Symbian C++ Application Programming Overview

F. Pérez, C. Carrión, E. Montón, V. Traver
ITACA Institute, Polytechnic University of Valencia (Spain)
fraperod@itaca.upv.es

Abstract

This paper offers an introduction to the development of applications for mobile devices, and in particular for the development of applications created using Symbian C++.

Mobile development is limited by the nature of mobile devices and wireless technologies. Due to this special environment conditions, the choose of an Operating System prepared for handle specific issues related with resource constrained devices is a key factor to successfully overcome the limitations of the mobile development world.

Symbian Operating System (OS) is described and compared with other well-kwon mobile operating systems. Furthermore, the main issues related with Symbian C++ programming are exposed, and the security of Symbian devices are discussed.

1. Introduction

There are many limitations imposed on the mobile application development by the mobile devices and the wireless networks. Most of these are obvious, such as narrow bandwidth, heterogeneous networks, and limited resources on the devices.

The applications running on the mobile devices need to be prepared for more or less frequent disconnections. The battery might run out, the user might be moving out of the coverage area of the network, or some external network failure might occur. This means that compared to applications running on workstations connected to a wired network, mobile applications need to take more control over how to handle network failures; they cannot just assume that these are rare and that the user can react to them immediately whenever they occur.

Distributed applications for mobile devices almost always have to make many kinds of implementation compromises. It is not a good idea to make the mobile client applications complex and resource consuming, but it is not a good idea to exchange a lot of data across an insecure and slow network either. Furthermore, the solution that seems to result in a good balance in one type of device might be totally inadequate in another.

The limitations are not always just hardware-related; the software environment brings in restrictions as well. For example, the PC world has support for many different programming languages, each better than the other for solving certain kinds of programming challenges. Support for only a fraction of these has yet been made available on mobile devices. Symbian[1] devices currently have built-in support for native C++, Java, and Python programming languages and through a third-party solution from AppForge for .Net Visual Basic and Visual C#[1].

Symbian is an advanced, multitask, open, and flexible operating system built up specifically for mobile phones. The objective of Symbian is to drive standards for the interoperation of data-enabled mobile phones with mobile networks, content applications and services.

Symbian is an operating system open for third-party development, which implies that the programming APIs needed for the development of applications are available for free. as well as standard languages such as C++ and Java, SDKs, tools, documentation, books, technical support and training. Symbian OS has a rich set of APIs for independent software developers, partners and licensees to write their applications.

In this paper a first approach to programming Symbian Devices using C++ is provided.

2. Symbian OS Architecture

Symbian has been designed taking special care regarding resource management, scalability and interface customization for specific mobile phones. All Symbian devices share common system core components and also most of programming APIs, which represent approximately the 80% of the system. The difference in APIs appears mainly in User Interface (UI) which can be defined by each vendor and represents the remaining 20% of system APIs. Figure 1 shows Symbian's structure.

Therefore, main effort when porting from one device type to another resides in the adaptation of the UI part of the application. The most extended interface types among Symbian devices are Nokia Series 60[3] which is based on one-handed keyboard input, and Sony-Ericsson UIQ interface[4] which is based on the use of a tactile screen, integrating fully-featured personal digital assistant (PDA) capabilities with traditional mobile phone single unit.

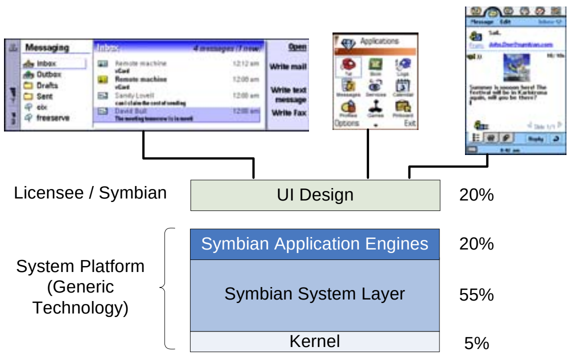


Figure 1 – Symbian Structure

Symbian Multimedia Framework (MMF) allows developers to write efficient and powerful plug-ins. Codec API is provided and MMF supports codec plug-ins, with separate format and controller plug-ins. Format plug-in decodes or encodes a specific data format (e.g., MP3) and controller handles, e.g., a specific file format (say WAV) and creates the necessary source and sink objects and their connections.

There is a large networking and communication subsystem, which has three main servers - ETEL (EPOC telephony), ESOCK (EPOC sockets) and C32 (responsible for serial communication). Each of these has a plug-in scheme. For example ESOCK allows different ".PRT" protocol modules, implementing different types of networking protocol scheme. There is also a wide support related to short-range communication links too, such as Bluetooth, IrDA and USB.

Java Virtual Machine: is built in as part of Symbian System offering: robustness, functionality and platform integration and performance, both in terms of speed and memory footprint. Symbian, in both its Java implementation and its core operating system, has done an optimal use of constrained memory resources. The Java implementation is highly optimized at all points in the chain. This includes graphics, libraries, and of course the VM.

2.1. Multi-Tasking

In smartphones, multitasking is especially valuable since users will frequently want to do things such as downloading e-mail while talking or looking at a Web site. As 2.5G and 3G network packet data services become more common, multitasking smartphone applications will become a must.

Symbian OS is an event-handling system extremely efficient in power consumption. The events are managed through the *active objects*, which encapsulate the association between making a request, and handling its completion: each active object is responsible for making and handling just one outstanding request. They provide non-preemptive multi-tasking, which makes multi-threaded programming unnecessary for most applications and servers.

In addition, since the programmer uses active objects within the Symbian OS itself for system service access, it's relatively simple to access system resources such as the file server, messaging server, and sockets server with a straightforward API call.

2.2. User Interface

Symbian OS offers different kinds of user interfaces(UI). Even though the user interfaces themselves are maintained by other parties, the base classes and substructure ("UIKON") for all UIs are present in Symbian OS, along with certain related servers (for example, a view server which controls transitions between different phone user interface screens). There's a lot of related graphics code too, such as a window server and a font and bitmap server.

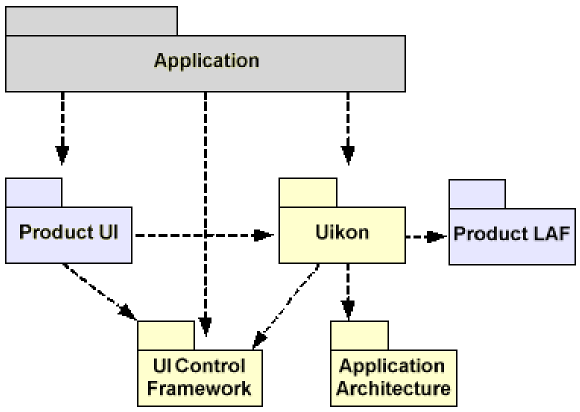


Figure 2 - User Interface Architecture

Figure 2 shows the Architectural relationships in Symbian OS regarding user interface. The two key components are *Uikon*, a generic core UI framework, which is present on all Symbian OS phones, and a *Product UI*, UI libraries developed by a Symbian OS licensee for a particular phone or range of phones.

An extra element of UI specialization for licensees is provided by the *Product Look And Feel* (LAF) library. This sets the appearance, such as colour and size, of Uikon controls.

These components are layered above the UI Control Framework, which defines the abstract concepts of the user interface at a basic level, and the Application Architecture, which defines an abstract framework for applications.

Together, these components provide to applications: a programming framework that provides services required by all applications, such as initializing the application at start-up, channeling user input to the correct part of the application code, and interacting with the shell and file system controls and dialogs for applications to use. Development Kits produced by licensees supply an appropriate real Product UI.

2.3. Communication Architecture

In the last few years we can observe a rapid and continuous evolution of mobile communication systems. The applicable technologies are, for example, 3G UMTS, GPRS, IrDA or Bluetooth. Although they are totally different transport techniques, at the application layer they all can be handled with sockets. After initializing the current communication technology, there is a socket on the server and client side that is needed to be connected. There are different discovery services depending on the technology currently used to select the remote device before establishing the connection. On each side of the communication, there is a socket to read and Moreover, Symbian includes support for multi-homing – the ability to be connected to two connections at the same time (e.g. WiFi and EDGE) – so you may be browsing using EDGE but downloading email at the same time on WiFi, for example.

3. Comparisons with Other Systems

Symbian's approach to resource management is a key part in fulfilling the requirements of handheld devices. This can be seen when comparing Symbian with other well-known systems.

3.1. Windows CE

From a programming perspective, Windows CE's greatest attraction is that it uses an extended subset of the Win32 APIs found in desktop versions of Windows. Experienced desktop Windows programmers can port their code (if it is written at a fairly basic Win32 level) to Windows CE without much adaptation.

But this attraction is also Windows CE's greatest weakness. The Win32 APIs were not designed to handle errors on every resource allocation, or to clean up properly. Nothing is done to reinforce good habits in Win32 programmers, especially in the desktop environment, which has become more and more forgiving. Programmers don't always check return codes, and don't always handle them well.

Because Windows CE is marketed to developers on the basic assumption that programmer don't have to change much in its existing Win32 code, there's no option to implement memory management that would alter every function call in existing source code. Instead, Windows CE's designers implemented a retrospective monitor/sweeper system. The CE system monitors available memory and, when it is running low, sweeps through all applications, asking them to release memory. This works with only loose cooperation between the system and applications. Put in another way, imposes only minimal requirements for change on existing Win32 programs.

When Windows CE's monitor detects that only a small amount of memory is still free, or when an allocation is about to fail due to an out-of-memory condition, the system sends a *WM_HIBERNATE* message to all applications. Each application must then try to free up some memory. The user may also be asked to close some applications. Ideally, this works fine, and the system continues on its way.

But if the applications don't free memory, or the user doesn't close some applications, the system takes more radical action: it may kill applications, with data loss. Applications are killed in reverse Z order – that is, the application that's furthest in the background is killed first. This can be easily verified this by experimentation with a Windows CE device, and the only really safe way to deal with it is to save all application data when an application losses focus.

Windows CE's memory management is not tailored to the requirements of small systems, nor is it in any sense industrial strength. It is possible to write individual industrial-strength applications, libraries or servers for Windows CE, but the system provides little support for doing so, and in practice few applications take the necessary precautions[5]

3.2. Palm OS

Like Symbian, Palm OS was designed from the outset for small, memory-constrained environments.

The APIs and system design convey this clearly, and every Palm programmer knows this from the outset.

Palm OS maintains a very small memory footprint by simple application design, and by closing applications when there are not in foreground. When an application is closed, it is required to save data. Most application data is stored in the Palm database manager, which handles saving as a routine part of the application.

Although Palm OS doesn't provide special cleanup support, cleanup comes at regular intervals anyway, because applications are so frequently closed. The Palm system design is very well suited to its task, though it would be difficult to scale this approach to a larger environment; for example, to support multitasking without closing down background applications, or to support a large number of active servers.

Windows CE can be compared quite usefully with Palm OS. Palm applications in the background are effectively in deep hibernation. Windows CE's shallow hibernation doesn't protect background applications from getting killed, so when Windows CE applications go into the background they have to save data – like Palm applications, in order to be truly safe. Palm OS's system design has the advantage of being fit for its intended purpose, but it's hard to scale up. Windows CE can clearly be scaled up (as it can be expected of a system that was initially scaled down!), but can only be made fit for its intended purpose by applications taking Palm-like precautions.[5]

3.3. Linux

Linux is starting to take part of the mobile market, specially focused for new emerging Asian markets (mainly China).

Linux main advantage is its low cost, being royalty-free, seems a proper option for manufacturer facing new feature rich mobile devices. Availability of the full source code allows manufacturers to develop customized-looking terminals.

The main Linux's drawback is that it has been designed for PC/servers, not for mobile devices with reduced memory, processor, and battery resources.

Linux is generally regarded as a more complicated and sophisticated operating system compared with the more intuitive "user friendly" operating systems such as PalmOS and Microsoft's PocketPC. Therefore adaptation of user interface has also to be done by vendors.

Another Linux inconvenient is that no single company supports mobile Linux, so there is no singular, uniform standard for hardware/software developers. This situation may lead to application compatibility problems among different manufacturers of Linux devices.

In order to solve these problems some standardization organization has been founded:

OSDL: Mobile Linux Initiative works in the standardization related to Linux lower layers.

The Linux Phone Standards Forum (LiPS) works in the definition of common interfaces for application's services and its behavioural (or non-functional) requirements for those services, in order to allow the development of compatible applications for different Linux devices.

On the other hand, some companies are selling Linux based solutions for mobile devices' software. The most representative companies offering Linux based solutions are Montavista Linux, Trolltech, PalmSource and WindRiver.

As example of available Linux devices, highlight that Motorola has started manufacturing mobile phones based on a Linux/Java software platform (which are being distributed in China). The platform released combines MontaVista's Linux OS, Trolltech's Qt/Embedded (now known as Qtopia Core) GUI platform, and Motorola's own proprietary phone stack. Third-party applications in Motorola's Linux devices are only allowed in Java.

3.4. Java

The Java Platform, Micro Edition (J2ME) provides a robust, flexible environment for applications running on consumer devices, such as mobile phones, PDAs, TV set-top boxes, printers and a broad range of other embedded devices.[11]

Configurations are composed of a virtual machine and a minimal set of class libraries. They provide the base functionality for a particular range of devices that share similar characteristics, such as network connectivity and memory footprint. Currently, there are two J2ME configurations: the Connected Limited Device Configuration (CLDC), and the Connected Device Configuration (CDC).

The J2ME platform can be further extended by combining various optional packages with CLDC, CDC, and their corresponding profiles. Created to address very specific market requirements, optional packages offer standard APIs for using both existing and emerging technologies such as Bluetooth, Web services, wireless messaging, multimedia, and database connectivity. Because optional packages are modular, device manufacturers can include them as needed to fully leverage the features of each device.

The main drawbacks of using Java in mobile applications are the execution speed of the java applications, due to it is a semi-interpreted language, run-time RAM utilization because the Java Virtual Machine has to be loaded also in order to run a Java application and also some limitations dealing with access to device's hardware resources.

4. Symbian C++

Symbian native programming using C++ does not uses Standard C++. By the time that Symbian system was being defined (1997) C++ Standard was still not available.

Symbian C++ has not supported Standard C++ Exceptions until Symbian version 9. Instead a Trap and leave mechanism is used.

Symbian C++ does not support multiple inheritance in general, only allowed with pure abstract classes (interfaces), which are named usually starting by a "M".

Standard Template Library (STL) from C++ has not been supported in Symbian until version 9. This support has being introduced to easy porting of existing code, but its use is discouraged.

Symbian C++ is a strongly Object Oriented language which also defines its own Class Naming and coding style. It is strong focused on correct resource allocation and error condition handling while application execution. Possible Memory allocation errors are taken into account in each single function call.

The Cleanup Stack is a mechanism used for releasing dynamically allocated resources in case an error may happen while executing a function. Each faction which may leave (exit because of a memory allocation error) is named placing an "L" suffix (i.e. *FunctionL*).

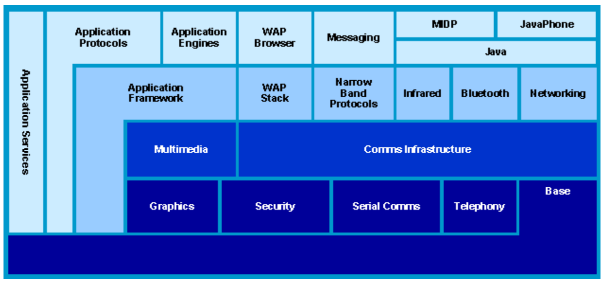


Figure 3 – Main Symbian APIs Classification

4.1. Memory

Symbian OS uses a strict managing memory system. Memory, as a limited resource is carefully handled, particularly in the event of error conditions. For this reason, exception handling and memory management are closely tied together in the Cleanup Support API. This API has three key concepts: exception handling, cleanup stack, and general cleanup item.

4.2. Coding Idioms

Symbian OS defines its own class types instead of using standard C++ or STL classes. Symbian OS uses a simple naming convention which prefixes the class name with a letter (T, C, R or M). The categories are used to describe the main properties and behavior of objects of each class.

T classes: Classes that do not own dynamically allocated resources. Behave much like the C++ built-in types.

C classes: Symbian OS requires that all classes that own dynamically allocated resources derive from a standard base class: *CBase*. Such classes by convention have a C prefix to their name, and so are referred to as C classes. C classes, and their associated allocation and cleanup idioms, are fundamental to Symbian OS.

R classes: Resource types. The "R" which prefixes an R class indicates a resource, which is used but nowt owned by the application.

M classes: M class is an abstract interface class. The only allowed multiple inheritance in Symbian C++ is through M classes.

5. Symbian Application Architecture

The GUI applications developed for Symbian mobiles usually use the Model-View-Controller pattern, which can be seen as differenced layers with well-defined interfaces.

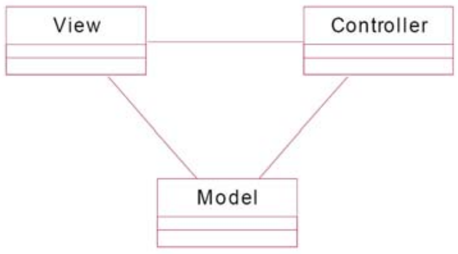


Figure 4 – Model View Controller Design Pattern

Figure 4 shows the structure of the model-view-controller pattern. The model class is in charge of storing and managing the application data, it contains the logic of the application, and also handles the communications and file-based data storing. The view class function is to format and show the data contained in the model to the user. And finally the controller class' functions deal with user interaction events handling.

This design of applications makes easy-to-change one of the components without the need for modifying the rest of the application. This is especially important when facing the development of applications which will have to be ported to different kinds of Symbian User Interfaces.

Figure 5 shows how the model-view-controller pattern as usually is implemented in Symbian C++ applications.

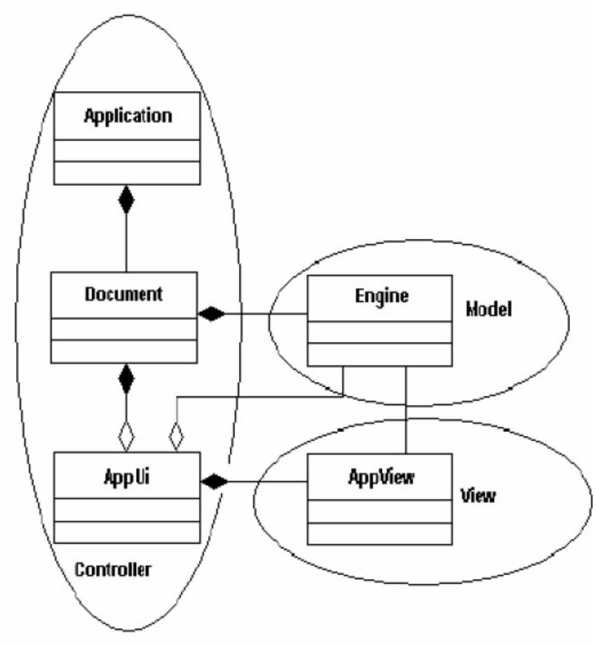


Figure 5 - Basic GUI Symbian Application

6. Mobile Communication Architecture

The most common architecture used for communications in general mobile application (Symbian or not) is showed in Figure 6.

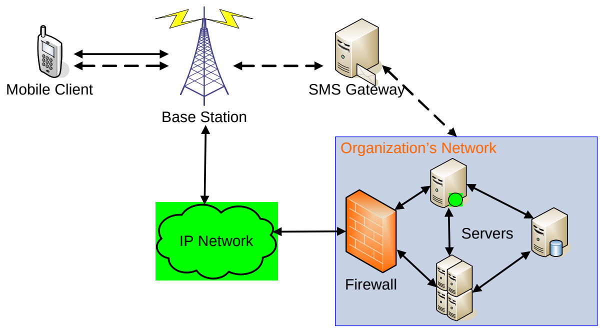


Figure 6 - Typical Mobile Application Architecture

6.1. Connection Establishment

In the typical client/server model it is the server's duty to be running and available at all times and the client's duty to initiate connections to the server whenever needed. This type of interaction is called the "pull" technology (the client effectively "pulls" information from the server whenever needed).

If the resources on the mobile device allow it, the client application can be up and running as well and the connection can be kept alive all the time. This typically requires at least so-called small keep-alive messages to be sent across at regular intervals.

However, there are situations where the client needs to receive information from the server as soon as it becomes available but cannot keep the connection open while waiting. Frequently waking up to poll the server to check for updates is very inefficient, which is when the use of the so-called "push" technology comes in handy. In this case, the client application is either running idle or not running at all, until the server sends a message which triggers the operating systems to wake up the target client application and pass the message to it. The target application can then take appropriate actions to handle the message with or without user interaction.[6]

6.2. Short Message Service

If an application can cope with asynchronous, infrequent, but reliable data transfer, the Short Message Service (SMS) might be the way to go. SMS datagrams are especially useful when the client application needs to be able to receive notifications or small amounts of data from the server as soon as the data becomes available.

The biggest restriction is that each datagram can only contain a maximum of 140 bytes (8-bits) of binary data or 160 (7-bits) text characters. The APIs i support concatenating multiple SMS datagrams, but still the user has to pay separately for each of them.

There are many ways to enable your server-side system to send and receive SMS datagrams. For example, you can subscribe and connect your server to a mobile service provider’s SMS gateway service or fit your server-side system with an additional device capable of receiving and sending SMS datagrams[6]. This first situation is illustrated in Figure 6.

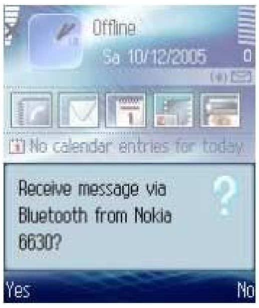
7. Security and Malware

Symbian OS has been subject to a variety of viruses, the best known of which is Cabir. Usually these send themselves from phone to phone by Bluetooth. So far, none have taken advantage of any flaws in Symbian OS - instead, they have all asked the user whether they would like to install the software, with somewhat prominent warnings that it can't be trusted.

Cell phone virus is a proof-of-concept application that might describe same as computer viruses that install itself into the targeted device and executes its malicious code to “infect” the phone with preset command.

Currently, Cell phone viruses are spreading using: Bluetooth, Multimedia Messaging, and faked applications in warez/Shareware software.

Bluetooth Wireless Technology: it is capable replicates itself in 10metre Bluetooth wireless range and search for bluetooth devices that are active in discovery mode. Upon detected it will pop up as Screen as shown in Figure 7.



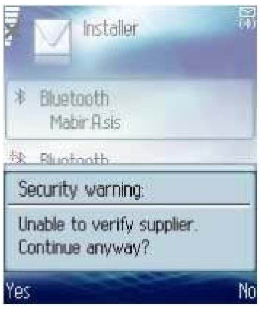
No calendar entries for today

Receive message via Bluetooth from Nokia 6630?

Figure 7 - Bluetooth message reception query

If user clicks yes and he may facing risk that he will be infected by this suspicious file since he didn’t practice well in mobile security knowledge and he may proceed to the installation process.

User should be aware that installing application that without valid certificate will cause them facing a very high risk of cell-phone-viruses infection and they should only install those applications which are trustworthy. Example of a security warning message is shown in Figure 8.



Security warning:

Unable to verify supplier. Continue anyway?

Figure 8 - Security warning from unsigned application

Multimedia Messaging Services (MMS): This year January 2005, a new type of mobile viruses able to spread itself via Bluetooth but also MMS has been causing public attention and AV firm pretend this is the most effective way for mobile viruses to replicate themselves. Besides, it is able to generates different codes to send itself via MMS by scanning user phonebooks contacts that might causing other innocent users with less expose to mobile security knowledge get confused and proceed to the installation process which giving opportunities to cell-phone-malware to executes itself. Anyway, user should aware of third party application that doesn’t contain any valid certificates that might be a virus, as shown in Figure 8.

Faked games, applications and security patches at Warez/Shareware sites: this is also a way that cell phone viruses developers used to spread their stuff at which usually most people like to browse those site to get “free” stuff and were not aware that actually it has been packed with mobile trojan/malwares inside them.

However, of course, the average mobile phone user shouldn't have to worry about such things, furthermore Symbian OS 9 is adopting a capability model. Installed software will theoretically be unable to do any damage (such as costing the user money by sending network data) without being digitally signed - thus making it traceable. Developers can apply to have their software signed via the Symbian Signed program.

8. Symbian OS 9.x

There are two significative changes in Symbian OS 9.x. The first one is the break in binary compatibility. This means that programs written in C++ for previous versions of Symbian OS will not run on OS 9 without making some changes to the source code and recompiling. So your Series 60 version 3 device (which will use Symbian OS 9) won’t run current Series 60 version 2 applications.

The second one is at the security level. Previously, once an application was installed on your phone, it had full access to all the API's in the OS. Now, applications do not have automatic access to all the API's. Those involving critical functions (or which could run up your mobile bill) will now be restricted.

There are two different types of API (also know as capabilities): unprotected (60%) and protected (40%). The protected capabilities can be further divided into three groups: basic, extended and phone manufacturer approved. Applications that use either extended or phone manufacturer approved capabilities must be Symbian Signed or it will not be possible to install them on target devices.

Basic protected capabilities are those which may be broadly understood by the user (such as Bluetooth connectivity). The phone manufacturer can set these capabilities as subject to being authorised by the user (whether a capability is set as user-authorisable is dictated by the manufacturer's security policy for the device which can vary between markets and operators). If a capability is user-authorised then it is not necessary for an application to be signed. However if a manufacturer has chosen not to set a capability as user-authorisable then an application must be Symbian Signed in order for it to be installed. In practice it is recommended that most applications using basic capabilities should be signed.

Applications that use only the unprotected APIs or use user-authorisable basic capabilities can be installed although, as with existing Symbian phones if they are not signed, they will show a security warning on installation. Applications using the unprotected capabilities, basic capabilities or extended capabilities can be signed through the standard Symbian Signed program. The manufacturer approved capabilities are the sensitive seven - DRM, NetworkControl, MultimediaDD, TCB, AllFiles, CommDD and DiskAdmin and are capabilities which have particular implications for device data integrity and security. [8]

So a trojan intrusion will only be possible if you give it permission to do so, and after you've also given permission for an 'unsigned' application to be installed. For applications you do trust that are unsigned, you can give them permission ahead of time in the new Applications Manager, and you won't be pestered by the permission requests. This is much like the current security model for Java MIDP on some Symbian powered phones. Of course all this assumes that the end-user is sensible enough to say no when something suspicious happens and permission is asked for a program.

9. Development Environments

Free Software Development Kits (SDK) are available for download from the web. They allow the development of third part applications. Also Integrated Development Environments (IDE) are available for free in order to be able to develop Symbian applications.

9.1. SDK

9.1.1. S60 Platform

Series 60 Platform SDK for Symbian OS, for C++ allows C++ developers to quickly and efficiently run and test applications for devices that are compatible with the S60 Platform. The SDK delivers all the tools required to build C++ for Symbian OS applications. The tool's package contains the S60 device emulator, API implementations, documentation, and sample applications. The S60 Platform device emulator allows applications to be run and tested without a device.

9.1.2. S80 Platform

Series 80 Platform SDK for Symbian OS enables C++ application development for devices based on Series 80 Platform. The SDK is compatible with the Nokia 9300 and Nokia 9500 Communicator and includes wide range of tools, APIs, libraries, and documentation, as well as an emulator. The SDK includes Application Wizard for generating skeleton applications for Symbian OS SDKs and phones and the Nokia Connectivity Framework for communication with other Nokia SDKs and products supporting Nokia Connectivity Framework connectivity.

9.1.3. UIQ Platform

The UIQ SDK is a development kit intended for software developers and organizations developing applications and services for mobile phones that incorporate the UIQ platform (Sony Ericsson & Motorola). The UIQ SDK contains binaries and tools to facilitate building and deploying applications for UIQ-based mobile phones. It also contains comprehensive reference and guidance materials, examples, and a PC-based emulation of the UIQ platform to enable fast development turn-around times. Application development in both C++ and Java is supported.

9.2. IDE

Visual Studio .NET is a development environment built from the ground up to enable integration through XML Web services. By allowing applications to share data over the Internet, XML Web services enable developers to assemble applications from new and existing code, regardless of platform, programming language, or object model. With Visual Studio .NET 2003 you can now construct applications for a variety of mobile devices, including Pocket PCs, Tablet PCs, mobile phones, and more.

CodeWarrior for Symbian OS is designed for application developers programming for Symbian OS phones from OEMs including Sony Ericsson, Motorola, Nokia, Samsung, Siemens and Panasonic. It contains the same powerful suite of CodeWarrior application development tools used by many Symbian OS licensees, delivering everything you need to build native C++ applications for Symbian OS phones. Using CodeWarrior Development Studio, Personal Edition, you can: develop, compile and debug C/C++ applications for Symbian OS smart phones within a single environment; develop applications for multiple targets simply by plugging in device-specific Symbian OS SDKs; streamline development with integrated Symbian build components; test and debug host and target applications (both source- and assembly-level) and shared libraries - even debug multiple executables simultaneously; navigate and edit code instantly via the graphical class browser.[12]

The Carbide product provides best-in-class tools for mobile developers across many development technologies, including Symbian C++. Carbide.C++ provides a complete set of application development tools to target the S60 and UIQ SDKs.

Carbide.C++ provides a complete set of application development tools needed to target the S60 and UIQ SDKs and to build and deploy applications to devices. It is anticipated that customers using CodeWarrior for Symbian OS, will be provided with migration paths to Carbide.C++.

Carbide.VS is a set of tools that enable efficient Symbian OS C++ application development using the Microsoft Visual Studio .NET 2003 IDE and Symbian OS SDKs. Carbide.VS is targeted at developers with Visual Studio skills who want to create C++ applications for Symbian OS platforms. Carbide.vs support development for multiple SDKs and provides easy entry into Symbian OS C++ development with wizards and other automated functions that integrate with Visual Studio. User can get started with minimal manual configuration. Carbide.VS also contains functionality to automate several development tasks specific to Symbian OS. Carbide.VS.[12]

10. References

[1] Symbian URL: www.symbian.com

[2] AppForge – CrossFire. URL: http://www.appforge.com

[3] S60 Platform. URL: http://www.s60.com/

[4] UIQ Technology. URL: http://www.uiq.com

[5] M. Tasker, J. Allin, J. Dixon, J. Forrest, et Al, *Symbian Programming: Mobile solutions on the EPOC platform*, ISBN: 1-861003-03-X, Wrox Press Ltd. 2000

[6] "Enterprise: Developing End-To-End Systems", Forum Nokia, 2006. URL: http://sw.nokia.com/id/ebe4cf33-741d-4a4d-bf14-e28937090aa4/Enterprise_Developing_End-To-End_Systems_v1_0_en.pdf

[7] Symbian Developer Network. URL: http://www.symbian.com/Developer/

[8] All About Symbian - Symbian, Series 60 and UIQ unwrapped. URL: http://www.allaboutsymbian.com/

[9] The Linux Phone Standards Forum. URL: http://www.lipsforum.org/

[10] QTopia - Linux-based mobile phones. URL: http://www.trolltech.com/

[11] Java Platform, Micro Edition (J2ME) http://java.sun.com/j2me/

[12] Carbide. URL: http://www.forum.nokia.com/carbide