Arching Over the Mobile Chasm: Platforms and Runtimes

Sasu Tarkoma and Eemil Lagerspetz

***Abstract*—Mobile computing is one of the breakthrough technologies of today with over three billion mobile phones in use. As the computing power and capabilities of the devices are rapidly improving, software has become a crucial issue in the mobile marketplace. Indeed, the current trend is towards converged communication where Web resources integrate seamlessly with mobile systems. Platforms, runtimes, and middleware play a vital role in current distributed systems. This wide technology domain consists of infrastructure and support services for applications and application developers. Middleware typically provides support for various interoperable service deployment and execution related functions. In this article, we give an update on the state-of-the-art in the area and discuss current trends and research challenges.**

***Index Terms*—mobile computing, middleware, operating systems**

I. Introduction

Mobile devices are increasingly dependent on good software and ultimately good user experience. A lot of support services are needed for developing software in this operating environment, very different from desktop computing [1], [2], [3]. These are mainly provided in the middleware layer, lower than applications, but above the operating system and basic TCP/IP protocol stack. Thus middleware provides a level of indirection and transparency for application developers who save development cost and time, when they use standardized or well-known interfaces when designing their products. For example, Web sites and mashups, industrial systems, banking systems, and stock market systems rely extensively on middleware. The development time and cost has traditionally been high for mobile applications and services, operating in a more challenging environment than a typical fixed network. The wireless and mobile environment is not very reliable, has high latency, limited bandwidth, and many different terminal types. One specific implementation is therefore not necessarily usable for all mobile equipment on the market.

We identify five key requirements for mobile computing applications and services, namely:

- Accessibility for ensuring that data is accessible by all entities.
- Reachability for ensuring that mobile devices and the service and resources that they host are reachable (and thus accessible).
- Adaptability which is needed to support good user experience and cope when the computing environment changes. Context-awareness is a key part of adaptability.
- Trustworthiness, which is needed to maintain user confidence in the services and the service ecosystem. This requires facilities for assessing and establishing trust between entities.
- Universality, which requires the use of interoperable standards based protocols and formats. This is especially needed for loosely coupled systems that are combined at runtime.

Considering these requirements, there is strong motivation to develop mobile middleware solutions capable of transparently handling the inconveniences of the operating environment while supporting adaptability for current devices [4], [5].

II. Evolution

We divide mobile applications and services [6] into four generations. This allows us to outline some of the significant trends in the mobile application landscape. Early mobile phones in the 1980s provided only the basic voice services and 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 (SMS). Of these, the role of SMS has been paramount, but in Japan the iMode has also been instrumental, combining low cost data and messaging.

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

The third generation applications and services are supported by a more sophisticated environment. They are built on top of a platform offering services such as location support, content adaptation, storage, and caching. We are now witnessing the emergence of third generation applications on platforms that include Series 60, Java ME, Android, and the iPhone. These new devices that are able to support middleware and complex applications are often known as smartphones.

If the feature set of a mobile Web browser is considerably limited due to device constraints it is called a microbrowser. Limitations might be encountered in the processing power or display capabilities of mobile devices. However, since 2006, there has been a marked increase in the number of mobile devices capable of supporting Web browsers with advanced features, such as CSS 2.1, JavaScript, and Asynchronous Javascript (AJAX). It is evident that the two separate universes of wireless telecommunications and the Internet are on converging evolutionary paths.

The fourth generation of applications has not yet arrived. Still, we can briefly sketch the expected properties of these future applications in the light of recent proposals in research and standardization communities. The fourth generation is expected to be adaptive not only in terms of application behaviour and content, but also regarding the networking stack and wireless interface. Always-on connectivity, multi-mode communications, mesh networking, and adaptive network interfaces and physical communication media will be important parts of future mobile computing devices.

The following list summarizes the evolution of mobile applications and services with approximate dates for the generations:

- *1st (1990-1999).* Text messages (SMS) and mobile data. Speeds up to tens of Kbps.
- *2nd (1999-2003).* Limited browsers, WAP, iMode, and MMS. Speeds up to 144Kbps.
- *3rd (2003-2008).* Mobile platforms, middleware services. Series 60, J2ME, Android, iPhone. Speeds up to several Mbps.
- *4th (2008-).* Adaptive services, user interfaces, and protocols. Context-awareness, always-on connectivity. Speeds up to hundreds of Mbps. Emergence of app stores. Versatile devices: smartphones, pads. Cloud-assisted applications with social networks.

III. Mobile Platforms

In this section, we consider the following mobile platforms and their middleware aspects: Android, BlackBerry, iPhone, Java 2 ME, Kindle SDK, Maemo and MeeGo, Palm WebOS, Symbian, and Windows Mobile. At the end of this section, we summarize the key features of the platforms and discuss their differences.

A. Android

Android is an operating system and software platform for mobile devices, based on the Linux OS. Android has been developed by Google and the Open Handset Alliance consisting of over thirty companies. The platform allows development of managed code using a Java-like language. The language follows the Java syntax, but does not provide the standard class libraries and APIs, instead the language utilizes libraries and APIs developed by Google.

Figure 1 presents the Android architecture. The architecture 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 used by various components of the Android system. The capabilities of these libraries are exposed to developers through the Android application framework APIs.

The Android runtime is responsible for the execution of the custom Java bytecode. The runtime includes the Core Libraries and the Dalvik Virtual Machine and on top of the libraries and the runtime, the application framework, which consists of various managers. On top of the managers reside a number of bundled applications: for example an email client, SMS program, calendar, maps, browser, and contacts, all written using the Java programming language. Developers utilize the same API that is also used by the built-in core applications. Android emphasizes component reuse and any component can publish its capabilities, which can then be utilized by other components if security constraints do not prevent this.

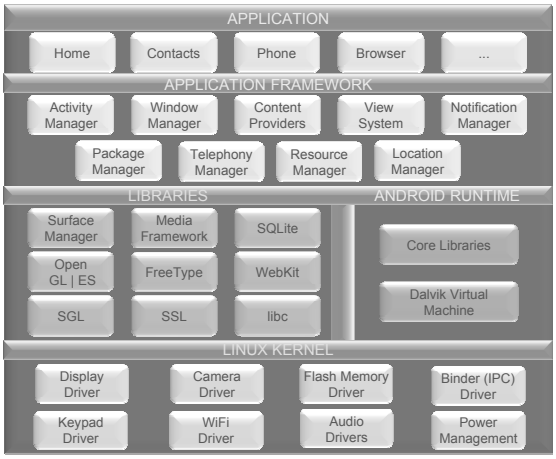

Fig. 1. The Android architecture.

B. BlackBerry

RIM's BlackBerry devices are based on a proprietary operating system. The current version (v4) supports Java MIDP 2.0 applications and synchronization with various productivity suites. The communications model is based on the enterprise servers that act as email relays. The servers utilize RIM's Network Operation Center (NOC) in order to send and receive messages to and from the mobile devices. Since proprietary NOC is used, mobile push can be implemented in an efficient manner.

C. iPhone

The iPhone OS is a mobile operating system developed by Apple Inc. for their iPhone, iPod touch, and iPad products. The OS is derived from Max 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 presents an overview of the MacOS X architecture, which has been adapted for the iPhone architecture. The iPhone system is built on an ARM processor and the Core OS (Darwin) includes the XNU kernel and system utilities. The XNU kernel includes POSIX support, networking, and file system support, as well as device drivers. Above the kernel, we have the system utilities. Above the operating system, we have the layered middleware, namely core services, application services, API layer, and finally the GUI (Aqua).

The iPhone OS is based on four abstraction layers, namely the Core OS layer, the Core Services layer, the Media layer, and the Cocoa Touch layer.

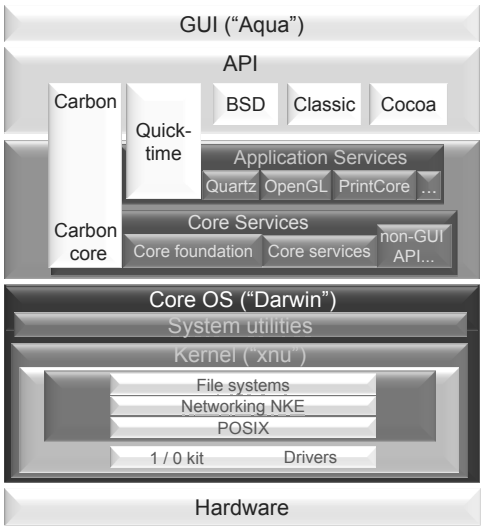

Fig. 2. The iPhone architecture.

The Core OS layer includes the OS X Kernel, TCP/IP networking stack and the Sockets interface, power management, file system, and security features. The Core Services layer includes Mac OS X application programming interfaces that are below the Media and Cocoa Touch layers. These include APIs to services such as networking, threads and Web. This layer also provides embedded SQLite database and support for geolocation. The Media layer pertains to the I/O interfaces of the device, mainly audio mixing and recording, video playback, OpenGL, and animation support. Finally, the Cocoa layer supports multi-touch events and controls, with an interface for accelerometer input and support for localization (i18n) and camera. The iPhone OS's user interface is based on multi-touch gestures such as swiping, tapping, pinching, and reverse pinching. In some applications, internal accelerometers are used to alter the screen orientation when the device is rotated on its y-axis.

Apple provides the SDK as a download without fee, but approval and payments are needed to release software for the iPhone platform in the App Store. There users are able to browse and download applications directly to iPhone, iPod touch, or iPad.

The five important APIs available for the developer are illustrated in the API layer in the diagram. They are Carbon, Quicktime, BSD/Posix, Classic and Cocoa. Carbon is a procedural API consisting of separate manager entities. Each Manager offers an API related to some functionality, defining the necessary data structures and functions. Managers are often interdependent or layered. Managers in Carbon include the file manager, resource manager, font manager, and event manager. The POSIX specifications define crucial operating system software interfaces and also a standard threading library API. The POSIX standard has three important parts, Kernel APIs, Commands and Utilities, and Conformance Testing. The kernel APIs include real-time services, threads, security interface, network file access, and network process-to-process communications.

Classic Environment is a backwards compatible hardware and software abstraction layer, no longer supported in the current Mac OS version. Cocoa Touch provides an abstraction layer of the iPhone OS, based on the Cocoa which is the native object-oriented application program environment for the Mac OS X. Cocoa's design follows Model-View-Control principles and its frameworks are written in Objective-C.

The iPhone OS 4.0 was announced in April 2010 and it supports multitasking for 3rd party applications. The key design principle is to offer APIs for specific background operations in order to be able 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. Third party push servers are supported for sending notifications to applications.

The recently announced iPad device is based on the iPhone OS and thus applications are developed using the iPhone SDK. The SDK supports the development of three types of applications, namely iPhone, iPad, and universal applications. An universal application determines the device type and then uses the available features based on conditional statements.

D. Java ME

Java ME is a portable solution for creating various mobile applications downloadable to mobile devices. Java Platform, Micro Edition (Java ME or previously J2ME) specifies a standardized collection of Java APIs for the development of software for small and resource-constrained devices. Target devices include consumer devices, home appliances, security, defense, automotive, industrial, industrial control, and multimedia. From 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, namely Connected Device Configuration (CDC) and Connected Limited Device Configuration (CLDC). The former is for high end PDAs and the latter is intended for mobile phones and other small devices. The configurations are then augmented by profiles, which define additional APIs for applications. The most common profile is the Mobile Information Device Profile (MIDP) aimed at mobile phones. Another well-known profile is the Personal Profile, aimed at consumer products and embedded devices.

The Java ME platform's Mobile Service Architecture (MSA) specification (JSR 248) defines a standard set of application functionality for mobile devices, covering also 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 the Connected Device Configuration.

To summarize, the Java ME is evolving into a versatile platform for mobile application development. The introduction of various JSRs and the MSA version 2 have gradually removed the early restrictions with MIDP applications and

won growing vendors support for the newer specifications. Moreover, software portability challenges between CLDC and CDC are being addressed in MIDP version 3.

For mobile computing, Mobile Sensor API, Contactless Communication API, and Location API support applications that are aware of their surroundings and context and compatible with adaptable and tailored content. For instance, Mobile Broadcast Service API supports the delivery of streaming multimedia to mobile phones. Converged communications support is provided by the XML API and *IP Multimedia Subsystem (IMS)* Services API [7]. Usability is a crucial selling point in mobile devices, as amply demonstrated by the iPhone. Usability themes are addressed by the Mobile User Interface Customization API and Scalable 2D Vector Graphics API.

E. Kindle SDK

Amazon has recently announced a Kindle SDK that can be used to develop Java-based active applications for the Kindle e-book readers. The Kindle SDK is based on the Java ME Personal Basis profile and Kindle-specific extensions. The APIs support basic UI, networking, and limited secure storage on the device[1].

F. Maemo and MeeGo

The Maemo platform from Nokia includes the Internet Tablet OS, which is based on Debian GNU/Linux and draws much of its GUI, frameworks, and libraries from the GNOME project. Maemo uses the Matchbox window manager, and like Ubuntu Mobile, it 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 may have a small keyboard, and central characteristics include a stylus and a touch-sensitive screen. Graphical interfaces must be designed with the touch screen in mind.

The Maemo SDK features a sandboxed development environment on a GNU/Linux desktop system largely built using a tool called Scratchbox. This environment behaves in a similar manner than the actual OS on the Nokia Internet Tablet devices. Currently C is the only official programming language for Maemo. The Maemo user interface architecture is based on the GNOME framework, especially the GTK+ widget set.

For hardware abstraction, Maemo provides the Hardware Abstraction Layer (HAL) with a shared library that has an API for device objects. HAL is capable of loading the right device driver, when a new device is detected, creating and maintaining /dev files, and tracking the status of devices. The Maemo platform includes the normal networking protocols, such as TCP/IP stack, OpenSSL library for network security, and and libcurl that provides HTTP access for applications. The D-BUS communication system is used as the primary channel between applications. Maemo also includes an SQL database, SQLite, that can be used to store user application data without centralized server process to connect into. The applications are built on top of the Hildon framework.

The latest development combines Nokia's Maemo platform with Intel's Moblin. The new combined system is called MeeGo[2]. Both Maemo and Moblin applications have been developed mainly with the GTK framework. Now this will be replaced by the Nokia's Qt framework. MeeGo is expected to run on both Atom and ARM processors and to support both netbooks and mobiles phones. MeeGo applications are written in C++ using the MeeGo SDK that includes Qt.

Figure 3 presents an overview of the MeeGo architecture. The architecture follows the typical design of having a hardware abstraction layer (HAL), OS base (Linux kernel, X), middleware, and then user experience (UX) related functions. The lowest HAL layer is provided by the device vendor and it includes kernel drivers and patches, kernel configuration, modem support, and other software related to the underlying hardware. The 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. Qt provides the basic APIs for GUIs, database, XML, networking, and a WebKit-based Web runtime. The Qt platform is available for a number of systems including Windows, MacOS, Linux, Symbian, and Windows CE.

The Qt API is implemented in C++ and most developers use C++. At the moment, C++ is the only language that can be used 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, calendar, alarms, sensors, etc.

G. Palm WebOS

The Palm's WebOS is a mobile OS running on the Linux kernel. The runtime system includes a WebKit- based browser, and applications are written using the JavaScript language. The WebOS follows the cloud-based services model, in which clients interact and synchronize directly with cloud-based services. Applications use a JavaScript-based framework, called Mojo, which provides common functions pertaining to UI, widgets, and data access. A typical WebOS application uses HTML 5 for presentation and audio/video. The applications are modelled using the Model-View-Controller architectural pattern in order to separate concerns.

The Palm WebOS is based on scenes that are pushed and popped into a scene stack. The top scene in the stack is visible to the user. The scenes are activated and deactivated by the execution framework. The scenes and applications use asynchronous notifications (W3C DOM events) to signal changes.

H. Symbian and Series 60

The Symbian OS is an open mobile operating system developed by Symbian Ltd. for ARM processors. The system includes a microkernel OS, associated libaries, user interface,

[1]http://kdk-javadocs.s3.amazonaws.com/index.html

[2]www.MeeGo.com

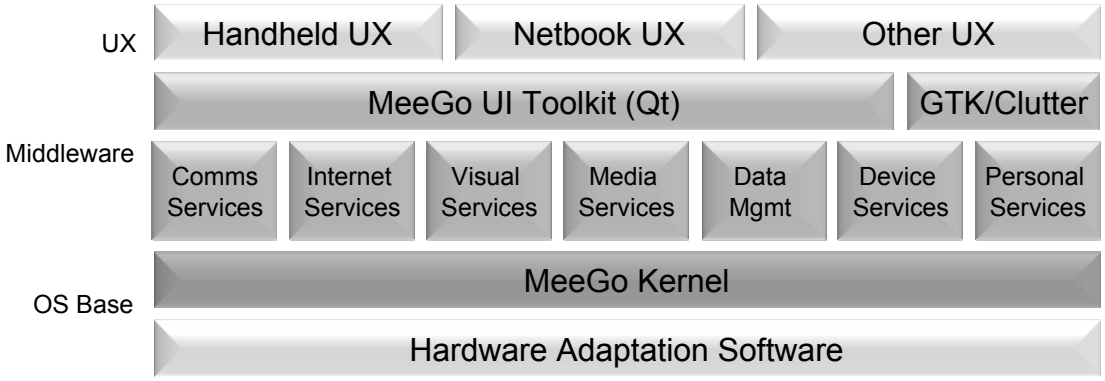

Fig. 3. The MeeGo architecture.

and 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. The choice of servers, a microkernel design, and event passing were motivated by three design goals, namely minimizing response times to users, maximizing integrity and security, and utilizing scarce resources efficiently.

Nokia has acquired the ownership of Symbian (2008) and established Symbian Foundation in order to provide royalty-free software for the mobile environment. In fact, the Symbian OS and S60, UIQ, and MOAP were made open source in 2010.

The Symbian OS has a microkernel architecture, which includes a scheduler, memory management, and device drivers. Other services, like networking, telephony, or filesystem support are placed in the OS Services Layer or Base Services.

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, Store, Central Repository, DBMS, and cryptographic services. The Base Services layer is responsible for basic connectivity and serial communications as well as telephony. The communications infrastructure has been developed on this layer and two prominent networking stacks are the TCP/IP and WAP stacks. The Web and WAP browsers are available for the respective protocol stacks.

The Symbian Web runtime is based on the WebKit system illustrated by Figure 4. The Java runtime and JavaPhone are available for applications. The figure does not include SyncML support, also a core feature.

The native language of the Symbian OS is C++, but the language is not compatible with ANSI C++. The OS and applications are based on the Model- View-Control (MVC) design pattern, which support the separation of different functions. All Symbian applications are built up from three classes that create the fundamental application behavior: an application class, a document class, and an application user interface class. The remaining required functions, the application view, data model, and data interface, are created independently and interact solely through their APIs with the other classes.

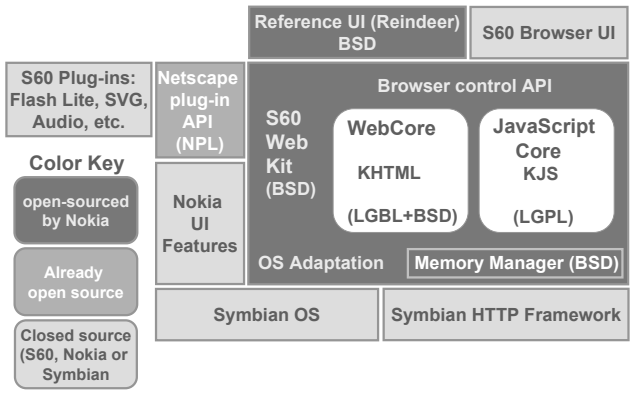

Fig. 4. S60 WebKit.

Symbian OS emphasizes resource recovery using several programming features, such as a cleanup stack and descriptors. The event-based nature of the OS allows the minimization of thread switching using a technique called active objects.

In addition to C++ native applications, Widgets are supported through the Nokia Web Runtime Widgets (WRT). The WRT environments follows the W3C Widgets specification and allows the installation and execution of widgets. The widgets can access device-specific features using a JavaScript API called Platform Services 2.0.

I. Windows Mobile and .NET Compact Framework

Windows Mobile 6 was released by Microsoft at the 3GSM World Congress 2007 and it comes in three flavours: 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 has been designed to integrate with Windows Live and Exchange products. Software development for the platform is typically done using Visual C++ or .NET Compact Framework. When native client-side functionality is not needed, server-side code can be used that is deployed on a

mobile browser, such as the Internet Explorer Mobile bundled with Windows Mobile.

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

The .NET Compact Framework is a subset of the .NET Framework and shares many components with the desktop software development environment. The framework includes an optimized Common Language Runtime (CLR) and a subset of the .NET Framework class library. The advantages of a managed environment such as CLR include creation of more trustworthy and platform independent software and the expectation is that managed components developed using .NET languages, such as C#, are used to create the applications. The disadvantage of managed code is the performance penalty in real-time environments. Garbage collection and Just in Time (JIT) compilation can introduce unexpected delays to program execution and cause stochastic behaviour. However, it is also possible to use the Win32 API with .NET CF, which allows direct access OS features.

J. Summary

Figure 5 presents an overview of the different mobile platforms and their properties. The first row titled "Development" indicates the development languages and the tools available for the platform. We observe that C, C++ and Java are currently the dominant programming languages for mobile devices. The second and third rows pertain to network scanning and interface control. These functions are interesting when an application needs to monitor and control the wireless communications.

"Background processing" denotes the multitasking capabilities of the platform. Energy and power monitoring and control is an important aspect of mobile platforms. Both multitasking and energy management features vary from system to system. Memory management and persistent storage are well supported across the platforms, as well as location information.

HTML 5[3] is the next version of HTML and is currently in development with the first public working draft of the specification available in January 2008 and completion expected around 2012. Browser vendors, including mobile browser vendors, are already implementing some HTML 5 features as they are defined. HTML 5 has some features that can significantly improve current mobile Web applications including network Web Socket API, advanced forms, offline application API, and client-side persistent storage (key/value and SQL). HTML 5 support divides the platforms. The iPhone platform has a very good support for HTML 5. The Session Initiation Protocol [8] is a key signalling protocol is 3G and 4G wireless access networks for session management. Some platforms expose a SIP API to developers.

Three of the surveyed platforms are fully open source, namely Android, Maemo/MeeGo, and Symbian OS. The platforms have varying systems for supporting 3rd party application installation and execution. Execution of privileged system functions requires certification or other means means of obtaining permission. The newer platforms are less fragmented whereas older systems are invariably fragmented.

[3]http://dev.w3.org/html5/spec/Overview.html

Platform	Browser	Rendering engine	Flash	Widget engine	Comments
Android	Android	WebKit	Yes (for Android 2.1)	No	
Apple iPhone	Apple iPhone Safari	WebKit	No	Yes	
BlackBerry	BlackBerry	Mango	No	Yes	Proprietary push technology
Opera Mobile	Symbian S60, Windows Mobile	Presto	Yes	Yes	Opera Mini supports other browser platforms using a proprietary proxy
Symbian S60	Symbian S60	WebKit	Yes	Yes	
Windows Mobile	Internet Explorer	Trident/MSHTML	Yes	No	Silverlight support planned

Fig. 6. Overview of smart phone browsers.

Figure 6 presents an overview of current smart phone Web runtimes and browsers, namely Android, iPhone, BlackBerry, Opera Mobile, Symbian, and Windows Mobile. MeeGo and PalmOS are based on WebKit. WebKit has support for HTML 5. Some key features of HTML 5 include:

- Better page structuring through new elements (e.g., section, header, footer, article, nav and dialog).
- A canvas element with 2D drawing API for dynamic graphics and animation.
- Direction provision for audio and video content.
- Client-side persistent storage (key/value and SQL).
- Offline application APIs.
- Editing and drag-and-drop APIs.
- Network Web Socket API.
- Cross-document messaging.

IV. Current State

The current platform landscape is heterogeneous and several different operating systems, programming languages, and interfaces are used, resulting in complex mobile software development and testing processes. A mobile platform needs to 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, Java ME MIDP, iPhone, and Android APIs cannot be extended easily by a third-party developer, meaning that it is easier to extend and modify functionality at the server-side instead of modifying the client.

For high-end mobile phones, Java ME MIDP with its primitive and limited set of data structures and no access to the file system is quite restricting for programmers. A more powerful Java ME configuration and profile can be used but these are not currently well supported on phones. Full Java 2 API and an easy way to access platform functions is the best option for Java developers. A MIDP developer can circumvent some of the limitations by using open source class libraries that provide the required data structures.

	Android Linux	Blackberry OS 5.0	iPhone OS (iPhone, iPod touch, iPad)	Java ME MIDP	Kindle SDK	MeeGo Linux	Palm WebOS Linux	Symbian Series 60	Windows Mobile .NET and Windows Phone 7
Development	Java, native code with JNI and C/C++	Java MIDP, Blackberry APIs	Objective-C	Java ME	Java, Personal Basis Profile	C/C++, Qt APIs, various	Applications with Web tech. (HTML 5), C/C++	C++, Qt, Python, various	C/C++, .NET, various
Network scanning	Yes	Yes (hotspot API)	No	No	No	Yes	Limited (Web apps)	Limited	Yes
Network interface control	Limited	Limited (hotspot API)	No	No	No	Yes	Limited (Web apps)	Yes	Yes
Background processing	Yes (services)	Yes	No (Yes for 4.0)	Yes (multi-tasking support in MIDP 3.0)	No	Yes	Yes	Yes	Yes, not supported for third party applications in WP7
Energy and power monitoring and control	Yes	Limited (battery info)	Monitoring since 3.0	No	No	Yes	Yes (battery status, inform duration of activity)	Yes	Yes
Memory management	Yes	Yes (low-memory events)	Yes	Limited	Limited	Yes	Yes (no for Web apps)	Yes	Yes
Persistent storage	Yes	Yes	Yes	Limited, exension	Limited secure storage	Yes	Yes (HTML 5 storage)	Yes	Yes
Location information	Yes	Yes	Yes	Extension	No	Yes	Yes	Yes	Yes
HTML 5	Yes, support depends on version	Yes, support depends on version	Yes	N/A	N/A	Depends on WebKit version	Yes	No (Widgets and Javascript API)	No
SIP API support	Limited	No	Limited	Extension	No	Yes	No	Yes	Limited
Open Source	Yes	No	No	No	No	Yes	No (some parts are Open Source)	Yes	No
3rd party application installation	Certificate, Android store	Certificate	Certificate, Apple AppStore	Certificate	Kindle DRM	Certificate	Certificate	Certificate	Certificate, app store (WP7)
Level of fragmentation	Some fragmentation	Minor fragmentation	Minor fragmentation	Fragmented	Not fragmented	Not fragmented	Not fragmented	Some fragmentation	Some fragmentation

Fig. 5. Overview of smart phone systems.

The convergence of mobile and traditional IT fields has caused the increasing popularity of Web technologies in the development and deployment of mobile applications. It is not particularly difficult to implement scalable Web applications, and this has led to a prospering service ecosystem exemplified by the many massively popular Web sites. Many applications can be implemented on the basis of current Web technologies, such as AJAX, REST, OpenID, OAuth, and other solutions. However, the Web protocols do not directly work well with mobile and wireless links and thus using current Web technologies only supports applications that are natural to realize using Web's request/reply interactions style and with minimal susceptibility to variations in latency. Indeed, asynchronous operation would be particularly useful in mobile applications that need to react to changes in the environment.

Web technologies and the TCP/IP protocol stack were not designed for wireless environments meaning that the performance cannot be optimal. Therefore current standardization and development efforts include for example making XML more suitable for mobile devices. Typically access to underlying system services is not possible. This has been addressed in some systems by providing specific APIs that expose some core functionality to applications. For example, Symbian and iPhone give programmers the possibility to access low level functions if C++ or Objective-C are used, although this may require certification.

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, on servers, using proxies and gateways, and through collaboration of the different entities. Ideally, services and software running on devices collaborate to realize a service for users. Also context-awareness brings forth several new challenges: context acquisition, privacy, and software testing and quality assurance. Testing adaptive and context-aware behavior requires new kinds of solutions and methods for assuring that software is working properly and generates desired user experience. Unfortunately, universal device and service discovery is still not available for developers. The current trend is to strive towards adaptive applications by both utilizing Web technology and platform

specific APIs.

As an example, we can take any application that needs to push information to consumers, such as email or RSS. Current technological constraints require that push-based applications are implemented using polling — not a very efficient way of doing push. The same constraints that make push difficult also create difficulties for VoIP calls and other media flows. The SIP protocol architecture [9] and related *Network Address Translation (NAT)* traversal specifications, solve most of these reachability issues; however, the resulting system is very complex. Although middleware and protocol stack extensions take care of the complexity for the developer, understanding of the environment and the protocols that are used is still vital for the system designer, developer, and tester.

V. Towards Common APIs

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

The base platforms can be viewed to offer the basic set of development APIs, such as those for Android, iPhone, MeeGo, Qt, Symbian, and Windows Mobile. Additional APIs are necessary to be able to support easy and cross-platform service creation.

One key aspect is the development language and environment. Web application development is largely done with JavaScript. Recent experimental results suggest that JavaScript-based application platforms can be executed on Web browsers; however, there are still a number of practical challenges pertaining to performance and browser limitations [10]. One of the key challenges is to allow a Web-based application to access local system variables, such as context variables. The need for security and privacy are paramount.

The PhoneGap is an open source development framework for building cross-platform mobile applications. The aim is to allow developers to build applications with HTML and JavaScript, and still allow them to utilize device-specific features in Android, iPhone, Symbian, and other devices[4].

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

The GSM Association has an initiative to define a commonly supported API for mobile operators to expose network information to Web application developers called OneAPI. These APIs will use both RESTful and Web Services interfaces. The first APIs to be implemented will be for messaging and location functions.

The Open Mobile Terminal Platform (OMTP) group defines requirements and specifications that aim towards simpler and more interoperable mobile APIs. BONDI is a browser and widget specification developed by OTMP in collaboration with W3C's WebApps working group. BONDI defines requirements governing Device Capability access by JavaScript APIs to promote interoperability and security of implementations[5]. The recent 1.1 release of BONDI is compliant with the W3C Widgets: Packaging and Compliance specification.

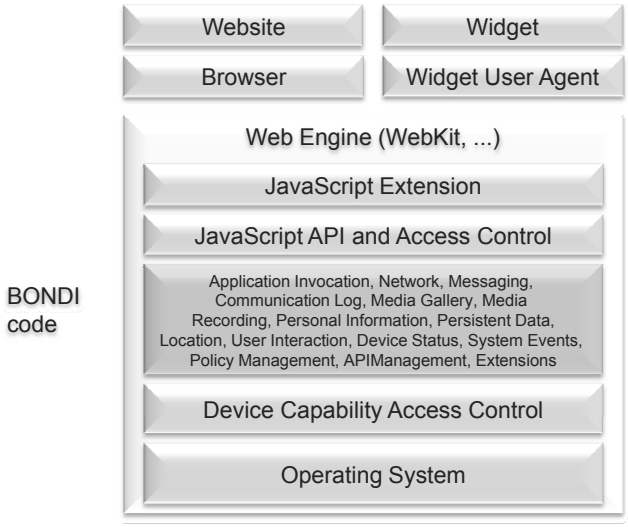

Fig. 7. The BONDI architecture.

The Open Mobile Alliance (OMA), a key mobile standardization organization, bases its browsing specifications on Internet technology, but limits profiles for constrained resources and user interfaces of mobile devices. For instance, it assumes reduced memory, processing power, bandwidth, and user-input methods. It defines application-level protocols, semantics, syntax, content formats, user-agent behavior and use of hypermedia transfer protocols.

The elements of OMA Browsing include:

- WAP Architecture (with or without proxy)
- XHTML Mobile Profile
- ECMAScipt Mobile Profile (including Document Object Model), providing for local application scripting capability (akin to JavaScript)
- Wireless Cascading Style Sheets (CSS)
- Binary XML (efficient communications)
- OMA Push (asynchronous server-initiated content delivery)
- XMLHttpRequest (forthcoming in OMA Mobile Browsing 2.4, an important Ajax method)

VI. Challenges

Mobile computing and software development faces a number of key challenges. Based on the survey in this article, we identify fragmentation as a key challenge. Fragmentation can happen on multiple layers and dimensions, for example on the operating system, platform and middleware, and service API layers. The current state of mobile software development is

[4]http://www.phonegap.com/

[5]http://bondi.omtp.org/default.aspx

that there are a number of significant operating systems and platforms that need to be supported and that these platforms have differing programming conventions, interfaces, and software distribution solutions. This raises the cost of software development and slows down the software ecosystem.

In addition to fragmentation, the nature of the APIs and the features that they expose of the underlying platform are widely differing. Most systems expose certain underlying system features, some of which require 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 Palm's WebOS W3C Events. One particular trend is to utilize URI-based conventions for naming system resources and services. This is extensively used in the Nokia Platform Services, WebOS and the BONDI architecture. An alternative albeit more radical solution to fragmentation would be to use virtualization to execute the whole mobile application software stack [11].

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. The battery capacity increases approximately 10% annually while the requirements increase in a more rapid pace. Especially Internet and Web 2.0 service usage consume vast amounts of energy and result in short battery lifetimes, and ultimately poor user experiences. Current research challenges include how to support energy accounting [12] and execute applications across mobile devices and cloud-based systems [13].

VII. Conclusions

A considerable amount of R&D has gone into solutions for different kinds of mobile and pervasive environments that support a wide variety of different applications. However, the solution landscape is still fragmented. The next step would be to support access to context information and enable more intelligent information processing on client devices. This is needed to fully realize the visions of pervasive and ubiquitous computing. On the other hand, the current distributed service trend is to build large computing clusters, server farms and datacenters, and then connect these with end users.

The separate development lines of mobile devices and datacenters should be parallel and working together as information processing can be distributed between mobile devices and datacenters. This evidently requires sophisticated rules for distributing application functionality and mechanisms for ensuring that private information is not leaked. The possibility for distributing computation opens up new ways for improving user experience and increasing mobile device remaining operating time.

Given that there are over three billion mobile devices on the market today and the projections indicate that the number will approach five billion in the near future, the prospects for mobile applications, service, and middleware appear to be very promising. To be able to handle such a large amount of users with possibly widely differing device types and characteristics necessitates interoperable and high performance platforms, and highly scalable and available fixed infrastructure.

One step towards extensibility and universality would be to employ a common and interoperable message bus that supports component discovery, capability negotiation, and communications. Both message passing, publish/subscribe [14], and tuple spaces have been proposed as the 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, also for enabling intra-device communications.

The iPhone OS is pioneering the mobile usage of HTML 5 and it remains to be seen how fast other mobile platforms adopt this new specification. HTML 5 in combination with custom JavaScript APIs would open a world of new possibilities for the development of portable and cloud-assisted mobile software.

Another approach would be to utilize virtualization techniques to support multiple operating systems and platforms on the same hardware, possibly at the same time. Virtualization can also be used to enhance security of a system. This is a future technology still maturing for mobile devices [11].

In this article we have identified the following key requirements for mobile computing: accessibility, reachability, adaptability, trustworthiness and universality. We maintain that mobile middleware plays a crucial role in ensuring that these requirements can be met. Of course, the middleware cannot meet these requirements alone, but needs support from the operating system and network protocol stack.

References

[1] M. Weiser, "Ubiquitous computing," *Computer*, vol. 26, no. 10, pp. 71–72, 1993.

[2] A. K. Dey, "Understanding and using context," *Personal Ubiquitous Comput.*, vol. 5, no. 1, pp. 4–7, 2001.

[3] K. Raatikainen, H. B. Christensen, and T. Nakajima, "Application requirements for middleware for mobile and pervasive systems," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 16–24, oct 2002.

[4] S. Tarkoma, Ed., *Mobile Middleware — Architectures, Patterns, and Practice*. Wiley, 2009.

[5] E. Oliver, "A survey of platforms for mobile networks research," *SIGMOBILE Mobile Computing and Communications Review*, vol. 12, no. 4, pp. 56–63, 2008.

[6] K. M. Dombroviak and R. Ramnath, "A taxonomy of mobile and pervasive applications," in *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2007, pp. 1609–1615.

[7] A. Cuevas, J. I. Moreno, P. VIdales, and H. Einsiedler, "The IMS Service Platform: A Solution for Next Generation Network Operators to Be More Than Bit Pipes," *IEEE Communications Magazine*, August 2006.

[8] H. Schulzrinne and E. Wedlund, "Application-layer mobility using SIP," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 4, no. 3, pp. 47–57, 2000.

[9] S. Berger, H. Schulzrinne, S. Sidiroglou, and X. Wu, "Ubiquitous computing using SIP," in *Proceedings of the 13th International workshop on Network and operating systems support for digital audio and video*, June 2003, pp. 82–89, http://www.cs.columbia.edu/IRT/papers/Berg0306_Ubiquitous.pdf.

[10] T. Mikkonen and A. Taivalsaari, "Creating a mobile web application platform: the lively kernel experiences," in *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*. New York, NY, USA: ACM, 2009, pp. 177–184.

[11] L. Rudolph, "A virtualization infrastructure that supports pervasive computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 8–13, 2009.
[12] S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich, "Apprehending joule thieves with cinder," in *MobiHeld '09: Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*. New York, NY, USA: ACM, 2009, pp. 49–54.
[13] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code offload," in *Proc. ACM Mobisys*, June 2010.
[14] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.