Write Once, Run Anywhere – A Survey of Mobile Runtime Environments

Sören Blom, Matthias Book, Volker Gruhn, Ruslan Hrushchak, André Köhler
Applied Telematics/e-Business Group, Dept. of Computer Science, University of Leipzig
Klostergasse 3, 04109 Leipzig, Germany
{blom, book, gruhn, hrushchak, koehler}@ebus.informatik.uni-leipzig.de

Abstract

The hype surrounding Web 2.0 and technologies such as AJAX shows: The future of distributed application development lies in Rich Internet Applications (RIAs), which are based on highly distributed components and characterized by the intensive use of communication networks, complex interaction patterns and advanced GUI capabilities. As service providers begin to tap into the mobile market by extending the reach of their established e-commerce systems to mobile devices, a core challenge is the choice of a runtime environment and middleware that adapts well to the existing architecture, yet is a safe investment for the years to come. This paper surveys the current state and the future of runtime environments suitable for developing RIAs for mobile clients.

1. Introduction

"Write once, run anywhere" – this slogan was introduced with the Java programming language ten years ago, the idea being to implement an application only once and then be able to run it on any platform, independently of its operating system. From today's perspective, Java was a great success. However, application infrastructures and runtime environments have changed significantly since then.

Current developments as well as recent experiences with mobile applications lead us to the assumption that the future of application development lies in Rich Internet Applications (RIA). The term "Internet application" emphasizes the fact that future applications will be based on highly distributed components and are characterized by the intensive use of communication networks. Software will be mashed up with third-party components and services at runtime, creating completely new applications ad hoc. The term "rich" refers to the fact that applications will support complex interaction patterns and advanced graphical user interface (GUI) capabilities.

Furthermore, application usage scenarios are also changing radically. Especially e-commerce applications on mobile devices will become increasingly important. But mobility causes some major concerns for application developers: a wide variety of devices to support, unstable network connections, limited bandwidth, high latency, security issues and many more challenges need to be considered.

Thus, manufacturers and resellers of mobile devices must decide how strongly the application developer should be supported or constrained by the software pre-installed on the devices. They must define which operating system and additional runtime environment components should be provided, and how much influence the application developer should have on this configuration. On the other hand, application developers must choose a suitable technology for implementing their application that is compatible with target devices, highly flexible, and supports the development of state-of-the-art GUIs.

Considering this, an interesting question is which runtime environments will be available in the future for running mobile software clients, supporting the "write once, run anywhere" paradigm, providing state-of-the-art GUIs and interaction capabilities, as well as solving or masking known mobility problems (distribution, connectivity, performance, reliability, and many more). In order to support this decision process, this paper provides an overview of current and future technologies positioned to address these issues. In the following section, we identify different distribution patterns supported by different runtime environments, and then discuss various runtime environment technologies in detail (Sect. 2). After summing up our findings, we discuss middleware addressing typical challenges found in the development of mobile applications (Sect. 3). We conclude with an evaluation of the presented technologies, and an outlook on their future use (Sect. 4).

2. Mobile Runtime Environment Classification

For the purpose of the following discussion, we define a runtime environment as the set of components that need to be present on a client in order to run applications im-

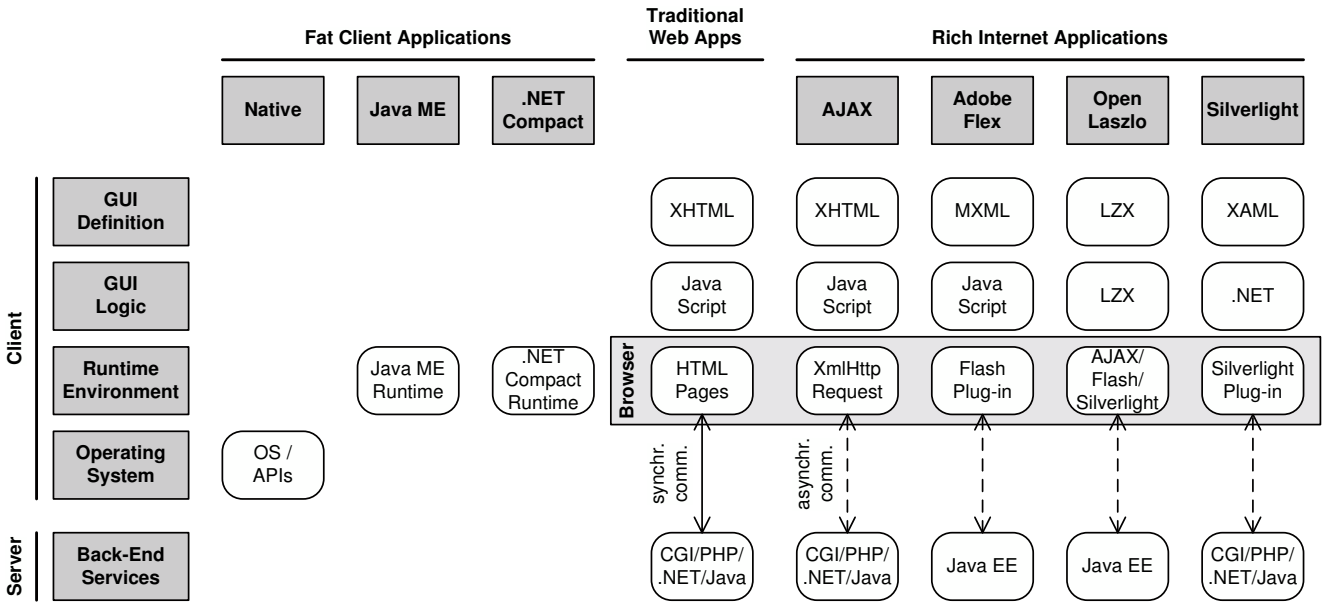

Figure 1. Mobile Runtime Environment Classification

plemented using a certain technology. Figure 1 presents a classification of the major players in the field of mobile runtime environments. As the figure shows, we can distinguish several classes of applications by the kind of runtime environment they require. We will give a short overview of these classes here, and then present the individual technologies in the subsequent sections.

Fat client applications are applications that are first downloaded and installed on the client device. The application then runs locally on the device. The client (i.e. the mobile device) is capable of some processing and potentially even limited persistent storage (caching). Installed applications may be implemented directly on the device's operating system as native applications, or on a generic runtime environment such as Java ME or .NET Compact. From a technical point of view, they are bound to a certain technology and put a fixed memory footprint on the client, but do not need a permanent network connection.

In contrast, web-based applications do not require any (or only lightweight) downloads, but can be accessed just like a web site. They form a special class of web applications insofar as they need to take into account the limitations that are unique to mobile devices, such as small screen size.

Traditional web applications communicate with the server using the familiar request-response interaction paradigm of the hypertext transfer protocol (HTTP) and the display features supported by the hypertext markup language (HTML). Recently, these technologies have been adapted with the aim to provide a richer, more responsive user experience that is similar to desktop-based applications. These Rich Internet Applications (RIAs) are based on technologies that provide (at least simulated) asynchronous client/server communication, improved usability and portability. They are characterized by the following aspects [1]:

1. RIAs are executed within a particular runtime environment, i.e. a software infrastructure required for the execution of the application that may offer base services such as GUI rendering, security, data and communications management. Typically, the runtime environment must be installed as a browser plug-in on the client.

2. The layout of user interfaces in RIAs is described using declarative languages, while the functionality of the interface (i.e. the interaction with the user) is implemented in a scripting language.

3. The RIA's business logic and back-end services are typically implemented remotely on a central server. This way, clients only need to handle immediate user interaction, but can refer to the server for all heavyweight processing and data management.

In the following subsections, we will discuss the most prevalent technologies in more detail.

2.1. Native Applications

Technically, the most straightforward way of developing a mobile application is by implementing it directly on the basis of the device's operating system (OS). Major products in this area are PalmOS, which still has a significant market share in the PDA market; Windows CE, which is targeted to

a number of embedded device classes as diverse as PDAs, automotive terminals and smart phones; and Symbian OS, which has had a strong focus on telephony features from its conception and comes in different "series" targeting different handset classes. The LiMo Foundation, a consortium of device manufacturers and network operators, has recently begun efforts to deploy Linux-based operating systems on mobile phones [7]. Similar plans are being pursued since quite some time by the Mobile Linux Initiative (MLI) of the Linux Foundation (formerly a working group of the OSDL consortium) [12].

Native applications are executable without the need for a particular runtime environment, and potentially run faster as they can directly access native features of the device through the application programming interface (API) of the operating system, but may also pose security risks for both the device and the application.

More central to the aim of our discussion, however, is the question of the "write once, run anywhere" potential of native applications. Since they need to be compiled for a particular OS, they are typically not as portable as applications built for a more generic runtime environment. In particular, different mobile OS may use different paradigms to achieve similar tasks such as display control, file management, multi-threading etc., some of which may even be entirely unsupported on other devices. However, more recent OS developments such as Android (an open platform for mobile devices forming a complete software stack including an operating sysstem, middleware and key applications [3]) may ameliorate this issue somewhat, as they specifically aim to provide the flexibility for making applications available across a wide spectrum of devices.

2.2. Java ME / .NET Compact

Sun Microsystems' Java platform has long been the prime example for "write once, run anywhere" runtime environments. The Java Platform, Micro Edition (Java ME) is a runtime environment tailored to the special features and restrictions of mobile devices. Microsoft's .NET Compact framework, a subset of the full .NET framework also geared specifically to mobile devices, has a very similar aim. Since both solutions are conceptually related, the following discussion can be generalized for either framework.

Due to the power and memory limitations of mobile devices, Java ME provides only the essential libraries and virtual machine capabilities [15, 16]. Applications developed using this technology are deployed as so-called MIDlets, which require (like all Java-based programs) a Java Virtual Machine (JVM, or just VM) as their runtime environment.

While Java ME benefits from its relationship to the desktop and server editions of the Java platform and thus technically allows the reuse of code fragments and developer skills, its portability is characterized by the challenge of striking a balance between catering to the lowest common denominator of all mobile devices, or tapping the potential of different device classes' features. This is not necessarily a fault of the VM, but a consequence of the diverse range of capabilities (such as display resolutions) offered by devices. As such, developers may find it difficult to realize the "write once, run anywhere" ideal in practice [17].

On the desktop and server platforms, Java has been notorious for comparably slow execution speed. On mobile platforms, where a highly responsive user interface is especially important, there are ongoing attempts to boost performance by compiling Java bytecode for execution in hardware (a survey of approaches can be found in [20]). Most recent is Sun's JavaFX Mobile platform [6], a Java-based operation and application environment based on a Linux OS that is explicitly positioned for the development of RIAs.

In contrast to most other runtime environments discussed in this paper, Java ME has the benefit of being independent of a permanent network connection, as all necessary logic can be deployed on the client device. As such, it is well-suited for rather isolated, compact applications whose purpose does not hinge on a permanent network connection. However, for applications that rely heavily on remote resources or content, and thus require a permanent network connection by virtue of their application domain anyway, Java ME cannot keep up with the flexibility and small footprint that the runtime environments for RIAs offer. As the continuing success of the Web 2.0 paradigm shows, tightly network-integrated applications are quickly becoming the norm. It is therefore to be expected that RIA technologies will become more suitable for the client, although Java technology will certainly remain a strong player on the servers that power network-intensive applications.

2.3. AJAX

Since a few years, interest in a new class of dynamic web applications has surged, largely fueled by the Web 2.0 hype: By using a combination of established web-based technologies, developers are creating applications with a look and feel that resembles desktop-based interaction patterns much more than the usual page-centric paradigm. Collectively termed AJAX (Asynchronous JavaScript and XML) [19], these technologies effectively hide the request-response communication taking place between client and server, and instead present users with a self-contained, highly responsive interface that allows them to manipulate and interact with the content directly.

To achieve this, AJAX applications rely on a number of prerequisites. To circumvent the interactive limitations of HTTP, AJAX applications employ fair amounts of JavaScript to control the behavior of the user interface and

implement minor application tasks. This presentation logic communicates with the server by exchanging HTTP requests that are extended over long periods of time to achieve the illusion of asynchronous communication.

The technologies that jointly constitute what may be called the "runtime environment" for AJAX applications are meanwhile incorporated in all major web browsers. Experiences with running AJAX applications on mobile phones are still quite limited – while technically feasible on sufficiently equipped smart phones, it may be a challenge to carry the usability of AJAX applications (which tend to make excessive use of the direct manipulation paradigm) over to mobile phones with their limited display and input capabilities. On the other hand, the iPhone's multi-touch display is a prime example of the usability potential inherent in direct manipulation interfaces – if device manufacturers are willing to give up the traditional ten-button/two-softkey input paradigm.

Regarding future developments, it seems likely that AJAX will continue to enjoy high popularity among developers and end-users. Its major advantage over other approaches is the dependency on just a very basic set of technologies that are built into any modern web browser, so it requires zero installation. Therefore, it comes closest to the "write once, run anywhere" vision – although it must be noted that the implementations of different web browsers contain enough subtle differences to make the development of complex portable AJAX applications a non-trivial challenge. The independence from a dedicated runtime environment (other than the standard web browser) makes AJAX especially attractive for mobile devices, as it does not introduce a persistent memory footprint of its own, and it can be executed in a controlled environment that is unlikely to expose the handset or network to serious threats.

2.4. Adobe Flash / Flash Lite

Adobe Flash (formerly Macromedia Flash) is a popular proprietary technology for the development of multimedia content that has been around for a long time by now. Flash content can be authored using the commercial development tool Adobe Flash CS 3, which includes Adobe Flash Video Encoder for the production of highly compressed video clips that have recently become very popular on the web. The contents and GUI description of a Flash application are stored in SWF format, a vector-based graphics and animation format. The application and presentation logic is implemented in ActionScript [2]. The runtime environment for Flash applications is the Flash Player, which is executed within a web browser. Through its combination of media content with programming logic, Flash is especially suited for the multi-medial representation of complex processes. With the help of ActionScript, it is possible to implement online games, multimedia tutorials, and A/V streaming.

For mobile devices, the Flash Lite technology has been derived from the basic Adobe Flash, and optimized to achieve a more lightweight runtime environment in terms of file size, memory footprint and performance requirements [8]. A disadvantage of Flash technology may be that its authoring tools are more geared towards designers than developers, and that the file format is proprietary. However, the multimedia capabilities and its installed base are virtually unparalleled, with many popular Web 2.0 applications using Flash technology. In the mobile world, Flash Lite is also quite well established [4].

2.5. Adobe Flex

Adobe Flex is a development framework that strives to make the traditionally designer-centric Flash technology more accessible to software developers by enabling them to build RIAs based on Flash technology. A Flex application is developed using the programming language MXML for the description of the user interface, and ActionScript for the implementation of the client-side logic. The Flex compiler will then translate this code into Flash bytecode that can be executed by any Flash runtime environment.

Flex can be especially attractive as a front-end for complex information systems, where the business logic can be implemented using Java EE on the server, which communicates with the Flex front-end through the so-called Flex LifeCycle Dataservice.

The use of Flex offers the advantage that the implementation of the presentation logic on the client allows for a highly responsive user interface, while the server-side business logic has easy access to back-end systems. However, it may be perceived as a disadvantage that the compiled Flash application needs to be downloaded by clients prior to use, and that a Flash Version 9 plug-in must already be installed on the client. The future relevance of the technology remains to be seen – a beneficial factor may be Adobe's decision to make Flex available as open source software from version 3 onwards.

2.6. Microsoft Silverlight

Silverlight [11] is Microsoft's attempt to introduce a browser extension into the market that shall provide the capabilities of the Windows Presentation Foundation (WPF) [13] to arbitrary platforms. The Mono Open Source Initiative is working on a port of Silverlight (code-named Moonlight) to Linux [9].

Silverlight consists of a browser plug-in which also includes a runtime environment for .NET (the .NET Framework for Silverlight) in version 1.1. Using the Silverlight technology, WPF user interfaces can be realized

as RIAs, which would enable large-scale reuse of application paradigms know from the Windows world. GUIs are described declaratively using the open and standardized XAML language, while different programming languages can be used to implement the GUI logic. Client-server communication is realized using JavaScript (similar to the AJAX approach); from version 1.1, limited .NET support for web services is also available. Since all GUI code is executed on the server and all communication is conducted across web services, there are no prerequisites as to which server platform should be employed.

Since Silverlight is still a young technology, disadvantages are its relative immaturity and sparse installed base among end-users. While Silverlight support is still low in the mobile realm, Microsoft is planning to provide native support under Windows Mobile.

2.7. OpenLaszlo

OpenLaszlo [10] is another platform for the creation of Rich Internet Applications. The technology is freely available as open source software under the Common Public License. The architecture is similar to Adobe Flex in that it does not require a runtime environment of its own, but employs existing runtime technologies: The GUI description and logic are encoded in the XML-based LZX format, which is translated by a compiler into Flash bytecode or DHTML. Either of these representations can be presented in a browser by clients. In addition, it is possible to generate applications for other runtime environments such as AJAX or Silverlight.

Benefits of OpenLaszlo applications are the use of an open, declarative language, which offers a great selection of standard GUI components, and the possibility to integrate it with server-side business logic implemented in Java. However, developers need to be aware of potential restrictions imposed by different target platforms that the OpenLaszlo application may be compiled to. Still, OpenLaszlo is attractive for its support of different target runtime environments.

3. Middleware for Mobile Application Support

Runtime environments for mobile applications, and especially technologies for RIAs, have the potential to open up a large universe of information to users through lightweight user interfaces. However, their network dependence makes them prone to temporary losses of connectivity, which may lead to anything from erratic application behaviour to loss or corruption of back-end data.

To ameliorate these problems that typically plague mobile applications, a number of middleware solutions exist that decouple the client- and server-side processing logic from the network by providing transparent caching and synchronization. In Fig. 1, these middleware implementations would be deployed on a client layer just below the web browser, where they can transparently manage the cooperation with remote components and handle communications with the server.

The recently-introduced Google Gears framework provides a mechanism for continuous execution of AJAX-based applications even under unstable network conditions [5]. It serves as a data management layer between the client-side application UI and the client's network layer. This data management layer comprises a connectivity switch, caching/synching logic and local database. When the network connection is lost, the connectivity switch will reroute all read/write operations from the application UI to the local database, which should be able to answer most requests for data, and can cache all data updates. Once the connectivity has been restored, the local database is synchronized with the server, and the regular connected operation resumes. The effectiveness of Google Gears' masking of connection losses depends heavily on the application domain and its reliance on external real-time information.

For more heavyweight mobile applications, a number of middleware solutions exist that support disconnected operation, caching/replication, and conflict resolution. Among the most interesting products are XMIDDLE and OSGi, two Java-based middleware implementations.

The XMIDDLE project [18] provides a data sharing middleware that allows caching and replication of data. Its data structure is represented as an XML-based document object model (DOM). XMIDDLE regards disconnection as usual in mobile environments, and therefore allows reconciliation among mobile clients that do not need to be connected to a central host. This mechanism makes it suitable for ad hoc networking models. However, XMIDDLE is solely concerned with the replication of data, but not of executable components.

The Open Services Gateway Initiative (OSGi) standard defines a small layer that allows multiple Java-based components to efficiently cooperate in a single Java VM. It provides mechanisms for remote component and dependency management, component life-cycle handling, as well as loading, running, updating and removing components. However, it does not provide support for data reconciliation after concurrent changes.

As a final example, the MobCo middleware [14] currently developed by the Applied Telematics/e-Business Group at the University of Leipzig in cooperation with Deutsche Telekom Laboratories strives to increase application availability and ubiquitous usage scenarios for mobile clients. MobCo offers a middleware and framework for the development of components that are able to run both on the server and the mobile client, but whose actual execution

location is determined at runtime. As the middleware is context-aware (regarding mobile hardware, network situation and application status), components are able to migrate from server to client or vice versa, depending on where they can experience better performance or increase system availability (in situations where the mobile client needs to switch to offline mode). This is combined with a persistence layer that is able to store and retrieve data from either local or remote locations transparently to the application. The optimal execution location for components will be determined through simulation, which will enable derivation and evaluation of component migration strategies. With such flexibility in the application's architecture, the MobCo middleware will allow distributed applications to cope much better with the heterogeneous and ever-changing conditions (in terms of network infrastructure, hardware, etc.) of mobile software, and thereby expand the number of contexts in which an application can successfully be run without having to write it more than once.

4. Conclusion

Our comparison of runtime environments shows that the future of mobile application development lies in Rich Internet Applications, which are based on highly distributed components and characterized by the intensive use of communication networks, complex interaction patterns and advanced GUI capabilities. In contrast, fat client applications can make a more efficient use of system resources and depend less on a permanent network connection. However, they support the "write once, run anywhere" paradigm only to a much lesser degree, as their closeness to the underlying system causes distribution and compatibility issues.

The more promising way for mobile devices thus seems to be reliance on the above-mentioned RIA technologies. AJAX is a prominent representative of this approach, as the fast-growing amount of commercial applications with this basis shows. From a software process perspective, OpenLaszlo shows what the future of Rich Internet Application development may look like: an open, declarative language that enables the generation of GUIs for different runtime environments, a great selection of standard GUI components, and the opportunity to smoothly integrate it with server-side business logic.

Finally, it is important to consider emerging specific middleware products for use in mobile environments. They are solving the typical mobility problems, which are of particular relevance when developing applications for demanding businesses use cases. In combination, open and highly flexible technologies for lightweight GUI development as well as highly specialized middleware products will be the crucial tools for developing the mobile Rich Internet Applications of tomorrow.

5. Acknowledgments

The Applied Telematics/e-Business Group at the University of Leipzig is endowed by Deutsche Telekom AG.

References

[1] Duhl, J.: Rich Internet Applications. IDC White Paper, 2003. http://www.adobe.com/platform/whitepapers/idc_impact_of_rias.pdf.

[2] ActionScript, Flash Developer Center, Adobe Systems Inc., 2007. http://www.adobe.com/devnet/flash/actionscript.html.

[3] Android - An Open Handset Alliance Project, 2007. http://code.google.com/android.

[4] Flash-Enabled Mobile Devices, Adobe Systems Inc., 2007. http://www.adobe.com/mobile/supported_devices/.

[5] Google Gears API Developer's Guide (Beta) – Architecture. Google, Inc., 2007. http://code.google.com/apis/gears/architecture.html.

[6] JavaFX. Sun Microsystems, Inc., 2007. http://www.sun.com/software/javafx/.

[7] LiMo Foundation, 2007. https://www.limofoundation.org.

[8] Mobile and Devices Developer Center, Adobe Systems Inc., 2007. http://www.adobe.com/devnet/devices/flashlite.html.

[9] Moonlight - Mono-based implementation of Silverlight, Mono Project, 2007. http://www.mono-project.com/Moonlight.

[10] OpenLaszlo, Laszlo Systems, Inc., 2007. http://www.openlaszlo.org.

[11] Silverlight, Microsoft Corp., 2007. http://silverlight.net.

[12] The Linux Foundation: Mobile Linux., 2007. http://www.linux-foundation.org/en/Mobile_Linux.

[13] Windows Presentation Foundation (WPF) community, Microsoft Corp., 2007. http://windowsclient.net.

[14] V. Gruhn and C. Schäfer. From Mobile Business Processes to Mobile Information Systems. *Software Architecture*, pages 296–299, 2007.

[15] S. Helal. Pervasive Java. *IEEE Pervasive Computing*, 01(1):82–85, 2002.

[16] S. Helal. Pervasive Java, Part II. *IEEE Pervasive Computing*, 01(2):85–89, 2002.

[17] D. S. Kochnev and A. A. Terekhov. Surviving Java for Mobiles. *IEEE Pervasive Computing*, 02(2):90–95, 2003.

[18] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Wireless Personal Communications*, 21(1):77–103, 2002.

[19] L. D. Paulson. Building Rich Web Applications with Ajax. *Computer*, 38(10):14–17, 2005.

[20] T. B. Preußer, M. Zabel, and P. Reichel. The SHAP Microarchitecture and Java Virtual Machine. Technical Report Tech. Rep. TUD-FI07-02, Fakultät Informatik, Technische Universität Dresden, April 2007.