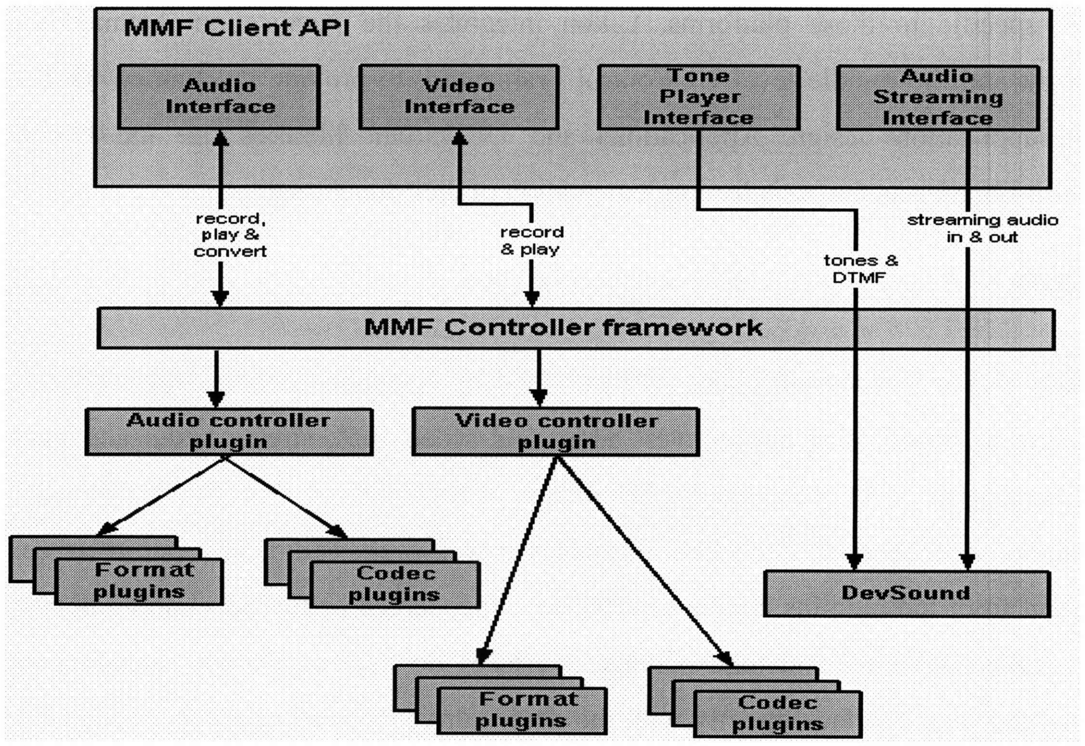


Fig. 15 Symbian Multi Media Framework[26]

An API to communicate using RTP (Real Time Protocol) protocol utilizing the Socket Server Interface is supported by Symbian. RTP is a multi media protocol built on UDP and used for transporting interactive audio and video for web-conferencing and VoIP applications. Symbian also provides an extensible Camera API that allows application programs to control and use the on-board camera of the device as well as allow hardware manufacturers to extend the API to provide proprietary extensions. APIs to perform bitmap transformations, image manipulation and conversion

Security

The platform security architecture on the Symbian Operating System addresses security threats from the distribution of malicious applications by preventing unauthorized access to user data and system services. It achieves this by establishing a firewall for protection

[26] http://www.symbian.com/developer/techlib/v9.1docs/doc_source/guide/Multimedia-subsystem-guide/N100EA/MMFClientOverview.gif (accessed Nov. 2006)

of systems servers through a capability-based access control model and creating a protected part of the file system that rogue applications won't be able to access. To ensure robust platform security, Symbian uses the concept of a Trusted Computing Base – the smallest set of architectural components that cannot be subverted and whose integrity is guaranteed[27]. On the Symbian platform this consists of the Kernel and E32 user library which are system-wide trusted and have full access to the device. A Trusted Computing Environment consisting of specific accesses granted to specific system servers (telephony, windowing, certificate management) is built on top of the Trusted Computing Base. Each component of the Trusted Computing Environment is given only specific capabilities it requires and nothing more – this limits the extent of damage that can be caused by vulnerability or misuse of privileges.

Certificate Management is used to trust software from third parties at installation stage – their certificates are verified against appropriate root certificates and required capabilities for their functionality are granted.

Telephony

Symbian provides an interface to the telephony subsystem on the phone through the CTelephony API. The API provides comprehensive information about the phone, current in-progress calls, carrier and network information and ability for applications to originate and answer calls. The API also enables applications to control a single or two voice calls at once. Various details about the call like start time, duration, status (dialing, ringing, on-hold), caller-id and called number, phone capabilities and settings are all provided through the API. Function calls made to the Telephony API are asynchronous and applications use the Symbian Active Object framework to control the API calls.

[27] Symbian Security Architecture document at http://www.symbian.com/developer/techlib/v9.1docs/doc_source/guide/N10022/SGL.SM0007.013_Rev2.0_Symbian_OS_Security_Architecture.doc (accessed Nov. 2006)

An Advanced Telephony API to the Operating System is exposed to phone manufacturers who require more advanced telephony services. This interface is not available to application programmers to prevent malicious use of the phone system. The basic Telephony API available through the CTelephony class also provides applications with the ability to be notified about changes in the phone's environments – battery, signal strength, call status etc. Symbian OS also provides an API for applications to send and receive faxes using the ETEL fax API and use contacts from the phone's Contacts' Database using the Phonebook Synchronizer server API. Alternatively, applications can use the Contacts Model API to store, manipulate and retrieve contact data.

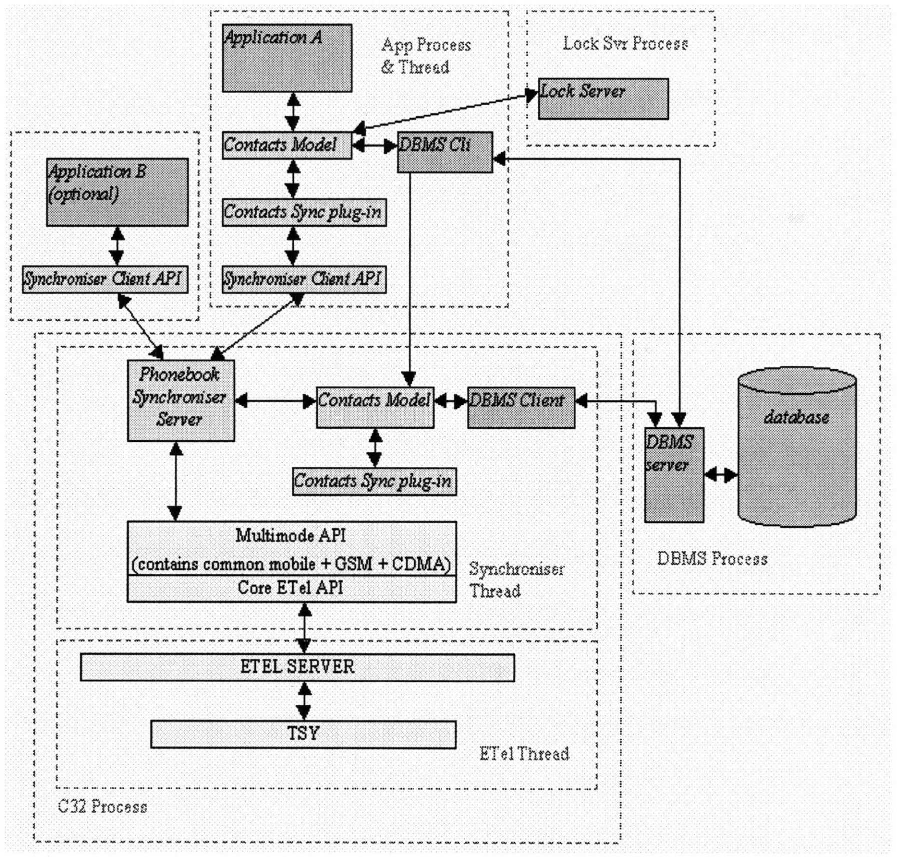


Fig. 16 Symbian Phonebook Server and Contacts Model Architecture[28]

[28] http://www.symbian.com/developer/techlib/v9.1docs/doc_source/guide/Telephony-subsystem-guide/N10142/phbk-architecture.gif (accessed Nov, 2006)

Application Engines

Application Engines is an API for applications to integrate seamlessly with and use data from built-in core applications like Contacts, Agenda. Other subsystems provide their own APIs for other applications to use their core data. This provides for a holistic user experience on the Symbian platform for the user that is consistent across all applications.

Text Handling

Symbian OS uses its own version of String libraries that are lighter than C++ strings and easier to use and maintain than C strings. It uses descriptors as the common interface to text and binary data in the program binary, process stack or heap. Descriptors prevent buffer overflow errors, but leave memory management issues to the application for efficient memory usage. Symbian uses the Unicode character set format to represent characters and provides APIs to convert between Unicode and other character sets. The API also allows installation new of plug-in modules to implement conversion between Unicode and character sets not already in the system.

Base UI Framework

Like other GUI operating systems, Symbian is event-based. Symbian applications perform actions in response to user or system generated events. Symbian provides OS level support for an event framework. The Symbian UI framework includes an active-scheduler that receives events and dispatches them to handlers of those events called active-objects. Thus, at a high level a Symbian application can be considered as a set of active objects performing tasks in response to events fed to it by the active scheduler. Applications conform to and provide implementations that adhere to the Symbian Application framework requirements – the Kernel uses this framework to handle application launch and creates the initial set of framework objects to launch the User Interface. This framework includes CONE – Control Environment consisting of base UI classes and APPARC – Application architecture and application data classes.

2.3 Windows Mobile Operating System – High Level Architecture

Windows Mobile is the operating system for Mobile devices developed by Microsoft and is found in three different flavors based on device capabilities – Pocket PC, Pocket PC Phone Edition and Smartphone. Pocket PC and Pocket PC Phone Edition devices are typical handheld devices akin to Palm devices and come with larger display areas and touch screen capability. They also have more powerful processors and come with mobile versions of Microsoft Office Applications – Word Mobile, Excel Mobile and PowerPoint Mobile. They might also come with WiFi hardware to enable users to connect to the Internet at Wireless Hotspots. Pocket PC Phone Edition powered devices come with all the capabilities and features of Pocket PC devices and add cellular phone capabilities. Windows Mobile Pocket PC Phone Edition devices are able to place/receive calls on a cellular network and also able to connect to the internet wirelessly over the cellular service provider's network. Windows Mobile Smartphone platform is geared toward typical cell phone devices – it is designed to be more compact, run on slower processors and interact with users on a smaller screen with no touch screen capability. Smartphone powered devices also don't come bundled with Mobile versions of the Windows Office Applications but third-party applications to provide these capabilities are available for the Smartphone platform.

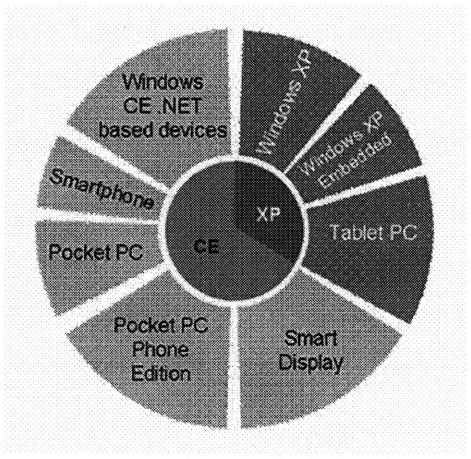


Fig. 17 Family of Windows Operating Systems[29]

[29] http://www.pocketpcmag.com/_archives/Jun06/images/Jun06_p56_1.jpg (accessed Dec 2006)

Windows Mobile is based on the Windows CE embedded operating system which has been designed to be a compact, ROM-based operating system incorporating a subset of the Win32 API used to program all Microsoft Windows Operating Systems – Windows 95, NT, Me, XP etc. Windows CE has been designed from the ground up to be a lightweight, low power consuming, multi-threaded, multi-tasking operating system with an optional user interface and adaptable to run on different hardware with different capabilities and run time environments. Another chief aim of the designers of Windows CE was to allow Win32 programmers of the desktop Windows Operating Systems to continue using the same APIs and programming paradigms – messages loops, event handling frameworks on the smaller Windows CE platform. The small footprint of Windows CE also required pruning the comprehensive set of Win32 APIs to suit the CE platform while adding a few new ones specific to embedded programming.

Existing frameworks for Windows development like MFC – Microsoft Foundation Classes can be used to develop Windows CE applications. Windows CE includes the MFC runtime library in ROM to reduce the footprint of applications developed using MFC. Although MFC programming is possible, Microsoft has stopped development work on this framework – instead, .NET is the development environment of choice on the Windows platform. The .NET environment consists of hardware independent, type safe run-time environment to run code in a secure way – applications are compiled into an intermediate language called Common Intermediate Language (CIL) that is compiled into machine language at run time called Just-in-Time (JIT) compilation. This enables applications to be written and compiled once, but still run on a variety of hardware supporting the .NET environment. Code that runs and makes calls within the .NET environment is called managed code while code that invokes calls directly on the operating system to access functionality not available within the .NET runtime is called unmanaged code. Microsoft has developed a limited version of the .NET environment for Windows CE called the .NET compact framework. Thus, application developers for the Windows Mobile platforms have the option to use the .NET runtime for development apart from using the Win32 API directly.

To summarize the Windows Platform for Mobile devices, Windows CE is the embedded operating system serving as the base for a host of consumer and industrial devices requiring a real-time, small foot-print operating system with a deterministic response to interrupts. A Windows Mobile device is Windows CE based, with a custom shell, user interface and a set of mobile productivity applications provided by Microsoft. Windows Mobile branded devices are certified by Microsoft to conform to a set of hardware and software standards and specifications. Three variants of Windows Mobile devices exist – Pocket PC, Pocket PC Phone edition and Smartphone. Since Windows CE is the base operating system for all Windows Mobile devices, it will be the subject of further discussion – its architecture, components, their major functionalities and programming interfaces.

This thesis will use the latest shipping version of Windows CE, version 5.0 as the basis for further study. Windows CE provides the Platform Builder which is a comprehensive set of tools to enable device OEMs to design, create, build and test a Windows CE based customized Operating System for specific target devices. Windows CE also includes the OEM Adaptation Layer (OAL) to enable OEMs to develop a customized Operating System using its Componentization support through code libraries, production quality device drivers and centralized configuration files to achieve consistent architecture across processor families and hardware platforms[30].

This Section discusses the architecture of the Windows CE Operating System based on a study of the MSDN Windows CE API documentation at http://msdn.microsoft.com and the book "Programming Windows CE .NET" by Boling. The architecture described here is a summarization of the Core OS concepts found in the official documentation which provides a more detailed explanation. This section uses the same terminology and description for the subsystems and components as the official API documentation.

[30] Microsoft Developer Network: Windows CE 5.0 Platform Builder online documentation at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceintro5/html/wce50oriWelcomeToWindowsCE.asp (accessed Dec 2006)

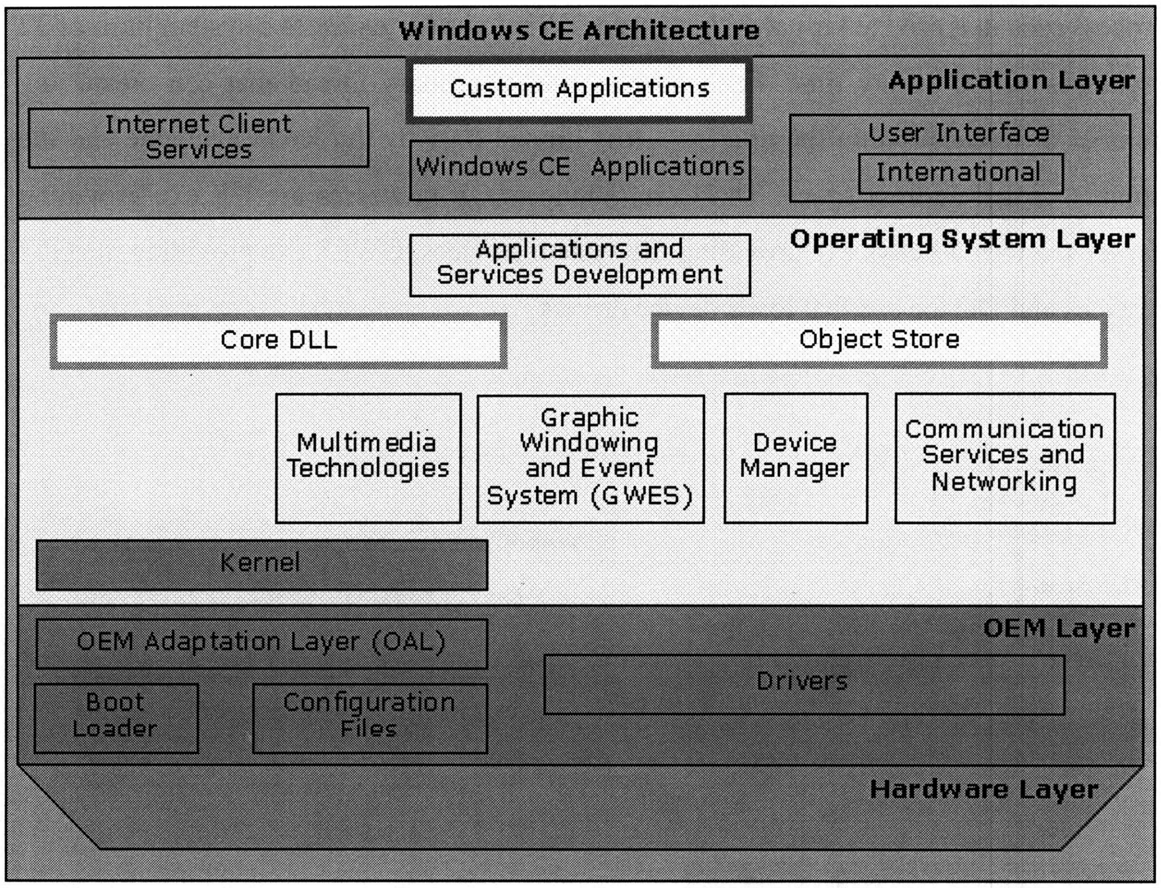


Fig. 18 Windows CE Operating System Architecture[31]

Kernel

The Kernel provides the core functionality for the Windows CE Operating System consisting of process/thread management and memory management. It exposes this functionality to applications through a comprehensive set of core Win32 APIs which are a subset of the APIs to Windows XP. This is because Windows CE doesn't support all process and memory management features that Windows XP supports – an example of this is the lack of environment variables and environment related APIs in Windows CE.

Windows CE is a multitasking (running multiple processes), multi-threading (running multiple threads of execution within a process), pre-emptive operating system where each

[31] http://msdn.microsoft.com/library/en-us/wceintro5/html/windows_ce_architecture.gif (accessed Dec. 2006)

process runs in a private virtual address space. Windows CE enforces a system limit of 32 processes active at any time. Each process runs a primary thread that can create any number of threads within the process – it is limited only by the RAM available and the process virtual address space. The Core Windows CE processes are NK.exe providing kernel services, FileSys.exe providing file system services, GWES.exe providing the GUI services and Device.exe that loads device drivers.

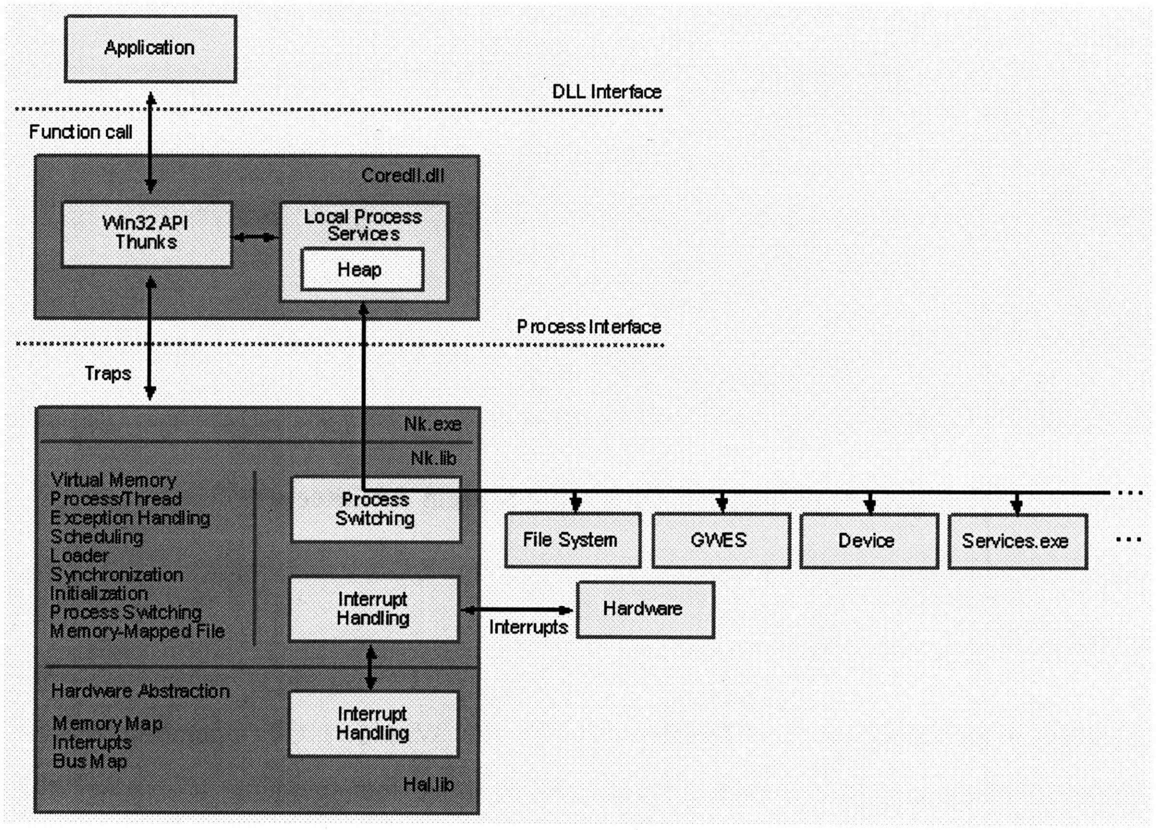


Fig. 19 Windows CE Kernel Interface[32]

In order to provide real-time application capability, Windows CE implements thread priority levels with priority inheritance; deterministic pre-emptive thread scheduling that always runs the highest priority thread ready-to-run (i.e. not blocked on something) and round-robins among threads with equal priority for a configurable period of thread quantum or time-slice. If a thread of higher priority becomes unblocked when a thread of

[32] http://msdn.microsoft.com/library/en-us/wcecoreos5/html/coreos.gif (accessed Dec. 2006)

lower priority is running, the lower priority thread is immediately suspended and the higher priority thread scheduled for execution.

Windows CE supports the entire set of Win32 synchronization objects provided by Windows XP (with the exception of file change notifications and waitable timers) that applications can use to coordinate multiple threads. These synchronization objects include Event objects (dual state objects – signaled or not-signaled that are used to signal task completion to another thread or process waiting on that task), Semaphores (to synchronize access to a resource from multiple threads), Mutexes (to co-ordinate access to resources across multiple threads) and Critical Sections (to protect blocks of code from being executed by two or more threads at the same time). Windows CE also provides applications with API calls for inter-process communication – this include Named Memory Mapped Objects (shared block of memory accessible to both processes at the same time) and Message Queues (First-in-First-out, unidirectional data queues used for sending data from one process to another).

Memory Management

Windows CE based devices use ROM to contain the Operating System image and built-in applications and RAM to use as the file system and program dynamic memory. The RAM based file system is called the object store and behaves as a permanent virtual RAM disk – it retains files stored in it even when the power is turned off and through soft resets. The program memory section of RAM is called the system heap and is used to store program heaps, stacks and executable code for running applications. The system heap area of RAM is reinitialized on soft resets.

ROM based, uncompressed programs that are marked XIP (Execute-in-Place) are run directly from ROM conserving memory space in RAM. All other object store and storage card based programs are loaded into RAM before execution. Windows CE implements a paged, virtual memory management system similar to Windows XP, but unlike XP that assigns a 2GB virtual address space to every process, the virtual address space of a

Windows CE process is 32 MB[33]. In a paged memory management system, the smallest unit of memory managed by the microprocessor is a page – a page is 1024 or 4096 bytes depending on the microprocessor architecture. Virtual Memory addresses accessed by an application in a page are translated to a physical page in RAM by the microprocessor.

Windows CE provides APIs for applications to directly manipulate Virtual Memory allocations in the application's virtual memory space – this is useful for applications requiring large blocks of memory. Applications that don't need the low level control of the Virtual Memory APIs can also rely on Windows CE APIs to manipulate the system provided default heap for dynamic allocations. Applications can also create their own separate heaps instead of using the OS provided default local heap – this is useful to avoid memory fragmentation. The Operating System also provides APIs for applications to retrieve direct pointers to static data contained in ROM – this conserves memory because data need not be copied to RAM.

The Windows CE file system implementation diverges most from the other Windows Operating System products. The RAM based compressed file system is implemented as a database containing files, system registry and other Windows CE databases. The API to the file system though is still the standard Win32 API whose implementation hides the file system details from the application programmer. The Operating System also supports multiple installable file systems through the installable file system (IFS) API to support plug-in storage cards or devices – it incorporates an IFS driver for the FAT file system to access ATA flash cards and hard disks.

Windows CE also supports most of the standard I/O functions in the Win32 API found in Windows XP. It also provides API support for creating and accessing memory mapped files, navigating and managing the file system and accessing the system registry. Windows CE provides a unique database API to store and organize simple groups of data that is on top of the standard Win32 API. The database functionality provided is not comparable to the enterprise SQL databases, but is sufficient for most mobile

[33] "Programming Windows CE .NET" by Boling, 2003, Microsoft Press.

applications that need to store and manage simple data lists. The functionality includes database creation, records with multiple columns and data types, record insertion/deletion and query, sorting and record access.

User Interface

The Graphical Windowing and Events Subsystem (GWES) is the central component in Windows CE acting as the interface between the user, the application and the kernel. GWES is a combination of the Win32 API, User Interface consisting of predefined controls (Windows Controls like Buttons, Edit Controls, Lists, Scroll bars and Static Controls) and the extended set of Common Controls (provided as a Common Control Library DLL and includes Command bar, Data/Time picker, Calendar Control, Progress bar, Tooltip, Tab and Tree Controls), and the Graphics Device Interface (GDI) libraries.

The Windows CE Operating System uses a Shell to define a consistent look and feel for the user interface on the device. It provides users with an interface to manage objects (such as files and folders, remote network drives) necessary for running applications and also manages the Operating System. The Shell provides the basic framework for a user interface that can be customized for a specific device. This is an important feature and allows OEMs to use the Windows CE Shell architecture to implement a variety of shells – this facility is used to provide the custom shells that define the look and feel for Pocket PC and Smartphone devices that are based on the same Windows CE platform. Microsoft provides the source code for presentation and user interface aspects of the Standard Shell (also known as the HPC shell) available to OS design developers for customization.

Windows CE supports the development of Custom User Interfaces also known as Skins to customize the appearance of controls and other UI elements. It accomplishes this by separating the drawing code for controls and non-client area windows from other code that implements their behavior. By tailoring the UI to conform to a specific form factor along with modifications to the shell, OEMs can develop User Interface that can differentiate their devices from others. Windows CE provides two skins by default that

OS designers can use – a Windows 95 skin that replicates the Windows 95 desktop UI appearance and a Windows XP skin to display a XP like UI.

Windows CE also enables OS designers to customize the appearance and behavior of specific UI components or replace them. It also enables customization of behavioral response to out-of-memory conditions, system startup windows device calibration of the touch screen. The size and location of message boxes and other dialog controls in the User Interface can also be customized through setting values for the system registry keys. The system registry also enables OS designers to customize the colors of the User Interface display elements. The Notification mechanism to notify registered applications of the occurrence of interesting events is also customizable and can be tailored to the hardware requirements of the target device.

A Windows CE application program is written and operates the same way as a traditional Windows desktop application program. It uses a subset of the Win32 API that the desktop application uses, but the API syntax and semantics are the same. Also, it contains the same message loop to receive event messages and reacts to those messages similarly with some differences. A major difference is that Windows Mobile application requirements specify that only one instance of a program should run at any time. If the user chooses to run a second instance, the second instance should check for the occurrence of a previous instance and bring it the foreground before it itself quits. This behavior is not enforced by the Operating System, but is a behavioral requirement that all bundled Microsoft applications for the Mobile platform implement.

The Graphics Device Interface (GDI) component of GWES provides the low-level painting support. It abstracts the device graphics capabilities and provides application programs with a display device context to which they draw[34]. Windows then transfers from the pixels from the virtual drawing area to the display device. The drawing functions provided under Windows CE are identical to those provided under Windows XP – these include drawing text, lines, curves, closed shapes and bitmap images. GDI

[34] "Programming Windows CE .NET" by Boling, 2003, Microsoft Press.

under Windows CE does lose a few functions from the desktop versions – these include drawing of complex shapes like Arc, Chord and Pie as well as all drawing functions using the current point.

Windows CE provides the same set of predefined window controls as other versions of Windows. It also provides complex controls through the common control library. The set of standard controls implement a tightly defined user and programming interface. Standard controls include Push Buttons, Check Boxes, Radio Buttons, Group Boxes, Edit Controls, List Box and Combo Box Controls, Static Controls (Labels) and Scroll Bars. Menu Controls in Windows CE are implemented differently than in other Windows versions. They are not part of the standard top-level window, instead they appear attached to a command bar or menu bar control that is manually created for the window. Windows CE also provides complex controls consisting of Toolbar, Status bar, Trackbar, Command bar, Date/Time Picker, Calendar, Progress bar, Tab and Tree controls through the common control library DLL. These common controls have been adapted for the small form factor displays of devices using Windows CE.

Windows CE supports a subset of the common dialog library provided with other desktop versions of Windows OS. A Dialog box provides the user with a predefined window class and window behavior and helps in reducing duplication of code. It hides the complexity involved in creating and managing user interface controls. The Dialog manager in GWES creates dialog boxes using an application provided template and implements default functionality for switching focus between controls and default actions for the Enter and Escape Keys. The common dialog library controls provided in Windows CE are the File Open/Save, Color and Print dialogs which are reformatted for the smaller display on Windows CE devices[35].

[35] "Programming Windows CE .NET" by Boling, 2003, Microsoft Press.

Networking

Windows CE provides Connectivity APIs for Serial, IR, Bluetooth communications as well as Networking APIs to connect to other computers over a TCP/IP network. The Windows CE Networking API is a subset of the vast number of networking functions provided with Windows XP. The Network Communication services provided with Windows CE can be classified into three broad categories – Local Area Networking (LAN) services, Personal Area Networking (PAN) services, Wide Area Networking (WAN) services, General Networking features and Servers.

The set of Windows CE network drivers, protocols and APIs enable three different entities to contribute toward the platform – OEMs to create network enabled devices such as Pocket PCs and SmartPhones, Application developers to create network-centric applications and services, and Hardware vendors to create networking hardware and drivers using the open Network Driver Interface Specification (NDIS) such as for 802.11, Bluetooth, CDMA and GPRS protocols. The Windows CE networking services architecture includes built-in support for wireless networking (Bluetooth and 802.11), Servers (FTP, Telnet, HTTP, PPTP etc.) and an updated implementation of the TCP/IP stack and Network Driver Interface Specification 5.1.

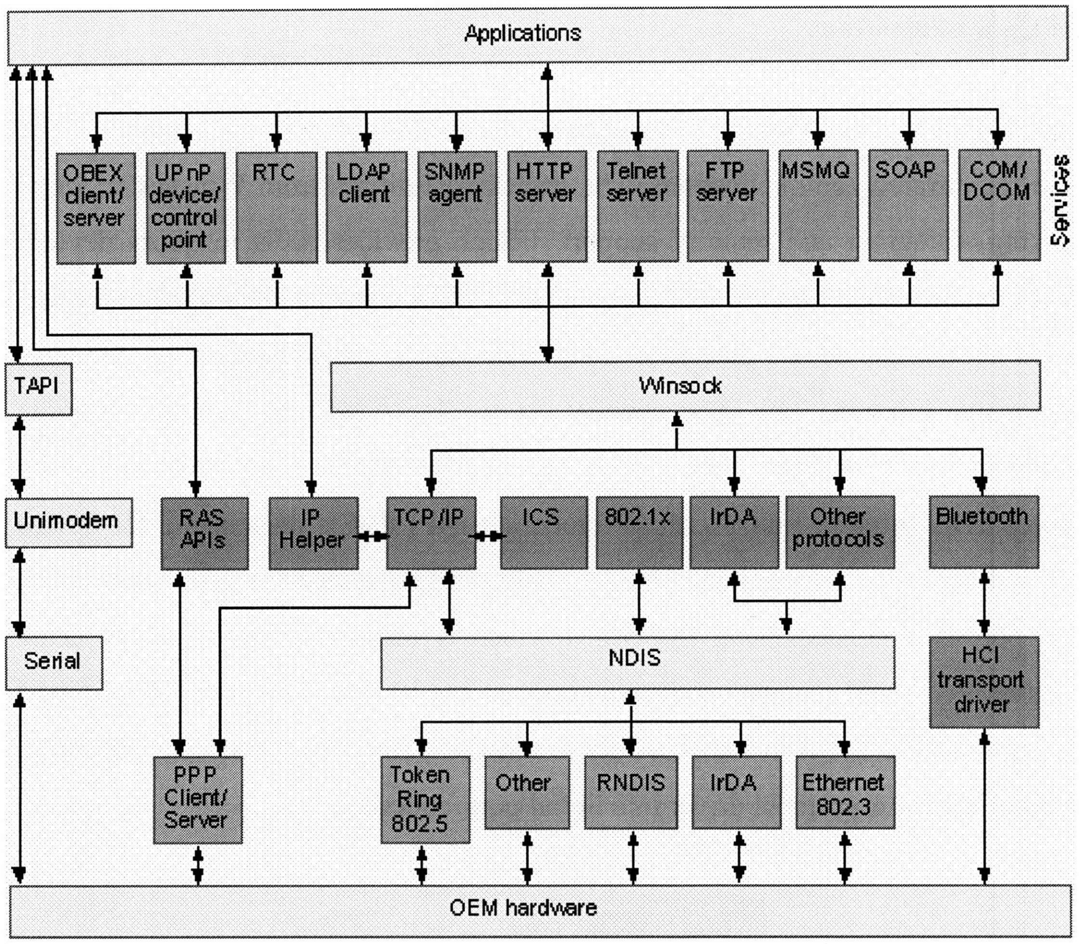


Fig. 20 Windows CE Communications and Networking Architecture[36]

The Windows Networking API is a provider and implementation independent interface to network resources – the WNet API calls are translated into network commands for a particular network protocol. The Windows Sockets API, WinSock provides a sockets interface to the underlying communications layer which could be IrDA, Bluetooth or TCP/IP. The Winsock API in Windows CE leaves out the asynchronous function calls to reduce the library size, but includes support for stream and datagram connections.

[36] http://msdn2.microsoft.com/en-us/library/ms880996.cmcommunicationarchitecture(en-us,MSDN.10).gif (accessed Dec. 2006)

Infrared Communications

Windows CE provides the IrSock API which is a socket-like API built on top of the IrDA stack used for infrared communications. The IrSock API differs from Winsock in terms of addressing, datagram and security support. IrSock provides APIs to query devices ready to communicate over an infrared port while the IrDA stack handles signal collision, interruption and remote device detection[37]. The IrDA stack implementation in Windows CE implements Infrared Link Management Protocol (IrLMP) to manage multiple connections and per-connection flow control, Information Access Service (IAS) and the lowest level IrDA Infrared Link Access Protocol (IrLAP) that is responsible for device detection, link establishment and data delivery between two devices. A serial connection to the IR port is simulated using IRCOMM which is configured transparently using IrSock. Using IRCOMM requires exclusive access to the IR link. The Serial IR Miniport driver is the interface between the IrDA stack and the serial driver and converts NDIS requests from the IrDA protocol driver into Serial port requests.

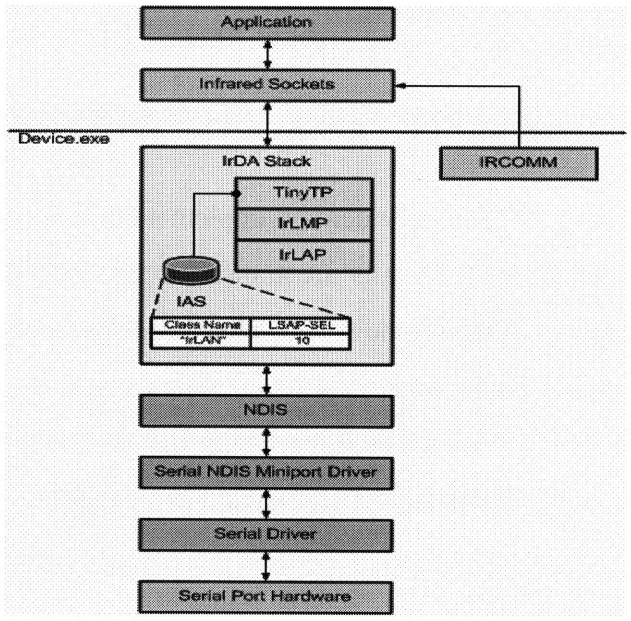


Fig. 21 Windows CE IrDA Stack Architecture[38]

[37] Windows CE 5.0 documentation at MSDN. http://msdn.microsoft.com
[38] http://msdn2.microsoft.com/en-us/library/ms900378.irstack(en-us,MSDN.10).gif (accessed Nov, 2006)

Bluetooth

Windows CE provides integrated support for the Bluetooth protocol. Even so, some OEMs use third party Bluetooth software on their devices instead of the built-in stack. Windows CE supports the Dial-Up networking, LAN access, Object Push and File Transfer Bluetooth profiles and lets OEMs add the capability to support additional profiles. The Windows CE Bluetooth protocol stack implements the functionality for devices to locate each other and establish a connection. The lower two layers in the Bluetooth stack in the diagram below are implemented in hardware, the rest are provided by Windows CE. Applications interact with the Bluetooth stack through one of two interfaces – the Winsock API and virtual serial ports.

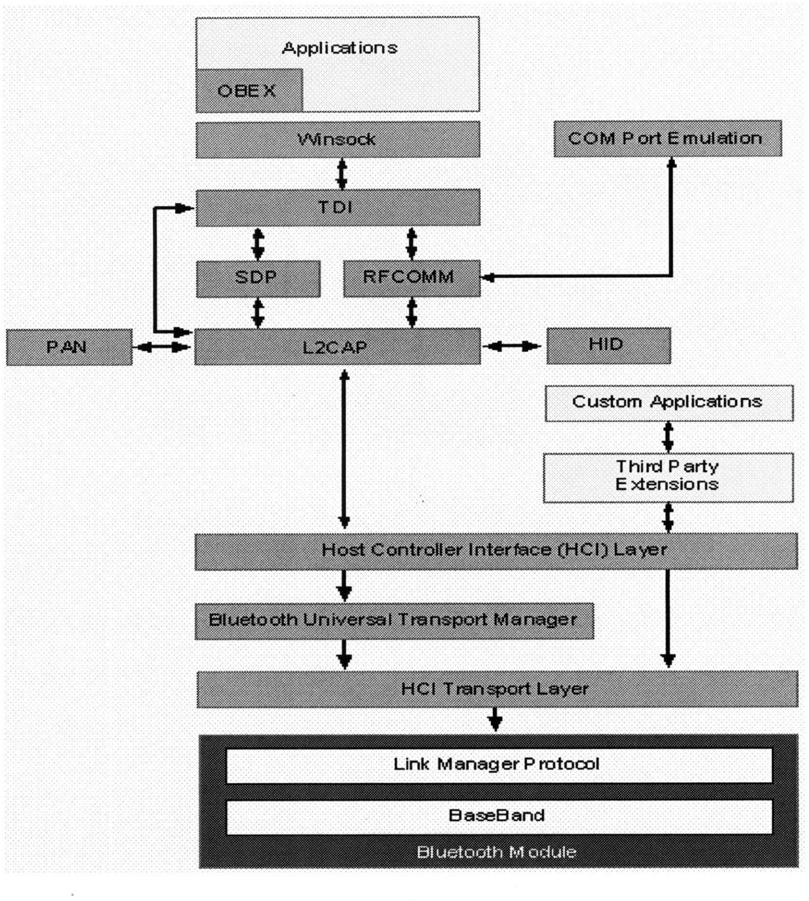


Fig. 22 Windows CE Bluetooth Stack Architecture[39]

[39] http://msdn2.microsoft.com/en-us/library/ms890956.btstackarch(en-us,MSDN.10).gif (accessed Nov. 2006)

Telephony

Windows CE implements a subset of the Microsoft Telephony Application Programming Interface (TAPI). TAPI is a set of APIs that abstracts the call control functionality implemented in different communication protocols using Telephony Service Providers (TSP) and provides a consistent programming interface to applications to make telephony connections[40]. This abstraction of the complexity involved in making connections with different telephony protocols and the use of TSP modules makes TAPI extensible for service providers and uniform to use for application developers. The Telephony Service Provider Interface (TSPI) in TAPI translates TAPI service requests into commands understood by the telephony hardware. Windows CE ships with the Unimodem service provider module and supports installable service providers such as VoIP, ISDN, CDMA etc. from independent software and hardware vendors and OEMs.

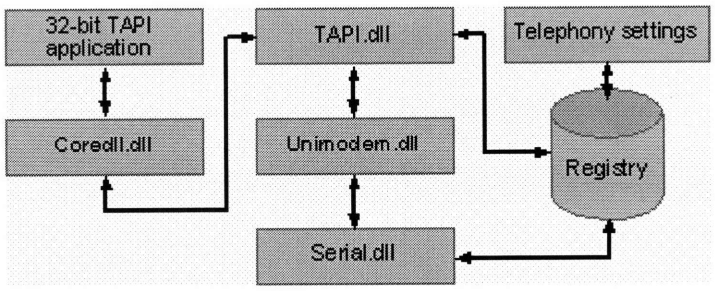


Fig. 23 Windows CE TAPI 2.0 Architecture[41]

TAPI implements comprehensive call control for both data and voice with support for supplementary services like call hold, forward, conferencing etc. It also provides support for call center management by associating data with calls, predictive port and queue management. The Pocket PC Phone edition and Smartphone shells provided by Microsoft add shell specific telephony APIs to simplify access to telephony functionality for applications. These APIs are limited in their functionality and work by wrapping the complex Windows TAPI calls, but are sufficient for simple applications that need to dial a number, view call records, and access the Short Message Service (SMS) subsystem.

---

[40] Windows CE 5.0 documentation at MSDN. http://msdn.microsoft.com

[41] http://msdn2.microsoft.com/en-us/library/ms881883.telephy(en-us,MSDN.10).gif (accessed Nov. 2006)

Security

Windows CE provides a comprehensive toolkit consisting of service providers, APIs and cryptographic libraries for application programs that can be used to secure communications, user applications and data stores. An optional module level security model is implemented by OEMs that designates modules as trusted or un-trusted. Windows CE doesn't support the thread and process-level security models supported by Windows XP[42]. The Windows CE security model is based on modules - .exe and .dll executable code – that are marked either trusted or un-trusted based on validation from the OEM Abstraction Layer (OAL). This validation is obtained when the operating systems loads the module. If the module is trusted, it can access anything in the system, else it is refused access to a few protected functions and registry keys. By leaving the mechanism to certify modules about their trustworthiness to OEMs, Windows CE lets OEMs choose an appropriate module verification scheme and implement a trusted environment for applications to run.

The Authentication Services provided by Windows CE include user authentication, credential management and message protection. This is implemented using the Security Service Provider Interface (SSPI) that allows addition of security packages implementing different authentication and cryptographic schemes like Kerberos and NTLM[43]. With SSPI, application programs can use the different security models without changing their interface to the security system. The Operating System also includes a Credential Manager to provide a single point to track and manage authentication information on the device and Passport Authentication to interact with Microsoft's Passport infrastructure.

[42] "Programming Windows CE .NET" by Boling, 2003, Microsoft Press.
[43] Security Overview in Windows CE 5.0 documentation at MSDN. http://msdn.microsoft.com

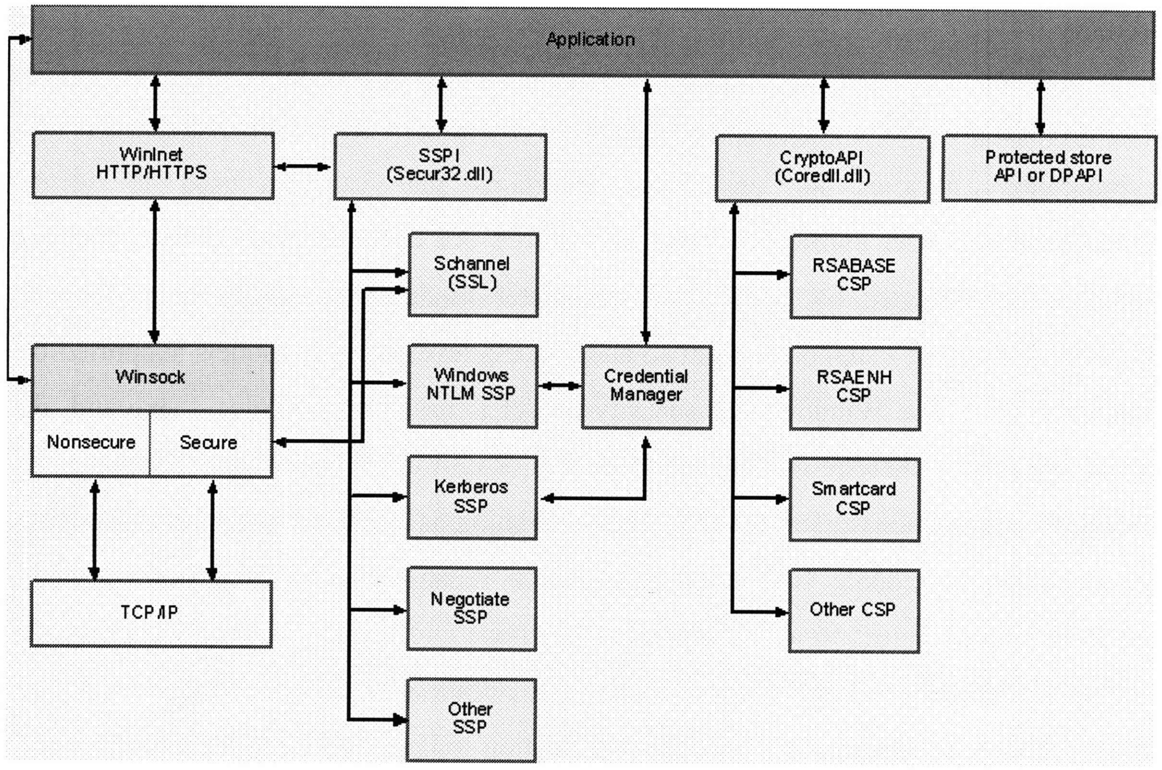


Fig. 24 Windows CE Security Services Architecture[44]

The Cryptographic services provided with Windows CE include pluggable Cryptography service providers (CSP), Cryptography APIs through CryptoAPI, certificate management, and development of customizable public-key infrastructures. CryptoAPI is a set of APIs that allow application developers to encrypt/decrypt data using both PKI and symmetric key cryptography, authenticate and manage digital certificates. Windows CE also implements the Smartcard Cryptographic Service Provider (CSP) for the smartcard subsystem. This is compatible with the Smartcard CSP for desktop Windows OS and provides applications to use CryptoAPI for the Smartcard subsystem.

[44] http://msdn2.microsoft.com/en-us/library/ms925958.security(en-us,MSDN.10).gif (accessed Nov, 2006)

Chapter 3

Evolution of the Mobile Platforms

This chapter will discuss the evolution of the Palm, Symbian and Windows CE Operating System Platforms over the years since their inception. Studying their evolution is interesting because it indicates the different architectures, technologies and subsystems that the Organizations decided to enhance or focus on. This in turn points to the vision into the future of the Organization at that point in time as well as their strategy to capture platform leadership through architectural enhancements of their platforms.

3.1 Palm OS Evolution

The Palm Operating System is a compact, 32-bit operating system designed specifically for handheld computers. A major theme of the development objectives since the first release of the operating system was to keep the user interface simple and the operating system extensible to licensees. The core of the operating system is kept compact to prevent feature bloat and enable it to run in environments with limited resources while its extensibility allows licensees to add differentiated features depending on their hardware.
Palm Operating Systems starting with Palm OS v1.0 and ending with Palm OS Garnet have been targeted to run on Motorola 68K processors. These versions of the Operating Systems are single-threaded and have retained the basic architecture since the original release. Palm OS Cobalt (version 6.x) is a complete rewrite of the original Palm OS designed to run on faster ARM processors, comes with a new OS architecture providing multi-threading support and a set of new APIs for multimedia and communications. Palm OS Cobalt retains compatibility with existing Palm OS 68K applications by including a run-time environment called Palm Application Compatibility Environment (PACE).

The different versions of Palm OS and their year of release are chronicled in the diagram below.

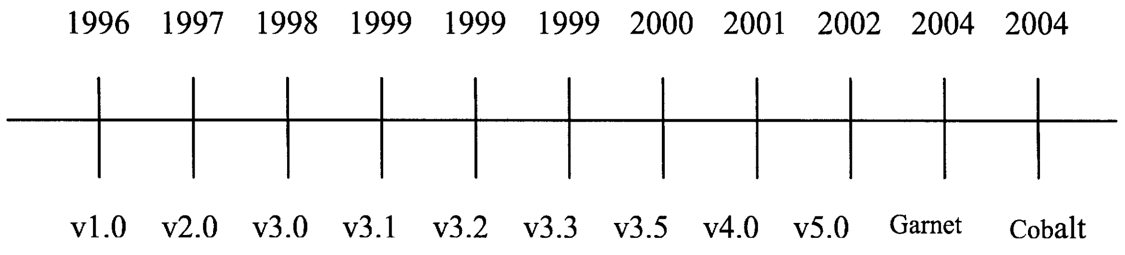


Fig. 25 Evolutionary Timeline of Palm OS

A study of new features added in each release of the Palm OS was done using the archived copies of web pages for http://www.palm.com using the internet archives web site http://www.archive.org. A study of various press releases since 1997 is available at the Palm web site, http://www.palm.com. Also, Palm maintains documentation regarding the developer SDKs and Operating System overviews at http://www.palmos.com/dev/tech/oses/. A historical timeline and changes in each Palm OS version starting from Palm OS v2.0 is provided in the book "Professional Palm OS Programming" by Foster. The following table is a compilation from the above sources and provides an understanding of the evolution Palm OS went through since its original release, Palm OS v1.0 in 1996

OS Version	Release Year	Features
Palm OS v1.0	1996	• Releases Pilot 1000 and Pilot 5000 using the same Palm OS v1.0 designed for one-button access and desktop integration. • Stylus based touch-screen input with Graffiti handwriting recognition software and available on-screen keyboard. • Integrates with Microsoft Schedule+ and Lotus

		Organizer. • "Open" system – third-party applications can be added. • Expandable memory. • Includes Personal Information Manager (PIM) software with five applications. • Data storage using a database model with small chunks of memory that can be relocated anywhere. No file system available to minimize overhead. • 160x160 available display resolution. • Supports Metrowerks CodeWarrior as tool for application development.
Palm OS v2.0	1997	• TCP/IP support added. • GUI enhancements including addition of scroll bars. • New launch codes to enable integration with third-party applications e.g. for phone number lookup from Address Book application. • String manipulation and IEEE floating-point math functions added.
Palm OS v3.0	1998	• GUI enhancements including addition of bold and custom fonts and Progress dialog manager. • IR beaming added. • API changes to include dynamic user interface functions. • Inclusion of unique device ID on hardware with flash ROM. • Dynamic heap memory increased to 96K • Memory architecture changes to configure storage RAM as single heap instead of multiple heaps. • Support for standard MIDI files and asynchronous

		sound playback.
Palm OS v3.1	1999	• Support for contrast adjustment dialog box on Palm devices. • Support for Motorola Dragonball EZ processor. • Unicode support – characters are stored in two bytes. • GUI enhancements for text fields.
Palm OS v3.2	1999	• Clipboard API changes. • Alert dialog box to display application runtime errors.
Palm OS v3.3	1999	• Serial Connection Manager added to support flexible serial connection capabilities. Implementation of IRCOMM standard to support serial connections over IR. Desktop synchronization (HotSync) can be done over IR. • Improved connectivity options to connect to remote systems.
Palm OS v3.5	2000	• Includes the full functionality of OS versions 3.1, 3.2 and 3.3 which are not supersets of each other. • Dynamic heap changes based on memory available to the system. • Support for 1, 2, 4 and 8-bit color and grayscale and APIs to support Color. • Addition of Notification Manager for the system to notify registered applications about events in which they are interested. • Several GUI enhancements including addition of Command bars, Slider and Graphical controls, enhanced Gadget (custom widgets) support, Color Picker dialog, look and feel changes with displaying menu items. • Enhancements to support localization of applications

		without recompilation.
Palm OS v4.0	2001	• Support for Dragonball VZ processor. • Virtual File System (VFS) API introduced to support for a variety of storage schemes with a consistent interface. • Telephony API added to abstract differences between different phones and provides a network protocol independent common programming interface. • Ships with a GSM phone driver to support GSM phones. • Several enhancements to the web clipping application to support color, cookies and ability to download binary files. • Exchange manager enhanced to register exchange libraries and provide a common API for developers and UI for users to choose new transports such as SMS and Bluetooth.
Palm OS v5.0	2002	• Support for ARM processors that improve execution speed and application capability. • Existing Palm 4.0 and earlier based applications can still run on Palm OS 5.0 based ARM devices with the help of Palm Application Compatibility Environment (PACE). • Support for double density displays with resolution up to 320x320 pixel mode. • Enhanced sound support with new APIs to play and record sampled sound. • Inclusion of Web Browser 2.0 for rich wireless content and direct internet access without going through Palm.Net proxies for transcoding. Wireless

		download and installation of Palm applications is also supported. • Enhanced security with addition of encryption / decryption routines and APIs including 128 bit RC4, SHA1, RSA verify and SSL 3.0 / TLS 1.0 services. • COM support for Palm Desktop to allow developers to integrate with and extend Palm Desktop Software.
Palm OS Garnet (v5.4)	2004	• Palm OS Garnet (v5.4) is primarily a maintenance release consolidating all changes in Palm OS v5.2 and Palm OS v5.3 in several international languages. • Support for Graffiti 2 handwriting software through acquisition of Jot product from Communications Intelligence Corporation (CIC) – this has been available since OS v5.2. • Support for QVGA (Quarter VGA) or resolutions of 240x240. This is in addition to the double density (320x320) that was available since Palm OS v5.0 – available since OS v5.3. • API enhancements to Text Manager. • Support for Dynamic Input Area (stylus input area that can be minimized for application display or brought forward when handwriting recognition is needed, thus increasing available display area for applications). • Integrated Bluetooth and Bluetooth Manager UI features.
Palm OS Cobalt	2004	• ARM-native application development. • Complete rewrite of Operating System. • Uses Frameworks based architecture with improved componentization and modularity for licensees to

		provide customized solutions. • Support multi-threading, private virtual address spaces for applications for increased robustness and application flexibility. • Increases supported memory to 256MB for ROM and RAM. • Finally moves to GDI support to isolate OS drawing operations from low level graphics hardware. • Support for Dynamic Input Area (stylus input area that can be minimized for application display or brought forward when handwriting recognition is needed, thus increasing available display area for applications). • Multimedia enhancements to support a flexible architecture for audio and video – support for ADPCM/PCM, MP3 and MPEG1/MPEG4 becomes standard. • Comprehensive security enhancements through inclusion of authentication, authorization managers and pluggable security providers. Enables end-to-end secure communications through SSL/TLS and enhances application security through code-signing and secure desktop synchronization. • API changes include replacement of Palm-specific Net library with standard Berkeley sockets API. • Integrated Bluetooth support and new APIs for telephony call management. • Improved graphics for 2D rendering, complex drawing and anti-aliasing.

3.2 Symbian OS Evolution

The Symbian Operating System is a flexible and scalable operating system designed specifically for mobile phones. In order to cater to a wide variety of phone equipment with different form factors from a variety of manufacturers, the core of the operating system is separated from the user interface. This allows manufacturers to develop the right User Interface for their variety of devices and target different market segments. Currently four different user interfaces are provided – UIQ, Series 60, Series 80 and FOMA SW Platform. This thesis will trace the evolution of the Symbian OS itself and not the UI layer that has an evolutionary path of its own.

The different versions of Symbian OS and their year of release are chronicled in the diagram below.

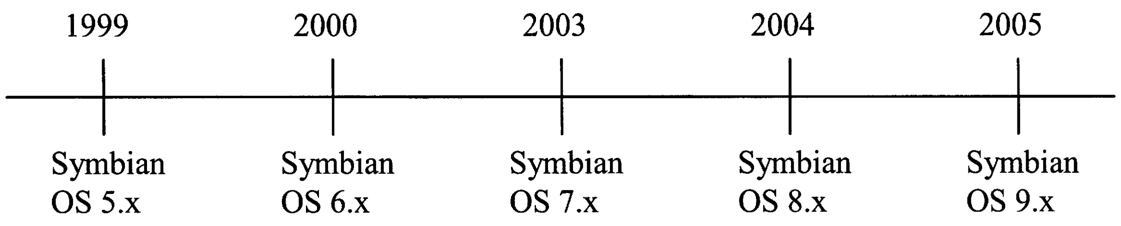


Fig. 26 Evolutionary Timeline of Symbian OS

The following table chronicles the technologies and features included with each new release of the Symbian Operating System. The table is compiled by analyzing the product functional descriptions starting from v8.0 onward at http://www.symbian.com/symbianos/releases/symbianosreleases.html and the combing archived internet pages for symbian.com at http://www.archive.org. This thesis traces the evolution of the Symbian OS itself and not the EPOC operating system from Psion that forms the basis for the first Symbian release. The first Symbian release v5.0 was in 1999 and was also called EPOC Release 5 or ER5. The current stable release of the Operating

System is v9.3 released in July, 2006. The release year in the table indicates the year the first major release of that version.

OS Version	Release Year	Features
Symbian OS v5.x	June 1999	• First release of Symbian OS 5 or ER5 is delivered as a set of components to device manufacturers. It consists of Base (runtime and tools for building ROM, emulator and Windows Utility package), Engine (component with any user interface which is used by application engines), Graphics (components for drawing, printing, fonts and views), and System GUI (GUI, System shell, control panel and other components that define look and feel). • Runs on ARM based CPUs • Provides a full multi-tasking infrastructure with process scheduling, memory and power management, timers, file system, keyboard, PC card, CF-removable media. • Development tools are provided as EPOC C++ SDK and OEM Adaptation Kit (OAK) for building ROM images for new devices. • Operating System is developed in C++ and application programming interface to the OS is exposed using C++ constructs. Provides a EPOC C Standard library with many of ANSI C facilities including string, file i/o, TCP/IP sockets, processes, pipes and blocking i/o. • EIKON is provided as the native GUI framework

		which is replaceable by device manufacturers with their own customized User Interface. • Designed for devices with color or grayscale displays with 640x240 screen sizes, alphanumeric keypads, and pen input via screen digitizer. • Connectivity options include TCP/IP, dial-up, telephony API with call control and phonebook support. • PC Connectivity for synchronization and IR support for data communication with other devices is included.
Symbian OS v6.x	2000	• Releases two reference designs for device manufacturers • Quartz is a tablet size, palm form factor with pen operation, 240x320 color screen, tablet design specific GUI and navigation, handwriting recognition and an integrated task-based application suite • Crystal is a powerful, keyboard-based wireless information device for professional and power users with keyboard operation, 640x200 screen with soft keys, GUI evolved from earlier communicator based versions and a comprehensive application suite. • Both reference designs use the same generic technology components – multi-tasking kernel, data management, graphics, communications, multimedia, and security. • Data synchronization with PC and server-based

		data. • Application development options include EPOC C++, Java, WAP and HTML. • Integrated wireless telephony for Communicators and Smartphones that combine data and voice seamlessly. • Integrated Bluetooth, WAP and GPRS based packet data. • Security options include full-strength encryption and certificate management, secure communications (HTTPS, SSL and TLS) and validating application installations. • Implementation of PersonalJava 3.0 and JavaPhone 1.0. • PC Connectivity enhanced through Symbian Connect connectivity package. • GUI subsystem componentized from single library to ease implementation of multiple reference designs. • Support for several audio and image formats through media server.
Symbian OS v7.x	2003	• Improves code reusability by introducing the ECom plug-in architecture. This is a generic framework for specifying plug-in interfaces, calling and writing plug-ins. • New transport framework architecture that provides a common interface to HTTP and WAP. • Support for IPv6 and simplified API to use secure sockets.

		• Media support and API to handle streaming audio, support for 2D hardware graphics accelerators. • Metrowerks CodeWarrior is supported for Symbian C++ development. • Support for Sun's Java MIDP APIs that are specialized for mobile phones. • Support for Multi-Media Messaging (MMS) that supports sending multimedia messages in a SMS fashion. • Network Adaptor Framework specified for licensees and third parties to use for development of adaptors for various network interfaces. • Support for SyncML client API to enable data synchronization with standards based SyncML server. • New telephony APIs to provide a common interface to multiple air interfaces including GSM, CDMA and WCDMA. • Support for USB, Multimedia card and hardware accelerator support added to base operating system.
Symbian OS v8.x	2004	• Releases two application compatible variants of the Operating System with different kernels – EKA1 is the legacy kernel evolving from v7.0 and EKA2 is the new hard real-time kernel with improved real-time capabilities. • Provides a Device management framework using OMA SyncML 1.1 to enable network operators and enterprises to remotely manage and

		configure Symbian OS phones reducing in-market deployment and management costs. • Adds Media Device Framework (MDF) to provide a hardware abstraction layer for multimedia hardware acceleration to deliver high performance multimedia applications such as video streaming. • Enhanced support of Java to allow developers to access native functionality of Symbian OS and become fully compliant with network operator Java requirements and Java Community standards. • Support for SDIO standard to allow licensees to incorporate SD memory and I/O cards.
Symbian OS v9.x	2005	• Includes RTP (Real time Transfer Protocol) stack natively. • Additional Device Management capabilities including support for OMA Device management to enable enterprises and operators to manage phones in the field. • Includes a capability based security model to enhance platform security against malware. Also includes support for application specific, private secure data stores. • Native support for WLAN 802.11 and High speed downlink data access (a 3G wireless protocol) added.

3.3 Windows CE Evolution

Windows CE is Microsoft's operating system for resource constrained embedded devices like industrial controllers, GPS devices, handhelds and home equipment. It is the base operating system for Windows Mobile devices that include Pocket PC, Pocket PC Phone Edition and Smartphone devices. The first release of Windows CE, version 1.0 was in 1996. It was primarily meant for handheld devices (smaller laptop-like devices without less memory and CPU power than traditional notebook computers) and shipped with lighter versions of Excel, Word, and Internet Explorer with reduced functionality to minimize code size. The current, stable version of Windows CE is version 5.0 and is the base operating system for the Windows Mobile family of devices. Windows Mobile 5 has faster, better productivity tools reflecting the higher CPU power and memory capabilities of current mobile device hardware. It has better graphics capabilities including implementation of Direct3D for gaming and multimedia applications, Digital Rights Management (DRM) integration, and default storage of user and application data in persistent, non-volatile storage.

The different versions of Windows CE and their year of release are chronicled in the diagram below.

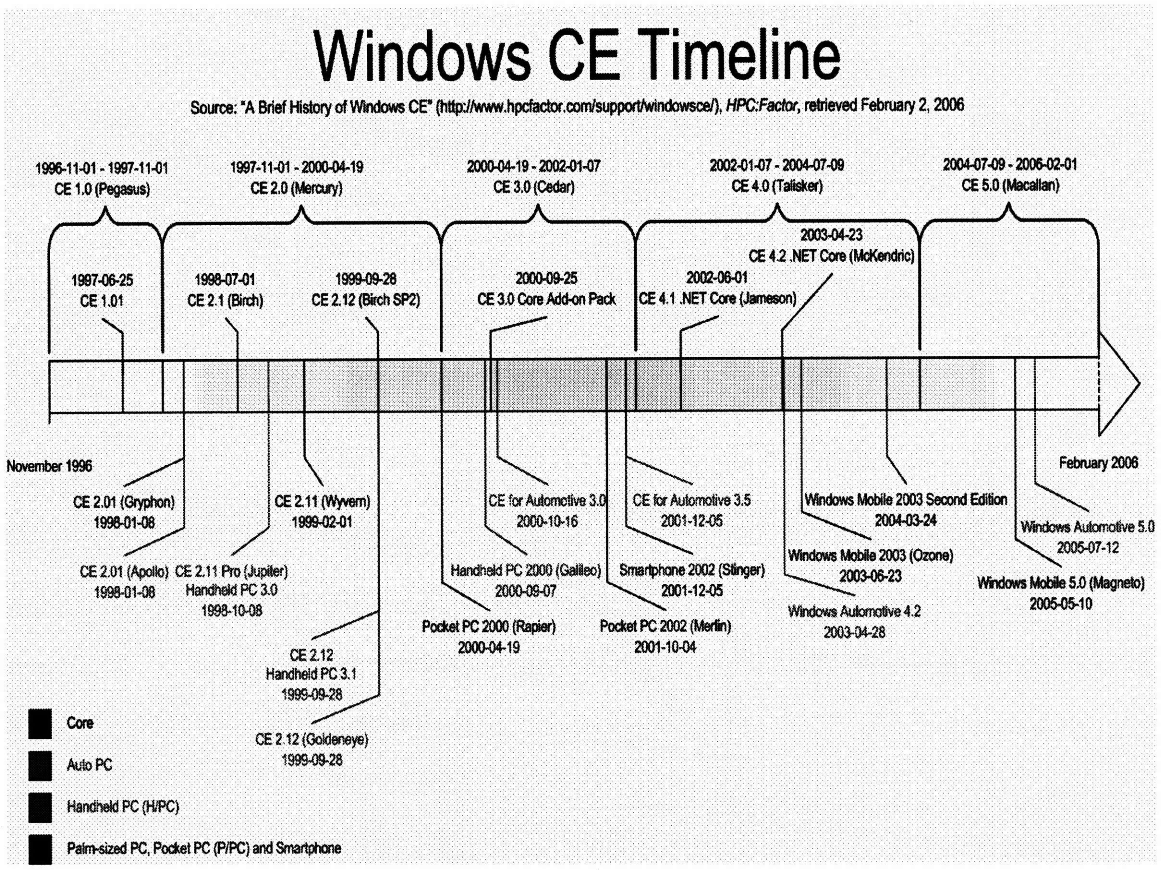

Windows CE Timeline
Source: "A Brief History of Windows CE" (http://www.hpcfactor.com/support/windowsce/), HPC:Factor, retrieved February 2, 2006
1996-11-01 - 1997-11-01 CE 1.0 (Pegasus)
1997-11-01 - 2000-04-19 CE 2.0 (Mercury)
2000-04-19 - 2002-01-07 CE 3.0 (Cedar)
2002-01-07 - 2004-07-09 CE 4.0 (Talisker)
2004-07-09 - 2006-02-01 CE 5.0 (Macallan)
1997-06-25 CE 1.01
1998-07-01 CE 2.1 (Birch)
1999-09-28 CE 2.12 (Birch SP2)
2000-09-25 CE 3.0 Core Add-on Pack
2002-06-01 CE 4.1 .NET Core (Jameson)
2003-04-23 CE 4.2 .NET Core (McKendric)
November 1996
February 2006
CE 2.01 (Gryphon) 1998-01-08
CE 2.11 (Wyvern) 1999-02-01
CE 2.01 (Apollo) 1998-01-08
CE 2.11 Pro (Jupiter) Handheld PC 3.0 1998-10-08
CE 2.12 Handheld PC 3.1 1999-09-28
CE 2.12 (Goldeneye) 1999-09-28
CE for Automotive 3.0 2000-10-16
Handheld PC 2000 (Galileo) 2000-09-07
Pocket PC 2000 (Rapier) 2000-04-19
CE for Automotive 3.5 2001-12-05
Smartphone 2002 (Stinger) 2001-12-05
Pocket PC 2002 (Merlin) 2001-10-04
Windows Mobile 2003 Second Edition 2004-03-24
Windows Mobile 2003 (Ozone) 2003-06-23
Windows Automotive 4.2 2003-04-28
Windows Automotive 5.0 2005-07-12
Windows Mobile 5.0 (Magneto) 2005-05-10
Core
Auto PC
Handheld PC (H/PC)
Palm-sized PC, Pocket PC (P/PC) and Smartphone

Fig. 27 Evolutionary timeline of Windows CE OS[45]

The following table indicates the technologies and features included with each release of Windows CE Operating System. This addition is incremental and application programs are backward compatible with the previous release of the Operating System. The table is an aggregation of data using analysis of the readme files provided with each developer SDK release corresponding to the release version of the Windows CE Operating System and a study of the comprehensive historical timeline maintained at http://www.hpcfactor.com/support/windowsce.

[45] http://upload.wikimedia.org/wikipedia/commons/c/cb/Windows_CE_Timeline.png (accessed Nov. 2006)

OS Version	**Release Year**	**Features**
Windows CE 1.0	Late 1996	• Simple “Organizer” Operating System. • Low power hardware mandated on OEMs. • 32-bit Operating System. • Mandated displays to be half-VGA or 480x240 with 4 grayscales and 2 bits per pixel • Windows 95 like interface. • Unicode used throughout OS to target international markets. • Terminal Emulation and Point-to-Point Protocol for e-mail. • Remote Networking and Remote Access Service Support • PC Sync client • Shipped with mobile versions of Excel, Word, Internet Explorer, Outlook and Microsoft Personal Information Manager (PIM).
Windows CE 2.x	Fall 1997	• Modularization of the Handheld Operating System to be called Microsoft’s first Embedded Operating System. OEMs could take different parts of the scalable, low footprint Windows CE OS and create diverse range of devices based on it using the Windows CE Embedded Toolkit (ETK). • Support for Color half-VGA and full-VGA (640x480) with 24 bits per pixel and True Type Fonts. • Internet Explorer, Word, Outlook, Pocket

		Access and PowerPoint were bundled with the OS. • Kernel and NDIS miniport driver model support for a broad range of peripheral devices including Ethernet adaptors, modems, IR, GSM. • Support for VGA adaptors and PCMCIA/CF memory modules. • Object store size increased to 16MB and support for Microsoft Message Queue added. • COM support for in-proc servers. • FAT32 file system and installable file system support • Software Input Panel (SIP) to display virtual keyboard. • Fast IR and USB support • Inclusion of the C runtime library in the Operating System to reduce the size of the OS and application programs. • Release of the first "Palm PC" based on Windows CE 2.01 without Pocket Office and Explorer. (Later renamed to "Palm-size PC" after a trademark infringement suit by 3com). • Includes "ActiveSync" for desktop synchronization.
Windows CE 3.x	Mid 2000	• Complete User Interface rework of the Palm-size, keyboard-less, Windows CE based devices to compete directly with Palm OS. The Windows CE, Palm-size devices started to be

		called Pocket PCs. • Pocket PC based devices replaced the Windows 95 like CE Shell with a Pocket PC shell reflecting the Microsoft's differentiation strategy for Pocket PC. • Core Performance enhancements including better real-time support, increased number of thread priorities (256), adjustable thread quantum, nested interrupt service routines and reduced kernel latencies – this led to development of enterprise class applications. • Adoption and development of Handheld PC devices started trailing off while Microsoft's entire marketing started to get Pocket PC oriented – the distinctiveness of a Handheld PC providing the Office Suite of productivity applications diminished with the increased functionality of the Pocket PC class of devices. • Full COM out-of-proc and DCOM support. • Object Store increased to support 256MB of RAM. • File Size limits increased to 32MB per file. • Improved multimedia support through media player. • Networking support for XML, PPTP (Tunneling protocol), ICS and remote desktop display. • DirectX API for graphics added.
Windows CE .NET	Early 2002	• Changes in virtual memory management which resulted in doubling of available virtual

		memory per application. • New Driver loading model and system services support. • Bluetooth, 1394 (Firewire) and 802.11 wireless networking support. • Support for IPv6, Winsock 2 API addition, Power management. • Support for running managed applications through inclusion of Microsoft's .NET runtime in the form of .NET Compact Framework. • Pocket PC Shell APIs and other functionality supporting menu bars, soft input panel (SIP – the virtual keyboard) were moved to the base OS – Windows CE to enable OEMs using Windows CE to support running PocketPC applications.
Windows CE 5.0	Mid 2004	• Support for Networked Media Device (NMD) to allow audio and video playback on home networks. • Supports Windows media DRM (Digital Rights Management) to allow Windows CE based devices to access protected media content. • Digital Video Recorder (DVR) functionality added to support creation of set top boxes. • Increased support for Platform Builder tools including Production Quality OEM Abstraction Layer (OAL) for certain hardware platform Board Support Packages and Production Quality device drivers to simplify

		and bring faster-to-market custom, embedded OS designs. • 32-bit Cardbus support for PCMCIA. • Support for Direct3D Mobile – a subset of Direct3D API on desktop systems – to provide drawing services for Direct3D middleware. This enables better graphics support for games and multimedia applications. • Additional functionality and managed code APIs added to .NET compact framework. • Improved connectivity with Microsoft Exchange Server and Active Directory Server to enable tighter desktop integration. • Inclusion of Windows Messenger. • Improved WLAN support for 802.11 to build native WLAN Access Points and Stations. • Keyboard Layout and Input Method Editors for Multiple languages. • Security support for applications enhanced – added security APIs for cryptography and authentication libraries. Adds Local authentication system for modular device locks like smart card plug-ins.

3.4 Analysis

The chronology of Palm OS evolution shows the incremental features being added to the OS to support evolution in the device hardware itself. The original Palm OS architecture changed little since the first release in OS until 2004, when support for ARM processors led to complete restructuring of the OS in the form of Palm OS 6.x or Cobalt. In contrast, the original Symbian architecture supported ARM in 1999 along with multitasking and virtual memory for increased OS stability. This original architecture has changed little, but evolved consistently with additional features to support new device capabilities adhering to the same plug-in architecture. The first release of Windows CE was based on the original Win32 architecture from Windows 95 and has retained that basic architecture. It has evolved from pure handheld (HPC) support in 1996 to current Smartphone and Pocket PC shell support based on the same base Operating System. Since the first release of Windows CE in 1996, the focus of Microsoft's innovations for the mobile device market seems to be in branding – or specifying – a set of standards and specifications in hardware and user interfaces for device manufacturers. This seems to reflect Microsoft's desire for device users to see a standard Windows-like interface on their devices, as an extension to their Windows OS desktop.

Connectivity

Palm OS Connectivity support evolved from initial desktop PC synchronization (1996), through TCP/IP (1997) and IR (1998). Palm OS v2.0 added TCP/IP support for Palm devices to connect to the Internet, but internet browsing support was in the form of transcoded HTML for Palm specific device displays using Web Clipping. Direct connectivity without going through Palm.Net came only in Palm OS v5.0 in 2002. Palm OS telephony API support came with v4.0 in 2001 and integrated Bluetooth support came with v5.0 in 2004. Symbian support for TCP/IP, dialup telephony, telephony API for call control, IR and desktop synchronization came with the very first Symbian OS release in 1999. The second Symbian OS release in 2000 added integrated support for Bluetooth, WAP and GPRS. The very first release of Windows CE in 1996 had PPP (Point-to-Point

Protocol) to access email, remote access and remote networking support. In order to support a broad range of peripheral hardware, Windows CE adopted a NDIS miniport driver model with the second Windows CE release in 1997. Fast IR and USB support came in the same release. Integrated Bluetooth, Firewire and WLAN (802.11) support came in 2002. The release of Windows CE 5.0 in 2004 included support for devices to participate as a networked media device for home media connectivity.

Symbian OS has provided the most open and standards based architecture for external connectivity and communication since its original release, whereas Microsoft relies on using the same standard Windows-like interface to communications and interfacing to external devices. Palm OS has the greatest increase in communications and connectivity capabilities since its first release. This seems to reflect a desire on the part of the OS developers and designers to stay true to the "simple OS" philosophy and see a Palm OS device as a basic personal organization device.

Security

Palm OS added comprehensive API support for security including addition of cryptographic libraries for encrypting/decrypting came in ver5.0 in 2002. The second release of Symbian OS in 2000 added integrated support for high strength cryptographic libraries for encryption/decryption and certificate management. Symbian OS v9.1 in 2005 added a capability based security model with installation of Symbian-signed applications to protect against malware. Windows CE 5.0 in 2004 added OS support for security with built-in cryptographic and authentication libraries.

All three Mobile Operating Systems have similar levels of security support with built-in cryptographic and authentication libraries and application programming interfaces to these libraries. Symbian was the earliest to add this support and Windows CE the last.

User Interface

The first User Interface in Palm OS used innovate display controls specific to the small screen available in handhelds. The next big step in innovation came with Color support in Palm OS 3.5 in 2000. This release also added several User Interface controls and GUI enhancements and APIs to support color. Several Multimedia enhancements including enhanced sound support and double density displays appeared in Palm OS v5.0 in 2002. Symbian OS supported color natively in its very first release in 1999. The second release, in 2000 added support for different display resolutions and screen sizes, stylus or keyboard based input in two reference designs for device manufacturers. Extended graphical support including 2D hardware accelerators, streaming audio and synchronization using open-standards based SyncML was also added in 2003. Evolution of Multimedia support continued with releases in 2004 and 2005 to support video streaming including addition of a media device framework to provide hardware abstraction for multimedia hardware acceleration. The first Windows CE release in 1996 had a Windows 95 shell and reflected Microsoft's desire for the Handheld PC (HPC) to be seen as a desktop extension. The second release in 1997 added full VGA color support. This release also heralded Microsoft's strategy for a small, real-time OS for embedded devices - the Operating system was repackaged and re-componentized for device manufacturers to adapt for their specific use. It was thus, no longer a phone or handheld operating system, but a base for a wider family of consumer devices. Reflecting the growing availability of Pocket PC applications, Microsoft folded Pocket PC shell specific APIs into the base Windows CE OS to allow different device manufacturers to run the same Pocket PC applications. Windows CE 5.0, released in 2004 has native support for Direct3D to enable development of highly functional games and multimedia applications.

Core OS (Kernel) Features

Multi-threading support, virtual memory and framework architecture with componentization for application programming came only with Palm OS Cobalt (v6.x) in

2004. This was also the first release with improved multimedia support including 2D graphics rendering, complex drawing with anti-aliasing, and advanced audio. Abstraction of API interfaces to support multiple implementations transparent to client applications started with Palm OS v4.0 in 2001 - it included VFS API to provide a storage-independent interface to the file system and exchange manager to support a transport independent messaging interface.

The very first release of Symbian OS in 1999 had the foundation for modern mobile operating system architecture. This release had a microkernel with multithreading and virtual memory support, abstraction and separation of windowing layer from the core OS to allow UI customization, and core device services running in user mode to support robustness. A plug-in architecture to improve code-reusability with a generic framework for specifying services and using them was added in 2003 with the ECom plug-in architecture.

The Windows CE Operating System has been designed since its first release to be a compact, real time operating system incorporating a subset of the Win32 API used to program desktop Microsoft Windows Operating Systems. It is a multi-threaded, multi-tasking operating system with virtual memory to separate application processes from tripping on each other's memory and ensures OS robustness. The chief aim of the designers of Windows CE was to allow Win32 programmers of the desktop Windows Operating Systems to continue using the same APIs and programming paradigms – messages loops, event handling frameworks on the smaller Windows CE platform.

Palm OS and Symbian OS support Java applications by providing a Java runtime and ensure compatibility with Java community standards. In contrast, Windows CE uses .NET as the technology to provide hardware independent, type safe run-time environment to run code in a secure way.

Chapter 4

Platform Analysis of Mobile Operating Systems

The Mobile Operating Systems studied in this thesis have a common theme – they have all been specifically designed for mobile, small form factor, computing devices with a display interface that provide communication and data organization capabilities. These Operating Systems differ in their architecture, their core design goals, and services offered to users and device manufacturers.

The Mobile device ecosystem consists primarily of three different entities – Consumers, Device Manufacturers and Operating System providers. The reality of voice and data integration along with the rapid increase in computational power offered in small form factor has led to increasing convergence of voice-centric mobile devices like cell phones and data-centric mobile devices like Personal Digital Assistants (PDAs). Network Service Providers and Operators like NTT DoCoMo play an important role in shaping the complete wireless connectivity experience with added applications and services for end users. Thus, they also constitute another important entity in this ecosystem. An Operating System vendor needs to address the needs of all these entities through innovation and evolve the mobile communication platform.

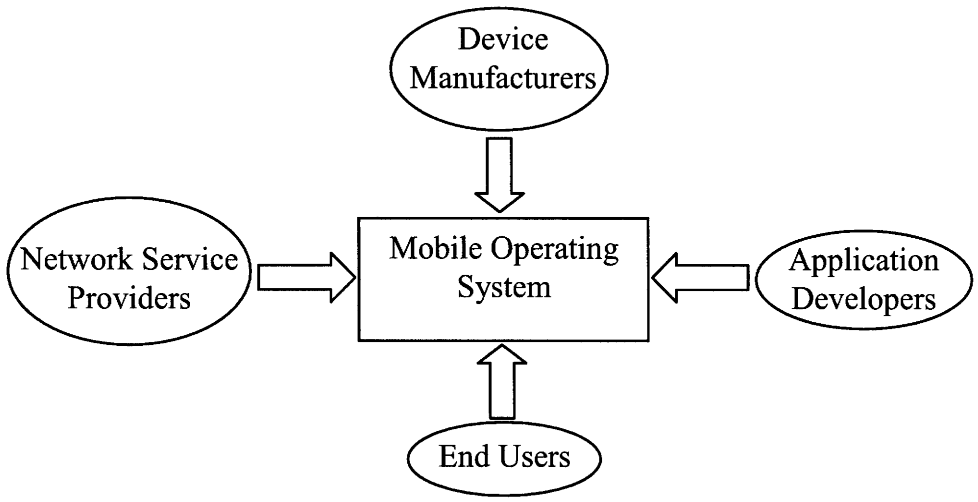


Fig. 28 Mobile Platform Eco-System

Competition among device manufacturers in the mobile device space has resulted in increasing device functionality offered to customers and thus to complexity in the Operating System software that manages the hardware and interfaces with end users. Improved connectivity and multimedia capabilities of these devices has allowed Network Operators and third-party service providers to develop value added service offerings for end users. Separation of concerns is becoming increasingly necessary in this environment to compete effectively – no longer can device manufacturers make their own Operating Systems for their devices as was the case in the days of the early personal organizer devices.

Mobile Operating System providers play an important role in the Mobile device ecosystem as they control the interface to the users and provide the platform for application developers of value added services. The OS providers aspire to gain platform leadership in this industry through innovation, an architectural vision and a strategy to carry together innovations in each of the other entities of the ecosystem for a coherent platform evolution. Using the second lever – product technology and architecture – in the Platform leadership framework proposed by Cusumano and Gawer, this thesis studies the degree of openness of a product's architecture, interface and intellectual property that a Company is ready to reveal to other component makers of the Platform.

4.1 Interface Design

Programming Interfaces are sets of interface descriptions such as function signatures or types of messages that can be received and accepted by a module. A Programming Interface provides for separation of concerns between two parties – the developers of the interface and its functionality, and the developers wishing to use it to build their own different functionalities[46]. Implementation of the functionality behind the interface can be changed as needed while keeping the interface provided stable. Thus, it results in a written contract between two parties and enables their communication. A modern programming interface consists of the following components:

- **Function signature** – Name of the function, number and type of arguments
- **Interface constants** – Numbers and Names used to give a uniform meaning and context, for both the interface developer and user
- **Protocols** – A specific paradigm or pattern whose intent is well understood by the interface user and used correctly for proper application execution. This also includes assigning a uniform meaning to system resources like files and environment variables.

Application Programming Interface

An important characteristic of an API specification is that once specified and released for public usage, it is to be maintained and supported for a long time. This is done so that the API user investments in learning the API and usage are preserved. Thus, addition of new interfaces to the API specification can and should occur, but removal is not allowed. Still, a mechanism to evolve the API to accommodate enhancements in functionality and performance is to be needed. This requires development of an API specification that hides implementation details of the functionality behind the interface and provides tools and features that enable evolution of the API.

[46] How to Design a Module (API) – Netbeans API Tutorial at http://openide.netbeans.org/tutorial/api-design.html (accessed Nov, 2006)

Usability studies play a key role in measuring effectiveness of the API specification and help provide a framework for making design decisions. By comparing usability differences in API design choices while controlling biases toward a particular API, implementers can make decisions on the design of APIs that use these patterns[47]. API specifications also ensure portability by requiring all implementers of the API to conform to the common interface points. Applications written using the specified API should run without issue on all systems implementing the API. In a way, the API specification helps in setting a standard to which all implementations conform.

The Symbian OS API has been designed around providing application frameworks to developers. The native C++ interface to the Operating System which itself has been developed in C++ and uses these frameworks, results in an easy learning curve for developers interested in developing for the Symbian platform. Also, use of application frameworks and Symbian OS specific idioms helps in creating efficient, reliable and complex program implementations.

Microsoft's strategy for its API is to make it easy for legions of existing Win32 developers to develop applications for the Windows CE platform. The Windows CE API is a subset of the desktop Win32 APIs with the same syntax and programming idioms. Most of the existing desktop Win32 applications can be easily ported to run on Windows CE through adaptation to the smaller display size of Mobile devices.

The Palm Operating System has been written in the C programming language and has a C language API to access the system functionalities. The Palm OS API reflects the designers' motivation to give priority to lower program memory requirements than speed and efficiency of execution. Consequently, 16-bit integers are preferred to 32-bit integers, unwieldy bit fields are used to conserve memory and Palm's own implementation of standard ANSI C functions with different signatures (StrCopy instead of strcpy for

[47] Informing API Design through Usability Studies of API Design Choices: Stylos. J, IEEE Visual Languages and Human-Centric Computing, 0-7695-2586-5/06

instance) are used. The choice of the C programming language for application interface description means that Application programs have to simulate object oriented programming patterns without the native language support[48]. Thus, the Palm API designers had to use struct instead of class and use a type indicator inside the struct to hint about the type of object when the struct is a union of different types of objects. Providing a native C++ API, with the system interfaces modeled as classes and objects using Object Oriented techniques would provide a more natural interface to the Operating System, especially a GUI intensive OS like Palm.

Service Provider Interface

Another related interface specification provided by a platform developer that intends its product to be used by different vendors is the Service Provider Interface (SPI). An example of the SPI is the Device implementation side of the Serial Manager API provided by the Palm OS. The SPI is the common interface provided by the Operating System to implementers providing hardware or software services that in turn are used by Application programs.

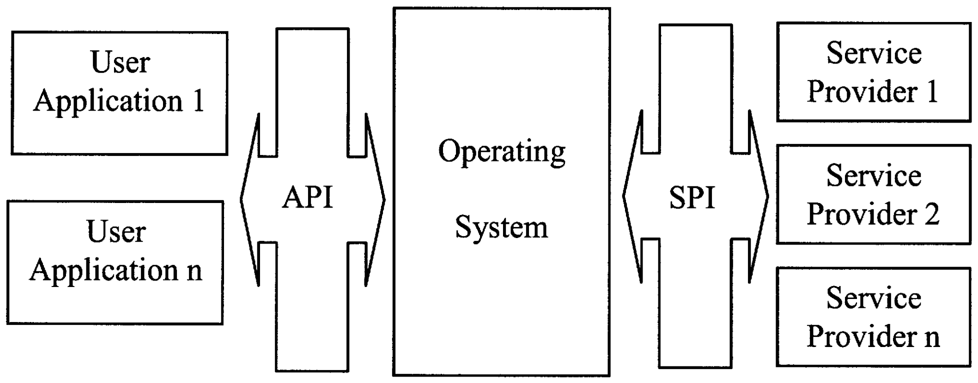


Fig. 29 Operating System Interfaces – API and SPI

[48] An application with variations as used in teaching a Palm programming course: Maurer, D. May 2003. Journal of Computing Sciences in Colleges, Volume 18 Issue 5

The SPI interface provided by Symbian follows the framework methodology it uses for application development. Symbian implements common service provider architecture with pluggable modules that OEMs use to adapt for their hardware. The most significant differentiated architecture that Symbian provides is the abstraction of the User Interface functionality through creation of the UI Control Framework middle layer. This allows device manufacturers to develop different concrete User Interface control libraries at the top layer to customize and differentiate their devices based on device capabilities – pen/keyboard based input, display size/resolution etc. Windows CE and Palm OS provide a similar Service Provider Interface for licensees to use for device customization. Microsoft specifies and imposes certain hardware and software look-and-feel requirements for its licensees to use the Pocket PC or Smartphone branding – thus, OEMs can't differentiate their devices on the basis of primary User Interface. All Pocket PC or Smartphone devices are required to have the same User Interface developed by Microsoft. Palm OS licensees can customize the User Interface provided on their Palm powered devices there is no architectural support in the Operating System to achieve this.

Documentation

Documentation is another key aspect of an API specification. It provides details in plain text to the API user of what the programming interface is supposed to provide as functionality. To be useful, the documentation should also include code examples, conceptual overviews, and definition of terms, programming guidelines, known bugs and workarounds[49].

Symbian, Windows and Palm provide comprehensive documentation of their APIs and OS architecture. This is delivered through their Software Development Kits (SDKs) as well as online on their web sites. Microsoft uses the wide availability of Win32 documentation to its advantage by using a subset of the Win32 API for Windows CE programming.

[49] API Documentation from Source Code Comments: A Case Study of Javadoc by Kramer, D. Oct 1999, Proceedings of the 17th annual international conference on Computer documentation

4.2 Architecture

Operating Systems designed for mobile devices have key characteristics that are quite different from those required for a general purpose desktop OS. There are a lot of differences among mobile devices – in terms of screen size, method of interaction (touch screen, keyboard, and voice activation), voice and data centric applications and capabilities. Adapting a desktop operating system for a mobile device by scaling down features and service causes fundamental design compromises in the mobile operating system. A good design follows from function and hence a mobile operating system should be architected keeping in mind the key characteristics of mobile devices – limited user interface – both input (keyboard) and output (display), connectivity (to different wireless infrastructures incorporating different technologies deployed by various operators), extensibility (by third party hardware and software vendors) and differentiability (to target markets and by licensing OEMs).

Mobile applications are a fast growing market and Organizations developing business and consumer applications are adapting their products and services so they can take advantage of opportunities in these markets – by adapting their existing product for mobile use, providing a mobile interface to their product, or developing new mobile products with characteristics specific for mobile use.

Successful Application development for the mobile device needs[50]

- Understanding of challenges in User Interface design for small screens
- Context of the environment in which the application is used – mobile tasks are generally short and quick. The interface should be easy to grasp and interaction with user be quick.

[50] Driving Devices: Lessons Learned in the Business of Designing Mobile UIs: Frank, B. ACM, Interactions, July & August, 2006. ACM 1072-5220/06/0700

- Development of mobile application is a) Context or use oriented b) Rapid for quick implementation and deployment, because the mobile landscape changes very fast
- A mobile application cannot be a clone of a desktop application. It uses the mobile device primarily as a communication medium to interact with a server and display information of interest to the user in an intelligent interface.

The Symbian Operating system has been designed for flexibility, customizability, efficiency, robustness, and communications-centric. It has been implemented as a multi-threaded, event-based, object oriented and preemptive Operating system written in C++. The Operating System has micro-kernel based architecture with core system functionality like scheduler and base user library that execute in privileged-mode and a file system and graphical windowing system running as processes in user-mode.

Palm specifies a similar event based architecture for application programs where the fundamental structure for the application is an event loop that receives events – generated externally or internally by the application – which the application interprets and performs actions in response to those events. Applications access the functionality of the Palm Operating System through the several different system function calls in the C programming language. These groups of function calls that work together to implement a feature are categorized as Palm OS Managers.

Windows Mobile has been architected to be a lightweight, low power consuming, multi-threaded, multi-tasking operating system based on the Windows CE embedded operating system. It is a real-time, small foot-print operating system with a deterministic response to interrupts and incorporates a subset of the Win32 API used to program all Microsoft Windows Operating Systems. A chief aim of the designers of Windows CE was to allow Win32 programmers of the desktop Windows Operating Systems to continue using the same APIs and programming paradigms – messages loops, event handling frameworks on the smaller Windows CE platform. Existing frameworks for Windows development like MFC – Microsoft Foundation Classes can be used to develop Windows CE applications.

Since Microsoft promotes .NET is the development environment of choice on the Windows platform, it has developed a limited version of the .NET environment for Windows CE called the .NET compact framework. Application developers for the Windows Mobile platforms have the option to use the .NET runtime for development apart from using the Win32 API directly.

API Documentation

Documentation of an API does not just include a reference to all the functions and data variables exposed by the Operating System. Documentation should also include well written tutorials explaining programming for the system from the ground-up, explanation of programming paradigms and patterns to be followed for development of application programs for the system as well as provision of an extensive library of program code demonstrating working applications on the system.

Palm does a good job on this front by providing users with the actual code used for all the built-in Palm applications. This code is not the simple hello-world variety provided in the documentation by various Platform APIs. The Palm application code is sufficiently complex that many production applications can be developed by understanding and isolating the structural parts from the Palm code. The Symbian documentation is more comprehensive and structured than Palm. It prepares developers for Symbian programming by explaining the architectural constructs and programming frameworks that Symbian developers using for developing the Operating System. Since several core services of the Symbian OS – windowing server, file server, messaging server are implemented as user-side services instead of kernel modules, the same service provider-consumer pattern can be taught and easily applied to application programming. Since the operating system has been written in C++ and the API itself is exposed through C++ classes, the documentation lends itself well to UML modeling. Complex relationships between different classes constituting the API can thus be well understood with the UML class diagrams that Symbian provides in its API documentation. The Windows CE method of documentation follows the well understood and familiar Windows platform documentation using MSDN. The MSDN web site provides a common web interface to obtaining API reference documentation for all Microsoft products. Use of the standard Win32 API for Windows CE application programming interface as well as support of the .NET runtime on the Windows CE platform allows developers to reuse their Windows programming skills on the Windows CE platform. Books and sample code from other Windows projects can be adapted for use on the CE platform though production quality sample code for the built-in applications (as with Palm) is not available.

Platform Framework for Application Development and Service Provision

Object Oriented Programming is a programming paradigm wherein a computer program can be expressed more naturally using objects from the problem domain. This allows for the design and development of small to large, complex programs that can be maintained and evolved easily. An object oriented software system uses encapsulation for information hiding with explicit public interfaces, inheritance and polymorphism as powerful tools to explain relationships among objects.

Symbian OS is designed and implemented using Object Oriented Concepts in the C++ programming language. The Programming Interface to the Operating System uses the same idioms and frameworks which allow Symbian application developers to design and develop complex programs that are maintainable and evolvable. The Windows CE API is a subset of the desktop Win32 APIs with the same syntax and programming idioms that allow existing desktop Win32 applications to be easily ported to run on Windows CE. Object Oriented Programming using the MFC framework is possible, but not promoted by Microsoft. OO Programming can be accomplished using the .NET runtime, but the abstraction caused the .NET environment running on low-power mobile hardware is not conducive for complex application development requiring deterministic runtime capabilities. The Palm OS API reflects the designers' motivation to give priority to lower program memory requirements than speed and efficiency of execution. The choice of the C programming language for application interface description means that Application programs have to simulate object oriented programming patterns without the native language support.

Event Handling

Symbian provides the Active-Object framework which is an encapsulation of the low-level Asynchronous Service Handling API. This framework allows application programs to be written to use a well-understood, more natural event framework for requesting services from the Operating System (or other programs) and perform actions based on request completion. The active-object framework is at the heart of Symbian programming – UI applications as well as complex applications performing external communications can be quickly developed using the framework. The active-object framework includes two components – the active-object encapsulating the service request mechanism including making the request and handling the response and the active-scheduler which implements the wait-loop for waiting for response to requests and dispatching them to the event-handlers. The Symbian active-object framework results in a non pre-emptive multiprocessing system without the overhead of multi-processing using threads. Even tasks requiring long processing times can be split up into reasonable fixed-size work chunks with the task scheduling itself with the active scheduler after completing a chunk of work. This results in light-weight multi-tasking since it avoids the costs of synchronization required for thread-based multi-processing as well as the kernel costs for launching/destroying threads. With active-objects, A Symbian application becomes a collection of co-operating objects than co-operating threads.

Client-Server Architecture for processing OS service requests

The Symbian Operating System is based on micro-kernel architecture with the kernel running in privileged mode and providing only a few key services – thread scheduling, message passing etc and the System servers running in user-mode – Windowing server, File Server etc. providing other key services to applications. This architecture yields better security – functionality is isolated to specific servers and security flaws in them can't affect the entire system. The implementers of a Symbian OS on a device are also given the flexibility of packaging the service provider threads into different processes of their choice based on their requirements of performance, security and economy.

Memory Management

Symbian provides interfaces for application programs to manage their program heaps. It provides convenient APIs so threads within a process can have their own separate heaps or share specific heaps. Symbian provides APIs so that individual threads in different processes can also share memory chunks. In addition to this, Symbian also provides low level APIs for effective heap management. Threads can create a new memory heap and walk through their heap of allocated and unallocated cells and determine that the heap is in a consistent state and not corrupt. These interfaces provide quite an open interface into Symbian's memory management architecture and provide a pretty low level API for application programs to effectively manage their memory.

Installable File systems

Symbian provides the DOS-like VFAT file system natively through the File Server's client API. In addition, it provides the ability for other file systems to be implemented and installed dynamically without reboot – the only requirement is that these file systems present a Symbian native interface to client programs on the device. This allows device implementers to provide new file systems like access to remote systems like Unix servers or IBM mainframes to distinguish their particular device.

The File Server provides a comprehensive API to access the file system functionality including opening, reading/writing files, listing drives, volumes and directories, parsing pathnames, deleting and moving files and directories, scanning and searching for file and directory names. Symbian also provides a notification mechanism to indicate file system and disk space changes.

Symbian also provides a stream store API to manipulate data in files as byte streams. A database manager built on the stream store that provides a file-based, high performance, relational database is also part of the Symbian platform.

Communications

In the Socket Communications API, Symbian provides support for multihoming that allows for multiple Circuit or Packet switched data connections to be active at a time. This allows for Symbian to support technologies such as W-CDMA and GPRS later releases that allow for establishing multiple sub-connections within a connection. This is achieved through the RConnection class which also provides applications with the ability to monitor the progress of a connection. This is especially useful for connections that take a lengthy setup time like Dial-up networking. The Symbian Socket Server API allows for dynamic installation of protocol modules while providing the same Socket Client API for application programs. New protocol families (a collection of related protocols – TCP/IP family includes UDP, IP and TCP, IrDA family includes IrMUX and TinyTP) providing different communication transports can thus be added to the sockets API while remaining transparent to client programs.

Plug-in Architecture for Service Provision

Providing an extensible framework for dynamic service deployment allows mobile operating systems to support evolution of applications and services. This requires support of a mechanism wherein standard interfaces can be specified, services implementing those interfaces are implemented and registered with an existing programming framework, and client programs can call those interfaces by dynamically selecting the appropriate implementation for the interface. This open architecture to dynamic deployment of service implementations provides an ideal platform for application developers and network service operators. It also supports device customization and differentiation desired by many OEMs licensing the Operating System.

Symbian provides a plug-in architecture for service deployment through its ECom framework. It provides an instantiation mechanism that forms the framework backbone and provides the services to identify and load the current run-time implementation.

Evolution

The Symbian OS provides advantages to both the phone manufacturer desiring to using Symbian OS on his phone while also enhancing the UI with proprietary features and to the application developer who wishes his application to run on a wide variety of phones from different manufacturers. This combination of compatibility (with application programs) and customizability (by device manufacturers) means that the Operating System has to maintain binary compatibility with its APIs while allowing derived platforms to add innovative and differentiating functionality to customize the OS. This is a challenge because there are three different evolutionary paths possible in such an environment:

1. Symbian Operating System evolves
2. UI layer and Other device customizations from the manufacturer evolve
3. Application program and functionality evolve

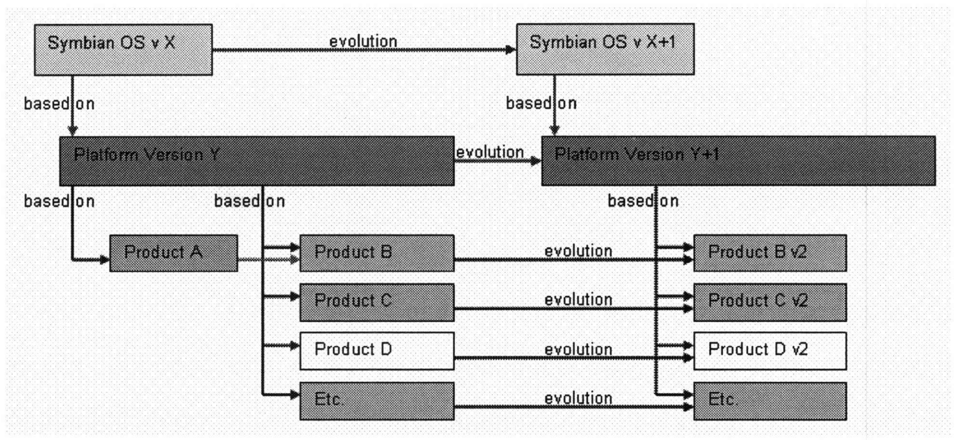


Fig. 30 Platform evolution[51]

[51] http://www.symbian.com/developer/techlib/v9.1docs/doc_source/guide/N1001E/extensiondll.html (accessed, Nov. 2006)

Symbian uses the extension DLL pattern as a method to allow licensees to extend the API while not introducing binary compatibility problems.

Security

Building the right security architecture in its platform can create competitive advantage for the Firm. Licensees and Application Developers for the platform would trust the Operating System as low risk option for deployment. Device manufacturers licensing the platform have a greater incentive to adopt the most secure platform because of the risk to their reputation from security breaches and violations. Tracing the evolution of all three Mobile Operating System in Chapter 3 shows that all three – Symbian, Microsoft and Palm have included security as a central component in their OS architectures. All three provide authentication and authorization services through interfaces to certificate management and cryptographic libraries for encryption/decryption.

The platform security architecture on the Symbian Operating System addresses security threats from the distribution of malicious applications by preventing unauthorized access to user data and system services. Windows CE supports an optional module level security model to be implemented by OEMs – the OS designates modules as trusted or un-trusted based on validation from the OEM Abstraction Layer (OAL).

Chapter 5

Conclusion

The Smart Mobile device market is experiencing rapid growth with the increased convergence of voice and data centric applications. This is being fueled by improvements in low-power, faster processing CPUs and advancements in flash and dynamic memory units. This improvement in hardware capabilities has allowed development of complex user interfaces, multimedia and communication capabilities on mobile devices. Modern Mobile Operating Systems manage this complexity in the mobile device by administering hardware resources and providing a platform for development of communication and multimedia applications.

The Mobile Operating Systems studied in this thesis differ in their architecture, their core design goals, and services offered to users and device manufacturers. Their design reflects the OS vendor's strategy toward the mobile platform and is decipherable based on a study of its architecture and application programming interface. Three conclusions can be made based on this study – each of them suggests a strategy to use to gain platform leadership through product architecture and degree of openness of interfaces.

1. **Design as a generic platform for extreme customization:** This is essentially the core principle followed in the design of Symbian OS. The Operating system is based on micro-kernel architecture with the OS providing only a few core services. User Interface, file and device management are implemented as user services in a pluggable infrastructure that permits new service implementations conforming to a generic interface to be added or replaced. OEMs licensing Symbian OS add their specific User Interfaces on top of the reference UI layer provided by Symbian – this aids in device customization and differentiation.

2. **Design to complement a broader platform:** Microsoft has designed the Windows CE platform to serve as the Operating System for smaller, limited resource, hardware environments while keeping the user interface familiar to Windows Users. Windows Mobile branded devices based on Windows CE are certified by Microsoft to conform to a set of hardware and software standards and specifications – this ensures that all Windows Mobile devices have the same Microsoft specified interface. Application developers for the platform use a subset of the Win32 API to program for Windows CE. Microsoft has even implemented the .NET runtime for Windows CE to allow developers to program in its type-safe, hardware independent runtime environment.

3. **Design as a platform for extensibility:** The Palm Operating System is a compact operating system designed specifically for handheld computing devices. It evolved over time to support phone specific features on handheld devices. Major design objectives since the first release of the operating system have been to keep the user interface simple and the operating system extensible to licensees and application developers. Developers have low level access to hardware bypassing the OS interface provided through its API – this allows developers to have direct control of processor, memory and interrupt hardware to develop extended functionality. With the increasing complexity in modern application programs and services, this original Palm OS architecture supporting handheld specific development became untenable. This has led to development of Palm OS Cobalt (or v6) which is a completely new architecture with multi-tasking support. This architecture is still PDA device specific with phone features added in, as opposed to the Symbian OS architecture which is mobile phones specific, with extended data organization capabilities.

The three mobile operating systems studied have each followed a different strategy for their product architecture. The Symbian Operating System has been designed to be open and extensible for a wide range of mobile phone devices. The architecture is "licensee-

focused" to enable phone manufacturers to provide differentiated services including phone-specific User Interfaces and custom hardware capabilities. The User Interface is designed as an extensible layer which provides basic interaction and display capabilities appropriate for small screen and limited input possibilities. Symbian uses industry standard SyncML for device synchronization and Java for application deployment.
The support for SyncML in the Operating System reflects Symbian's understanding of the needs of mobile phone network providers to remotely manage and configure Symbian OS phones to reduce in-market deployment and management costs. Support for Java by implementing all Java Community standards in the Operating System and providing native access to OS features from Java programs allows Java application developers to program rich, highly functional applications for the Symbian platform.

The Palm Operating System, on the other hand was designed specifically for handheld organizer devices. It hasn't evolved much since the original release in terms of serving as a platform for mobile devices – phone specific features are "add-ons" to the core operating system, application programming support is limited (no advanced programming frameworks, error handling support) and no OS support for multi-tasking or virtual memory for process address space separation. The Palm Operating System architecture is based on simplicity and efficiency – the User Interface is designed to be simple and intuitive, and APIs are sufficiently low-level for applications to gain performance efficiencies.

The Windows CE Operating System has been designed to serve as a base for a wide range of consumer devices, not just mobile phones or handheld organizers. Microsoft supplies a Platform Builder toolkit for OEMs to customize and package the Operating System components for their specific device. The Windows Mobile family of mobile Operating Systems uses Windows CE OS components as the base, but specifies a set of user interface and hardware capabilities for devices to be branded as Pocket PC or Smartphone devices. The Operating System architecture supports rich, functional application development using a subset of the Win32 APIs from desktop Windows OS.

Microsoft has also implemented a version of the .NET runtime instead of Java, for Windows CE to run .NET applications for hardware independence.

Symbian has designed the Operating System exclusively for mobile phones with integrated telephony support to seamlessly connect with other subsystems. The User interface is adaptable to both touch-screen and keypad based devices. Symbian's strategy for a mobile platform seems to be limited to phones – the architecture, design goals and implementation are all specific to mobile phone devices. The Palm Operating System architecture and implementation seems to be handheld organizer specific – it is only adaptable to devices with a touch screen. Phone support is an additional, but not core feature of the Operating System. The Windows CE Operating System serves as a platform for the broadest class of devices among all the three mobile Operating Systems studied. As an extension to the desktop Windows XP Operating System and base platform for embedded, consumer devices, Windows CE is a "component platform" in Microsoft's grand strategy of Operating System platforms for generic computing devices.

Microsoft's platform approach to Mobile Operating Systems seems to be the best among the three – it views mobile devices as a "class" and not specific devices providing particular functions. It provides a base Operating System that could be componentized and adapted for specific devices. Symbian focuses on a Mobile Operating System as voice communication specific – with network providers, operators and phone manufacturers. The design of the Symbian Operating System fits these goals very well – the application programming frameworks and plug-in architecture for phone device customizations are well suited for this.

The current architectures and evolution over time of the three Mobile Operating Systems studied, Symbian, Palm and Windows CE, truly reflect their vendor's strategy for a mobile device platform – mobile phone, handheld organizer or both.

Bibliography

Foster, L. Palm OS (Wireless + Mobile) Programming. IDG Books, 2002. ISBN 81-265-0173-1

Foster, L and Bachmann G. Professional Palm OS Programming. Wiley Publishing, 2005. ISBN 0-7645-7373-X

Harrison, R. Symbian OS C++ for Mobile Phones. Symbian Press, 2003. ISBN 81-265-0476-5

Boling, D. Programming Microsoft Windows CE .NET. Microsoft Press, 2003. ISBN 0-7356-1884-4

Cusumano, M and Gawer, A. Platform Leadership: How Intel, Microsoft and Cisco drive Industry Innovation. Harvard Business School Press, 2002. ISBN 1-57851-514-9

Rechtin, E and Maier, M. The Art of System Architecting. CRC Press, 2000. ISBN 0-8493-0440-7

Palm OS 68K API Documentation. Volume I and Volume II available as SDK download from http://www.palmos.com/dev/dl/. Accessed Aug 2006.

Symbian OS API Guide. Available at http://www.symbian.com/developer/techlib/v9.1docs/doc_source/index.html. Accessed Oct 2006.

Windows CE Reference. Available through the MSDN web site at http://msdn2.microsoft.com/en-us/default.aspx. Accessed Nov 2006.