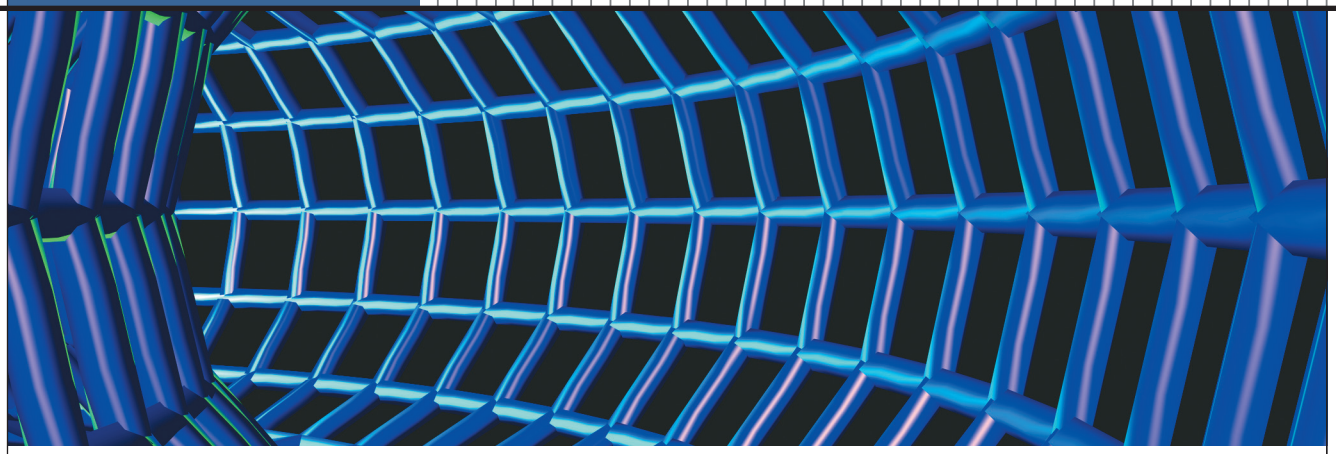

Arching over the Mobile Computing Chasm: Platforms and Runtimes

Sasu Tarkoma and Eemil Lagerspetz, *University of Helsinki*

Platforms, runtimes, and middleware play a vital role in an evolving mobile computing environment in which the trend is toward converged communication, where Web resources integrate seamlessly with mobile systems.

Mobile devices increasingly depend on reliable software to offer a good user experience. Developing software in this operating environment requires many support services,[1-3] which are mainly provided in the middleware layer. Middleware offers a level of indirection and transparency for application developers, who save development cost and time using standardized or well-known interfaces when designing their products.

Development time and cost have traditionally been high for mobile applications and services, which operate in a more challenging environment than a typical fixed network. The wireless and mobile environment is less stable, has high latency, limited bandwidth, and many terminal types. Therefore, a specific implementation is not necessarily usable by all mobile equipment on the market.

Managing high development costs, meeting the challenges of the wireless network, and supporting device mobility motivate mobile handset manufacturers and vendors to provide middleware solutions for easier development and a unified user experience in the fragmented mobile marketplace.[4,5]

MOBILE MARKETPLACE EVOLUTION

In the 1980s, mobile phones provided only basic voice services.[6] The first generation of mobile applications and services, introduced around 1991, were restricted by technology. The two key enablers for application development were the mobile data connection and the Short Message Service.

The second generation of mobile applications was supported by built-in browsers, such as the Wireless Application Protocol (WAP) and, more recently, lightweight Web browsers. This generation also introduced the Multimedia Messaging Service (MMS) for images, audio, and video.

A more sophisticated environment supports third-generation applications and services that are built atop a platform offering services such as location support, content adaptation, storage, and caching. Platforms supporting the emergence of third-generation applications include Symbian Series 60, Java ME, Android, and the iPhone iOS.

Although the fourth generation of applications is still emerging, we can briefly sketch their anticipated properties in light of recent proposals in the research and standardization communities. The fourth generation is expected to

 Published by the IEEE Computer Society 0018-9162/11/$26.00 © 2011 IEEE

Table 1. Overview of popular smartphone systems.

Property	Android Linux	iPhone OS	Java ME MIDP	MeeGo Linux	Symbian Series 60	Windows Mobile .NET and Windows Phone 7
Development	Java, native code with JNI and C/C++	Objective-C	Java ME	C/C++, Qt APIs, various	C++, Qt, Python, various	C# and .NET, Silverlight, various
Network and energy monitoring/ control	Several APIs	Limited API support, battery monitoring since 3.0	No	Several APIs, native calls	Yes	Yes (limited in WP7)
Background processing	Yes (services)	No (yes for 4.0)	Yes (multitasking support in MIDP 3.0)	Yes	Yes	Yes, not supported for third-party applications in WP7
HTML5	Yes, support depends on version	Yes	N/A	Yes, support depends on version	Yes, future versions	No, expected in future versions
SIP API support	Yes, support depends on version	Extension	Extension	Yes	Yes	No, possibly in future versions
Open source	Yes	No	No	Yes	Yes	No
Third-party application installation	Certificate, Android market	Certificate, Apple App Store	Certificate	Certificate	Certificate	Certificate, WP7 apps marketplace

be adaptive not only in terms of application behavior and content, but also in the networking stack and wireless interface. Always-on connectivity, multimode communications, mesh networking, adaptive network interfaces, and physical communication media will be important features of future mobile computing devices.

MOBILE PLATFORMS

Table 1 gives an overview of the different mobile platforms and their properties. C, C++, and Java are currently the dominant programming languages for mobile devices. Network scanning and interface control functions, which have varying levels of support in mobile platforms, are important when an application needs to monitor and control the wireless communications. Background processing, which denotes the platform's multitasking capabilities, and energy and power monitoring and control—two important aspects of mobile platforms—are fairly well supported across platforms. Both multitasking and energy-management features vary from system to system. Memory management and persistent storage are well supported across the platforms, as is location information.

HTML5, the next version of HTML, is in development. The first public working draft of the specification was made available in January 2008, and completion is expected around 2012. Browser vendors are already implementing HTML5 features as they are defined. Some HTML5 features, including the WebSocket API, advanced forms, offline application API, and client-side persistent storage (key/value and SQL), can significantly improve current mobile Web applications. The iPhone platform has good support for HTML5.

The Session Initiation Protocol (SIP) is a key signaling protocol in 3G and 4G wireless access networks for session management.[7] Some platforms expose a SIP API to developers.

Three platforms are fully open source: Android, Maemo/MeeGo, and Symbian OS. The platforms have varying systems for supporting third-party application installation and execution. Execution of privileged system functions requires certification or other means of obtaining permission. The newer platforms are less fragmented, whereas older systems are invariably fragmented.

Android

The Android operating system and software platform for mobile devices is based on the Linux operating system. Android was developed by Google and the Open Handset Alliance, which includes more than 30 companies. The platform allows development of managed code using a Java-like language that follows the Java syntax, but does not provide the standard class libraries and APIs. Instead, it uses libraries and APIs developed by Google.

Figure 1 shows the Android architecture. It is based on the Linux kernel and a set of drivers for the various hardware components, such as display, keypad, audio, and connectivity. Android includes a set of C/C++ libraries for use by its various components. The Android application

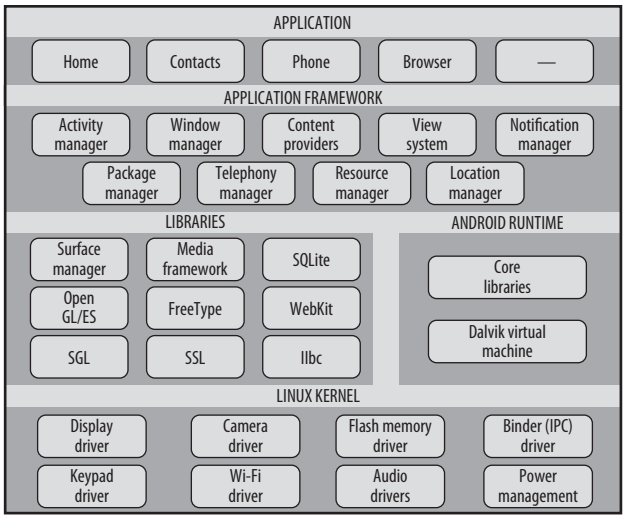

Figure 1. The Android architecture and its key components. Android is based on the Linux kernel and includes a set of C/C++ libraries, which are used by its various components.

framework APIs expose the capabilities of these libraries to developers.

The Android runtime executes the custom Java bytecode. The runtime includes the core libraries and the Dalvik virtual machine. Atop the libraries and runtime is the application framework, which consists of various managers. Several bundled applications reside atop the managers. These applications, which include an e-mail client, SMS program, calendar, maps, browser, and contacts, are all written in Java. Developers use the same API that the built-in core applications use. Android emphasizes component reuse, and any component can publish its capabilities, which other components can use if security constraints do not prevent this.

BlackBerry

RIM's BlackBerry devices are based on a proprietary operating system. Version 4 supports Java Mobile Information Device Profile (MIDP) 2.0 applications and synchronization with various productivity suites. The communications model is based on enterprise servers that act as e-mail relays. The servers use RIM's network operation center (NOC) to send and receive messages to and from the mobile devices. Because they use a proprietary NOC, the servers can implement mobile push efficiently.

iPhone

Apple developed the iPhone mobile operating system, or iOS, for its iPhone, iPod touch, and iPad products. The operating system is derived from Mac OS X and uses the Darwin foundation, built around XNU, a hybrid kernel combining the Mach 3 microkernel, elements of Berkeley Software Distribution (BSD) Unix, and an object-oriented device driver API (I/O kit).

Figure 2 gives an overview of the Mac OS X architecture, which was adapted for the iPhone architecture. The iPhone system is built on an ARM processor, and the core operating system (Darwin) includes the XNU kernel and system utilities. The XNU kernel includes Posix support, networking and file system support, and the device drivers. Above the operating system is the layered middleware—namely, core services, application services, the API layer, and finally the GUI (Aqua).

Apple provides the SDK as a free download, but requires approval and payment to release software for the iPhone platform in the App Store. There, users can browse and download applications directly to their iPhone, iPod touch, or iPad. Figure 2 shows the five available APIs: Carbon, QuickTime, BSD/Posix, Classic, and Cocoa.

Carbon is a procedural API consisting of a file manager, resource manager, font manager, and event manager. Each manager offers an API related to some functionality, defining the necessary data structures and functions. Managers are often interdependent or layered.

The Posix specifications define crucial operating system software interfaces and a standard threading library API.

The Classic environment, a backward-compatible hardware and software abstraction layer, is no longer supported in the current Mac OS version.

Cocoa Touch provides an abstraction layer based on Cocoa, the native Mac OS X object-oriented application program environment. Cocoa's design follows model-view-control (MVC) principles, and its frameworks are written in Objective-C. The Cocoa layer supports multitouch events and controls, and it has an interface for accelerometer input and support for localization (i18n) and a camera.

Announced in April 2010, version 4.0 of the iOS software supports multitasking for third-party applications. The key design principle is to offer APIs for specific background operations to optimize overall system performance. The new iPhone multitasking-specific APIs include support for background audio play, VoIP, location services, task completion, and fast application switching. For example, VoIP applications will be able to receive calls in the background. The APIs also support third-party push servers for sending notifications to applications.

The iPhone SDK supports the development of three types of applications—iPhone, iPad, and universal applications. A universal application determines the device type and then uses the available features based on conditional statements.

Java ME

Java Platform, Micro Edition (Java ME, previously J2ME), specifies a standardized collection of Java APIs for developing software for small and resource-constrained devices. Target applications include consumer devices, home appli-

ances, security, defense, automotive, industrial, industrial control, and multimedia. Since December 2006, the Java ME source code has been licensed under the GNU general public license.

A Java ME configuration specifies the virtual machine and the core libraries. There are two main configurations:

- the connected device configuration (CDC) for high-end PDAs, and
- the connected limited device configuration (CLDC) for mobile phones and other small devices.

Device manufacturers augment the configurations with profiles, which define additional APIs. The most common profile is the MIDP, aimed at mobile phones. The Personal Profile targets consumer products and embedded devices.

The Java ME platform's Mobile Service Architecture (MSA) specification (Java Specification Request [JSR] 248) defines a standard set of application functionalities for mobile devices, covering interactions between various technologies associated with the MIDP and CLDC specifications. An MSA version 2 device can use either CLDC 1.1 or CDC 1.1 as its configuration. The MIDlet execution environment is extended to CDC.

Java ME is evolving into a versatile platform for mobile application development. The introduction of MSA2 and various JSRs has gradually removed the early restrictions with MIDP applications and won growing vendor support. Moreover, MIDP version 3 addresses software portability challenges between CLDC and CDC.

Kindle SDK

Amazon offers a Kindle SDK for developing Java-based active applications for Kindle e-book readers. The Kindle SDK is based on the Java ME Personal Basis Profile and Kindle-specific extensions. The APIs support a basic user interface, networking, and limited secure storage on the device.

Maemo and MeeGo

Nokia's Maemo platform includes the Internet Tablet OS, which is based on Debian GNU/Linux and draws much of its GUI, frameworks, and libraries from the GNU Object Model Environment (Gnome) project. It uses the Matchbox window manager and, like Ubuntu Mobile, uses the GTK-based Hildon as its GUI and application framework. The Maemo platform is intended for Internet tablets, which are smaller than laptops but larger and more versatile than PDAs. A tablet might have a small keyboard, and its central characteristics include a stylus and a touch-sensitive screen. The touch screen is an important consideration for developers when designing graphical interfaces.

The latest development combines Nokia's Maemo platform with Intel's Moblin to form the MeeGo system. Both

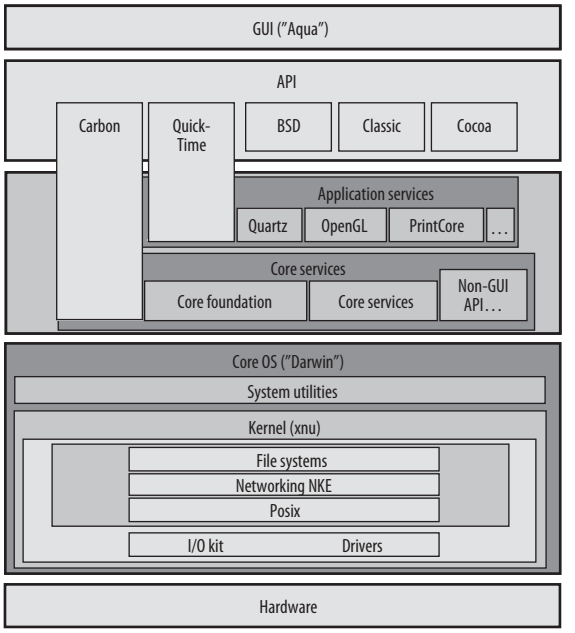

Figure 2. iPhone architecture and its main components. The architecture uses the Darwin operating system, which includes the XNU kernel and system utilities.

the Maemo and Moblin applications were developed with the GTK framework. However, Nokia's Qt framework has replaced GTK. MeeGo is expected to run on both Atom and ARM processors and to support both netbooks and mobile phones. MeeGo applications are written in C++ using the MeeGo SDK, which includes Qt. In February 2011, Nokia announced that its future smartphones will be based on the Windows Phone 7 platform.

The MeeGo architecture includes a hardware abstraction layer (HAL), an operating system base (Linux kernel, X), middleware, and user-experience-related functions. The lowest HAL layer, provided by the device vendor, includes kernel drivers and patches, kernel configuration, modem support, and other software related to the underlying hardware. MeeGo includes a set of components called the *content framework* to gather and offer user metadata to application developers.

Qt is a cross-platform application framework designed for building GUI applications. It provides the basic APIs for GUIs, databases, XML, and networking, and includes a WebKit-based Web runtime. The Qt platform is available for several systems, including Windows, Mac OS, Linux, Symbian, and Windows CE. The Qt API is implemented in C++, which most developers use. Currently, developers can only use C++ to create Symbian applications, although other language bindings are available for other platforms. The Qt platform is currently being extended to support device-specific APIs pertaining to location, calendars, alarms, sensors, and so on.

HP WebOS

HP's WebOS, originally developed by Palm, runs on the Linux kernel. The mobile runtime system includes a WebKit-based browser, and applications are written in JavaScript. The WebOS follows the cloud-based service model. The JavaScript-based application framework, called Mojo, provides common functions pertaining to user interfaces, widgets, and data access. A typical WebOS application uses HTML5 for presentation and audio/video. Developers create the applications using the MVC architectural pattern to separate user experience concerns from the application's data model and storage.

The WebOS is based on a scene metaphor in which an application consists of a set of scenes that facilitate presentation and user interaction. Scenes are pushed into and popped out of a scene stack. The top scene in the stack is visible to the user. The execution framework activates and deactivates the scenes. The scenes and applications use asynchronous notifications (W3C document object model events) to signal changes.

A mobile platform must be flexible and extensible not only in the distributed environment but also in the local environment.

Symbian and Series 60

Symbian's open mobile operating system is designed for ARM processors. The system includes a microkernel operating system, associated libraries, a user interface, and a reference implementation of common tools. Like many desktop operating systems, Symbian is structured with preemptive multitasking and memory protection. The multitasking model features server-based asynchronous access based on event passing. Three design goals motivated the choice of servers, microkernel design, and event passing:

- minimizing response times to users,
- maximizing integrity and security, and
- utilizing scarce resources efficiently.

Nokia acquired ownership of Symbian in 2008 and established the Symbian Foundation to provide royalty-free software for the mobile environment. The Symbian operating system was open sourced in 2010.

The base services layer is the lowest level reachable by user-side operations. It includes the file server and user library; the plug-in framework, which manages all plug-ins; a store; a central repository; a DBMS; and cryptographic services. The base services layer provides basic connectivity and serial communications as well as telephony. The communications infrastructure was developed on this layer, with two prominent networking stacks—TCP/IP and WAP. The Web and WAP browsers are available for the respective protocol stacks. The Symbian Web runtime is based on the WebKit system. The Java runtime and JavaPhone are available for applications.

The Symbian operating system's native language is C++, but the language is not compatible with ANSI C++. The operating system and applications are based on the MVC design pattern, which supports the separation of functions. The Symbian operating system emphasizes resource recovery using several programming features, such as a cleanup stack and descriptors. The operating system's event-based nature allows the minimization of thread switching using *active objects* that support asynchronous processing by encapsulating service request and request completion processing. In addition to C++ native applications, widgets are supported through the Nokia Web Runtime (WRT) widgets. The WRT environment follows the W3C widgets specification and allows widget installation and execution. The widgets can access device-specific features using the JavaScript Platform Services 2.0 API.

Windows Mobile and .NET Compact Framework

Microsoft released Windows Mobile 6 at the 3GSM World Congress in 2007. It comes in a standard version for smartphones, a version for PDAs with phone functionality, and a classic version for PDAs without phone features. Windows Mobile 6 is based on the Windows CE 5.0 operating system and integrates with Windows Live and Exchange products. Software development for the platform typically uses Visual C++ or the .NET compact framework. When native client-side functionality is not needed, software developers can use server-side code that is deployed on a mobile browser, such as Internet Explorer Mobile bundled with Windows Mobile.

The next version is the Windows Phone 7 Series (WP7) announced at the 2010 Mobile World Congress. WP7 focuses on user experience and does not support third-party software multitasking.

CURRENT STATE

The current platform landscape is heterogeneous, with several operating systems, programming languages, and interfaces in use, resulting in complex mobile software development and testing processes. A mobile platform must be flexible and extensible not only in the distributed environment but also in the local environment. The current and emerging platforms are still limited in this respect. For example, because third-party developers cannot easily extend Java ME MIDP, iPhone, or Android APIs, it is easier to extend and modify functionality at the server side than to modify the client.

The convergence of mobile and traditional IT fields has led to the increasing use of Web technologies in the development and deployment of mobile applications. The current Web technologies are suitable for mobile applications that conform to the Web's request/reply interaction style. However, in many cases Web protocols do not directly work well with mobile and wireless links. Indeed, asynchronous operation would be particularly useful in mobile applications that must react to changes in the environment.

Support for adaptive operation is an important trend in mobile applications and services. Adaptation can be realized in many ways—for example, on client devices or servers, using proxies and gateways, and through collaboration of the different entities, including services and software. Context awareness also introduces new challenges, such as context acquisition, privacy, and software testing and quality assurance. Testing adaptive and context-aware behavior requires new kinds of solutions and methods for ensuring that software is working properly and that it generates the desired user experience. Unfortunately, universal device and service discovery is still not available for developers. The current trend in developing adaptive applications is to use both Web technology and platform-specific APIs.

Thus, the current state leaves much room for improvement. No common APIs for network scanning and selection or network interface control exist. Background processing is supported on most platforms, but not all. Application-level energy awareness is far from ubiquitous. However, memory management, persistent storage, and location information are widely supported.

The marketplace has a clear need for a common API that unifies network connectivity, energy awareness, and the user experience. This should take a form that is easy to deploy on existing devices and most software stacks.

TOWARD COMMON APIS

One solution to the current challenge of fragmented device base and development tools is to provide common APIs for service and application developers. Indeed, both device manufacturers and telecom operators are actively involved in various API development and standardization efforts.

One key aspect is the development language and environment. Although recent experimental results suggest that JavaScript-based application platforms can be executed on Web browsers, several practical challenges pertaining to performance and browser limitations remain.[8] One challenge is determining how to allow a Web-based application to access local system variables, such as context variables. Security and privacy are paramount.

Web runtimes therefore must provide access to client-side platform APIs, such as the file system, geolocation, or camera. Previously, these APIs were exclusive to native applications. The industry is focusing on JavaScript and URL-based APIs to solve the API fragmentation problem. At least in theory, JavaScript APIs should be accessible to any content rendered by the Web runtime.

The Open Mobile Alliance, a key mobile standardization organization, bases its browsing specifications on Internet technology, but limits profiles for constrained resources and user interfaces of mobile devices. The GSM Association's OneAPI initiative aims to define a commonly supported API for mobile operators exposing network information to Web application developers.

The Open Mobile Terminal Platform group is pursuing a standardization activity that defines requirements and specifications for simpler and more interoperable mobile APIs.

The marketplace has a clear need for a common API that unifies network connectivity, energy awareness, and the user experience.

CHALLENGES

Mobile computing and software development face several important challenges. A key problem is fragmentation, which can occur on multiple layers and dimensions (the operating system, platform and middleware, service API layers, and so on). Currently, the available operating systems and platforms have differing programming conventions, interfaces, and software distribution solutions. This increases software development costs and slows down the software ecosystem.

In addition to fragmentation, the nature of the APIs and the features of the underlying platform they expose differ widely. Most systems expose certain underlying system features, some requiring authorization to access. For example, access to context information and networking services varies from system to system.

An asynchronous system-wide event bus is a basic solution for interconnecting various on-device components; however, there is no single standard for this. For example, Android and Java ME use Java-specific events, MeeGo uses D-Bus, and HP's WebOS uses W3C events. One trend is to use URI-based conventions for naming system resources and services. This approach is used extensively in Nokia Platform Services, WebOS, and other runtimes. An alternative, albeit more radical, solution to fragmentation is to use virtualization to execute the entire mobile application software stack.[9]

Energy consumption is one of the greatest challenges for current mobile devices. Energy and power continue

to remain the most limiting factors for the performance of mobile computing systems. Battery capacity does not increase as fast as the requirements. Internet and Web 2.0 services, in particular, consume vast amounts of energy, resulting in short battery lifetimes and, ultimately, poor user experiences. Current research challenges include how to support energy accounting and execute applications across mobile devices and cloud-based systems.[10]

A considerable amount of R&D has gone into solutions for different kinds of mobile and pervasive environments that support a wide variety of applications. However, the solution landscape is still fragmented. The next step to realizing the visions of pervasive computing is to support access to context information and enable more intelligent information processing on client devices.

Given that there are more than 3 billion mobile devices on the market today, with projections indicating that the number will approach 5 billion in the near future, the prospects for mobile applications, services, and middleware appear promising. Handling such a large number of users with widely divergent device types and characteristics necessitates developing interoperable and high-performance platforms as well as a highly scalable and available fixed infrastructure.

One step toward extensibility and universality is to employ a common interoperable message bus that supports component discovery, capability negotiation, and communications. Researchers have proposed message passing, publish-subscribe,[11] and tuple spaces as key components for mobile and pervasive software, but these ideas have not yet found their way into products and standardization. HTTP and runtime-specific APIs or local sockets are still the common denominator for communications and for enabling intradevice communications.

Although it has not yet been adopted on a large scale in the mobile marketplace, the HTML5 specification offers one approach for providing persistent storage and a satisfactory user experience. The iPhone iOS is pioneering the use of HTML5. It remains to be seen how fast other mobile platforms adopt this new specification. HTML5 in combination with custom JavaScript APIs would open a world of possibilities for developing portable and cloud-assisted mobile software.

Another approach is to use virtualization techniques to support multiple operating systems and platforms on the same hardware, possibly at the same time. Organizations could also use virtualization to enhance system security. This is a future technology still maturing for mobile devices.[9]

References

1. M. Weiser, "Ubiquitous Computing," *Computer*, Oct. 1993, pp. 71-72.
2. A.K. Dey, "Understanding and Using Context," *Personal Ubiquitous Computing*, vol. 5, no. 1, 2001, pp. 4-7.
3. K. Raatikainen, H.B. Christensen, and T. Nakajima, "Application Requirements for Middleware for Mobile and Pervasive Systems," *ACM SIGMobile Mobile Computing and Comm. Rev.*, vol. 6, no. 4, 2002, pp. 16-24.
4. S. Tarkoma, ed., *Mobile Middleware: Architectures, Patterns, and Practice*, John Wiley & Sons, 2009.
5. E. Oliver, "A Survey of Platforms for Mobile Networks Research," *ACM SIGMobile Mobile Computing and Comm. Rev.*, vol. 12, no. 4, 2008, pp. 56-63.
6. K.M. Dombroviak and R. Ramnath, "A Taxonomy of Mobile and Pervasive Applications," *Proc. ACM Symp. Applied Computing* (SAC 07), ACM Press, 2007, pp. 1609-1615.
7. H. Schulzrinne and E. Wedlund, "Application-Layer Mobility Using SIP," *ACM SIGMobile Mobile Computing Comm. Rev.*, vol. 4, no. 3, 2000, pp. 47-57.
8. T. Mikkonen and A. Taivalsaari, "Creating a Mobile Web Application Platform: The Lively Kernel Experiences," *Proc. ACM Symp. Applied Computing* (SAC 09), ACM Press, 2009, pp. 177-184.
9. L. Rudolph, "A Virtualization Infrastructure that Supports Pervasive Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, 2009, pp. 8-13.
10. E. Cuervo et al., "MAUI: Making Smartphones Last Longer with Code Offload," *Proc. Int'l Symp. Mobile Systems, Applications, and Services* (MobiSys 10), ACM Press, 2010, pp. 49-62.
11. P.T. Eugster et al., "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, 2003, pp. 114-131.

Sasu Tarkoma *is a full professor in the Department of Computer Science at the University of Helsinki. His research interests include mobile computing and Internet technology. He received a PhD in computer science from the University of Helsinki. Tarkoma is affiliated with the Nokia Research Center and Aalto University. Contact him at sasu.tarkoma@cs.helsinki.fi.*

Eemil Lagerspetz *is a researcher at the Helsinki Institute for Information Technology and a PhD student at the University of Helsinki. His research interests include mobile data management and data communications. Lagerspetz received an MSc in computer science from the University of Helsinki. Contact him at eemil.lagerspetz@cs.helsinki.fi.*