

Native and Java ME Development on Symbian OS

Sam Mason and Elise Korolev

Published by the Symbian Developer Network

Version: 1.0 – March 2008

1 INTRODUCTION	2
2 PERFORMANCE OF NATIVE AND JAVA ME APPLICATIONS	2
3 DEVELOPING WITH C++ AND JAVA ON SYMBIAN OS	3
3.1 APPLICATION START TIME.....	4
3.2 ATOMICITY OF INSTALLATION	4
3.3 ACCESS TO SYSTEM APIS	4
3.4 PORTABILITY AND FRAGMENTATION	5
3.5 SECURITY	7
3.6 TOOL AVAILABILITY AND LEARNING CURVES.....	7
4 JAVA ME ON SYMBIAN OS	8
5 PERFORMANCE ISSUES IN GAMES	8
6 MOBILE ECONOMICS	10
7 CONCLUSION	11

1 Introduction

This paper provides a general comparison of the two main technologies used for application development on Symbian OS at this time – Symbian C++ and Java ME. Since technologists are often polarized in their personal and professional opinions of these technologies, it is important to state that it is not the purpose of this paper to support either side of this debate. Rather, we argue that in a world of commercial realities, there are strong cases for using (and being proficient in) both.

While Java ME and Symbian C++ are currently the most widely used on Symbian OS, there are also a number of other choices available, such as runtimes like Flash Lite and Python. Since no single technology can meet all the needs of every developer, offering this variety allows developers to select the technology most suitable for their applications. This leads to a richer ecosystem and greater customer choice, as can be seen in the table below.

		Feature phones	S60/UIQ on Symbian	Technology Features & Notes
Development environment	Python		✓	Ease of development (scripting language)
	Java ME	✓	✓	MSA defines the next step in the Java platform evolution contributing to defragmentation. S60 on Symbian will support for JSR-248 subset which is mobile optimized and ensures cross vendor compliancy. Some variation between UIQ and S60 JSR implementations.
	Flash	✓	✓	Flash Lite is a very productive environment for highly graphical, compelling user interfaces and full end-to-end services. Flash Video support for the browser enables the Google and Youtube video services.
	C		✓	Standard C libraries - extending the native C++ development environment. Leverage on existing C skills. Portability from other systems. Re-use of existing C code and components
	C++		✓	Native apps provide better usability (which is especially important for mainstream users): the browser UI isn't adaptable enough. Native apps cope better with intermittent connectivity as in push email
	Widget		✓	Widget at its core is a Web page designed for a single specific function using familiar technologies like HTML, CSS, JavaScript, AJAX, as Web pages but runs, operates & behaves like a native application. Can be developed in days

Keeping in mind that our discussion mainly focuses on Symbian OS, we will use the term “native” to refer specifically to an application written using Symbian C++ and compiled to run on a particular processor such as ARM. This is necessarily more restrictive than the more general definition of a native application used in other sectors of the IT industry.

We will also use the term “feature phone” to refer to mainstream mobile phones. These usually run a proprietary operating system and allow only Java ME applications to be installed after market. Feature phones tend to focus on basic phone functionality and are not considered to be “smartphones”.

2 Performance of Native and Java ME Applications

Symbian C++ code is compiled into ARM instructions directly, so no interpretation is needed at runtime. Consequently, native C++ applications measurably outperform Java ME applications in almost all cases.

On the other hand, Java is an interpreted language, so Java ME applications (MIDlets) must execute in a virtual machine (KVM) that has to interpret the Java instruction codes (byte codes) into native code before executing the appropriate ARM CPU instructions. As with any layer of abstraction, this has an adverse effect on the runtime speed of Java ME applications – that said, it is important to note that the Java language itself was designed primarily for security and portability, and not for speed.

However on Symbian OS, the performance differences between native C++ and Java ME are now less than an order of magnitude. Symbian engineers have put a large amount of effort into speeding up Java ME applications by using techniques such as Dynamic Adaptive Compilation (DAC), which takes the most commonly executed pieces of byte code and compiles these at runtime into native binary code, and the use of lightweight threads within the KVM. An optimized port of Sun Microsystems' CLDC 1.1 Hotspot VM has shipped on Symbian phones since Symbian OS v8.

Given that mobile games remain the golden child of the mobile application commerce, it is important to look at the performance benefits that native development offers over Java ME development in the game sector. This is so important that we devote a separate section later in this paper to games performance, which is based on a case study from Ideaworks3D.

3 Developing with C++ and Java on Symbian OS

Symbian OS offers many features as a development framework, including fine-grained control over resource utilization, access to phone hardware and system services, a robust security model, and support for a wide range of industry standards. Some of the benefits and disadvantages of working on Symbian OS using native C++ and Java ME are summarized in the following table.

	Java ME on Symbian OS	Symbian OS C++
APIs	<ul style="list-style-type: none"> • Rich , easy to use , well documented and are familiar to thousands of programmers. • Designed collaboratively between various expert stakeholders in the industry (www.jcp.org) to ensure they are fit for purpose • 80% quicker application development than C++ 	<ul style="list-style-type: none"> • More APIs to play with you can obviously get at parts of the OS that Java doesn't cover • Richness of APIs mean that some applications cannot be done in Java, such as Skype
Battery and memory usage	<ul style="list-style-type: none"> • Instantaneous system/heap memory info only. • No access to battery information • No process control 	<ul style="list-style-type: none"> • Fine grain control over memory, process creation. • Potentially much better memory usage - every MIDlet launched requires a virtual machine running in its process this is ~1300 K. • Can query and monitor battery state
Access to phone functionality	<ul style="list-style-type: none"> • Limited on Java ME 	<ul style="list-style-type: none"> • Inter-process communication, accessing hardware, controlling other native applications such as the browser, etc. • Add assembler code to critical sections of your application
Execution	<ul style="list-style-type: none"> • slower execution speed than native 	<ul style="list-style-type: none"> • C++ applications run natively on the device, and are therefore much faster.

There are also a number of other points to consider when choosing between native C++ and Java ME development on Symbian OS, as we shall now discuss.

3.1 Application Start Time

A MIDlet requires the entire VM binary to be loaded into its process as part of the initiation sequence. Furthermore, the Application Management System (AMS) may need to be loaded if it is not already resident. This often results in MIDlet start times of 3-5 seconds. Given that studies have shown application start time to be a key factor in consumer satisfaction, this is a distinct disadvantage of using Java ME.

Obviously native applications do not suffer in the same manner. However we note that simply “going native” is not sufficient to guarantee rapid start up – the time taken is a function of the complexity of the application itself. An application that loads large data files or retrieves a large number of remote assets or real-time data on start up will necessarily take longer than a simpler one.

3.2 Atomicity of Installation

Native C++ applications can be updated on Symbian OS via the SWInstaller. This can be done either directly, via the package type of the SIS file, or implicitly, by updating/replacing operational data in the “imports” sub-directory of an application’s private directory. This allows a variety of after-sale support options (patches, updates, etc.) to be deployed to customers.

However the same cannot be said of Java ME MIDlets. While data updates to a given MIDlet are possible by downloading new data from a remote server (game levels, graphics, language translations, audio, etc.) and storing in the RMS, the MIDlet binary itself is atomic from the point of view of the SWInstaller. Therefore, in order to get an update, the new version must be installed over the old. There is, however, still an option to keep earlier data files as part of this process and a well designed MIDlet should be designed to support (or import and update) data from previous versions.

3.3 Access to System APIs

By virtue of being externally and centrally specified via the JCP, Java ME applications cannot take full advantage of everything any particular platform offers – and this includes Symbian OS. The APIs available within Java ME in the form of supported JSRs (Java Specification Requests) must be compared with a richer set of native APIs available to C++ developers.

Developers find that having access to the native APIs often enables them to better implement their applications, since they are a more integrated part of the OS and are not restrained by the KVM sandbox. By having more process and memory control as well as direct access to system services, native applications can make use of device resources such as windows, microphones and onboard cameras in a more efficient manner.

Contrast this flexibility with Java ME where if a JSR required for a project is not present on a particular phone then the developer either cannot work on the phone or must come up with some hybrid strategy to overcome the limitation (perhaps the required functionality can be bought or built as a Java library for example).

However using system APIs also binds developers into a specific platform and possibly even a particular operating system version which can then make porting across devices a problem. This is why Java ME was designed around a centrally managed platform-independent specification body.

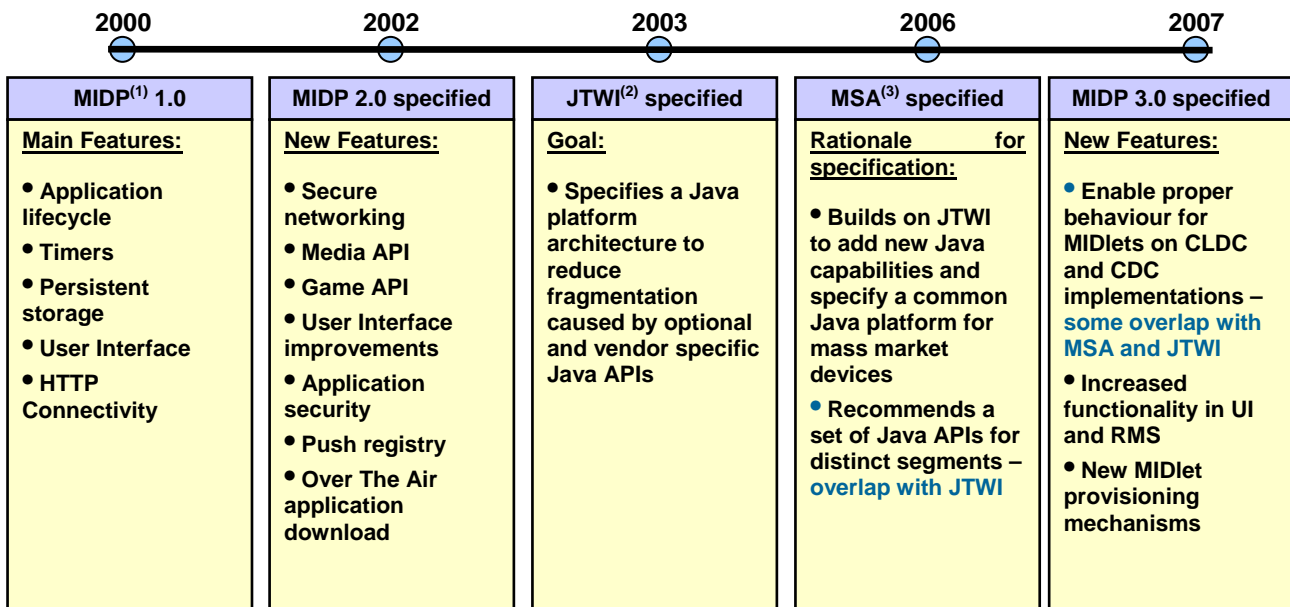
Essentially, whether portability is going to be a problem or not is going to depend on your project and your target market.

3.4 Portability and Fragmentation

“Fragmentation” has arguably been the largest and most costly problem in Java ME’s otherwise successful history. It arises because:

- In the real world, companies have their own agendas and are in competition. Consequently, when Java ME devices first hit the market, there quickly arose a number of proprietary libraries developed to address perceived “gaps” in the provided functionality. These mainly provided APIs for multimedia and game development and were shipped out on thousands of mobile devices well before standards could be finalized
- The implementation of JSRs varied greatly across devices - sometimes even when the devices came from the same manufacturer.
- No minimal mandatory set of JSRs was originally defined and so those chosen to be included in phone hardware by manufacturers were largely based on internal strategy.

There have been several attempts to create a consistent Java ME platform, as summarized below. While fragmentation continues, we are starting to see a large number of MSA (JSR 248) compliant phones emerging into the market. This should provide the economic drive for further JSR consolidation over the next few years.



- (1) Mobile Information Device Profile
 (2) Java Technology for Wireless Industry
 (3) Mobile Services Architecture

Where Java ME development on Symbian OS differs from mainstream Java ME is that Symbian has been able to tightly control the implementation and quality of the JSRs that ship with Symbian OS. These are well defined and map directly to operating system versions and consequently porting Java ME applications across a variety of Symbian OS phones is quite different to porting across a set of feature phones in the mass market.

In a specific case, Blaze, a leading games publisher, developed *The Da Vinci Code* game using Java ME. The game was targeted at all the major operators’ handsets, which amounted to more than 300 feature phones. As shown below, the overhead for porting across the set of proprietary

OS feature phones is an order of magnitude higher than when porting across a range of Symbian smartphones simply because the consistency of the JSRs under Symbian OS is so well defined.

	Java ME (Feature Phones)	Java ME (Symbian Phones)
Development	>300 50% project budget in overheads	1 5% project budget

Less obviously, developers of native C++ applications also encounter fragmentation albeit in a far more contained and predictable manner. There are a number of versions of Symbian OS, both prior to and following the version 9 binary compatibility break. Furthermore, there are significant differences between the UIQ and S60 UI platforms that have made porting across the range of Symbian OS devices a non-trivial undertaking.

An interesting comparison of the costs due to fragmentation in both technologies is presented in the following table. The table estimates the costs for porting a Java ME application across a set of feature phones and the costs for porting a native C++ application across a set of Symbian phones.

	Java ME (Feature Phones)	Symbian OS C++	
Investment	Design & Specification	0.5	0.75
	Development	1	2
	Port 1	0.125	0.1
	Port 2	0.125	0.1
	Port 3	0.125	0.1
	Port 4	0.125	0.25*
	Port 5	0.125	0.1
	Port 6	0.125	0.1
	Port 7	0.125	0.1
	Port 8	0.125	0.1
	Port 9	0.125	0.1
Porting costs	1.25	1.05	

* When porting to one of the other Symbian UIs

The costs associated with porting Symbian applications to new phones on the same UI are minimal, allowing primarily for retesting & tweaking only, while porting to each Java target typically costs 10-15% of the original development budget. (In this analysis, any server side development is omitted.)

In the case of Symbian C++, 25% extra in budget comes from a change in UI (for example, moving from S60 to UIQ). Even if there is no new work to be done, developers need to test the application again which can take up to 10% of the original budget; hence the 0.1 in the last column above. This is a conservative number but is a reflection of developer experience.

3.5 Security

The usability of many Java applications can suffer as the security model for certified applications can overload the user with prompts. Currently, Java ME applications use signing and a tiered security model to control access to protected APIs.

The benefit when installing native C++ applications on Symbian OS is that Symbian Signed's certificates are on all S60 and UIQ phones. This means that signing to Symbian Signed guarantees a good user experience during both install and at runtime.

At present, security policies on MIDP 2.0 devices, depending on the device configuration, can force the subscriber to grant/deny access to messaging, auto start, file read, file write, camera access and network access functionalities numerous times. This depends on whether the MIDlet has been signed or not (and consequently what protection domain it is part of) and what limitations (if any) have been imposed by the operator and/or manufacturer. Since security policies for protection domains can be set at both the manufacturer and operator levels this adds another layer of complexity to the security model.

On the other hand, Symbian Signed, the security program for native C++ applications, takes a more pragmatic approach. Applications declare their required capabilities and checks are performed at installation time. Applications that have been signed do not prompt the user when accessing protected APIs which makes for a much better user experience.

3.6 Tool Availability and Learning Curves

Availability of tools should not be a deciding factor in this discussion. Java ME development can be done using Eclipse, Netbeans or Sun's Wireless Toolkit (WTK), to name just a few.

For Symbian C++ development there is Carbide.c++ from Nokia, plug-ins for Visual Studio 2003 and 2005, and Metrowerks CodeWarrior. In addition, there are a number of free SDKs for S60 and UIQ development.

In many ways, Java ME really is just a version of Java Standard Edition squeezed down into a virtual machine that can run with a few hundred kilobytes. An experienced Java developer could probably learn most of the Java ME libraries, designs and strategies in a couple of days. A beginner with a decent textbook may take a few weeks.

Symbian C++ on the other hand is a much steeper learning curve. Even experienced C/C++ programmers have struggled with Symbian concepts such as descriptors, the cleanup stack, two-phase construction and active objects. Because it requires a specialized and advanced skill set, it can be both difficult and expensive to find proficient Symbian OS programmers in the current market. It can also be even more difficult to train junior programmers in Symbian C++ as many lack the experience for the more advanced concepts.

In the commercial world, these can be important and deciding factors when choosing development technologies.

4 Java ME on Symbian OS

Unlike other operating systems, Symbian has a very rich Java environment, meaning more capabilities than on most feature phones. Furthermore, as already mentioned, the Java ME implementation shipped with Symbian OS is best-of-breed, so MIDlets will generally execute at quite acceptable speeds on current smartphones.

Symbian OS still holds the following advantages compared to feature phones when it comes to the management of Java ME applications:

- **Robustness:** process separation is there for a reason; without it, if a game MIDlet triggers a native bug in the implementation of a JSR library, it can take out the VM and require it to be restarted. On a feature phone this will cause all running MIDlets to die without notification, potentially losing user data. On Symbian OS, MIDlets are fully protected from one another.
- **Platform integration:** S60 and UIQ are already full multitasking environments with facilities for managing an arbitrary number of running processes. On Symbian OS, a user does not have to learn anything new to manage MIDlet multitasking; MIDlets are handled in exactly the same way as native applications. On feature phones, MIDlet multitasking often doesn't sit well with the native UI. Instead it is common for there to be a separate "Java" application with its own task management UI.
- **No arbitrary limitations:** feature phones typically have to divide up the memory assigned to various features at device creation time, and the memory assigned to Java is fixed. On Symbian OS, because MIDlets are process-based and managed like native applications, they co-operate with the system-wide memory management policy. This allows complex and resource-intensive MIDlets to be run on Symbian OS using the full capabilities of the device without hitting artificially imposed limits.
- **Future proofing:** although feature phones can now just about handle CLDC/MIDP applications via enhanced VMs, they still cannot handle CDC-based Java platforms like the forthcoming JSR 249: Mobile Service Architecture Advanced and JSR 232: Mobile Operational Management, and there is little prospect of them ever doing so. In contrast, CDC/Foundation is already shipping on a number of Symbian phones.
- **Architecture:** on Symbian OS, Java ME applications are treated as first-class citizens. MIDlets completely integrate into APPARC and the view server, so switching between MIDlets and native applications is identical.
- **Fragmentation:** as already discussed, the available set of JSRs is well defined and highly optimised. This largely alleviates fragmentation problems that plague other platforms.

5 Performance Issues in Games

Mobile games are a large part of the mobile application market; in 2006 alone, sales of mobile games reached roughly \$2.5 billion. One of the biggest factors in consumer satisfaction for many types of games is the quality of 3D graphics. Taking a look at 3D game development, the 3D rendering from Java ME is poor when compared to native applications, as explained by Ideaworks3D (Source: Games conference 2007):

3D rendering with Java ME (MIDP 2.0 with JSR 184, JSR 239, Mascot Capsule)

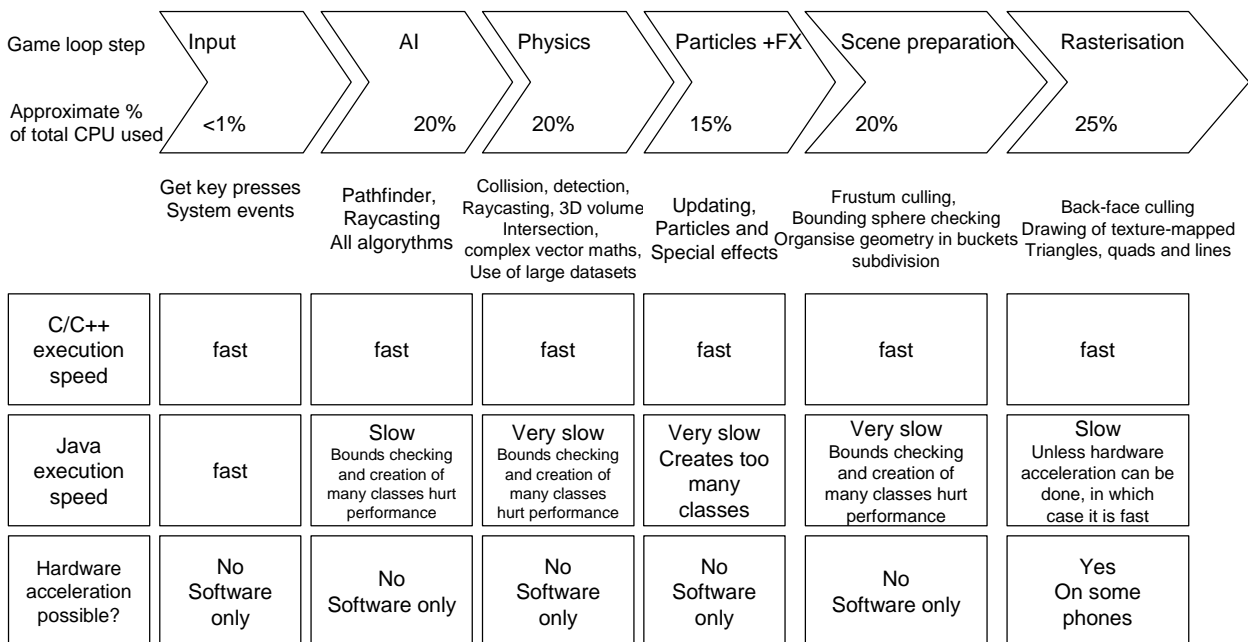
- A game is written in Java byte code and runs in an interpreted mode on the VM.

- Performance today is highly variable, and often gated by interaction of Java VM with a native 3D engine for rendering. First generation JSR 184 points to the need for hardware acceleration to achieve acceptable frame rate, in particular as screen resolution approaches QVGA (320 x 240 pixels).

Native C/C++ (S60, WIPI, Linux, Windows Mobile)

- A game is authored “to the metal” in C or C++ and is compiled to the OS platform so it can execute directly to the ARM instruction set.
- I3D’s evaluation of all native OS platforms points to uniform performance on comparable ARM processors. Minimal code optimization is needed to compile for high performance on native platforms.
- Smartphones are broadly at parity with the performance of Nintendo DS without additional hardware acceleration.

Here’s a comparison of the performance of code written in native C++ and in Java ME for different parts of game execution:



Source: ideaworks3D

Ideaworks3D concluded that only native C++ applications can achieve console-level game performance:

- Console-level games cannot perform at sufficient speed in Java, even with hardware acceleration.
- Games mechanics are generally too complex to perform well in Java byte code.
- A truly console-class game will require more complex game engine features than any current generation Java and JSR 184 games.
- Executing game code in Java byte code would mean that the game runs only as fast as the main loop, and so would not take advantage of the full speed of Open GL ES hardware.
- Mobile 3D games should be therefore split into two performance categories:

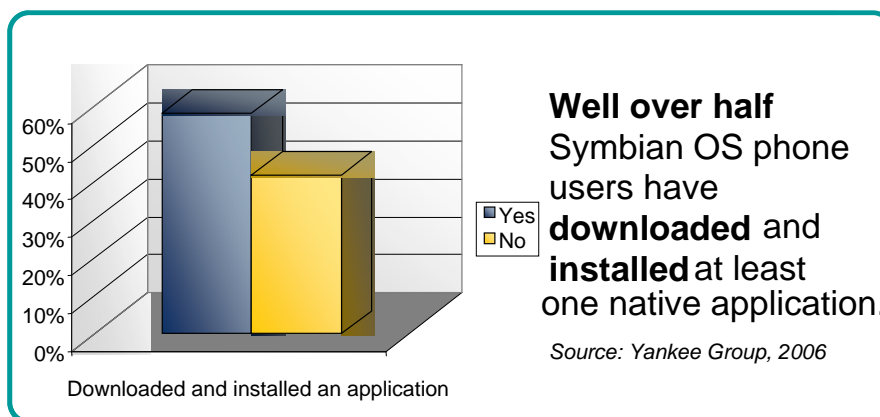
- Java ME for “casual” 3D experiences
- native C/C++ for console-level game experiences.
- Content developers/publishers must choose the right design for each platform, depending on the desired effect.

There should be no surprises in the results presented above. Native applications, whether on Symbian OS or not, out-perform Java ME applications in the mobile game space especially in the area of 3D rendering. However, there are two points that the reader should still keep in mind:

- Console-level quality is not always economically feasible or even desirable and many developers and companies are not aiming quite this high.
- Java ME is the largest platform for game development. The proportion of games written and sold worldwide using Java ME, with or without 3D graphics, is significantly higher than that for games written using any other technology

6 Mobile Economics

As shown in the graph below, smartphone users tend to install more native applications written in native C++ than in Java ME.



However, it is possible that this is simply an artefact of the search process as the average user with a Symbian device is more likely to Google using keywords like “Symbian” and “Software” when looking for applications online.

Native applications span a much wider range of categories and command a higher price – for example, the average selling price of a typical Java ME application (usually a simple game) is of the order of £3.50 whereas Symbian C++ titles are on average almost three times that.

Open OS-based devices application metrics	Symbian-based devices application metrics
<ul style="list-style-type: none"> • Avg. number of applications purchased per order: 1.30 • Avg. selling price of content: \$20.35 • Number of new content providers: 258 • Number of new content titles: 3,303 	<ul style="list-style-type: none"> • Avg. number of applications purchased per order: 1.21 • Avg. selling price of Symbian content: \$20.90 • Number of new Symbian content providers: 165 • Number of new Symbian content titles: 2,397

Native C++ and Java ME applications are increasingly available for download over the air. Novintel estimates that there are over 900 independent distributors delivering applications written in native C++ for S60 and Java ME applications over the air:

- An estimated 503 service providers (55 %) offer both native C++ applications for S60 and Java ME applications.
- 358 service providers (39 %) are estimated to offer only Java ME application download services.
- 56 of the independent distributors (6 %) focus exclusively on native C++ applications for S60.

7 Conclusion

This paper has compared some of the factors to consider when choosing which of the two main technologies – C++ or Java ME – to choose for developing an application for Symbian smartphones. As we described, Symbian C++ is a more expensive skill set than Java ME, and ultimately the development costs alone may be the deciding factor when choosing which technology to use. It is quite possible to decide that it is preferable to have a Java ME version of a given application that only runs at, say, 75% of the speed of the equivalent native one, but has less than half of the development costs because of the reduced development cycle.

If you then take into account that most Java ME applications that run on Symbian OS will most likely also run largely unchanged on Nokia's Series 40 feature phones too, then the potential return on investment for a given development project may very well overshadow the more esoteric technical hurdles when viewed from a marketing and budget standpoint. Java ME applications will reach a much wider audience than native applications can by their very nature. Even if you are only targeting Symbian OS phones, there are often valid reasons to use Java ME for development.

Conversely, there are many reasons to prefer native development over Java ME on Symbian OS, especially for applications where performance is crucial (for example, 3D console-level games) or where the application use cases require low level access to the rich set of operating system services exposed by Symbian OS, such as the messaging sub-system, the DBMS, image manipulation and conversion, direct screen access, event logs, call interception and management and much more.

As a real world example, Google Maps is available as a native C++ application for Symbian OS v9 (S60 3rd Edition smartphones only) which noticeably outperforms the more generic Java ME version. As is usually the case, this is largely due to Java's memory footprint which directly affects memory intensive operations such as panning and zooming images. Furthermore, the native application allows routes and favourite maps to be saved locally, has better integration with the GPS radio and faster access into the address book.

It is also interesting to note that, as mentioned earlier, the faster start-up time of the native application had the greatest impact on user satisfaction. For more information, see www.allaboutsymbian.com/news/item/6077_New_Google_Maps_with_GPS_supp0.php.

Symbian C++ and Java ME complement each other as mature, robust and commercially viable development platforms and we have discussed a number of valid economic reasons for using either technology in the marketplace of the real world.

There are many reasons to take either path but the choice you eventually make must necessarily depend on what you are trying to achieve.

Want to be kept informed of new articles being made available on the Symbian Developer Network?

[Subscribe to the Symbian Community Newsletter.](#)

The Symbian Community Newsletter brings you, every month, the latest news and resources for Symbian OS.



Sam Mason

Sam Mason's company, [Mobile Intelligence](#), is an Australian-based software engineering company specialising in application development for mobile phones and other resource-constrained devices. Sam was a contributor to the recently published Symbian Press book ***Games on Symbian OS: A Handbook for Mobile Development***, and is currently working on another book. He became the first person to sit the supervised Accredited Symbian Developer (ASD) exam in Australia, in March 2007, and wrote about the experience for the Symbian Developer Network, in a paper called [Taking the ASD Exam](#).