University of Groningen
Computing Science

Master of Science Thesis

A Survey of Mobile Platforms for Pervasive Computing

by

Hielko van der Hoorn

Supervisor: Marco Aiello
Second Supervisor: Alexandru C. Telea

Groningen, 2010

Contents

Contents i

1 Introduction 3

2 Related Work 5

3 Qualitative Analysis 9

3.1 The Quality Attributes . . . 10
3.2 The Operating Systems . . . 12
3.2.1 Android . . . 12
3.2.2 BlackBerry OS . . . 16
3.2.3 iPhone OS . . . 19
3.2.4 Symbian . . . 24
3.2.5 Windows Mobile . . . 28
3.3 Summary . . . 31

4 Quantitative Analysis 33

4.1 Experimental Setup . . . 34
4.2 Emulator Results . . . 36
4.2.1 Encryption . . . 36
4.2.2 Decryption . . . 37
4.2.3 XML Parsing . . . 38
4.3 Host System Dependency . . . 39
4.4 Results on Mobile Devices . . . 40
4.4.1 Android . . . 40
4.4.2 iPhone OS . . . 41
4.4.3 Symbian . . . 42
4.4.4 Side by Side . . . 43
4.5 Summary . . . 44

5 Conclusion and Future Work **47**
5.1 Future Work 49

Bibliography **51**

A Emulator Benchmark Results **57**

B Smartphone Benchmark Results **59**

List of Symbols and Abbreviations **61**

List of Figures **62**

List of Tables **63**

Acknowledgments

I would like to thank the following people for making this master thesis possible: my supervisor Marco Aiello for his ideas, his valuable feedback and the fast response to emails, no matter the time of day. My second supervisor Alexandru C. Telea for providing detailed and insightful criticism. Gerard Drenth en Timo Schinkel for lending out their smartphones for my experiments. My friends and family for providing welcome distractions while working on my thesis, and at the same time pushing me to finish it.

Chapter 1

Introduction

Mark Weiser proposed the term "ubiquitous computing" around 1988. In one of the first papers on the subject he stated "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it [51]." This idea is the basis behind pervasive computing, also known as ubiquitous computing, ambient intelligence and everyware. The vision of pervasive computing is to have multiple networked devices doing all kinds of everyday tasks, usually in the background without the user being aware of the work that is being done.

An example is a house that adjusts the lighting and heating based on the body temperature of the occupants and the level of light outside. Another possible application is to monitor someone's health using various sensors and alert health care when something goes wrong. The most common example is probably the intelligent fridge. This fridge should be aware of the contents inside, so that it can warn users when food is out-of-date, suggest possible menu's that can be created with the available food, and order new food when certain supplies are running low.

Figure 1.1: 'Smart' fridge

Although the idea of pervasive computing is not new, the technology is not yet widespread. The intelligent fridge as envisioned above is not yet available, and most pervasive computing applications have not left the research state. Smartphones are however becoming increasingly popular and since almost everyone carries their phone with them all the time, they are becoming one of the first viable platforms for pervasive computing [40].

In this thesis we address the question *"What are the most important features of a mobile OS for ubiquitous computing applications and how do current mobile OS perform with respect to these features?"*

A mobile OS can run on a variety devices such as mobile phones, smartphones, PDAs and handheld computers. Since smartphones are touted to be the first viable platform for pervasive computing we will focus on the most popular - based on marketshare - mobile operating systems in use on todays smartphones.

Before addressing the research question an overview of the state of art of pervasive computing is provided in Chapter 2 and how this relates to the work done in this paper. After this introduction the most important smartphone operating systems and the characteristics that play the biggest role in ubiquitous computing applications are identified in Chapter 3. In the remainder of the chapter the various operating systems are analyzed based on these criteria. After the qualitative analysis in Chapter 3 the different operating systems are also quantitatively analyzed in Chapter 4 using a self-developed benchmark. Finally some conclusions are drawn in Chapter 5.

Chapter 2

Related Work

The idea and vision behind ubiquitous computing was born in 1988 when Mark Weiser published his paper called "The Computer for the 21st Century" [51]. Since that date the interest and research into the subject of ubiquitous computing has grown immensely. Worldwide there are a several dozen research centers that are focused on ubiquitous computing, and some of them have a special focus on the mobile aspect of ubiquitous computing.

A key requirement of ubiquitous computing is networking, and since the user should not be bothered unnecessary tasks like network configuration, device and service discovery should be automated. The second key requirement is a 'natural' user interface. A interaction paradigm appropriate for the ubiquitous case has yet to emerge. Research is being done on topics such as context awareness, speech recognition and computer vision. A smartphone - with a mobile OS at its core - would be the platform that provides services such as computational power, interaction, communication and storage on which these technologies could be employed.

MIT tries to tackle various obstacles of ubiquitous computing at once with the Oxygen project. The vision of the project is to bring abundant computation and communication, as pervasive and free as air, naturally into people's lives [36]. To make this possible MIT has identified various technical challenges that must be solved. The system needs to be embedded in our world, be flexible to deal with users moving, devices coming online and going offline and it has to understand the intentions of users.

Since the environment where pervasive computing applications live is highly dynamic the underlying network technology should be designed to cope with this. *Grid* is a routing protocol that is part of the Oxygen project. It is a technology to set up self-configuring ad-hoc wireless networks without fixed infrastructure such as base stations or access points [31]. A second technique goes by the name *Span* and is a designed to save power in multi-hop ad hoc networks by turning off idle nodes while preserving network capacity and global topology [11]. Other network technologies developed as part of the Oxygen project include *Chord*, a scalable distributed lookup protocol for peer-to-peer networks and the Intentional Naming System (INS) for dynamic resource discovery based on properties and attributes of the resources and data [12, 9].

Interaction with pervasive computing applications is another focus point of research. The vision is that humans can communicate with computers in a similar way as with one another. Speech should be recognized, understood by the computer, and based on what the user wants a meaningful spoken reply should be synthesized. MIT has developed several spoken language interfaces that are accessible by phone. Jupiter provides weather information, Mercury and Pegasus both provides airline flight information and Voyager provides tourist and travel information in the Boston area. All these systems are build using the same technologies; *Summit[23]* for speech recognition, *Tina[46]* for natural language understanding, *Genesis[47]* for language generation and *Envoice[52]* for speech synthesis. Computer vision is the second part of the interaction equation. With the help of cameras it is possible to track the location of a person in the room and combined with face detection applications can perceive where people are looking [14].

MIT is also doing research into the hardware that is needed to make pervasive computing a reality. The result of this research is the H21, a mobile device equipped with a microphone, speakers, display, camera and wireless network access [48]. The prototype is based on a Compaq iPaq PDA that is extended with a camera, an accelerometer, a FPGA chip and additional I/O capabilities. The overlap in features between this prototype and a modern smartphone is remarkable. A FPGA chips is probably the only hardware that is not available on a smartphone, but the CPU of a smartphone is a lot faster. The prototype H21 has a 200Mhz StrongArm processor while for example the latest iPhone - the 3GS - has a 600MHz ARM11 processor.

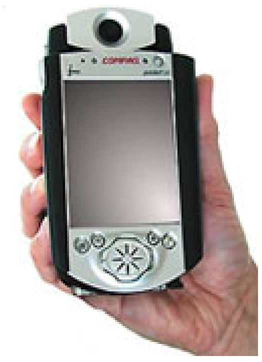
Figure 2.1: H21 device

Carnegie Mellon University also has an ambitious ubiquitous computing project named Aura. In this project the focus is on creating an environment that minimizes distraction on a user's attention. The hypothesis is that not processor, memory, disk or network is the most valuable resource in a computer system, but human attention [20]. To reduce user distraction the system needs to be *proactive* and *self-tuning*:

anticipating requests from higher layers and observing demands and adjusting performance accordingly. A scenario that is envisioned is a user that is trying to send some email's at the airport before taking a flight, but the wireless connection is too slow to send them in time. Aura will notice this, suggest to the user where more wireless bandwidth is available, so the task the user wants to perform can be completed.

The list of technologies being explored as part of the Aura project is too long to list. They overlap for a large part with the Oxygen project such as intelligent networking, multimodal user interfaces, intelligent workspaces and capturing high-level user intent. The most important capabilities of Aura are supporting user mobility and shielding users from variations in resource availability. For this a new layer of system abstraction has been introduced that sits between the user and individual applications called Prism. It is in charge of executing tasks, based on the user intent. This means for example when a user is on the move replacing a certain service with another one, or reconfiguring them.

In Europe the Amigo project focuses on developing standardized middleware and intelligent user services for the networked home environment [2]. At the moment different industrial players create different kinds of electronics for consumers to use in home each with different standards and form factors. The goal of the Amigo project is to develop standards that will make it possible to connect electronics from different domains such as dish washers, radios, televisions, (smart)phones and the computer. The second part of the Amigo project is the development of attractive services that are not possible with the current non-networked systems.

Compared to the Oxygen and Aura projects Amigo has a smaller scope and a more clearly defined goal, but at the same time to make the networked home and the intelligent services a reality a lot of similar problems have to be solved. Network configuration and service discovery should be automatic and dynamic while awareness about where the user is and what he wants is required for smart services. The Amigo project ended in 2008 and has resulted in an open-source software layer that can be used to create context aware networked applications for the home environment. Several sample services also have been developed such as *home care and safety* where calamities such as gas and water leakage are detected. Personal health is also monitored and food and exercise patters are suggested. The *home information and entertainment* system has been developed to show how applications can be created that can be accessed by the user on any device, no matter where they are. The *extended home environment* offers communication facilities between the home environment and persons in other environments [16].

An European project, that has the University of Groningen as one of its participants, with similar goals is the SM4ALL project. This project also focuses on creating a middleware platform for smart embedded services at home. The goal is to use peer-to-peer (P2P) technology to create a scalable solution that can handle adding and removing new devices and services on-the-fly. A special focus will be on ontologies for describing service capabilities to make dynamic configuration and composition of the services possible [1]. To establish interoperability between various types of embedded devices web services are used. Using web services on smartphones is with relative little effort possible thanks to the various toolkits are available for this task. Selecting the right toolkit is important since the performance varies [45].

Security and privacy also have been identified as a key requirement in the SM4ALL project because a smart home environment will contain a lot of sensitive information about the user, especially if it has been designed to support the elderly or the sick. A goal of the project is to develop a home that showcases how intelligent services can help those that are disabled.

Chapter 3

Qualitative Analysis

In order to determine the characteristics of the various operating system for ubiquitous computing applications a look at the most important quality attributes for this set of applications is taken in this chapter. In Chapter 4 the operating systems are evaluated based on quantitative measurements.

In the qualitative analysis the five most popular smartphone operating systems based on several theoretical properties are compared. Six quality attributes that are important for pervasive applications running on smartphones have been selected. The concurrency model, the networking capabilities and the security and privacy features are especially relevant for ubiquitous computing. The cost, memory use and power consumption are on the other hand important for the fast majority of mobile applications. The reasons for choosing these quality attributes are discussed in more detail in Section 3.1. The evaluation of the quality attributes is based on the API and SDK documentation provided by the manufacturers of the operating systems, but some other sources have been used as well.

The operating systems are compared from the perspective of a third party developer, and as such it is important to note that there is a difference between what each operating system truly supports and what is legally possible. For third party developers it is often not possible to access the full capabilities of the operating system. Not because it is impossible, but because it would break the license agreement. Apple arguably has the most strict set of rules in place, and states for example in Section 3.3 of the SDK agreement [8]:

> Applications may only use Published APIs in the manner prescribed by Apple and must not use or call any unpublished or private APIs.

As this quote illustrates the iPhone OS is capable of far more things than the developer is allowed to use. The technical part of these restrictions are easy to bypass [15], but breaking the license agreement means that a developer of the third party software cannot legally distribute his product to customers. For serious applications the bounds of the license agreement determine how suitable an operating is for a certain application just as much as the feature set of the operating system itself. Where possible it is pointed out where limitations are mainly the result of legal aspects instead of the characteristics of the operating system.

3.1 The Quality Attributes

The various operating systems are compared on a qualitative basis using six different criteria, in alphabetical order.

Concurrency Model

The concurrency model of the various operating systems is important because pervasive computing applications often have to perform tasks in the background without interrupting the user. For desktop applications the ability to run multiple applications and services concurrently is taken for granted, but when dealing with the limited resources of a mobile phone the availability of such functionality is not standard. The most important issue is the question if it is possible for a third party developer to run multiple processes and threads. Another characteristic that will be discussed is if the operating system supports running real-time processes.

Cost

The second quality attribute is probably the least important in our comparison, but a comparison without considering the licensing costs of the different options is not complete. A complicating issue with the cost comparison is the fact that most mobile operating systems are bundled directly with a mobile phone, and are not available for purchase as just the software product. Because of this the cost has to be measured as the average licensing fee per phone. Sometimes this data is published by the company selling the operating system otherwise estimations from third party research firms have to be used.

Memory Use

Although the computational capabilities of the average smartphone far exceed those of the more traditional mobile phone, resources are still very limited compared to a desktop computer. The operating system is in charge of the memory management and it has a direct impact on how much memory third party applications can access. The memory requirements of the operating system itself are a second factor that influence third party applications. The focus will be on the RAM use since most modern smartphones are equipped with multiple gigabytes of flash memory for persistent storage. The characteristic of the flash memory that will be discussed is if it can be used for virtual memory, or not.

Networking

Ubiquitous computing applications are often centered around multiple devices that are connected to each other in a network, making the networking aspects of the different mobile operating systems an important quality attribute. All the operating systems that are discussed in this thesis can create a TCP/IP connection and are therefore capable of connecting to most networked services. There are however differences with respect to the support for (wireless) networking protocols, various levels of support for different standards in the default Application Programming Interfaces and the permissions third party applications are granted.

Power Consumption

Mobile devices have a limited battery capacity, and that means that the power consumption of the operating system and third party applications is a major concern. All handheld devices share the same Achilles' heel: the battery [33]. The fundamental vision behind pervasive computing is that devices - smartphones in our case - work in the background, wirelessly connecting to other devices, exchanging data while performing useful tasks for the user. While it is maybe acceptable that the mobile phone battery is depleted after a few hours of calling, it is not acceptable that a pervasive computing application that is working in the background has a similar effect on the battery life of the smartphone. It is crucial that the mobile operating system has features that increase battery performance, such as the use of low power states when running low priority background processes.

Security and Privacy

The last important quality attribute that is discussed are the security and privacy features of the mobile operating system. Most pervasive computing applications store private information about the user like his location, his preferences or his health. Because of the sensitive nature of such data it is desirable that the operating systems contains features for protecting it. This could happen at several levels, for example by restricting other application on the phone from accessing this data or offering encryption functionality. Also important is functionality to protect against common attack vectors such as a man-in-the-middle attack, and protecting user data when the physical device is lost.

3.2 The Operating Systems

Five different operating systems are examined; the most popular ones. These smartphone operating systems are Android, iPhone OS, BlackBerry OS, Symbian and Windows Mobile. Iphone OS and Android are both newcomers on the smartphone market while Symbian and Windows Mobile have been around for years. BlackBerry OS also has been around for years, but smartphones based on this operating system have only been targeted to business users until recently. Symbian is by far the most popular operating system in this list and has a well fortified position with a market share of 51 percent [21].

3.2.1 Android

Android is the newest operating system that will be discussed. The first public beta version was released on 12 November 2007, but the first mobile phone with the operating system did not see the light until 23 September 2008. The Android operating system is developed under the umbrella of Google and is at the core based on Linux. There are however several layers between the Linux kernel and the Application layer, as can be seen in Figure 3.1.

The lowest layer - the red layer - of the Android operating system is the Linux kernel. It includes the hardware drivers for the various components of the mobile phone and it provides the core system services such as memory management, process management and the network stack. Android is build upon Linux version 2.6.

The green layer consists of libraries that can be used by every android application (subject to security constraints [25]). Available libraries are for example WebKit and

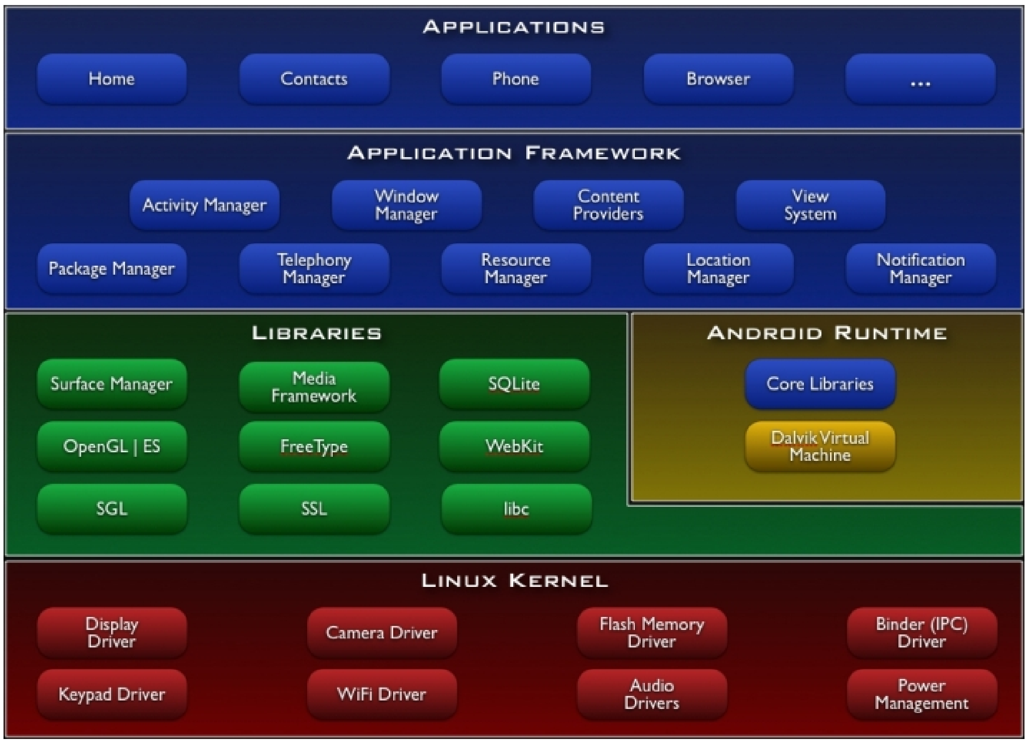

Figure 3.1: Overview of the Android architecture [25]

SQLite: providing respectively an embeddable web view and a lightweight database engine.

Above the green layer is the blue layer: the application framework and the applications. The application framework consists for a part of packages that are part of the Java SE 5 specification. Because of this Java developers can easily develop applications for the Android platform, but the Android SDK is not compatible with Java SE 5. There are for example Android specific packages needed for the user interface and for access to resources such as databases, files and the GPS location. Only core packages such as *java.math* or *java.util* are the same, but for the other functionality similar packages are available in most cases. A tool that automatically converts Java ME packages to Android packages is accessible online*. It is not possible to develop native applications in C or C++ as a third party developer.

Concurrency Model

Android is based on Linux, and offers almost the same features with respect to the concurrency model. Running background services is possible for third party devel-

*Available from http://www.netmite.com/android/

opers. The big difference is that services are not guaranteed to be running. When memory is low the operating system will try to kill processes that are not critical for the user experience. In practice this means that the application that is running on the foreground will be kept alive together with the services that are used by this application. Background processes that have no direct impact on the user experience may be killed at any time [26]. This kind of behavior definitely could cause problems for some pervasive computing applications that would rely on a background process running on a smartphone.

Cost

Android is developed by the Open Handset Alliance with Google as the main force behind the project. The Android operating system is build on the Linux kernel and the source code - including the standard Android applications - is available under the Apache license. It can be downloaded[†], modified and used for free. It is not a copyleft license, meaning that the source code may be used in proprietary software. For developers of third party applications the licensing details are not important: individual applications can be distributed using their own licenses.

Since the source code of Android is available for free the costs of using the operating system are minimal, but hardware specific device drivers are required. This is also limiting efforts from enthusiasts to get Android running on mobile devices shipped with other operating systems.

Memory Use

Android is based on the Linux kernel that has proven itself to be able to run with minimal hardware resources. Applications on Android are however not executed as native applications, but are run inside the Dalvik virtual machine. The Dalvik VM is a register based virtual machine that is optimized for low memory requirements and to allow multiple VM instances to run at once. The Dalvik VM is created to run Java applications, but it is not a Java virtual machine. It runs Java applications that are converted to the Dalvik Executable file format. One of the other big differences between Dalvik and other Java virtual machines is the lack of a just-in-time compiler. Although Dalvik has been optimized for low memory requirements the need to run every application inside a virtual machine is without a doubt costing memory compared to a architecture that uses native applications and the lack of a just-in-time compiler could hurt the performance.

[†]Available from http://source.android.com/

Networking

Android is marketed as a mobile phone with 'always on' Internet. Android phones are almost always connected to the Internet using the cellular network or a nearby WiFi network. Internet connectivity is therefore almost always available. Programmatic control over the WiFi connection is excellent. Using the Android API applications can scan for WiFi networks, connect to WiFi networks and keep the WiFi radio active when the device returns to idle [26]. User preferences cannot be overridden; when the WiFi functionality is switched off or when airplane mode is enabled creating a wireless connection is not possible.

In the first versions of Android Bluetooth functionality was completely non-existent due to certification issues as stated in the release notes of the Android beta 0.9 SDK:

> Due to significant API changes in the upstream open-source project and due to the timeline of getting certain Bluetooth profile implementations certified, a comprehensive Bluetooth API will not be possible or present in Android 1.0.

Since the release of Android 2.0 in October 2009 developers finally have access to the Bluetooth interface. It is possible to turn on the Bluetooth interface without user interaction and perform device discovery. Before communication is possible between devices they have to be paired first, requiring user interaction [26]. This would make Bluetooth an inconvenient choice for ubiquitous computing applications since user interaction is not desirable when it can be avoided.

Networking functionality can be used using standard Java packages such as *java.net* or *org.apache.http* and secure communications can be setup using packages such as *javax.security* or *javax.net.ssl*. With this the basic building blocks to connect to almost anything are available.

Power Consumption

Power considerations are build directly into the Android platform [39]. Applications can monitor the battery status and control energy saving features. It is possible to force the device to go to sleep or maintain a specific power level. Using the *PowerManager* class an application can control the power state of the device, but it is advised to use the API with care. Without outside influences the operating system will try to run in the lowest power mode possible, something an application can prevent from happening with the use of this class.

Security and Privacy

The Dalvik virtual machine is a key component of the Android operating system and it plays a major role in the security of the operating system. A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user [24].

This policy is enforced using the Dalvik virtual machine and the underlying Linux platform using Unix User Identifiers and file permissions. Unlike most Linux desktop operating systems, where applications from the same user run with the same UID, every application runs in its own virtual machine in a separate process with its own UID. This means that Android applications cannot access the code or data from other programs, and that all data sharing has to be done explicitly.

Besides running every application in a sandbox the security of the operating system is further enforced by limiting the permissions of every installed application. By default Android applications have no permissions to do anything that could impact the user experience or the data on the mobile phone. If an application wants to access the protected features such as the phone book or access the GPS functionality the developer needs to declare the required permissions. When the application is installed the user has to grant the application access to those features before the application can use them.

Because of the rigged security model the harm that can be caused by a 'bad' application is limited. The first serious security flaw that was found in Android in October 2008 and made it possible to run any code with the privileges of the standard web browser application [17]. This is a big problem, but the impact of the issue was limited by the security model. A hacker could for example not access private data from other applications or dial the phone directly.

Even in a worse case scenario where the user grants a malicious application rights to do almost anything, private data from other applications is still not accessible because of the Unix file systems with different UIDs for every application. This could be a major advantage for ubiquitous computing because of the often privacy sensitive nature of these applications.

3.2.2 BlackBerry OS

BlackBerry OS is a proprietary operating system that has been developed by Research In Motion to be used exclusively in combination with the BlackBerry smartphone family. The first BlackBerry smartphones were created with business professionals in mind, and offered functionality such as wireless synchronization with Microsoft Exchange. BlackBerry smartphones are still mainly used in the enterprise market, but in the past years various models that targeted the consumer market have also been released. In the first quarter of 2009 the RIM Curve overtook Apples iPhone as the best selling consumer smartphone in the USA [27].

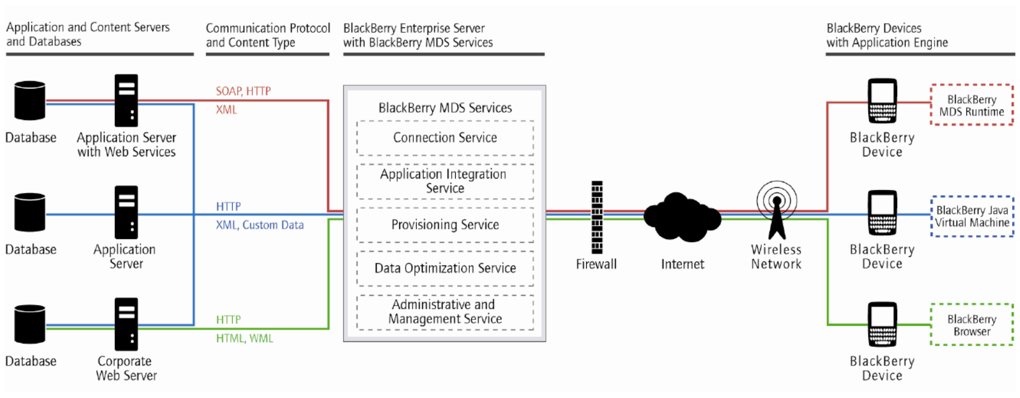
Application and Content Servers and Databases
Communication Protocol and Content Type
BlackBerry Enterprise Server with BlackBerry MDS Services
BlackBerry Devices with Application Engine
Database
Application Server with Web Services
Database
Application Server
Database
Corporate Web Server
SOAP, HTTP
XML
HTTP
XML, Custom Data
HTTP
HTML, WML
BlackBerry MDS Services
Connection Service
Application Integration Service
Provisioning Service
Data Optimization Service
Administrative and Management Service
Firewall
Internet
Wireless Network
BlackBerry Device
BlackBerry MDS Runtime
BlackBerry Device
BlackBerry Java Virtual Machine
BlackBerry Device
BlackBerry Browser

Figure 3.2: The Blackberry MDS Runtime enviroment [43]

BlackBerry OS is proprietary and not many details about the inner workings of the operating system have been disclosed by RIM. There is of course documentation about the SDK that is available for developing applications for the OS, but it does not give technical information about the underlying OS. Creating applications for a BlackBerry device can be done in two ways. First of all is possible to create Java applications that run on the device using a proprietary JVM. BlackBerry supports the Java ME MIDP standard as defined in the JSR 118 specification[‡]. Also available are various Java API extensions for tasks such as persistent storage, networking and application integration. The second option is to create a Mobile Data Service (MDS). This class of applications is optimized to receive push data from some kind of application service, and display this data to the user. A schematic overview of the MDS runtime is provided in Figure 3.2. As is visible there are different options to use MDS applications: They can run in the web browser, the JVM or the MDS runtime. Important to note is that the presence of the BlackBerry Enterprise Server is required for these applications. It is not possible to develop native applications in for example C or C++ as a third party developer.

Concurrency Model

BlackBerry OS is multithreaded and many applications and processes can run at the same time. Applications can create background threads to handle computationally

[‡]Available from http://jcp.org/en/jsr/detail?id=118

intensive tasks or network communication without locking the main thread. When an application is closed background threads are allowed to continue to be active. The fact that this functionality is available is not a surprise, the OS has been designed with running background network processes in mind. Standard applications for instance use this for synchronizing emails and calendar events, but third party applications can also use these APIs [44].

Cost

BlackBerry OS is a proprietary operating system that is exclusively used in combination with the BlackBerry smartphone family. There is no data available about the costs of the operating system, because it cannot be licensed by other parties or bought without a BlackBerry in the store.

Memory Use

The Java Virtual Machine handles the memory management on the BlackBerry for most third party applications, unless the specialized MDS runtime is used. The JVM handles allocating memory, swapping data between SRAM and flash memory and garbage collection. To deal with the limited memory capacity of smartphones a special low memory manager is running. When the number of available object handles or the amount of free flash memory drops below a certain threshold the low memory manager tries to free existing memory. Standard applications as well as third party applications should work with this interface, and try to delete low and medium priority data when the low memory listener receives an event [42].

Networking

BlackBerry OS is created with background network processes in mind. Creating a standard HTTP(S) or TCP socket to establish a connection over a wireless network is no problem. The design behind the BlackBerry OS is however focused on offering a connection to the company intranet or the Internet at all times, not on connecting with devices in the local vicinity of the user as is the case in a lot of pervasive computing applications. The API does not include functionality to automatically discover and connect to WiFi networks: A WiFi connection can only be used if it has been setup by the user in the past. Functionality to discover and connect to Bluetooth devices is available though.

Power Consumption

Details about the internals of the BlackBerry OS are scarce, and no details have been published about the power management features of the operating system. Developers get no special tools to control the power management of the device. It is possible to retrieve information about the status of the battery, but that is all that is possible.

Security and Privacy

Security has been historically a focus point of the BlackBerry OS, but the available security features are tailored towards the use in an enterprise environment. When the BlackBerry Enterprise Server is used all communication between the smartphone and the outside world is for example encrypted, and network administrators can install and remove applications remotely as well as change policies that control what applications and what the user can do. A possible policy is to encrypt all user data on the device.

For BlackBerry users in a non corporate environment there are still various security measures in place. BlackBerry applications can only access memory that is used by the JVM, access to the virtual memory or the persistent data from other applications is not possible unless access has been specifically granted. Application with these capabilities could potentially be harmful and have to be digitally signed by RIM. The same is true for various other APIs that have been labeled as sensitive so that an audit trail is available when abuse of these APIs is detected [44].

3.2.3 iPhone OS

Apple introduced the first generation iPhone June 2007 in the USA. The phone uses the so called iPhone OS that is similar to Mac OS X. The basic structure of the operating system is displayed in Figure 3.3.

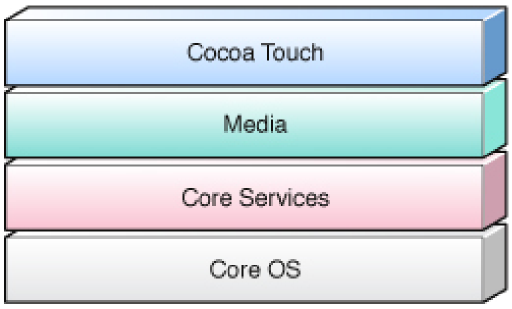

Figure 3.3: Overview of the iPhone OS architecture [3]

The lowest level of the operating system is formed by the Core OS layer. Additional abstraction layers are provided by the Core Services layer, the Media layer and the Cocoa Touch layer [3]. The Core OS layer contains the Mach kernel, hardware drivers and is in charge of managing the memory system, the scheduler, the file system, the network and the interprocess communication. It also contains the Security framework to protect and secure program data.

An abstraction step up is the Core Services layer. It includes the basic framework for Objective-C programming, access to the network availability and state of the device, access to the location information and the address book. the Media layer contains multiple frameworks to work with audio, video, 2D graphics and 3D graphics. The Cocoa Touch Layer is the highest level of abstraction available and offers the basic building blocks to create graphical event-driven applications for the iPhone OS. Objective-C is required to access the higher level API's, but since Objective-C is a *strict* superset of C it is possible to freely include C code in any Objective-C class.

As already quickly mentioned in the introduction of Chapter 3 a lot of the functionality that is available on the iPhone operating system is restricted by Apple using the SDK agreement and by making not all API's publicly available [8]. In particular Section 3.3.1 and 3.3.2 limit possibilities for developers by limiting the use of the iPhone API's to only those published by Apple and restricting the types of executable code that may be run on the device. A plug-in architecture is for example not allowed and so are specific types of applications such as those offering Voice over Internet Protocol (VoIP) functionality using the cellular network.

Concurrency Model

In order to maximize the amount of memory available to the 'foreground application' Apple has restricted the multitasking capabilities of the iPhone. Apple allows only one third party application to run at the same time [7]: when the user goes back to the home screen the application is terminated, state information is saved and when the user goes back the application is relaunched. This is a serious obstacle for pervasive computing applications since those often have to do work in the background without user intervention.

The limitations imposed by Apple are artificial; around ten applications/services are often running concurrently on the iPhone, but only Apples own applications and services are allowed to do this. Only one third party application can run at the same time, and it has to be in the foreground. Services and applications that do run concurrently are for example core services such as calling, email checking, music playback, Bluetooth and MobileMe synchronization. Apples own applications such as Safari are also allowed to continue running when enough free memory is available.

Cost

The iPhone OS is developed in house by Apple and cannot be licensed by third parties. Speculating about the costs is therefore a futile exercise, although it's worth noting that large parts of the operating system are build from pre-existing software. Big parts of the OS - including the kernel - are borrowed from the Mac OS X operating system. Other functionality is provided by open source products such as SQLite and WebKit, both also part of the Android operating system.

Memory Use

The first generation iPhone and the iPhone 3G both offer 128MB of memory while the iPhone 3GS offers 256MB. Approximately 11MB of the 128MB is used as VRAM and a decent chuck is used by the operating system itself, leaving approximately 76MB as user memory [32]. Unlike it's desktop counterpart the iPhone OS does not have a swap file for virtual memory, so when the RAM is full there is really no memory available anymore (and the application will be forced to close by the operating system). Because of this the iPhone 3GS has quite a speed advantage when running the standard Apple applications that support multitasking. On the older iPhone models the mobile Safari browser is almost always closed when another application is launched, but on the 3GS it can continue running in the background so switching back to this application can be almost instantaneous.

Networking

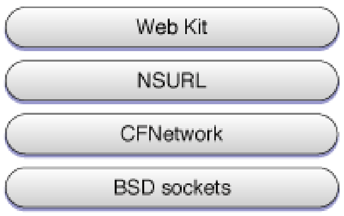

Figure 3.4: CFNetwork and other software layers on Mac OS X [6]

The core of the networking functionality inside the iPhone OS is based on BSD sockets. Several higher layers of abstraction are also available as can be seen in Figure 3.4. The CFNetwork layer allows developers to create sockets and streams, and by allowing these low level network building blocks applications can connect to anything inside the network. Although applications can be created to connect to almost any service that is running in a network, applications have (almost) no control over what network to connect to. Applications are not allowed to connect to a WiFi network without user interaction. This means that applications are usually limited to 'just' an Internet connection using the cellular network.

Apple announced on March 17, 2009 the major features of the latest generation iPhone OS, version 3.0, that has been publicly released in the summer of 2009. This version of the operating system makes it possible for applications to create peer to peer networks using Bluetooth. The SDK includes functionality to automatically discover other devices running iPhone OS and will be able to connect seamlessly to them without pairing [5]. The technology behind this is Apples Bonjour service discovery protocol that allows setting up networks without any configuration (also known as Zeroconfig). It is not possible to connect to non iPhone OS devices that do not support this standard.

Power Consumption

Apple has created the iPhone OS with power consumption in mind, and because of this several big restrictions are in place in what kind of applications can be developed as already mentioned in Section 3.2.3. Only Apples own 'core services' can run in the background, while third party applications are terminated as soon the user returns to the home screen. The main reason for not allowing background processes is to protect the battery performance of the iPhone.

According to Apple running 'a popular instant messaging client' in the background of an Android phone, a Windows Mobile phone or BlackBerry reduces standby battery life with 80 percent or more [5]. The alternative that Apple is going to offer for background processes in iPhone OS 3.0 are push notifications. These can be sent over the Internet to the mobile phone to notify the user of some new information. This makes it for example possible to create instant messaging applications that can receive messages when the instant messaging client is not running, but it seems an unpractical model for most pervasive computing applications. By nature these applications are often context aware and require two way communication between the phone and devices in the local environment: something that is not possible with push notifications.

Security and Privacy

For developers the iPhone OS offers several API's to implement security features. Just like it desktop counterpart, iPhone OS uses BSD (Berkeley Software Distribution) and CDSA (Common Data Security Architecture) to implement security services. Low level features, for example file access permissions, are implemented by the BSD kernel, a form of the UNIX operating system. CDSA provides higher level functionality such as encryption, authentication and secure data storage. CDSA is an open source standard with its own API, but this is not directly accessible because it does not follow standard Macintosh programming conventions. Developers have to use several Apple API's that call the CDSA API. See for an overview of the security architecture Figure 3.5.

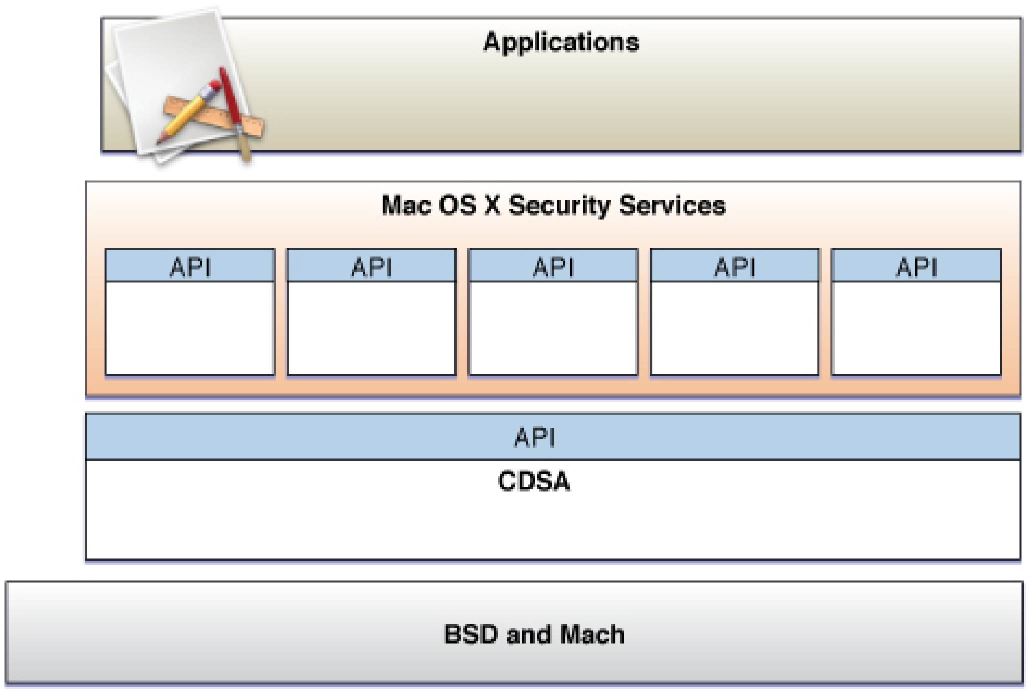

Figure 3.5: Mac OS X security architecture overview [4]

The iPhone OS runs applications such as the dialer and browser with root access. This means that when a security vulnerability is exploited in one application the whole operating system could be compromised. This is unlike Android and BlackBerry OS where applications are shielded from the operating system, and other applications, by running it inside a virtual machine. Since the introduction of the iPhone OS several security holes have been found - and fixed by Apple with firmware updates - that could allow a hacker to gain full control over the mobile phone. In the beginning of 2008 security expert Rik Farrow showed for example how a Safari exploit could be used to gain root access to the iPhone and install a sound recording application that could be used to spy on the user [13].

Although running applications with root rights as a default is a big risk, privacy sensitive data collected by an ubiquitous computing application can be protected to some extend using the APIs that are build on top of CDSA. These APIs make it possible to store, send and receive data encrypted.

3.2.4 Symbian

Symbian is the most popular smartphone operating system with a market share of 51 percent in the second quarter of 2009 [21]. Although Symbian is by far the biggest player on the smartphone OS market, its market share declined rapidly the last two years - with more than 20 percent - because of developments such as the introduction of the iPhone and Android. There are several variants of the Symbian OS, since the operating system itself does not include a user interface. Symbian is available as S60[§], UIQ[¶] and MOAP(S)[‖]. Nokia, founded the Symbian foundation in June 2008 with the intent to unite Symbian OS, S60, UIQ and MOAP(S) to create one open mobile software platform [37, 18], but for now the platform remains divided.

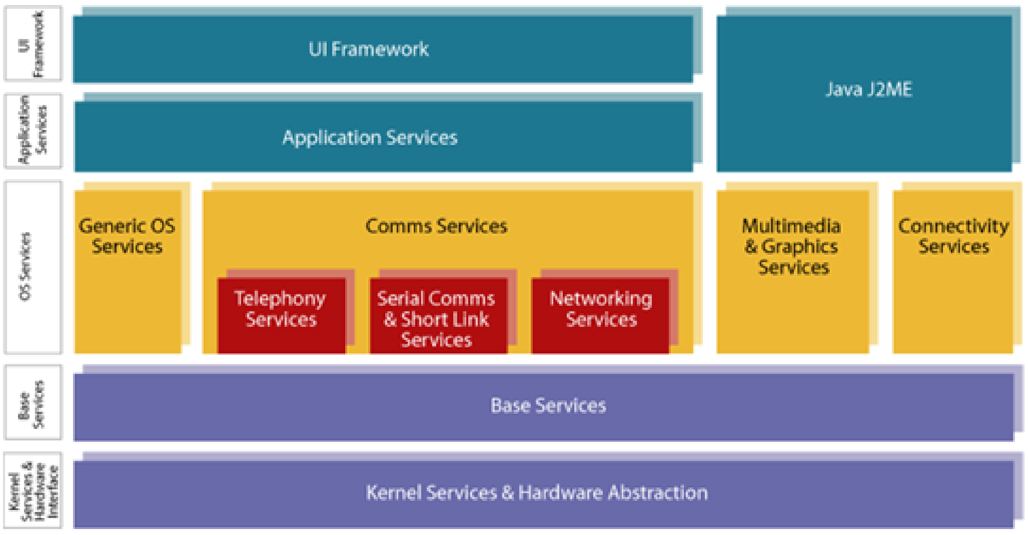

Figure 3.6: Overview of the Symbian architecture [38]

Symbian OS has it roots in Psion's EPOC that was originally developed for PDA's. Symbian has been specifically engineered for smartphones, unlike most other operating systems discussed in this article that have been derived from larger systems. The operating system can be decomposed into several layers as can be seen in Figure 3.6.

The top layer is the user interface framework that is used by the GUI implementation of the operating system itself (S60, UIQ, MOAP(S)) and also by applications. The second layer is the application services layer. This layer provides services that are required by all applications and generic services such as data synchronization and HTTP transport. Below this layer is the OS services layer that extends the base services layer below. The OS Services layer contains features such as the task scheduler,

[§]Series 60 User Interface
[¶]User Interface Quartz
[‖]Mobile Oriented Applications Platform (Symbian)

telephony services, the TCP/IP stack and multimedia services. The base services layer contains the lowest level of user mode services. The file server resides at this level as well as the user library that applications need to use to interface with the kernel. The lowest level is formed by the kernel and hardware drivers.

Concurrency Model

Just like the other smartphone operating systems Symbian is multithreaded. This functionality is also exposed to third party application developers, making it possible to run processes in the background. Not all types of applications have access to these APIs, but for C++ developers running background processes is not a problem. The Symbian OS kernel supports hard real-time tasks, but no real-time scheduling guarantees are given for user processes.

Cost

With the creation of the Symbian foundation in 2008 Symbian is on its way to become open source under the Eclipse Public License (EPL). The foundation is committed to moving the Symbian platform to open source in the next two years meaning that the platform is going to be free for anyone [18]. This is however not yet the case and at the moment phone manufacturers have to pay a license fee for using the operating system, although costs have been dropping the pasts years. In the second quarter of 2008 the average royalty cost per phone was 3,4 dollar, a drop of 21 percent in comparison with the previous year [50].

Memory use

Symbian was designed for devices with limited resources. Because out-of-memory errors can occur anytime when memory is allocated developers need to cope with these errors and make the program go back to an acceptable and stable state when these occur. To make this possible Symbian is equipped with leaves and a cleanup stack. Symbian does not support exceptions like standard C++, but supplies its own idioms. When a leave (read: exception) occurs objects that are allocated on the heap would not be cleanup automatically. To deal with these possible sources of memory leaks pointers to all objects that are created should be stored on the cleanup stack. This allows the system to automatically clean up unused objects.

Networking

Symbian provides a limited set of functionality for connecting to wireless networks. It can scan for wifi networks and retrieve information about the cellular network that it is connected to. It is however not possible to programmatically control the interfaces [39]. Bluetooth functionality is also limited: Bluetooth I/O is only possible with devices that are securely paired, meaning that user input is required before communication can take place. This is however not a Symbian specific problem. When a connection is set up the developer has full control over the data that can be transmitted. It is possible to specify the interface that should be used, and Symbian OS sockets are similar to BSD sockets.

Until recently the sockets API was not exposed to third party developers, only higher communication layers such as HTTP were available. All new devices based on the Symbian platform now support Open C and Open C++, and a part of this standard is a low level socket API. Older devices using Symbian S60 3rd Edition or later can be equipped with these libraries by installing a separate plug-in.

Power Consumption

Symbian OS arguably offers the developers to most control and insight on the power consumption of the telephone. Nokia offers the Nokia Energy Profiler[**] that can be used to monitor the power consumption of a Nokia Symbian phone. Symbian also contains a framework that can be used by third party applications to adjust the power mode, receive notifications when the power mode changes and wakeup/shutdown the kernel.

Security and Privacy

Just like the other smartphone operating systems Symbian has several security features in place that should protect integrity and privacy. The main threat that is addressed by the Symbian security architecture is the distribution of malicious applications. This is done by requiring programs to be signed before they can be installed, and once installed access to resources is restricted. Because applications are reviewed before they are signed the risk of installing malware is minimized, but it is not fully effective since there are ways for users to get unsigned applications running. In this aspect Symbian OS is remarkable similar to iPhone OS where applications that are distributed using the App Store are first reviewed by Apple and after a jailbreak this restriction can be circumvented.

[**] Available from http://www.forum.nokia.com/Resources_and_Information/Tools/

The second part of the Symbian security architecture is restricting the API's applications can use. Rights to access certain capabilities are granted when the application is installed. Some capabilities can be approved by the user, while applications that require access to the most sensitive capabilities require approval by the phone manufacturers. An overview of the different capability levels and the API's within these levels can be found in Figure 3.7.

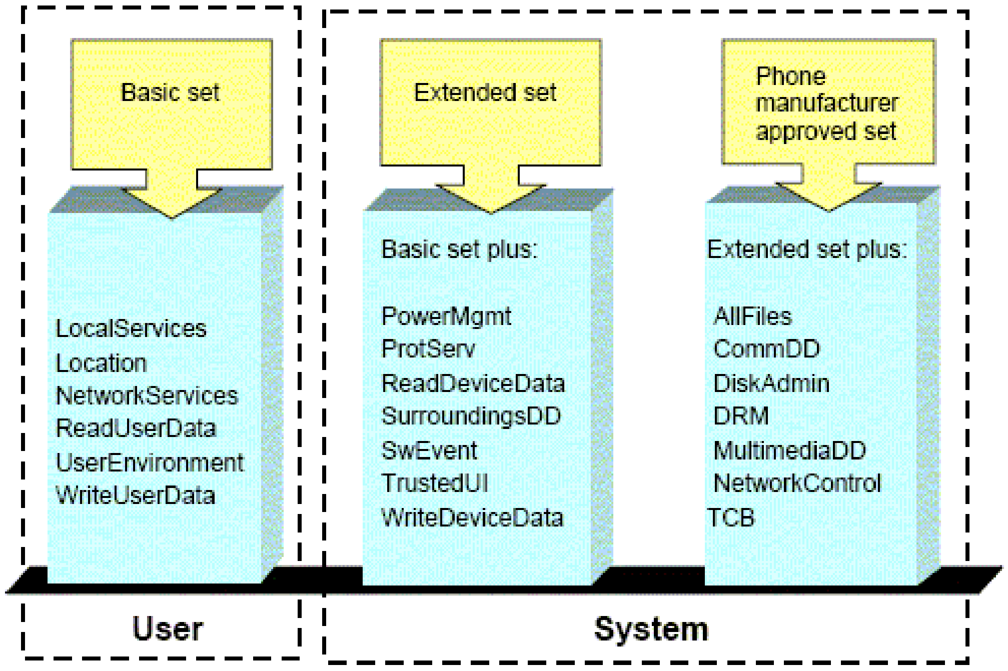

Figure 3.7: Overview of the Symbian capabilities [49]

System components and third party applications only get access to the capabilities that they require. Because of this a vulnerability or a misuse of privileges in a single component has a limited impact. Access to API's is restricted for most applications and so is access to the file system. Every application has a private directory as well as a public directory that is read-only for other applications. The other parts of the file system, except the /sys directory are fully accessible.

Besides the previous discussed operating system features Symbian provides developers with various API's to implement security features. Cryptographic, hashing and random number generating algorithms are all available as a default. Setting up secure network connections using SSL is also not a problem.

3.2.5 Windows Mobile

Windows Mobile has been developed by Microsoft to run on a variety of mobile devices. The operating system is based on the Win32 API, and has been designed to offer a similar look and similar functionality as its desktop counterparts. An schematic overview of the kernel is provided in Figure 3.8. The kernel used by the operating system is based upon Windows CE, an operating system designed for handhelds and embedded systems. Windows CE is a modular operating system where the developer can choose what functionality he wants. With just the kernel the operating system is a few hundred kilobyte in size, but components like a web server or support for the .NET Compact Framework can be added. The big difference between Windows CE and Windows Mobile is that the set of components that are used are fixed by Microsoft so that APIs are consistent between all Windows Mobile devices. APIs are however not identical. All phones have to support a minimal set of functionalities, but phone manufactures are free to include additional APIs in the OS image.

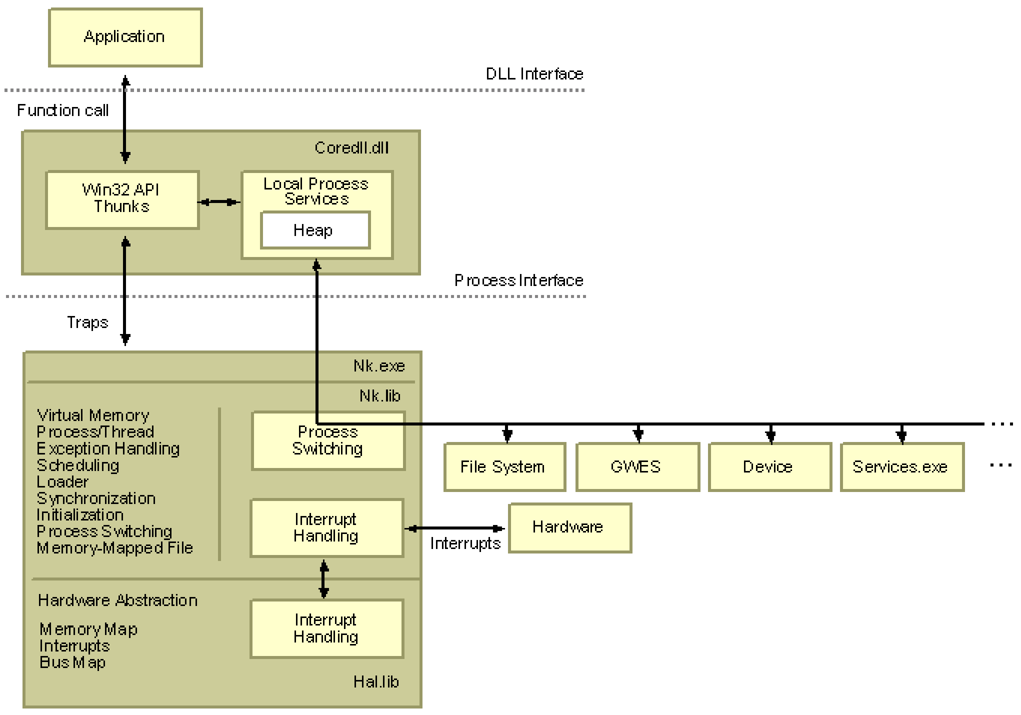

Figure 3.8: General structure of the Windows Mobile kernel [34]

Concurrency Model

Windows CE has been developed for embedded devices and this gives Windows Mobile an interesting characteristic that other smartphone operating systems are missing:

the kernel offers real-time performance. This means that there is a guaranteed upper bound on the execution time of high priority threads. To use this functionality the Win32 API has to be used since the .NET Compact Framework includes a garbage collector that freezes all threads during garbage collection. Since the win32 API is based on technologies from more than 25 years ago working with it should be avoided if possible. Microsoft itself describes it as an arcane, cranky API prone to memory leaks [35].

The multi-threading functionality offered by Windows Mobile in the .NET Compact Framework is fully featured, and offers all functionality you could wish for as a developer. Running multiple threads and multiple processes in the background are not a problem.

Cost

Microsoft licenses Windows Mobile to a wide variety of phone manufacturers. Companies such as Acer, HTC, LG Electronics, Samsung, Sony Ericsson and Toshiba all develop Windows Mobile phones. How high the licensing fees are for the various Windows Mobile versions is not made public by Microsoft. Market research firm Strategy Analytics estimates that the fee ranges from 8 dollar to 14 dollar [30].

Memory Use

Windows Mobile 6 is based on Windows CE 5.0, and is a 32-bit OS with a maximum of 32MB of virtual memory available for a single application. These days it is not uncommon to see smartphones with 128MB or more internal memory, but this cannot be used by a single application. A lot of memory is by default used by the operating system to keep applications running after the user closed them, a feature dubbed smart minimize by Microsoft. These applications can be closed by the OS though when memory is running low.

Noteworthy is that Windows Mobile offers programmers extremely low level access to the operating system. With Windows Mobile developers can access the Win32 API directly. This increases development options, but also risks the creating of applications with memory leaks. Most operating system objects that are created with this API have to be cleaned up manually by the developer. The managed wrappers available inside the .NET Compact framework are built to make sure that the underlying Win32 objects are cleaned up correctly. Symbian and iPhone OS also support C/C++, but the API's for accessing the operating systems features are cleaner defined and more shielded from third party developers. Symbian has the cleanup stack to prevent memory leaks while Objective-C objects can be put in an autorelease pool that is cleaned up after an event loop is completed. Smartphone operating systems that limit developers to high level programming languages running inside a virtual machine have almost no memory leak problems since the garbage collector takes care of this.

Networking

Windows Mobile offers various APIs for creating network connections. Since Windows Mobile allows developers to access very low level APIs basically everything imaginable is possible. Creating standard sockets is not a problem, and full control over the Bluetooth and WiFi connection is available. In the managed .NET Compact Framework this functionality is not supported by default, but it can be added easily by using freely available wrappers[††].

Power Consumption

Windows Mobile includes a Power Manager API that give developers influence on the power state of the device. An application can request a certain power state, but there is no guarantee that the status will be changed. The driver is ultimately responsible for the power state, and has to take other running programs in account too. Besides setting a general power state for the device separate hardware components can also be controlled. Changing the brightness of the LCD or switching off the Bluetooth radio are possibilities. An API for monitoring the battery status is also included in Windows Mobile.

Security and Privacy

As made clear in the previous sections Windows Mobile resembles its desktop counterpart in many ways. Developers have access to many low level APIs giving application developers almost full freedom. The flipside of the coin is that the build in security measures of the operating system are limited. Applications are not sandboxed and protected from each other. Applications cannot directly access memory in the process space of another process, but with techniques such as DLL injection it is possible to run code within the address space of another process. The process space security is designed to protect applications from programming bugs in other applications, but it is not sufficient to shield applications from malicious applications. Once a trusted process is launched there are essentially no limits on what it is allowed to do. It is

[††]Bluetooth .NET library available from http://www.codeplex.com/32feet/

however possible that applications that access trusted APIs need to be signed before access is granted, but this depends on the implemented security policy.

3.3 Summary

The various operating systems all have strong and weak points: it is certainly not possible to say that one of them is the obvious best for running pervasive computing applications. It is however possible to pinpoint the least suitable operating system in the list. In Table 3.1 a simplistic overview of the strong and weak points of the different operating systems is provided. With the exception of iPhone OS all operating systems score two or three stars in the various categories.

The artificial restrictions imposed by Apple with regards to running background processes make the current version of iPhone OS not suitable for most pervasive computing applications. The other four operating systems have no problems running multiple applications at the same time or running background processes so they all score three stars. It is very possible that Apple will relax these restrictions in a future version of the OS, making it more suitable for a wider range of applications [22].

	Concurrency	Cost	Memory	Network	Power	Security
Android	⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆
BlackBerry	⋆⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆
iPhone OS	⋆	⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆
Symbian	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆
Windows	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆

Table 3.1: Summary of quality attributes

Android is at the moment the only free open source operating system in this list, so it gets three stars while the competition only gets two. All operating systems have various technologies in place to deal with the limited memory capacity of smartphones, but both Android and BlackBerry OS get two stars instead of three because development of native applications is not possible and all third party applications are executed in a virtual machine.

Networking is very important for most pervasive computing applications, but almost all operating systems have limitations with regards to automatically connecting to WiFi or Bluetooth networks. Windows Mobile does not have these kind of limitations and gets a three star rating. Android also has a very strong feature set in place since Android 2.0 and gets a three star rating as well, even though the usefulness of the Bluetooth feature is limited. This is however true for most Bluetooth devices since they need to be paired before a communication channel can be setup.

Most operating systems have APIs in place to give developers insight in the battery status of the device and the power mode. BlackBerry OS is the exception to the rule with an API that only gives information about the battery status, and no control over the power mode.

Security is an aspect that gets an reasonable amount of attention in mobile operating systems. Android and BlackBerry OS both only run applications in a sandboxed virtual machine environment and therefore get the maximum score. This does not mean that the other operating systems are particularly weak. Symbian has an elaborate system in place for signing applications and Windows Mobile also supports this, although the system is less sophisticated.

Chapter 4

Quantitative Analysis

Besides the qualitative analysis in Chapter 3 the performance of the various operating systems also has been examined using a benchmark that performs various tasks that are common in pervasive computing applications. The variety of possible tasks that a pervasive computing application can perform is practically endless, but in almost all instances communication between various devices is an integral part of the application. The benchmark consists of three tasks that are all crucial with regards to communication, namely:

- Encryption
- Decryption
- XML parsing

These tasks are not only relevant for communication between various services and devices, but are also often used locally. When dealing with sensitive information it is desirable to store the user data encrypted and XML files are frequently used to store data like configuration settings and user interface details.

A challenge in developing a benchmark for various operating systems is that creating a cross platform application with a shared code base is not an option. All operating systems support different programming languages, and even when the language is the same the surrounding libraries are completely different. Blackberry OS and Android for example both run Java applications, but out of the 135 Android Java packages and 77 Blackberry OS Java packages just 7 packages have the same namespace.

Every operating system does provide libraries to facilitate the three tasks that have been selected for the benchmark. This makes it possible to create with relative ease a benchmark that is both relevant and fair. It is relevant because it is using the libraries that are part of the operating system and almost every other developer would use when including this kind of functionality. Especially on the platforms that do not support native applications from third party developers it is probably a futile exercise to try to create a higher performing alternative.

At the same time there is no real risk that the benchmark is unfairly optimized for a certain platform. Implementing a high performance algorithm would require expert knowledge on subjects such as memory management, the language used and the runtime environment. By relying on the included libraries for the heavy lifting in the benchmark a possible skill difference of the benchmark developer on various platforms is negated.

The benchmark does not directly test the performance of the kernel of the underlying operating system, but it is in charge of tasks such as memory management, thread scheduling and file access during execution.

4.1 Experimental Setup

The benchmark has been implemented for each of the five smartphone operating systems that are discussed in this thesis. An overview of the target platforms and the programming languages used is provided in Table 4.1.

	Development platform	Programming language
Android	Android SDK 2.0.1, Release 1	Java
BlackBerry	BlackBerry SDK v5.0 Beta 5	Java
iPhone OS	iPhone SDK 3.1.2	Objective-C
Symbian	S60 5th Edition SDK	C++
Windows	Windows Mobile 6.5 SDK	C#

Table 4.1: Overview of the various programming languages used

The first three platforms offer the developer no real choice with regards to the programming language that can be used to develop applications. Symbian offers more options, but running Java applications in the background is not easily possible and it would require that the Java virtual machine is always loaded in the memory. Windows Mobile also offers various options. Using a language supported by the .NET Compact Framework is a logical choice since it is available on every Windows Mobile phone and offers developers a high-level programming environment and access to a large number of useful API's. Other programming languages are available for Symbian and Windows Mobile, but in most cases it is required that the user installs an additional runtime environment before these applications can be executed.

The implementation of the benchmark is straightforward. The structure of the benchmark is displayed in Figure 4.1. Every part of the benchmark is run ten times to reduce variance in the results and acquire more accurate results. Most processor time is spend in the subroutines of the system libraries that handle encryption, decryption and XML parsing. The benchmark code handles tasks such as setting timers to measure results, open data streams and call system libraries. Both the encryption and the XML parsing subroutines work on a XML file with a size of 144KB.

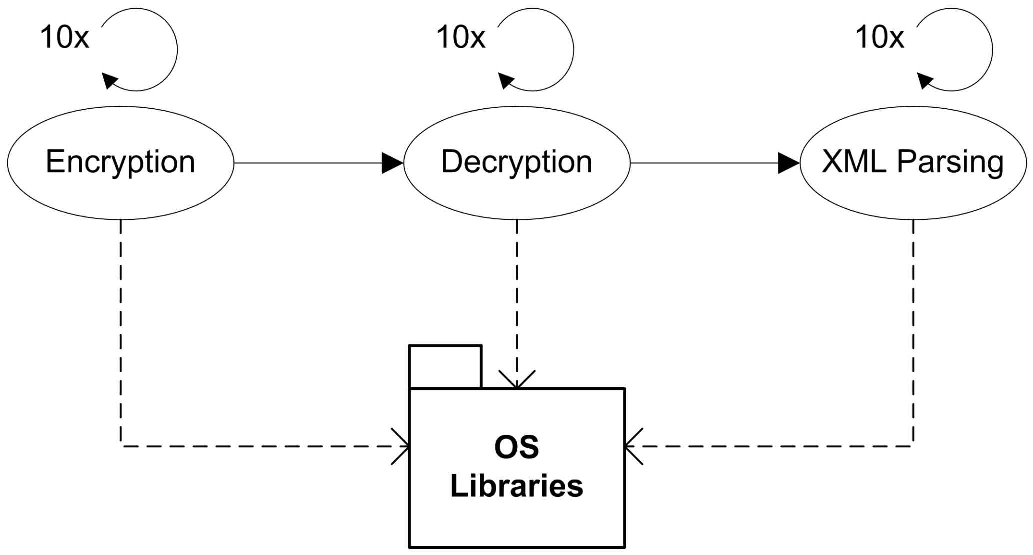

Figure 4.1: The structure of the benchmark

No significant differences in implementation exist between the various platforms. The Symbian emulator offers no timers with a resolution above five milliseconds, and as will become clear in the next section; that is not enough to generate meaningful results. Because of this each Symbian benchmark is run one thousand times before calculating the average execution time. The second noteworthy difference is present in the iPhone version of the benchmark. The iPhone SDK does not provide an Objective-C library that enables parsing an XML file to a DOM tree. It is possible to use a C library, but to use this effectively within an Objective-C application a small wrapper should be written. The performance impact of this is presumably insignificant since Objective-C is a superset of C.

For four out of the five smartphone operating systems an emulator that runs under Windows is available, the exception is the iPhone emulator that is only available for Mac OS X. Because of this the iPhone emulator benchmarks have been run on a different system and are not directly comparable with the other benchmark results.

The main specifications of the benchmark systems can be found in Table 4.2. After examining the performance on the emulators the benchmark will also be executed on various mobile devices, and the question if the emulators are a suitable tool for performance evaluations will be answered.

	Windows 7	Mac OS X Snow Leopard
CPU	Intel Core 2 Extreme X9770 3,2GHz	Intel Core 2 Duo 2,13GHz
RAM	4GB DDR2-800	2GB DDR3-1066
HD	Intel X25-M Postville 160GB SSD	128GB SSD
Video	ATI Radeon HD 4870 X2	NVIDIA GeForce 9400M

Table 4.2: The specifications of the benchmark systems

4.2 Emulator Results

As already stated the iPhone OS results are not directly comparable with the other results because those benchmarks have been run on a different host system. But although the host system is significantly slower, the iPhone emulator is very fast compared to most other emulators as is visible in this section. The raw benchmark results are available in Appendix A.

4.2.1 Encryption

In the encryption benchmark a 144KB file is encrypted using 256-bit AES encryption. AES is a widely used encryption standard that is used in secure communication standards such as IPSec and SSL and a wide variety of applications that require secure data storage.

Figure 4.2 shows that the time that is required to encrypt the file varies widely between the operating systems. Android is by far the slowest with a time of more than 3 seconds, while Windows Mobile is roughly 5 times faster and BlackBerry OS 10 times faster. The difference between the Android results and the Symbian and iPhone OS results are even more staggering. They set a time of respectively 5.8 and 5.4 milliseconds, more than 500 times faster. Converted to KB/s the slowest performer achieves a speed of 47KB/s, while the fastest performer manages to encrypt the file with a speed of 26820KB/s (26MB/s).

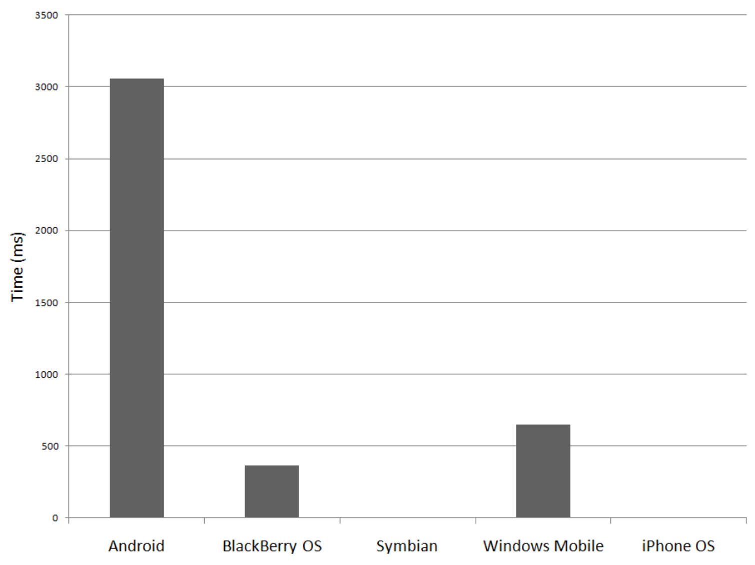

Figure 4.2: Encryption benchmark results

4.2.2 Decryption

The decryption benchmark is the mirror image of the encryption benchmark, and uses the output from this benchmark as input to recreate the 144KB plain text file.

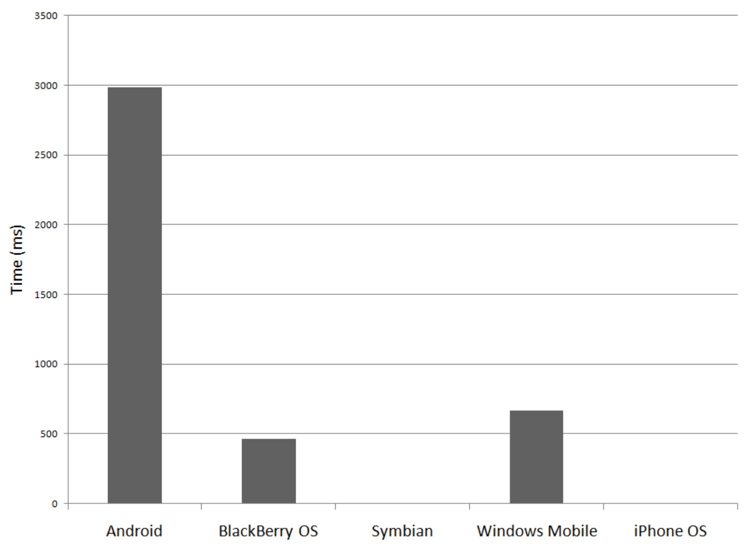

Figure 4.3: Decryption benchmark results

As expected the results of this benchmark are similar to those obtained by the encryption benchmark. The Android emulator is still very slow compared to the BlackBerry OS and Windows Mobile and the Symbian and iPhone emulators blow the competition out the water. The biggest difference between the two benchmarks is that BlackBerry OS needs 27 percent more time for the decryption than for the encryption while the differences for the other operating systems are less than 3 percent.

4.2.3 XML Parsing

As the name implies the XML parsing benchmark measures the time needed to parse a single XML file to a DOM tree. The biggest component that is measured is the parsing time to create the memory structure. The same 144KB big XML file from the previous benchmarks is used.

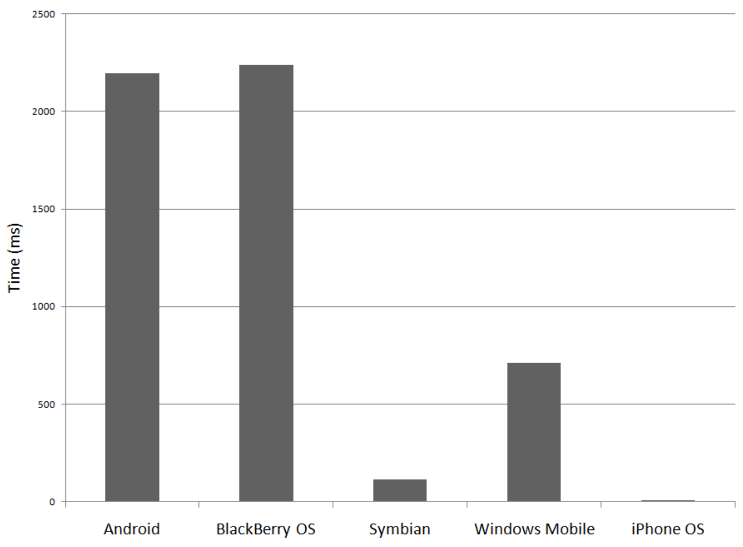

Figure 4.4: XML parsing benchmark results

The XML benchmark paints a picture resembling the previous benchmarks, but as can be seen in Figure 4.4 there are large differences at the same time. BlackBerry OS is this time the slowest operating system, closely followed by Android. Windows Mobile takes the third place while Symbian and iPhone OS are again significantly faster than the competition. While iPhone OS remains several hundred times faster than the slowest operating system, the gap between Symbian and the other operating systems is smaller. Symbian is less than 20 times faster than the slowest performer while iPhone OS is still almost 300 times faster.

4.3 Host System Dependency

The wild differences between the benchmarks results on the emulators are unexpected. The fact that benchmarks that run using an interpreted language (Java and C#) are slower than those directly executed by the CPU (C++ and Objective-C) is as expected. It should however be noted that it is not impossible for Java application to perform better than C++ applications because dynamic compilation allows for optimizations that a static compiler cannot perform [41].

Performance differences of several orders of magnitude hint that the emulators themselves are not a reliable tool to assess the performance of real world devices. In the accompanying documentation of the various emulators remarkable little is said about the performance, which is a vague sign in itself.

To test if the different emulators try to emulate phone performance the benchmarks have been executed again, but this time with the clock speed of the processor increased from 3.2GHz to 3.6GHz. If the benchmark results increase linearly with this clock speed increase that would either mean that the emulator has no upper limit on the performance, or that a modern desktop PC has insufficient performance to emulate the smartphone. In both cases the emulator is not a suitable tool for estimating the performance of the smartphone.

	Benchmark	3.2GHz Results	3.6GHz Results	Difference
Android	Encryption	3057.4	2672.4	14%
	Decryption	2987.3	2688.0	11%
	XML Parsing	2197.6	1795.8	22%
BlackBerry	Encryption	362.7	320.0	13%
	Decryption	462.8	411.7	12%
	XML Parsing	2237.7	1962.0	14%
Symbian	Encryption	5.798	5.528	5%
	Decryption	5.912	5.700	4%
	XML Parsing	113.7	107.4	6%
Windows	Encryption	650.5	564.7	15%
	Decryption	664.9	583.3	14%
	XML Parsing	712.0	640.3	11%

Table 4.3: Benchmark results in milliseconds with various processor speeds

In Table 4.3 the benchmark results when the processor of the host system runs on 3.2GHz and 3.6GHz are visible. No iPhone emulator results are included since the iPhone emulator does not run on under Windows, and the Macbook used for the iPhone benchmarks is not suitable for tinkering with the processor speed.

As is visible all tested emulators react significantly when the processor speed of the host system is increased. The difference between 3.2GHz and 3.6GHz is 12.5 percent, and most results scale perfectly with the increased processor speed. The Android XML benchmark gets an unexplained additional speed bump, while the Symbian benchmarks lag behind a little. A possible explanation for the latter is that not only the processor speed, but also the memory speed could be a limiting factor. But it is clear is that none of the emulators are a suitable tool to assess real world performance, and Chapter 4.4 shows that the iPhone emulator is not an exception to the rule.

4.4 Results on Mobile Devices

Chapter 4.3 makes it clear that the emulators are a poor tool to investigate the performance of the various smartphone operating systems. The only option is running the benchmark on real phones, and that is exactly what is done. We did however not have access to Windows Mobile and BlackBerry hardware, so just three out of the five smartphone operating systems are tested in this section. The various phones used and their main specifications are listed in Table 4.4 and the raw benchmark results are available in Appendix B. Note that the y-axis scales of the different graphs are not identical, and thus not directly comparable.

	HTC Hero	iPhone 3G	Nokia N95
OS	Android 1.5	iPhone OS 3.1.2	S60 3rd Edition
CPU	Qualcomm MSM7200A	Samsung ARM	TI OMAP 2420
CPU Speed	528MHz	412MHz	332MHz
Memory	288MB RAM	128MB RAM	64MB RAM
Storage	512MB ROM / Micro SD	16GB flash	160MB flash

Table 4.4: The specifications of the tested smartphones

There are significant differences in the hardware specifications of the tested smartphones so results are not directly comparable. Normalizing the results based on the CPU clock speed would give an approximation on how the operating systems would perform on identical hardware, but differences in memory speeds, bus speeds and processor architecture mean that if the results are close no conclusion can be drawn.

4.4.1 Android

While the benchmark was developed with the latest Android SDK 2.0.1 and the HTC Hero was running Android 1.5 no modifications to the benchmark code were necessary. Figure 4.8 shows that the results obtained by the emulator are roughly comparable with those obtained by the real hardware, but the performance differences

Figure 4.5: HTC Hero

Figure 4.6: iPhone 3G

Figure 4.7: Nokia N95

are not consistent. Encryption and decryption are respectively 9 and 14 percent faster while XML parsing is 48 percent slower.

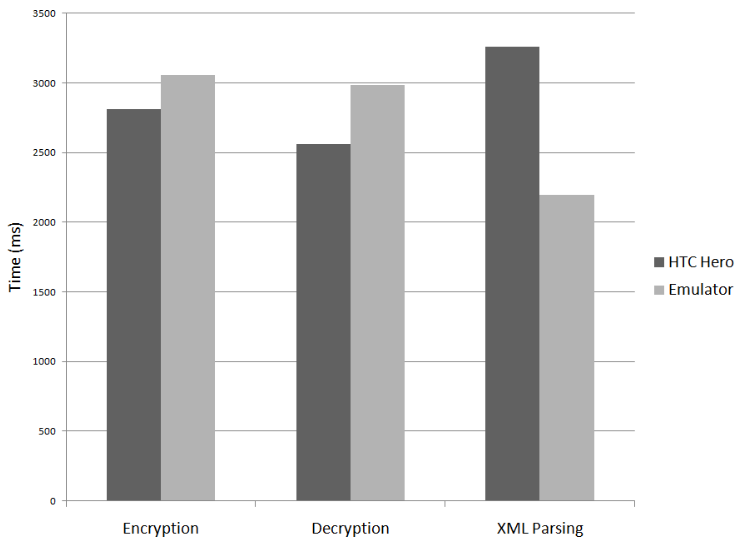

Figure 4.8: Benchmark results on HTC Hero

4.4.2 iPhone OS

The tested iPhone was running iPhone OS 3.1.2, the same version as the SDK used to develop the benchmark, so no changes were required to deploy the application. As the fastest operating system on the emulator it is no surprise that the real world results are significantly slower.

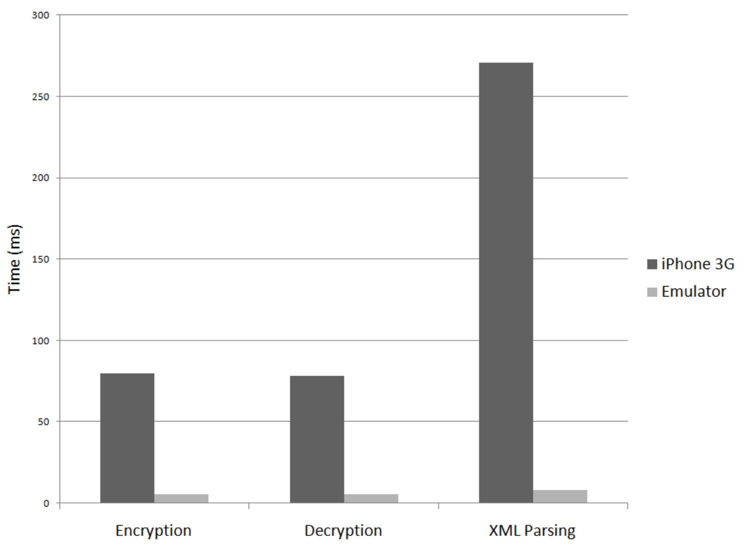

Figure 4.9: Benchmark results on iPhone 3G

Encryption and decryption both are almost 15 times slower, and just as on the HTC Hero, XML parsing is relatively slower on real hardware than on the emulator. The iPhone 3G is approximately 34 times slower when parsing XML, more than twice the difference when encrypting or decrypting as is illustrated in Figure 4.9.

4.4.3 Symbian

The Symbian benchmark was developed for Symbian S60 5th Edition while the Nokia N95 is running S60 3rd Edition FP1. Because of this the XML parsing benchmark had to be removed since it used a library that was not included in the earlier version of the operating system, and no similar library was available as well. The second difference is that the real device offers timers with a high resolution, removing the need to run every benchmark one thousand times before calculating the average execution time.

Just as the iPhone emulator, the Symbian emulator is extremely fast as illustrated in Figure 4.10. But as expected, on real hardware the performance does not come close to those results. The time needed to encrypted and decrypt a file rose with a factor of respectively 26 and 22. Although this is obviously a very big difference, the speed on real hardware is still very respectable.

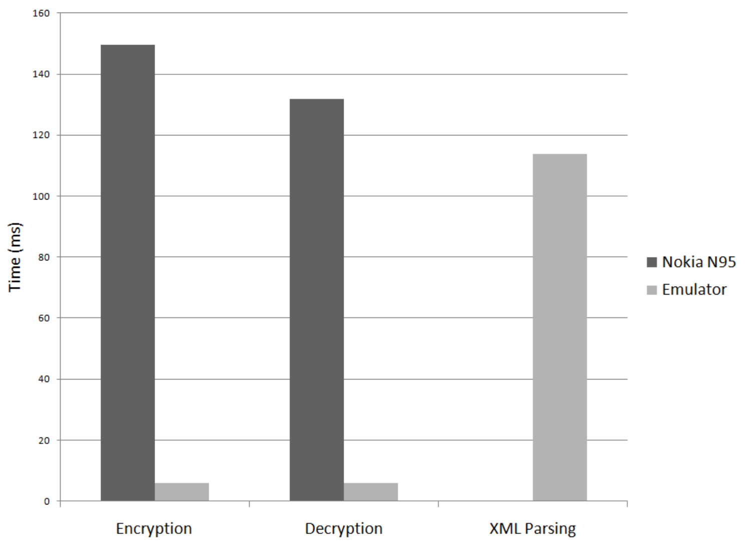

Figure 4.10: Benchmark results on Nokia N95

4.4.4 Side by Side

A look at the results on the previous pages show that while the benchmark is a lot slower on real hardware when using Symbian or iPhone OS, Android remains slow. The results of the three operating systems can be seen side by side in Figure 4.11.

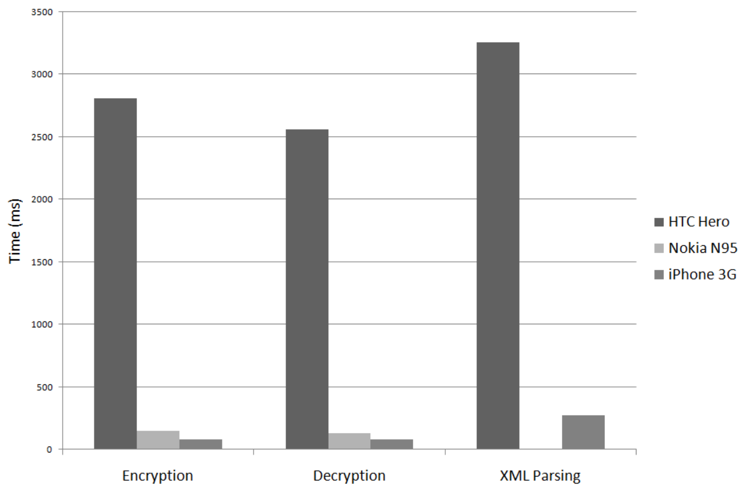

Figure 4.11: The results of the three smartphones side by side

On the emulator both Symbian and iPhone OS managed to perform several hundred times faster than the Android operating system. On real hardware the difference is smaller, but still remarkable large. Compared to the iPhone 3G, which is equipped with a slower processor, the HTC Hero is respectively 35 and 33 times slower when encrypting and decrypting. The difference when parsing an XML file is smaller, but the iPhone is still a factor 12 faster.

The differences between the iPhone 3G and Nokia N95 are small when we consider that the Nokia N95 is an older smartphone with a slower processor. Since there are many unknown variables that influence the results such as memory speeds and the architecture of the processor we have to settle with the conclusion that the iPhone OS offers performance that is in the same ballpark as Symbian.

4.5 Summary

The results obtained in Chapter 4 are all over the place, but nevertheless some conclusions can be drawn.

The first conclusion is that all the tested emulators are not a suitable tool for performance measurements. The performance of the emulators vary based on the clock speed of the processor of the host system, implying that they do not strive to emulate a certain, fixed, performance level. The extremely high speeds of the Symbian and iPhone emulator hint that the emulator layer in those cases is very thin, and that the benchmark is almost executed directly on the host system. The Android emulator performs closer to real hardware, but the differences vary seemingly random with different tests.

Although the emulator results are not reliable, there is one characteristic that appears to be translated to real world results. The operating systems that have fast emulators are also fast on real hardware, and slow emulators indicate slow phones. An explanation for this could be that the additional layers that make the emulator slow, are also present on the real phone. The two fast performers both use a static compiled language that is executed directly on the phone while the other three smartphone operating systems have been benchmarked with dynamic compiled languages. But as already mentioned, there is no fundamental reason why dynamic compiled languages should be slower than static compiled languages [41].

From the three slow emulators we have tested only Android on real hardware, so it is unclear if BlackBerry OS and Windows Mobile would behave the same. There is in fact a good reason to believe that Android performs worse than the competition in this regard. As discussed in Section 3.2.1 Android applications are all run inside the Dalvik virtual machine that run Java applications that are converted to the Dalvik Executable format. The Dalvik virtual machine is designed to work with limited resources, making it a good fit for smartphones, but at the moment no just-in-time compiler is available.

The omission of a JIT compiler means that no runtime optimizations are done by the virtual machine, resulting in poor performance. Recently an experimental version of a JIT for the Dalvik VM has been released by the Dalvik team [10]. The experimental JIT for Android is trace-based meaning that only hot code traces are compiled and not complete methods as is often done in server-class JITs. How much difference a trace-based JIT can make is illustrated by the TraceMonkey engine that executes JavaScript for the Firefox browser. Depending on the benchmark used a speed increase between a factor 1.83 and 22.5 is observed [19]. The Dalvik VM JIT is not finished, but preliminary results show that this experimental version is capable of doubling the performance of some processor intensive operations [28].

Related work with regards to benchmarking mobile operating systems has been done by Sander Kikkert. In his bachelor thesis he investigates the performance of web services on Android, Symbian and Windows Mobile [29]. The web services benchmark is not as processor intensive, the biggest bottleneck seems to be setting up and destroying network connections. The results do show a similar picture, although some of the differences are less extreme. The Android emulator is again by far the slowest and is capable of processing 1 request per second. The Windows Mobile emulator is 10 times faster while the Symbian emulator has the highest throughput with 33 requests per second.

The benchmark has also been carried out on a PDA with Windows Mobile 5.0. The device performed - depending on the network connection used - between a factor 3 and 4.5 better than the emulator. If the benchmark developed for this thesis would show a similar result the performance of Windows Mobile would be in the same order as Symbian and iPhone OS.

Chapter 5

Conclusion and Future Work

Pervasive computing is a broad field that is largely undiscovered and identifying what characteristics make a mobile operating system suitable for pervasive computing applications is not a trivial task. A wide variety of applications are possible, and arguably the most attractive applications have yet to be developed. Nevertheless seven criteria for comparing the five smartphone operating systems have been identified. Six out of the seven criteria have been evaluated qualitatively while the performance has been evaluated with quantitative measurements.

- Concurrency model
- Cost
- Memory use
- Networking
- Performance
- Power consumption
- Security and privacy

Three quality attributes are derived from the specific features that a lot of pervasive computing applications share while the other quality attributes are more general in nature and apply to most mobile applications.

The concurrency model, the networking API's and the security and privacy features are all very important for ubiquitous computing since these applications often work in the background, invisible to the user, and communicate wireless with local devices in the neighborhood. Security and privacy is also a major concern as these applications deal with where the user is, what he is doing and what he wants. Cost, memory use, performance and power consumption are relevant for almost every mobile application.

	Concurrency	Network	Security
Android	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆
BlackBerry	⋆⋆⋆	⋆⋆	⋆⋆⋆
iPhone OS	⋆	⋆⋆	⋆⋆
Symbian	⋆⋆⋆	⋆⋆	⋆⋆
Windows	⋆⋆⋆	⋆⋆⋆	⋆⋆

Table 5.1: Summary of pervasive computing specific quality attributes

An overview of the strong and weak points of the different operating systems with respect to the pervasive specific quality attributes is provided in Table 5.1. Android is the only OS that scores high in all categories, and is together with Windows Mobile the only operating system that gives developers good control over both Bluetooth and WiFi wireless network connections. BlackBerry OS and Android stand out because of the solid security model that shields applications from each other. Symbian does not stand out, but unlike iPhone OS it does not lack support for running background processes for third party developers.

	Cost	Memory	Performance	Power
Android	⋆⋆⋆	⋆⋆	⋆	⋆⋆⋆
BlackBerry	⋆⋆	⋆⋆	⋆⋆	⋆⋆
iPhone OS	⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆
Symbian	⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆
Windows	⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆

Table 5.2: Summary of general quality attributes

The quality attributes in Table 5.2 are broader in nature and show for most quality attributes no big differences between the operating systems. Android scores high in the cost category since it is at the moment the only true open source operating system in this comparison. With respect to memory use Android scores a bit lower, together with BlackBerry OS, because both operating systems require applications to be run inside a virtual machine. BlackBerry OS also gets a two star score in the power consumption category since it gives the developer no control over the power mode of the device.

The truly interesting results are found when examining the performance. As discussed in detail in Chapter 4 Android performs a lot slower than both Symbian and iPhone OS. On first sight this is unexpected as the core tasks of the operating system like thread and memory management are not that resource intensive. But in case of mobile operating systems, when you choose an OS, you not only choose a kernel, but also what libraries you can use, what languages are available, the compiler and/or the available virtual machine. The benchmark measures a combination of all these aspects, and depends on the standard libraries for the performance measurements.

This makes it is possible that Android with the Dalvik virtual machine is significantly slower than both iPhone OS and Symbian when doing computationally intensive work. The rating of Windows Mobile and BlackBerry OS have to be taken with a grain of salt since neither have been benchmarked with real hardware. Emulator results indicate that they could have performance issues, but unlike Android they at least have a virtual machine that does just-in-time compilation. Windows Mobile, unlike Android and BlackBerry OS, also offers developers the possibility to run unmanaged applications and to use third party libraries.

Not a single operating system is perfect, and it depends on what characteristics are important for a specific application to make the best match. Windows Mobile scores good in most categories, and is especially attractive since it gives developers full access to the Bluetooth and WiFi interface. Android also offers good networking features, but the subpar performance could be a serious problem for some applications. Symbian and BlackBerry OS both are reasonable suitable for most pervasive computing applications. Iphone OS is the only operating system that is not attractive since it does not support background processes.

The mobile operating landscape is quickly evolving, and in just a few months time this picture could change. The Android developers have already announced that they are working on a JIT compiler for the Dalvik virtual machine and it is rumored that the next generation iPhone OS will support multi-tasking and background threads [10, 22]. When this happens all major smartphone operating systems would be offering the features that are required for the majority of pervasive computing applications, making smartphones really the first viable platform for pervasive computing. Flexibility in connecting to local wireless networks would be the biggest remaining obstacle.

5.1 Future Work

The conclusion leaves some questions unanswered. First of all it would be interesting to know how Windows Mobile and BlackBerry OS would perform on real hardware, and the comparison could be improved by selecting smartphones with matching hardware specifications. The benchmark itself could also be improved by adding more tests that span a wider range of tasks. Comparing for example the speed of algorithms related to speech recognition, speech synthesization or image processing could add value since the current tests only represent a small part of the possible tasks that a pervasive computing application could face. Tests that focus on core tasks of the operating system such as process swapping, file I/O, memory allocation and low-level socket access would benefit the benchmark as well. The current benchmark relies heavy on the libraries supplied with the operating system, and it is possible that better alternatives exist for the standard libraries.

A good addition would be to not only measure computational performance, but also energy efficiency. A lot of pervasive computing applications will be idle most of the time, only periodically connecting to networks and checking if there are tasks that should be done. Developing a benchmark that imitates this usage pattern, while using various network technologies, would be a helpful tool to assess how suitable a smartphone OS would be for pervasive computing.

Another area were the research could be expanded are the operating systems itself. It would be interesting to take a look at the newcomers on the smartphone market such as WebOS or Maemo. Similar research could also be carried out for operating systems that are not designed for smartphones, but for various types of embedded devices, like Windows CE or TinyOS.

Bibliography

[1] Sm4all project. Online, Sep 2009. http://www.sm4all-project.eu/. [cited at p. 8]

[2] Amigo. Ambient intelligence for the networked home environment. Online, Sep 2009. http://www.hitech-projects.com/euprojects/amigo/. [cited at p. 7]

[3] Apple. *Iphone OS Technology Overview*, Oct 2008. [cited at p. 19, 20, 62]

[4] Apple. *Security Overview*, Oct 2008. [cited at p. 23, 62]

[5] Apple. Apple march 17 event. Online, 2009. http://events.apple.com.edgesuite.net/0903lajkszg/event/index.html. [cited at p. 22]

[6] Apple. *CFNetwork Concepts*, Mar 2009. [cited at p. 21, 62]

[7] Apple. *iPhone Human Interface Guidelines*, Mar 2009. [cited at p. 20]

[8] Apple. iphone sdk agreement. Iphone SDK, Mar 2009. [cited at p. 9, 20]

[9] Magdalena Balazinska, Hari Balakrishnan, and David Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery. In *In Proceedings of the First International Conference on Pervasive Computing*, pages 195–210. Springer-Verlag, 2002. [cited at p. 6]

[10] Bill Buzbee. Dalvik jit compiler. Online, Nov 2009. http://tiny.cc/6ILCi. [cited at p. 45, 49]

[11] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *ACM Wireless Networks*, 8(5), September 2002. [cited at p. 6]

[12] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schloss Elmau, Germany, May 2001. IEEE Computer Society. [cited at p. 6]

[13] Chris Dannen. Technology: Hacking the iphone for espionage. Online, 2009. http://www.fastcompany.com/blog/chris-dannen/lab/technology-hacking-iphone-espionage. [cited at p. 23]

[14] Shakhnarovich Lee Darrell, G. Shakhnarovich, L. Lee, and T. Darrell. Integrated face and gait recognition from multiple views. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 439–446, 2001. [cited at p. 6]

[15] Dev-Team. Pwnagetool release info. Online, 2009. http://blog.iphone-dev.org/post/74278878/close-the-stable-door. [cited at p. 10]

[16] Maddy D.Janse. Ist-2004-004182 amigo - amigo final report, Sep 2008. [cited at p. 7]

[17] Independent Security Evaluators. Exploiting android. Online, 2008. http://securityevaluators.com/content/case-studies/android/index.jsp. [cited at p. 16]

[18] Symbian Foundation. Online, 2009. http://www.symbian.org/. [cited at p. 24, 25]

[19] Andreas Gal. Tracing the web. Online, Aug 2008. http://andreasgal.wordpress.com/2008/08/22/tracing-the-web/. [cited at p. 45]

[20] David Garlan, Daniel P. Siewiorek, and Peter Steenkiste. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1:22–31, 2002. [cited at p. 6]

[21] Gartner. Gartner says worldwide mobile phone sales declined 6 per cent and smartphones grew 27 per cent in second quarter of 2009. Online, Aug 2009. http://www.gartner.com/it/page.jsp?id=1126812. [cited at p. 12, 24]

[22] Boy Genius. Apple iphone os 4.0 features detailed. Online, Jan 2010. http://tiny.cc/M7gJi. [cited at p. 31, 49]

[23] James R. Glass, Timothy J. Hazen, and I. Lee Hetherington. Real-time telephone-based speech recognition in the jupiter domain. pages 61–64, 1999. [cited at p. 6]

[24] Google. Security and permissions in android. Online, 2008. http://code.google.com/intl/nl/android/devel/security.html. [cited at p. 16]

[25] Google. What is android? Online, 2008. http://code.google.com/android/what-is-android.html. [cited at p. 12, 13, 62]

[26] Google. *Android reference documentation*, Mar 2009. http://developer.android.com/reference/. [cited at p. 14, 15]

[27] The NPD Group. Rim unseats apple in the npd groups latest smartphone ranking. Online, May 2009. http://www.npd.com/. [cited at p. 17]

[28] Jim Huang. Benchmark of dalvik vm on beagleboard/armv7. Online, Nov 2009. http://tiny.cc/SWMSB. [cited at p. 45]

[29] Sander Kikkert. Performance of web services on mobile phones, 2010. [cited at p. 45]

[30] Sravan Kundojjala. Symbian's dominance fading as mobile software platform market becomes more crowded. Online, Jul 2009. http://www.strategyanalytics.com. [cited at p. 29]

[31] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 120–130, Boston, Massachusetts, August 2000. [cited at p. 6]

[32] RoughlyDrafted Magazine. iphone 2.0 sdk: The no multitasking myth. Online, 2009. http://www.roughlydrafted.com/2008/03/13/iphone-20-sdk-the-no-multitasking-myth/. [cited at p. 21]

[33] Lawrence S Brakmo Marc A Viredaz and William R Hamburgen. Energy management on handheld devices. *Queue*, vol. 1:pp. 44–52, Oct 2003. [cited at p. 11]

[34] Microsoft. Msdn: Kernel overview. Online, 2009. http://msdn.microsoft.com/en-us/library/aa909237.aspx. [cited at p. 28, 62]

[35] Microsoft. Msdn: Selecting a windows mobile api. Online, 2009. http://msdn.microsoft.com/en-us/library/dd630621.aspx. [cited at p. 29]

[36] MIT. Project oxygen overview. Online, Jun 2004. http://oxygen.lcs.mit.edu/Overview.html. [cited at p. 5]

[37] Nokia. Mobile leaders to unify the symbian software platform and set the future of mobile free. Online, Jun 2008. http://www.nokia.com/A4136001?newsid=1230416. [cited at p. 24]

[38] Nokia. *S60 5th Edition C++ Developer's Library v1.3*, 2009. [cited at p. 24, 62]

[39] Earl Oliver. A survey of platforms for mobile networks research. *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 12:pp. 56–63, Oct 2008. [cited at p. 15, 26]

[40] Michael Rohs Rafael Ballagas, Jan Borchers and Jennifer G. Sheridan. The smart phone: A ubiquitous input device. *IEEE Pervasive Computing*, vol. 5:pp. 70–77, Jan-Mar 2006. [cited at p. 4]

[41] Kirk Reinholtz. Java will be faster than c++. *SIGPLAN Not.*, 35(2):25–28, 2000. [cited at p. 39, 44]

[42] RIM. Low memory manager in the blackberry java development environment. BlackBerry Developer Newsletter, Jun 2005. [cited at p. 18]

[43] RIM. Blackberry mobile data system: Technical overview. Online, 2006. http://na.blackberry.com/eng/services/mobile.jsp. [cited at p. 17, 62]

[44] RIM. Blackberry java application: Fundamentals guide. Online, 2009. http://na.blackberry.com/eng/developers/. [cited at p. 18, 19]

[45] Daniel Schall and Marco Aiello. Web services on embedded devices. *International Journal of Web Information Systems*, 2:1–6, 2006. [cited at p. 8]

[46] Stephanie Seneff. Tina: a natural language system for spoken language applications. *Comput. Linguist.*, 18(1):61–86, 1992. [cited at p. 6]

[47] Stephanie Seneff and Joseph Polifroni. Formal and natural language generation in the mercury conversational system, 2000. [cited at p. 6]

[48] Ken Steele, Jason Waterman, and Eugene Weinstein. The oxygen h21 handheld. *SIGARCH Comput. Archit. News*, 30(3):3–4, 2002. [cited at p. 6]

[49] Symbian. Platform security - a technical overview. Online, Sep 2006. http://developer.symbian.com/wiki/display/pub/Technical+papers. [cited at p. 27, 62]

[50] Symbian. Symbian reports first half and second quarter results for 2008. Online, Sep 2008. http://www.symbian.com/news/pr/2008/pr200810096.asp. [cited at p. 25]

[51] Mark Weiser. The computer for the 21st century. *ACM SIGMOBILE Mobile Computing and Communications Review*, 1999. [cited at p. 3, 5]

[52] Jon R.W. Yi, Jon Rong wei Yi, and James R. Glass. Natural-sounding speech synthesis using variable-length units. In *Proc. ICSLP*, pages 1167–1170, 1998. [cited at p. 6]

Appendices

Appendix A

Emulator Benchmark Results

This appendix contains the execution times of the individual benchmark runs on the various emulators. Every test has been executed ten times on every emulator and the averages of these results are used for graphs in Chapter 4.2. Times are in milliseconds, and lower times are better.

	Encryption	Decryption	XML Parsing
Android	2961	2931	2075
	3118	3001	2174
	2961	3035	2177
	3028	2965	2129
	3008	3031	2092
	2955	2938	2113
	3438	3009	2159
	2995	3006	2114
	3064	2937	2071
	3046	3020	2872
BlackBerry OS	365	464	2234
	362	464	2235
	362	461	2237
	364	464	2239
	361	462	2233
	362	463	2302
	364	462	2251
	361	462	2216
	362	463	2215
	364	463	2215

Table A.1: Emulator benchmark results in milliseconds

	Encryption	Decryption	XML Parsing
Symbian	5.950	5.940	114.15
	5.725	5.910	117.35
	5.725	5.900	114.20
	5.720	5.855	112.80
	5.675	5.865	112.75
	5.860	5.920	112.95
	6.100	5.880	113.10
	5.765	5.940	112.85
	5.755	5.975	113.45
	5.705	5.935	113.30
Windows Mobile	867.7	776.7	1266.3
	592.3	650.6	652.9
	595.2	642.4	624.8
	729.8	637.8	655.7
	585.7	658.0	668.1
	586.1	646.4	638.2
	586.9	686.9	653.9
	631.6	646.3	661.4
	582.9	642.7	649.6
	746.3	661.4	649.3
iPhone OS	5.545	4.333	9.298
	6.596	4.008	7.753
	3.525	6.618	7.735
	6.158	3.362	7.761
	6.811	4.056	7.693
	3.773	10.983	7.682
	4.338	3.513	7.804
	9.378	5.336	7.690
	3.668	7.519	7.677
	3.905	3.255	7.668

Table A.1: Emulator benchmark results in milliseconds

Appendix B

Smartphone Benchmark Results

This appendix contains the execution times of the individual benchmark runs on the three tested smartphones. Every test has been executed ten times on every ddevice and the averages of these results are used for graphs in Chapter 4.4. Times are in milliseconds, and lower times are better.

	Encryption	Decryption	XML Parsing
HTC Hero	2670	2591	3358
	2728	4071	3203
	2611	2616	3940
	2678	2583	3250
	2642	2600	3191
	2526	2594	3355
	3108	2725	3370
	3033	2971	3158
	2527	2545	3391
	2951	2536	3162
iPhone 3G	60.9	55.9	206.7
	50.4	45.7	275.8
	197.8	129.8	307.9
	306.2	402.2	517.5
	125.5	122.5	236.9
	486.4	112.9	339.1
	93.7	113.2	328.1
	88.9	89.5	329.5
	56.2	87.0	275.7
	98.7	100.1	334.9

Table B.1: Smartphone benchmark results in milliseconds

	Encryption	Decryption	XML Parsing
Nokia N95	150.6	117.1	n/a
	147.2	118.3	n/a
	148.4	118.1	n/a
	153.1	116.4	n/a
	150.5	117.5	n/a
	146.4	117.6	n/a
	150.0	116.9	n/a
	151.1	117.7	n/a
	146.1	120.1	n/a
	148.8	146.7	n/a
	149.7	131.9	n/a

Table B.1: Smartphone benchmark results in milliseconds

List of Symbols and Abbreviations

Abbreviation	Description	Definition
API	Application Programming Interface	page 11
BSD	Berkeley Software Distribution	page 22
CDSA	Common Data Security Architecture	page 22
JIT	Just-in-time	page 44
JVM	Java Virtual Machine	page 14
MDS	Mobile Data Service	page 17
MIDP	Mobile Information Device Profile	page 17
MOAP	Mobile Oriented Applications Platform	page 24
P2P	Peer-to-peer	page 8
S60	Series 60 User Interface	page 24
UID	User identifier	page 16
UIQ	User Interface Quartz	page 24
VoIP	Voice over Internet Protocol	page 20
VRAM	Video RAM	page 21
Zeroconfig	Zero Configuration Networking	page 22

List of Figures

1.1 'Smart' fridge 3

2.1 H21 device 6

3.1 Overview of the Android architecture [25] 13
3.2 The Blackberry MDS Runtime enviroment [43] 17
3.3 Overview of the iPhone OS architecture [3] 19
3.4 CFNetwork and other software layers on Mac OS X [6] 21
3.5 Mac OS X security architecture overview [4] 23
3.6 Overview of the Symbian architecture [38] 24
3.7 Overview of the Symbian capabilities [49] 27
3.8 General structure of the Windows Mobile kernel [34] 28

4.1 The structure of the benchmark 35
4.2 Encryption benchmark results 37
4.3 Decryption benchmark results 37
4.4 XML parsing benchmark results 38
4.5 HTC Hero 41
4.6 iPhone 3G 41
4.7 Nokia N95 41
4.8 Benchmark results on HTC Hero 41
4.9 Benchmark results on iPhone 3G 42
4.10 Benchmark results on Nokia N95 43
4.11 The results of the three smartphones side by side 43

List of Tables

3.1 Summary of quality attributes 31

4.1 Overview of the various programming languages used 34
4.2 The specifications of the benchmark systems 36
4.3 Benchmark results in milliseconds with various processor speeds . . . 39
4.4 The specifications of the tested smartphones 40

5.1 Summary of pervasive computing specific quality attributes 48
5.2 Summary of general quality attributes 48

A.1 Emulator benchmark results in milliseconds 57
A.1 Emulator benchmark results in milliseconds 58

B.1 Smartphone benchmark results in milliseconds 59
B.1 Smartphone benchmark results in milliseconds 60