AT&T Developer Program

Emerging Mobile Application Architectures

White Paper

Revision	**1.0**
Revision Date	June 19, 2008

This document and the information contained herein (collectively, the "**Information**") is provided to you (both the individual receiving this document and any legal entity on behalf of which such individual is acting) ("**You**" and "**Your**") by AT&T, on behalf of itself and its affiliates ("**AT&T**") for informational purposes only. AT&T is providing the Information to You because AT&T believes the Information may be useful to You. The Information is provided to You solely on the basis that You will be responsible for making Your own assessments of the Information and are advised to verify all representations, statements and information before using or relying upon any of the Information. Although AT&T has exercised reasonable care in providing the Information to You, AT&T does not warrant the accuracy of the Information and is not responsible for any damages arising from Your use of or reliance upon the Information. You further understand and agree that AT&T in no way represents, and You in no way rely on a belief, that AT&T is providing the Information in accordance with any standard or service (routine, customary or otherwise) related to the consulting, services, hardware or software industries.

AT&T DOES NOT WARRANT THAT THE INFORMATION IS ERROR-FREE. AT&T IS PROVIDING THE INFORMATION TO YOU "AS IS" AND "WITH ALL FAULTS." AT&T DOES NOT WARRANT, BY VIRTUE OF THIS DOCUMENT, OR BY ANY COURSE OF PERFORMANCE, COURSE OF DEALING, USAGE OF TRADE OR ANY COLLATERAL DOCUMENT HEREUNDER OR OTHERWISE, AND HEREBY EXPRESSLY DISCLAIMS, ANY REPRESENTATION OR WARRANTY OF ANY KIND WITH RESPECT TO THE INFORMATION, INCLUDING, WITHOUT LIMITATION, ANY REPRESENTATION OR WARRANTY OF DESIGN, PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, OR ANY REPRESENTATION OR WARRANTY THAT THE INFORMATION IS APPLICABLE TO OR INTEROPERABLE WITH ANY SYSTEM, DATA, HARDWARE OR SOFTWARE OF ANY KIND. AT&T DISCLAIMS AND IN NO EVENT SHALL BE LIABLE FOR ANY LOSSES OR DAMAGES OF ANY KIND, WHETHER DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, SPECIAL OR EXEMPLARY, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF GOODWILL, COVER, TORTIOUS CONDUCT OR OTHER PECUNIARY LOSS, ARISING OUT OF OR IN ANY WAY RELATED TO THE PROVISION, NON-PROVISION, USE OR NON-USE OF THE INFORMATION, EVEN IF AT&T HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES OR DAMAGES.

© 2008 AT&T Intellectual Property. All rights reserved.

Revision History

Date	Revision	Description
6/19/2008	1.0	First release of this paper

Table of Contents

Figures

Tables

1. Introduction

This paper discusses emerging mobile application architectures for handheld platforms. The intent is to educate developers on the strengths and weaknesses of new mobile development approaches. In particular, the paper examines service-oriented architecture (including Web services), Java, Web 2.0 methods, and database synchronization tools. These new architectures provide developers greater flexibility in how they develop applications, and can make mobile applications more efficient and effective.

1.1 Audience

The target audience of this white paper is software developers, IT developers, architects, and managers familiar with desktop or server development who are new to mobile development.

1.2 Contact Information

E-mail any comments or questions regarding this white paper via the AT&T Developer Program. Please reference the title of this paper in the message.

1.3 Resources

AT&T Resources

AT&T Developer Program: http://developer.att.com

Mobile Application Development: http://developer.att.com/mobiledevelopment

Device Information: http://developer.att.com/devicedetails

RIM BlackBerry Information: http://developer.att.com/rim

Windows Mobile Information: http://developer.att.com/windowsmobile

Symbian Information: http://developer.att.com/symbian

Java Information: http://developer.att.com/java

Palm Information: http://developer.att.com/palm

Mobile Middleware: http://developer.att.com/middleware

Wireless Reference Architecture Material: http://developer.att.com/WRA

Security Guidelines: http://developer.att.com/security

Certified Application Catalog: http://developer.att.com/certifiedsolutionscatalog

devCentral Resource on Platforms and Operating Systems

1.4 Terms and Acronyms

The following table defines the acronyms used in this document.

Table 1: Terms and Acronyms

Term or Acronym	Definition
API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
ASMX	ASP.NET Web services
ASP	Active Server Page
CDC	Connected Device Configuration
CF	Compact Framework (Microsoft .NET)
CLDC	Connected Limited Device Configuration
CSS	Cascading Style Sheets
CTP	Community Technology Preview
DOM	Document Object Model
eRCP	Embedded Rich Client Platform
eSWT	Embedded Standard Widget Toolkit
HTTP	Hypertext Transport Protocol
HTTPS	Secure Hypertext Transport Protocol (also S-HTTP)
Java ME	Java Platform Micro Edition
JSON	JavaScript Object Notation
JSR	Java Specification Request
JTWI	Java Technology for the Wireless Industry
MIDP	Mobile Information Device Profile
MOM	Mobile Operational Management
MSA	Mobile Service Architecture
MSF	Microsoft Sync Framework
OSGi	Not an acronym. Dynamic module system for Java. Managed by the OSGi Alliance.
RIA	Rich Internet Applications

SOA	A service oriented architecture such as Web Services using SOAP
SOAP	Initially stood for Simple Object Access Protocol
SyncML	Synchronization Markup Language
SVG	Scalable Vector Graphics
SWF	Macromedia Flash File Format
URI	Uniform Resource Indicator
W3C	World Wide Web Consortium
WPF	Windows Presentation Foundation
WSDL	Web Services Description Language
VM	Virtual Machine
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

2. Service Oriented Architecture (SOA)

The first architecture topic is Service Oriented Architecture, which is a model for creating a distributed system of services and service consumers. There are several technical definitions of SOA such as provided by:

- OASIS SOA Reference Model - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.
- Open Group SOA - http://www.opengroup.org/projects/soa/
- CBDI Definition - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmaj/html/aj1soa.asp

Practically speaking, SOA is simply a method for sharing common resources in a network among different entities that consume data. These entities can either be end-user applications or other services, the principle being that the consumer of the service need not know the details of the service; only what facilities it provides via its published interface.

Web services have become nearly synonymous with the Service Oriented Architecture even though a Web service is simply a specific type of SOA. In particular, a Web service provides a published interface using a WSDL (Web Services Description Language) file and provides communication between services and consumers using SOAP messages layered on HTTP (Hypertext Transport Protocol) or HTTPS (Secure HTTP). The primary reasons for its dominance are:

1. *Open nature of protocol* – Because the Web services standards are open, the services and clients are not necessarily tied to a single programming language or operating system. For example, a Web service running on Windows developed in C# can as easily communicate as a client written in C++ on a Unix based platform.

2. *Availability of tools and libraries* – Nearly every single programming language has a library that implements the SOAP and WSDL protocol, and there are many tools for creating SOAP services and WSDL files.

The ease and availability of Web service implementation has allowed it to grow quickly and dominate the field of Service Oriented Architectures.

Because of its popularity, it is natural that mobile platform developers will want to utilize Web services in their mobile applications. So far, however, the utilization of Web services on the mobile platform has been limited because of the following factors:

1. *Limited processing power* - Web services demand more processor resources than other networking protocols.

2. *Bandwidth availability and cost* – the SOAP messaging protocol is more verbose than other kinds of networking protocols.

3. *Battery consumption* – Greater network usage consumes battery life faster.

4. *Feature availability* – The tools and libraries necessary for implementing Web service clients on the mobile phone have only recently become available, and only on more powerful platforms such as smartphones.

However, as mobile phone processing power and battery life continue to improve, these factors become less significant. Furthermore, many of the mobile phones now on the market provide built-in support for Web services.

Generally, there are two models the mobile developer can choose from when creating an application that uses Web services. The first is to make the Web service calls directly from the mobile application and the second uses middleware to proxy the request from the phone to one or more Web services on back-end systems. Each model has advantages and disadvantages, which we present in the following two sections.

2.1 Web Services on the Handset

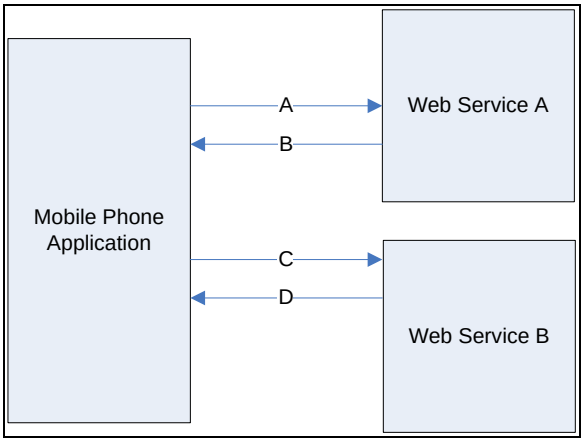
In this model, the application makes direct requests to one or more Web services, as shown in Figure 1. This model has the following advantages:

1. *Faster time to market* – If the tools are available to create the Web client on the phone, then it is relatively easy to create a Web-service-enabled application.

2. *Simple architecture* – It consists of just the application and the Web service. There is nothing between the application and the Web services it consumes.

3. *Scalability* – The scalability of the application is dependent only upon the scalability of the Web services it uses.

Figure 1: Web Services on the Handset



This model does have some disadvantages:

1. *SOAP requires more bandwidth than equivalent binary protocols* - Studies report typical factors in the range of 4 – 5 times larger[1]. Depending on the nature of the application, this may or may not have a major impact on the perceived performance of the application in particular and the device in general. If the application only occasional fetches a small amount of data from a Web service, then the overhead of using the Web service directly is not significant, and this can be a desirable approach.

2. *A single logical operation may require more than one Web service call* - Each request contributes a set overhead of bytes contributing to an overall increase in bandwidth and processor usage.

To create the client side of the Web service on the mobile phone, the phone platform must provide support for Web services. While it is possible to create a SOAP-enabled client side interface without the aid of tools or libraries (using basic socket commands and hand-written serialization/de-serialization code), doing so is complex and prone to failure. Fortunately, many mobile phones now provide some kind of direct support for Web services. Table 2 show the status of Web service support on the major mobile phone platforms.

[1] For example, see http://www.sics.se/~thiemo/ASWN2003.pdf.

Table 2: Web Services Availability (Based on Currently Available Versions)

Platform	Language	Version	Notes
BlackBerry	MDS Runtime	4.1.3	Via proxy
	Java	4.3.0	JSR 172
Windows Mobile	.NET CF	1.0	Built in support for Web services
	Native C/C++	Various	Mobile gSOAP
Symbian	Java ME	Various	JSR 172
	Carbide VS C/C++	2.0.1	Nokia WSDL to C++ converter for S60

The BlackBerry MDS runtime does not provide direct support for Web services running on the mobile device, and is mentioned here for completeness. Instead, it follows the middleware model described in the following section where the BlackBerry Enterprise Server with MDS Services acts as the middleware application and thus supports Web services via proxy.

Many mobile devices support a version of Java Platform Micro Edition (Java ME); however, in order to support Web services, the device must also have built-in support for JSR 172 (the Java Web Services Specification). You can check on JSRs supported on AT&T devices at http://developer.att.com/devicedetails.

You can also filter on JSR 172 on Sun's Java ME device table: http://developers.sun.com/mobility/device/pub/device/list.do?filterIds=1000. This link provides up-to-date information about Web service enabled devices. Sun does not list any BlackBerry devices because BlackBerry does not support Java ME; it has its own implementation of Java. Nevertheless, version 4.3.0 of the BlackBerry Java Development environment does support JSR 172.

Windows Mobile .NET CF provides thorough support for Web services on the device. This support has been available since .NET CF version 1.0 and in Visual Studio 2003. Creating a Web service enabled application is as easy for mobile devices as it is for standard Windows based applications. In Visual Studio 2008, for example, the developer creates a Smart Device Project, targeting it to the desired platform and language (e.g. Windows Mobile 6 Smartphone and C#). Then, the developer can immediately add a Web reference. The Web reference can be either local or remote and can be either a WSDL file or an ASMX

(ASP.NET Web services) file. Once complete, the developer then creates the application making the external Web service calls as necessary.

For C++ programming in Windows Mobile, you might consider the gSOAP toolkit (http://www.cs.fsu.edu/~engelen/soap.html.) Given a WSDL file, the developer runs a couple of programs (wsld2h and soap2cpp) to generate the client classes that allow direct calls to the Web service from a native application.

Symbian has native support for Web services via the Nokia WSDL to C++ converter.[2] This works in a fashion similar to gSOAP by using a WSDL file to create the necessary classes for the developer to use in creating a Web-service enabled application.

2.2 Web Services Security

The security of Web services is a big concern for developers. Although providing a detailed discussion of security is outside the scope of this document, there is an important consideration: SOAP, as a protocol, provides no security for protecting private data. The developer is responsible for ensuring that the communications between the mobile device and the server either contains no sensitive data or that the data is protected.

The available security measures depend on the development environment used by the developer. The NET.CF developer can choose to encrypt individual SOAP messages using WS-Security or to use transport level security such as HTTPS encryption or a VPN. WS-Security is a an enhancement to the SOAP protocol and allows SOAP messages to be individually signed (to protect its integrity), to be encrypted (to protect private information), and to be authenticated. WS-Security is usually better than HTTPS as it does not consume as much communications overhead, and it can be selectively applied to sensitive SOAP messages.

Alternatively, the .NET CF developer can choose to encrypt the channel over which the SOAP messages travel using encryption methods of a developer's choosing. This has its advantages as it is generally available and is well understood as a technique for protecting data. The developer must decide whether to encrypt all SOAP messages. If only some are to be encrypted, then,

[2] See http://www.forum.nokia.com/info/sw.nokia.com/id/5ddeb939-c4e4-4e64-8f25-282e1e86afed/Nokia_WSDL_to_Cpp_Wizard_for_S60.html. Supported on devices that use Nokia S60 3rd Edition.

the application will need to open both a secure and an insecure channel and ensure that sensitive requests only go out the secure path.

There are no libraries available for using WS-Security for Java ME devices. Apache does provide a Java library called WSS4J that supports WS-Security, but this is not available for the Java ME platform. In principle, the developer could develop their own WS-Security protocol, but generally using HTTPS or a VPN will be a much simpler option.

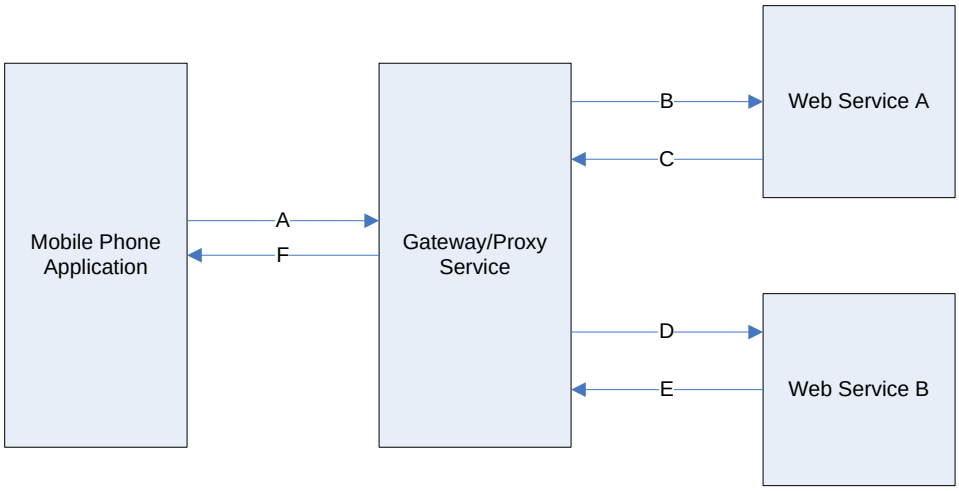
2.3 SOA between Middleware Servers and Backend Applications

In this model, the mobile phone does not directly make the Web service requests from the application. The application instead proxies the requests through an intermediary service that carries out the actual Web service requests. Figure 2 presents this model. This approach has certain benefits:

1. *Reducing the number of over-the-air requests* – A single logical operation in the mobile application may result in several Web service calls. Pushing the operation to the gateway reduces the number of over-the-air requests to a single round-trip call. Figure 2 shows this; a single remote request (arrow A) results in two Web service calls (arrows B and D).

2. *Binary over-the-air protocol* – Web service calls are relatively verbose. The developer can create a binary over-the-air protocol between the mobile phone and the Gateway/Proxy service. Representing the data in this form minimizes bandwidth usage while still allowing the developer to take advantage of existing Web services.

Figure 2: Web Services Proxied through a Gateway



There are disadvantages to this model:

1. *Longer time to market* - The developer must create and maintain the Gateway/Proxy service, increasing the time to market.

2. *Additional points of failure* – The gateway/proxy service adds an additional point of failure to the system.

3. *Scalability issues* – The developer must design the gateway/proxy service carefully to ensure that it is scalable. On a busy system with a large number of mobile phones, the gateway/proxy service could become a serious bottleneck in the system.

As mentioned in Section 2.1, BlackBerry MDS adopts this model (see http://na.blackberry.com/eng/services/mobile.jsp). In BlackBerry's case, the Gateway/Proxy service shown in Figure 1 is BlackBerry's Enterprise Server with BlackBerry MDS services. The developer creates the MDS application that runs on the phone and this is tied to a service the developer creates that runs on the Enterprise server. The developer is then free to use Web services from the Enterprise server with the knowledge that the data exchanged between the phone and server will be in a compact binary representation.

There are also a number of mobile-middleware platforms that developers can consider that employ this gateway/proxy architecture. See AT&T information on this topic at http://developer.att.com/middleware.

2.4 Weighing the choices between SOA models

Ultimately, the single greatest factors that affect the choice between the two models are the number and size of Web service requests required by the application. If the number of requests in a time interval is small and the size of the requests is not too large, then, the simplicity of the Web Service-on-the-handset model outweighs the complexity of the middleware model. On the other hand, if an application uses the network intensively, uses multiple Web services or if the size of the messages is large, then it may be better to create a middleware-based application to reduce the load and bandwidth of the application on the mobile phone.

Table 3 summarizes these considerations.

Table 3: Best Uses of Web Services

Type of SOA Model	Best Use Circumstances
Web Services on the handset	Relatively small number of requests Amount of data communicated relatively small
Web Services via proxy	Multiple Web services being accessed Large messages sizes Large amount of network usage

3. Java ME Evolution

Java ME has become a powerful development environment for mobile devices since Sun Microsystems introduced it to the community in 2000. This section presents the ways that Java capability is evolving, and how developers can take advantage of these developments.

Fragmentation remains a serious consideration to this day with Java ME. Java's principle of "write once, run anywhere" has not been as successful on mobile devices as originally hoped. There are several kinds of fragmentation affecting the platform:

1. *Device Level* – The sheer diversity of different types of mobile devices makes it difficult to target a single development and operational environment for all kinds of devices.

2. *JSR Ambiguities* – Ambiguities in the JSRs result in slightly different and sometimes incompatible implementations by manufacturers.

3. *Optional and Manufacture APIs* – The JSRs that make up Java ME on mobile devices allows certain optional APIs, as well as allowing manufacturers to create their own specific APIs.

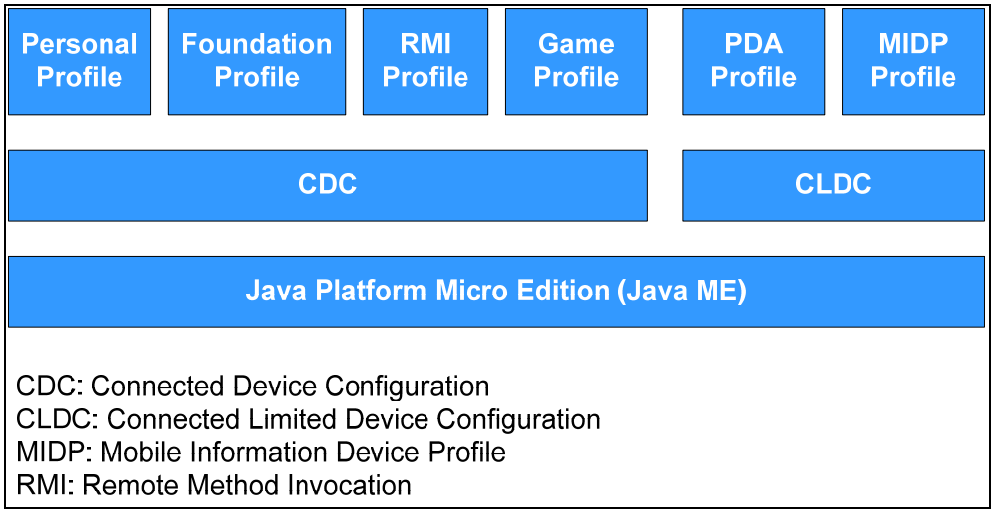
As early as 2003, developers recognized the problem of fragmentation and first addressed it with JSR 185 (Java Technology for the Wireless Industry).

Since then, new efforts to address fragmentation include the Mobile Service Architecture JSRs (JSR 248 and 249), MIDP 3 (JSR 271), and Mobile Operational Management (JSR 232). There is also now further consolidation in that new JSRs are being developed to work with both CDC and CDLC.

3.1 Java ME Architecture

In the mobile space, Java ME consists of two distinct platforms. These are Connected Devices and Connected Limited Devices, specified initially in JSR 36 and JSR 30 respectively, as shown in the following architecture diagram.

Figure 3: Java ME Architecture



We describe the two configurations below. However, it is important to note that most developers will be working with interfaces provided at the profile level, and hence the configuration type does not necessarily play that important a role.

Connected Devices, or CDC devices, are mobile devices with higher-end capabilities. These devices typically have 32 bit processors, about 2 Megabytes of RAM and 2.5 Megabytes of ROM. Examples include higher-end handhelds and set-top boxes. Connected Limited Devices, or CLDC devices, are lower-end mobile devices. These devices can have 16-bit processors, and as little as 160K RAM. Examples include mid-tier and low-tier mobile phones.

See Figure 4 and Figure 5 below for a timeline of the major JSRs on CLDC and CDC devices. The double triangles in these figures represent those JSRs that are complete while the solid lines are those JSRs that are still in development.

Figure 4: Evolution of Connected Limited Devices

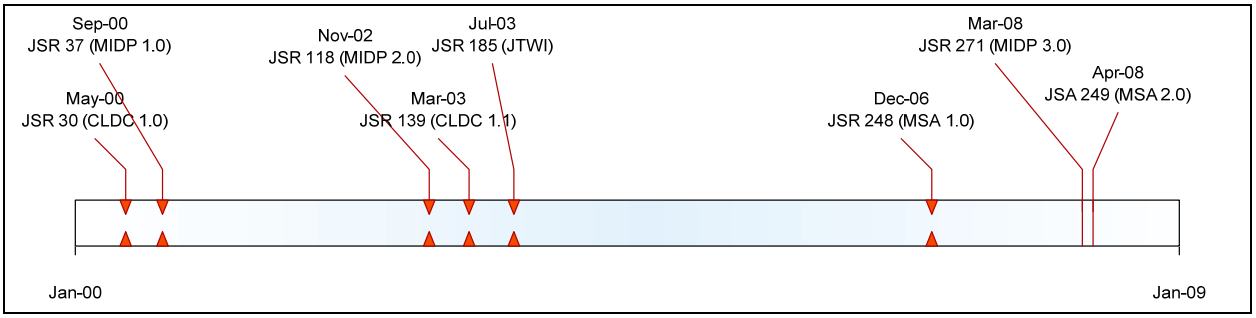
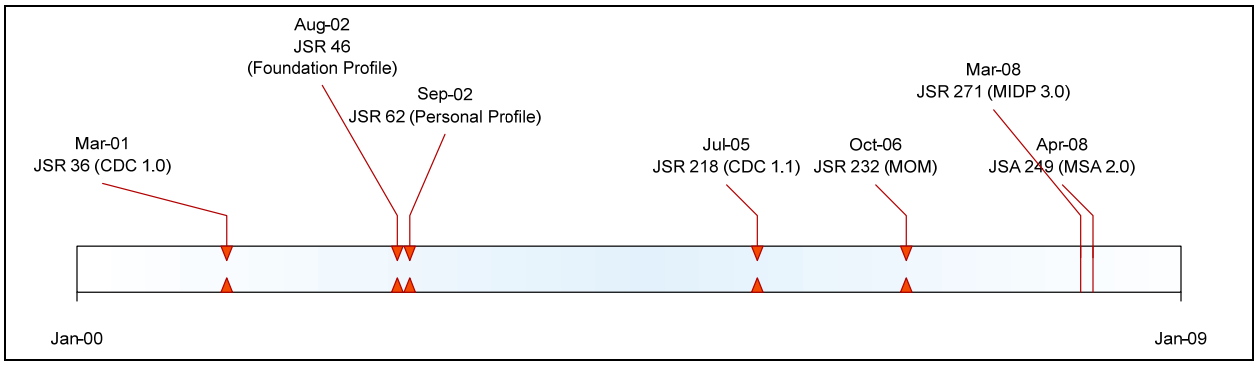


Figure 5: Evolution of Connected Devices



The following tables, Table 4 and Table 5, summarize the JSRs noted in the previous two figures.

Table 4: Major CLDC-Related JSRs

JSR	Name	Purpose
30	CLDC 1.0	Defines the configuration and virtual machine (VM) for resource-constrained devices.
37	MIDP 1.0	Defines the application runtime environment.
118	MIDP 2.0	Enhances MIDP 1.0 by • Adding an enhanced security model. • Adding support for HTTPS.

JSR	Name	Purpose
		• Adding a gaming and multimedia API.
139	CLDC 1.1	Makes CLDC more compliant with JVM specs with features such as improved error handling.
185	JTWI	Addresses API fragmentation and delivers a predictable and clear specification.
248	MSA 1.1	Addresses fragmentation by defining a predictable and interoperable development environment.
271	MIDP 3.0	Major goals include: • Enable support for both CDC and CLDC. • Improve interoperability across implementations. • Expand existing MIDP functionality.

Table 5: Major CDC-Related JSRs

JSR	Name	Purpose
36	CDC 1.0	Defines the configuration and VM for higher-end mobile devices.
46	Foundation Profile	Defines the set of APIs for applications running on devices with the CDC configuration.
62	Personal Profile	Defines the profile for those devices that require a high-degree of Internet connectivity.
218	CDC 1.1	Provide updates to the existing CDC configuration.
232	MOM	Mobile Operational Management (MOM) introduces the OSGi platform to mobile devices. It allows mobile devices to evolve and adapt their capabilities by installing new components on demand.

JSR	Name	Purpose
249	MSA 2.0	Major Goals: • Continue work in minimizing fragmentation by further defining the development environment. • Extend MSA to CDC devices. (Note that MSA 2.0 also supports CLDC devices) • Provide a backward compatible path from MSA 1.1 and JTWI.

3.2 Java ME Trends

Development efforts in Java ME include consolidation of the language so that applications are more portable across implementations. The following two sections discuss in further detail two projects that focus on consolidation and evolution of device capability:

1. Mobile Service Architecture (MSA), JSR 248 and 249.
2. Mobile Information Device Profile (MIDP) Version 3, JSR 271.

However, these are not the only area of activity for Java ME. There are other developments as well, including:

1. Mobile Operational Management (MOM), JSR 232.
2. Embedded Rich Client Platform (eRCP).
3. JavaFX Mobile.

We discuss these and other Java topics below.

AT&T's Java emphasis is on implementations that employ MIDP and Mobile Service Architecture.

3.3 Mobile Service Architecture (JSR 248 and 249)

Mobile Service Architecture is an important effort to standardize the two device platforms (CDC and CLDC) and to define predictable platforms. It consists of two JSRs. The first, JSR 248, targets CLDC device types while the second, JSR 249, targets both CLDC and CDC device types. Essentially both of these have the same overriding goals:

1. Minimize fragmentation of mobile Java implementations.

2. Enable the use of MSA to a wide range of mobile devices.

3. Ensure compatibility between CDC and CLDC devices.

To achieve these goals, the MSA defines a complete set of JSRs a device must support in order to be MSA compliant. This makes it much easier for developers to target specific device types with their applications. In addition, MSA adds clarifications, requirements and recommendations to the underlying JSRs for implementers, with the goal of maximizing interoperability.

The following table shows the different JSRs specified by JSR 248 (MSA 1.1) and JSR 249 (MSA 2.0). MSA 2.0 is backward compatible with the MSA 1.1 full set, MSA 2.0 Subset is backward compatible with MSA 1.1 Subset and MSA 2.0 Limited is backward compatible with the JTWI 1.0 API set.

Note: information in section about MSA 2.0 is based on preliminary draft versions of specifications. These are subject to change.

Table 6: Mobile Service Architecture

JSR	Name	Low Device Segment		Mid Device Segment		High Device Segment	
		JTWI	MSA 2.0 Limited	MSA 1.1 Subset	MSA 2.0 Subset	MSA 1.1 Full	MSA 2.0 Full
281	IMS Services *						X
280	XML						X
258	UI Customization *						X
257	Contactless *						X
272	Mobile Broadcast *						X
256	Sensor						X
180	SIP					X	X
177	SATSA – PKI *					X	X
177	SATSA - CRYPTO					X	X
172	Web Services					X	X
234	Multimedia Suppl.				X	X	X
293	Location API 2.0 *				X	X	X
211	Content Handler				X	X	X
177	SATSA - APDU *				X	X	X
238	Internationalization		X		X	X	X
287	Vector Graphics 2.0			X	X	X	X
184	3D Graphics			X	X	X	X
082	Bluetooth *		X	X	X	X	X
075	File and PIM		X	X	X	X	X
205	Messaging 2.0		X	X	X	X	X
120	Messaging 1.0	X					
135	Mobile Media	X	X	X	X	X	X
271	MIDP 3.0				X		X
271	MIDP 2.1		X	X		X	
118	MIDP 2.0	X					
139	CLDC/CDC	X	X	X	X	X	X

* indicates conditionally required

The MSA divides devices into three types of devices: high-level devices, mid-level devices, and low-level devices. A high-level device is one that supports all of the JSRs defined in the MSA. Mid-level devices are not as powerful or do not have as many features, so they support a smaller sub-set of JSR's. For example, a high-level device must support Web services (JSR 172) while a mid-level device does not have to. Low-level devices are least powerful and support the smallest set of JSR's. For example, a mid-level device must support 3D graphics (JSR 184) while a low-level device does not need to.

All devices must support a common set of JSRs in order to be MSA compliant. Some of these are mandatory and some are conditionally mandatory. A device must support a mandatory JSR to be MSA compliant. For example, all MSA compliant devices must support JSR 139 (e.g. they must be CLDC 1.1. compliant). A device need not support a conditionally mandatory JSR if the facility is not available. For example, if a device supports Bluetooth, then it must support JSR 82 (Java APIs for Bluetooth) to be MSA compliant.

JSR 248 is now completed while JSR 249 is still in early draft release. You can see JSR248 compliant devices be visiting Sun Microsystems's device table at http://developers.sun.com/mobility/device/pub/device/list.do?filterIds=1034.

3.4 Mobile Information Device Profile Version 3 (JSR 271)

The Mobile Information Device Profile (MIDP) version 3, not only adds significant new functionality, but is also an important step in further consolidation of Java implementations. Its most important feature is that it works with both CDC and CLDC types of mobile devices. In the past, a MIDP application, a MIDlet, was only guaranteed to only work on CLDC device types, specifically MIDP/CLDC compliant devices. There was no support for CDC-compliant mobile devices, as they did not support MIDP. With MIDP 3, this will change. A mobile device can support either CDC or CLDC (or both) and be able to support MIDP. This means that the developer can target a much wider set of devices with a single MIDlet. In addition, the JSR also states that a MIDP 3 compliant device must also be backwards compatible to earlier versions of MIDP. This immediately allows existing MIDlet applications to run on CDC compliant devices.

There are other enhancements to the MIDP as well, including (but not limited to):

1. Tightening of the API specifications to improve cross platform inter-operability.

2. Background and multitasking execution.

3. Enable shared libraries in MIDlets.

4. Increasing functionality, such as better support for devices with large displays and secondary displays.

5. Inter Midlet communication.

The capabilities of MIDP 3.0 should make it suitable for many enterprise applications that were not previously feasible.

This JSR is in public review of the early draft stage. There is no schedule for when it will reach final stage. Note that JSR 249 mandates MIDP 3.0.

3.5 Services Framework (JSR 320)

Currently under development, this is an important specification that will define service and management framework APIs for managing applications and services, and will support both CDC and CLDC-based devices. Services include those with no user interfaces and applications with user interfaces. The framework will provide for both local and remote management. Specific capabilities include:

- Install/Uninstall applications and services
- Start an Application/Service
- Stop an Application/Service
- Discover an Application/Service
- Monitor an Application/Service
- Service Application/Alarming

JSR 320 will be used in conjunction with MIDP 3.0 capable devices, and can also be used in conjunction with devices that support JSR 249. JSR 320 includes many of the capabilities contained in JSR 232, discussed below.

3.6 Scalable Vector Graphics (JSRs 226 and 287)

The JSR 226 and JSR 287 specifications define an API for rendering scalable 2D vector graphics that include files based on the W3C (World Wide Web

Consortium) Scalable Vector Graphics (SVG) format. The target platform is CLDC/MIDP devices. JSR 287 is backward compatible with JSR 226, and adds features such as the ability to create and modify animations, and expanded support of SVG Tiny version 1.2.

With JSR 287, it will be possible in a Java environment to create rich user interfaces comparable to those that use Adobe Flash.

Note that the Mobile Service Architecture specifications, JSR 248 and JSR 249, require implementation of JSR 287.

3.7 Mobile Operational Management (JSR 232)

Mobile Operational Management (MOM) is another area of development for Java ME. JSR 232 is the application of the OSGi framework to the mobile space. This framework is a somewhat overlapping framework to that provided by MIDP. Its primary advantage is that it simplifies the deployment and management of applications.

The framework adds an additional degree of flexibility that is lacking in mobile phones today. Currently, a developer is constrained by the libraries that are available on the device. A JSR 232-enabled device would allow in principle the developer to dynamically extend additional facilities into the framework. For example, the developer could install a more powerful graphics library instead of the default graphics libraries.

JSR-232 provides for components that can be installed on demand. These include:

- Active elements with no user interaction (services)
- Active elements with user interfaces (applications)
- Shared libraries (native and Java)

In addition, to further extend the capabilities of the phone, the developer would not need to create or wait for a JSR to be complete (a long process), but instead could extend the capabilities of the framework by adding a plug-in module with the requisite facilities.

Compliance with JSR-232 requires conformance with:

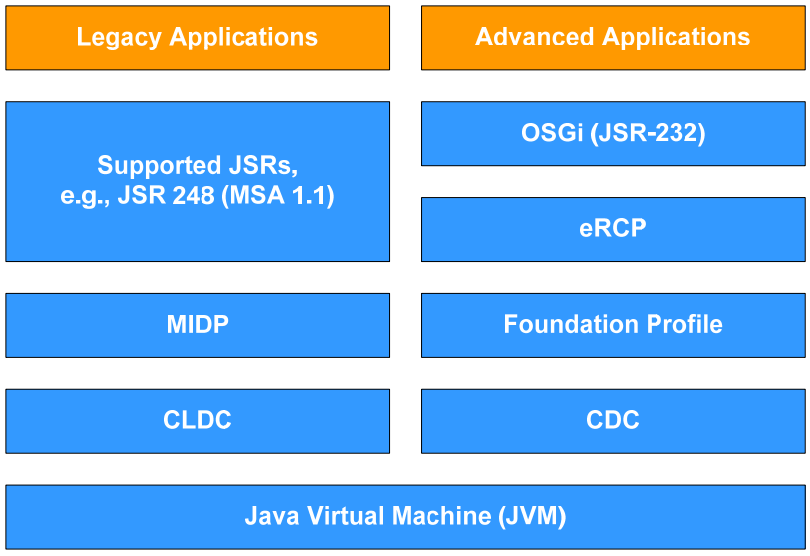
- OSGi Service Platform Core Specification Release 4 [OSGi Core]
- OSGi Service Platform Mobile Specification Release 4 [OSGi Mobile]
- Backward compatible later versions of these two specifications.

A drawback to this framework is that it does not work with lower-level devices. JSR 232 targets devices that are CDC compliant. A developer wishing to create an application for CLDC compliant devices would need to use the MIDP framework instead.

3.8 Embedded Rich Client Platform

To take advantage of the OSGi model without having to wait for the arrival of JSR 232 enabled devices, a developer can use the Embedded Rich Client Platform (eRCP, see http://www.eclipse.org/ercp/). The eRCP is already available for a number of mobile devices and gives the developer the immediate benefits of the OSGi model without having to wait for JSR 232.

Figure 6: OSGi/eRCP Stack



The eRCP contains the following components:

- *Core Runtime* – The Eclipse core that provides the OSGI framework for the applications.

- *eSWT* - Embedded Standard Widget Toolkit. This brings the SWT graphical toolkit to mobile devices.
- *eWorkbench* – A container for eRCP applications.
- *eJFace* – An extension to eSWT so eRCP applications can integrate with the eRCP workbench.
- *eUpdate* – An interface for dynamically updating software.

As with JSR 232, the full implementation of eRCP only works on higher-level devices (e.g. CDC device types). However, the eSWT component has been ported to CLDC devices as a plug-in for MIDP-based applications, thus enabling rich graphics on lower-tier phones.

eSWT is partitioned into two parts: the Core and the Expanded set. The Core contains the minimal set to run a eSWT application and allows the developer to target the widest set of devices. The Expanded set targets higher-level devices with larger screen sizes and greater capabilities.

Thus, the developer has the following choices when working with eRCP:

1. *Core eSWT* – The developer can use this to develop applications for CLDC/MIDP device types. This gives the developer the widest possible range of target devices for their application.
2. *Expanded eSWT* – The developer can use this if the application requires larger screen size but does not need the full capabilities of a CDC device type.
3. *eRCP* – The developer can use this to develop a full eRCP application on a higher level device. Currently, the device must be CDC compliant.

Two vendors supporting eRCP and OSGi today for mobile platforms are IBM, with its J9 JVM, and Nokia which supports this platform on some of its high-end handsets. Supported platforms include:

- Windows Desktop
- Windows Mobile 2003/5/6
- WinCE 5.0 Professional
- Nokia Series 80

3.9 JavaFX

JavaFX is a Java platform that Sun Microsystems is developing to span a range of client device types, including mobile devices, desktops and other consumer devices. The runtime environment is built largely on Java SE.

JavaFX provides capabilities that compete with Adobe Flash/Flash Lite and Microsoft's Silverlight. It consists of a scripting language called JavaFX Script and a scripting engine. On mobile devices, JavaFX Mobile is the scripting engine. The goal of JavaFX is to provide an easy to learn language for creating Rich Internet Applications (RIA). Although a skilled Java Developer can do this currently with existing tools such as Swing, JavaFX will make this easier while still being able to leverage the facilities available in Java.

JavaFX will co-exist with Java ME, and a phone running JavaFX will be able to provide both JavaFX APIs as well as Java ME APIs. JavaFX is still in the early stages of development and, at present, there are no JavaFX enabled phones. Details of APIs are not yet available.

4. Mobile Web 2.0

Another aspect of mobile application development seeing new application architectures is with Web-based approaches. The term 'Web 2.0' loosely describes the set of technologies, business models, and design patterns for creating successful and interactive Web sites. There is, however, no single technical definition of the term. Much debate has occurred about the meaning of the term since Tim O'Reilly first introduced it in 2005[3]. A search of the Internet for definitions of 'Web 2.0' results in terms like 'buzzword', 'revolution', 'Web-based applications' and even 'democracy'. Perhaps the most appropriate definition for this discussion is the one given by O'Reilly:

> *Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as platform, and an attempt to understand the rules for success on that new platform. Chief among those rules is this: Build applications that harness network effects to get better the more people use them[4].*

Although specific rules may be hard to pin-down, successful Web-based companies have developed services and applications with certain characteristics:

1. Appeal to a wide audience (e.g. Flickr, MySpace).

2. Use of the latest technologies to provide users with a valuable and enjoyable experience (e.g. YouTube, Google Maps, ITunes).

3. Use of data in new and interesting ways (Amazon and Netflix for their recommendations).

Mobile devices are at the cusp of a major change in how people interact with the Web. To date, most people interact with the Web via their home or work computers. Mobile devices that are Web-enabled are changing that paradigm as more people use the Web from their mobile devices. The concepts of Web 2.0 will apply equally well to mobile access.

How does this apply to the mobile Web developer? The characteristics of successful Web sites apply equally well to the mobile space, with the following challenges:

[3] http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
[4] http://radar.oreilly.com/archives/2006/12/web-20-compact-definition-tryi.html

1. *Physical constraints and variety of the mobile devices* – The constraints and varieties of mobile devices make it a challenge for the developer to create an application or service that is available to a large number of mobile users.

2. *Availability of new technologies* – Mobile Web browsers undergoing rapid change and new technologies such as AJAX (Asynchronous JavaScript and XML) are just now becoming available for mobile devices. However, the level of support varies among devices.

3. *Connectivity considerations* – Mobile devices are not always connected, so applications must take this into account. Those that must be able to operate in a disconnected mode are not necessarily good Web candidates.

In the following three sections, we investigate these characteristics as applied to mobile Web-space.

4.1 Mobile Web Best Practices

Appealing to the widest possible audience means that the user should be able to use the application or service from any device weather it is mobile or desktop. The W3 organization has a mobile working group (http://www.w3.org/Mobile/) that has established a set of Web Best Practices (http://www.w3.org/TR/mobile-bp/) to help the developer in creating Web sites that appeal mobile users. This consists of a set of sixty practices for delivering content to mobile users. Some of the best practices recommended by this document apply equally well to standard Web applications (e.g. Error messages should be informative and provide a link back to useful information), but others are specific to the mobile Web content. In particular, this document addresses several issues that the developer must consider when creating mobile Web content:

1. Screen sizes are smaller with mobile devices.

2. Bandwidth can be expensive, especially with usage-based plans.

3. Browser support varies by device.

Rather than repeat what is easily available from this resource, we simply highlight some of the more important concepts:

1. *Thematic Consistency - Ensure that content provided by accessing a Unified Resource Identifier (URI) yields a thematically coherent experience when*

accessed from different devices.

A user should be able to use a single URI to access a Web site using different devices, but should have the same level of experience. The canonical example of this is that a user could a particular Web site from either a mobile or a desktop browser. From the desktop, the user would experience the full graphical interface the site offers while from a mobile device it might be a simpler text-oriented interface. The actual consequence of this for the developer is that he or she must detect the browser type using the HTTP header 'User-Agent' in the HTTP request and set the return content accordingly.

2. *Capabilities - Exploit device capabilities to provide an enhanced user experience.*

   This means that the developer needs to be knowledgeable about the capabilities of different devices and browsers and take advantage of them when available. For example, a developer should not exclude the use of AJAX if one browser does not support AJAX and another does. Furthermore, if a mobile device supports full graphic capabilities (such as the Safari Browser), the developer should not limit the application to being text only for the sake of a more limited mobile browsers.

3. *Content Types – Send content in a type that a device supports and prefers.*

   The developer should ensure through detection of the supported types (e.g. via the HTTP Accept header) and through testing that a device supports the content type.

4. *Character Encoding – Send content in an encoding that a device supports and prefers.*

   As with the previous section, a Web service should not return content to a device that cannot support it.

5. *Cookies – Do not rely on cookies being available.*

   Web sites regularly use cookies to provide session management, user identification and user preferences capabilities. Unfortunately, cookies support in mobile browsers is not complete. The developer should test to see if cookies are supported on the device, and if not, provide alternative means

for required capability (e.g., session management can be accomplished via a URI decoration).

6. *Cache Headers – Use cached headers where appropriate.*

   The use of the Cache-Control header is more critical in the mobile Web as bandwidth is more expensive than in the standard Web. As a result, the developer should carefully consider what content can be cached by the device (e.g. Cascading Style Sheets [CSS] and images) to reduce the amount of content that is refreshed from the server during a page request.

7. *Input – Minimize the amount of input from the user.*

   Given the limitations of mobile device input, the developer designing Web forms should limit the amount of input from the user wherever possible. This means, for example, using selectable lists and radio buttons instead of free text input.

As an added benefit, the Mobile Best Practices Working group is creating a 'mobileOK trustmark'. The Best Practices statements in the Mobile Web Best Practices document can have conformance statements built around them. As a result, tests can be created to ensure that a Web site conforms to the document. If a Web developer creates an application conforming to the document, then, it could earn the mobileOK trustmark and could, as a result, achieve distribution to a wider audience.

4.2 AJAX and Other Technologies

AJAX has become almost synonymous with the term Web 2.0 and for good reason. Many Web sites developed for desktop platforms use some form of AJAX technology and some of the most compelling Web sites (e.g. Google Maps) use it extensively. Meanwhile, mobile devices are becoming more powerful and the browsers available to them are becoming more capable. Many mobile Web browsers, at least on smartphone platforms, either now have or will have built in support for rendering full Web pages and built in support for AJAX. It is inevitable that Mobile Web 2.0 applications will use AJAX as well.

When we refer to AJAX, we mean a set of technologies for creating interactive Web applications. An AJAX application includes the following:

- *JavaScript/ECMAScript* – Accesses the Document Object Model (DOM) to dynamically change and interact with user's display.
- *XML* – Typically (though not required) used as a communications protocol with the server.
- *XHTML/HTML* – Used as the markup for the user's display.
- *CSS* – Used for styling the user's display.
- *DOM* – Contains the markup and styling information in a structure that is accessible via JavaScript.

The success of AJAX rests on the fact that it communicates asynchronously with the server using the XMLHttpRequest Object. This allows the Web application to transparently request data from the server and selectively update the user's display dynamically. It also allows interaction with data downloaded by the Web application that can be stored locally, not requiring every user operation to wait for a response from a server.

To exploit AJAX in mobile devices, the mobile browsers must support the AJAX technologies listed above. However, knowing which mobile browsers support AJAX is not an easy task. The variety and capabilities of Web browsers is large. Ultimately, the developer will need to test their applications with a set of target devices in order to verify that the application works. Table 7 is a sampling of some of the more popular browsers (including some Beta browsers) with their support for AJAX. What is important to understand about this table is that although many of the browsers claim to support AJAX, few of them document the precise level of support. Specifically, few of them document the Web standards that the browser supports. What is clear though is that the direction for all mobile browsers is for full eventual AJAX support.

Table 7: Mobile Browser Support for AJAX

Browser	Platform	AJAX Support
BlackBerry Browser	BlackBerry	No
IE Mobile 6	Windows Mobile	Yes
S60 Browser	Symbian S60	Yes
S40 Browser	Nokia S40	No
Opera Mobile 8.65	Symbian, Windows Mobile	Yes
Opera Mini 4	Windows Mobile BlackBerry Symbian	No
Skyfire Beta	Windows Mobile	Yes
Minimo Beta	Windows Mobile	Yes
Safari	iPhone	Yes
Blazer 4	Palm	Yes

There are several considerations for the developer in order to utilize AJAX today in mobile devices. Most important is to keep in mind the limitations of the mobile device. What this means practically is the following:

1. *Avoid the use of XML as the communications protocol* – XML is a heavy protocol and parsing XML on the phone is expensive in terms of battery life. Instead, use a more lightweight protocol such as REST along with HTML. Some people call this AJAH instead of AJAX[5]. The advantage is that the Web application can immediately inject, with minimal processing, into the DOM the

[5] http://jimmyzimmerman.com/blog/2006/08/ajax-vs-ajah.html

HTML that the server returns. In cases where AJAH is not appropriate, then the developer can use JSON (JavaScript Object Notation) instead. This protocol represents JavaScript objects in a compact form that is easy for JavaScript to parse.

2. *Avoid the use of JavaScript libraries not specifically designed for mobile devices* – These libraries (e.g. Google Web Toolkit) provide a convenient library of code that eases the development of AJAX applications. The problem with these is that they rely on devices with fast processors and large amount of memory. Furthermore, the libraries tend to be large and the browser must pull the library to the device first in order for the application to start. This consumes bandwidth and increases the start-up time for the application.

3. *Minimize the size of CSS files* – The Web application must pull any CSS files it references, even if the styles are never used. The developer should eliminate unused styles and ensure that the size of the CSS files are as small as possible.

4. *Use on-demand asynchronous requests* – Instead of requesting data at intervals (set by timers) or within a loop, the developer should design the Web application so that it only makes requests as needed by the user. (e.g., it should never request data that the user does not specifically require).

5. *Test AJAX applications* – Given the diversity of mobile Web browsers, the developer should test the Web application not only within emulators but also on the devices themselves.

4.3 Widget Technologies

Note that mobile Web browsers are not the only way to take advantage of AJAX on mobile devices. There are several new and interesting projects underway to take advantage of AJAX while avoiding the variability found in mobile Web browsers. The developer creates widgets that can execute outside the browser on the mobile device. The widget is a redistributable package that is executed on the device by an application (a widget player).

The widget can thus be installed and run like any other native application. The advantage of these projects is that they combine the advantages of AJAX with the consolidation of a single widget player (instead of multiple browsers). Furthermore, because these applications run outside of the browser, they are not

subject to the same restrictions as the browser, so accessing device APIs is possible (e.g., a Camera API). These projects are:

- *MOJAX (*http://mojax.mfoundry.com/display/mojax/Main+Page*)* – Runs on any J2ME CLDP/MIDP device.
- *Opera Widgets* (http://widgets.opera.com/) – Available as part of Opera 9.5 for mobile phones. It will run on any device that supports Opera Mobile 9.5.(currently, only Windows Mobile is listed as supported).
- *Bling (* http://www.blingsoftware.com/index.html*)* - Available for J2ME and BREW devices.
- *Nokia Web Runtime* (http://www.forum.nokia.com/main/resources/technologies/browsing/widgets.html*)* – Available for S60 3rd Edition FP2 Phones.

5. Mobile Mash-Ups

A mash-up is a kind of Web application that combines data from external sources to create an entirely new Web service. For background information, refer to http://www.ibm.com/developerworks/xml/library/x-mashups.html and http://en.wikipedia.org/wiki/Mashup_(Web_application_hybrid)). Although not required, the data sources are typically some kind of published API, such as those provided by Google (http://code.google.com/) and Yahoo (http://developer.yahoo.com/). Examples of mash-ups are available from directories of mash-up sites such as http://www.webmashup.com/. The benefit of mash-ups is that the developer can use information that is readily available on the Internet in the form of Web services in a new and innovative application.

Generally, a mash-up will consist of three different components: the external data sources, the web site hosting the mash-up, and the browser on the client's device. The mash-up can either be an entirely server side component (following the classic Web-design philosophy), but in general it is designed as a combination of server side components (to do the heavy lifting) and client-side scripting (using AJAX or similar technology). This architecture results in a Web application that is responsive to the user's needs and is scalable in that it spreads the load of a busy mash-up across the client browser, the external data sources and, only minimally, the Web site hosting the mash-up.

Because of the state of the mobile technology, there are few if any examples of mobile mashups. However, because a mashup in general is a Web application and in many cases is an AJAX Web application, the guidelines for developing Mobile Web 2.0 applications discussed above apply here as well. In addition, the developer should be aware of the following issues:

1. *Web Service APIs* – Many of the publisher's of Web services provide JavaScript API libraries to make it possible to easily create AJAX-based mash-ups. However, these were designed with desktop devices in mind. As a result, they may not be portable to the mobile device and if they are, may be too heavy, in terms of memory and bandwidth usage, for the mobile device.

2. *Security* – JavaScript enforces the *Same Origin Policy* on asynchronous requests. A common method to avoid this (with cooperating Web service providers) is to use JSON along with dynamic scripting to fetch data from the external provider (e.g. http://developer.yahoo.com/common/json.html). This is

a lightweight method for creating mash-ups that avoids the use of heavy-weight script libraries. The problem is that this is a known exploit that can allow a hacker to access private information. This vulnerability applies equally to mobile and desktop scenarios.

To avoid these issues, the developer should adopt a more conservative approach. The developer should consider the use of a Web service to do the computationally-intensive portion of the Web application. The mashup site would host the service so it would be accessible from the client. It would be responsible for requesting data from external resources, aggregating the results and returning the data to the client side application.

The developer should avoid the use of JavaScript client libraries provided by publishers of Web services unless they designed specifically for mobile devices. Lastly, for security reasons, the developer should avoid the use of dynamic scripting unless the developer clearly understands the security model for the application.

6. Adobe Solutions

6.1 Flash and Flash Lite

There are several versions of the Flash runtime for mobile devices: Flash for PocketPC, Flash Lite 2.1, and Flash Lite 3.1. See http://www.adobe.com/products/flashlite/. All of these runtimes play multimedia content and applications in the form of SWF (Macromedia Flash File Format) files developed using Adobe's Flash development environment. The primary difference for each of is the platforms they support. The details on each version are shown below in Table 8. For a comparison of the feature sets in each release, see http://www.adobe.com/devnet/devices/articles/flash_lite_feature_comparison.pdf.

Table 8: Mobile Flash Player Versions

Runtime	Supported Platforms	Flash Version	Action Script
Flash for Pocket PC	Windows Mobile 5, Pocket PC	7	2.0
Flash Lite 2.1	Symbian, BREW, Windows Mobile 5	7	2.0
Flash Lite 3.1	Symbian, Windows XP, Redhat Linux	8	2.0

Flash Lite 3.1 is the latest release in the Flash Lite player series. Adobe released it to manufacturers in March 2008 with devices planned for the end of 2008. It added significant improvements over Flash Lite 2.1 including the following:

- H.264 support – This allows the player to support the latest MPEG 4 encoding standard.
- Flash 8 support – This includes the ability to play FLV files, which was unsupported in Flash Lite 2.1.

- Faster performance – ActionScript performance is improved by 15% to 20% over Flash Lite 2.1 and rendering performance is improved by 20% to 25% over Flash Lite 2.1.

The developer has several considerations for developing a mobile flash application. One advantage is that because the players are so similar, an application written for one player should run without significant changes on another player. The developer should only need to recompile the code for the targeted device configuration. This is an improvement over Java ME where fragmentation of the environment has complicated wide scale deployment of applications. If the developer is trained in Flash technologies, then transitioning to mobile devices should not be too difficult. On the other hand, the learning curve is very steep if the developer has little or no background in Flash.

Development requires Adobe Flash CS3 Professional along with Device Central CS3. Flash CS3 allows the developer to create SWF movies and target them at specific configurations (e.g. Flash 7 Pocket PC). The developer can then test their applications with Device Central CS3 against a variety of emulated device profiles.

6.2 Adobe Integrated Runtime

Adobe Integrated Runtime is an application that allows developer to create standalone rich Internet applications. It currently does not support any mobile devices, but we mention it here for completeness and because Adobe says that it will support mobile devices in the future. With AIR, a developer can create a Web application using a combination of Web technologies such as HTML, AJAX, Flash or Flex to create a rich Internet application that runs as a desktop application. In this way, AIR is very similar to Mozilla Prism: it allows Web applications to run on the desktop. However, it is distinct in that the code (e.g. the HTML, JavaScript, Flash, etc.) is delivered to the user device in the form of a single download that can execute at any time without necessarily having access to the Internet.

6.3 Adobe Mobile Platform

This is a new suite of products offered by Adobe to mobile operators (see http://www.adobe.com/mobile/platform/index2.html). It consists of the following components:

- Flash Cast – A client-server solution that allows operators to deliver customized content via channels to mobile devices.

- Flash Home – A client-server solution that allows operators to create customized home screen to mobile devices.

Both can work independently of each other, but in tandem can provide the mobile device user with a better experience. Flash Home presents the mobile device user with a home screen that is customized by the mobile operator. Using this system, the user can configure operator-provided channels for viewing. For example, the user could configure their phone to receive a weather channel.

The mobile device must have the Adobe Mobile Client installed for the user to take advantage of either of these products. The Adobe Mobile Client is a Flash based client runtime optimized for a wide range of mobile devices. It is built on Flash 6 with a subset of ActionScript 1.1 and 2.0 language elements.

Adobe did not design this system for application development by individual developers. It is more a tool for mobile device operators to use for creating a portal through which their users can access exclusive content.

7. Microsoft Silverlight

Silverlight is a cross-browser and cross-platform plug-in that allows the developer to create rich internet applications. It works on Internet Explorer, Firefox, and Safari browsers and on Windows and Mac OS X platforms. It is a direct competitor to Adobe Flash, Adobe Flash Lite and JavaFX. For background information, see http://www.microsoft.com/silverlight/.

There are several different versions of Silverlight available. See http://en.wikipedia.org/wiki/Microsoft_Silverlight for a listing of releases. Currently, Silverlight 1 is the release-to-market version while Silverlight 2 is available for download as a pre-release version and as a Beta 1 candidate. At present, neither of these is available for mobile devices. However, Microsoft plans to soon release (second quarter of 2008) a version of Silverlight for mobile devices. Based upon Silverlight 1, Silverlight for Mobile will not have the advanced features available in version 2 (such as ASP.NET support). Initially, it will only be available for Windows Mobile 6 running IE6, but Nokia and Microsoft recently announced plans to make it available to S60 and S40 based Symbian platforms.

Adobe Flash is leading the race in these rich internet applications, so it is worthwhile to compare some of the benefits and drawbacks of Silverlight to Flash. For a more extensive comparison, see http://weblogs.asp.net/jezell/archive/2007/05/03/silverlight-vs-flash-the-developer-story.aspx and http://silverlight.net/forums/t/3015.aspx. Silverlight does claim several advantages over Flash:

1. Animation – Animation is done using time based Windows Presentation Foundation (WPF) animation model whereas Flash is frames-based involving transformation matrices.

2. XAML – Silverlight uses XAML (Extensible Application Markup Language), an XML based mark-up language for creating the Silverlight application. Flash movies are stored in a binary format, not directly accessible to the developer.

3. Language support – Silverlight supports JavaScript for its development language in Version 1 and adds ASP.NET languages (e.g. C#) for Version 2. This means that developers have the potential for code reuse in Silverlight applications. Flash only supports ActionScript, which is used in no other scenario except Flash, limiting code reuse.

4. Tools – The developer can use Visual Studio to create Silverlight applications, a tool familiar to most Windows developers. Flash uses Adobe CS as its development platform.

The main drawbacks to Silverlight are:

1. Maturity of the product – It is relatively new and does not have the same experienced developer community.

2. No Mobile Device support – At present Silverlight Version 1 is still in Beta for mobile devices. Flash Lite runs on a variety of mobile devices.

Silverlight will be an appealing choice for those Web developers who have never before developed in Flash, but want to create a rich internet application. The learning curve is steep for learning Flash and the platform for creating a Flash movie (Adobe CS) is an expense consideration. If the developer already owns Visual Studio, then getting started in Silverlight is simply a matter of installing a plugin to Visual Studio and working through the demos. Silverlight also takes advantage of the existing knowledge base for JavaScript so the developer does not need to learn another language (e.g. Flash ActionScript). Future enhancements (specifically adding support for ASP.NET languages) to Silverlight will also give developers a powerful reason to develop Silverlight based applications.

8. Google Gears

Google Gears is a browser plug-in that enables developers to create Web applications that work in offline as well as online modes. It works with either Firefox or Internet Explorer. It is essentially a synchronization tool between a local and remote data store. In the event that the application loses network connectivity, a Google-Gears-enabled application will still be able to operate using the local data store. Google Gears is also available for Windows Mobile 5 and 6 devices running Internet Explorer Mobile.

The plug-in consists of three modules:

1. *LocalServer module* – This module caches and serves up application resources locally. A configuration file determines which resources are cached and which are not.

2. *Database Module* – This module serves as the data store for the plug-in. It is an SQLite database engine.

3. *WorkerPool Module* – This module contains the background threads that synchronize the local store with the remote store.

Any Web application can take advantage of Google Gears to enable offline access. However, the developer must modify the Web application, integrating with the Google Gears API in order to enable such access. Full information on how to install and use Google Gears is available at http://code.google.com/apis/gears/. In summary, the developer chooses what files are available offline, how to treat offline versus online states (e.g. modal versus modeless) and how frequently to synchronize with the remote system.

For the mobile Web developer, the main considerations in using Google Gears are how new the platform is, and the impact it will have on the device. The mobile version of Google Gears uses the same architecture as desktop version. The developer must consider:

1. *What should the application cache* – More caching is not necessarily better. More caching requires more local data storage. It also requires greater CPU cycles (to manage the data) which can potentially result in slower performance on a mobile device.

2. *How should the application synchronize data and at what interval* – The developer has two choices: modal and modal-less. Modal requires that the user choose when to place the application in offline or online mode. The advantage of this is that synchronization is a simpler task. Modal-less means that the application assumes it is offline and a worker thread runs in the background to keep data synchronized when a connection is available. This is advantageous when a connection is unreliable or not always available, but it forces the CPU to do more processing (by running a background thread and monitoring the system data).

9. Synchronization

Synchronization is the process that brings into agreement data located in separate systems through a process of comparison and exchange. On mobile devices, synchronization is important for two reasons: as a backup measure in case the mobile device is lost or stolen and as a method for facilitating applications.

The basic architecture for synchronization is relatively straightforward and usually follows the client-server model. The system involves the following steps:

1. *Client connection to server* – The client (e.g., the mobile device) connects to the server to begin synchronization.

2. *Change detection* – The two systems go through a process that detects the differences between the two systems.

3. *Conflict management* – If there is a conflict between the two systems (e.g., a file edited on both the remote and local systems) then it is handled by merging the change, overwriting the change or ignoring the change.

4. *Data exchange* – The two systems exchange data so that when the process is complete, the data on the two systems is identical.

A wide range of synchronization products and technologies characterizes the industry. Currently, there is no single common method or protocol for synchronizing devices. Synchronization depends as much on the device as the environment in which it operates.

Providing a comprehensive survey of all synchronization applications and technologies is beyond the scope of this paper. Instead, we focus on the most common ones the developer is likely to encounter:

- ActiveSync – Synchronization on Microsoft CE based platforms
- Microsoft Sync Foundation – The next generation synchronization from Microsoft
- SyncML – The industry led synchronization protocol.

- Other Vendors – A brief overview of products from Nokia, Oracle, and Sybase

9.1 Microsoft ActiveSync

On Windows CE based mobile devices, the developer can use ActiveSync (as documented at http://msdn.microsoft.com/en-us/library/ms879772.aspx) to synchronize with Windows desktops or Microsoft Exchange. On Windows XP and earlier OS, the application called ActiveSync handles direct synchronization between mobile devices and the desktop. On Windows Vista, Windows Mobile Device Center takes over this task. In either case, ActiveSync on the mobile device drives the synchronization process. For synchronizing over-the-air, the Exchange ActiveSync protocol allows mobile devices to synchronize with an Exchange server directly, without going through the desktop. As long as the device can connect to the Internet, it can synchronize its E-mail messages, calendar and contact lists with an Exchange server.

ActiveSync follows the client-server architecture: it consists of a *service manager* (the server) and *service provider* (the client). The service manager runs on both the desktop and mobile device and allows service providers to connect to and synchronize with them. ActiveSync provides a set of service providers for common applications and data types. For example, it provides data types for Outlook synchronization (e.g. appointments and contacts data types) and for File synchronization (file data types). In addition, the developer can create service providers to suit their application's data synchronization needs. The service provider consists of two pieces, a *device provider* and a *desktop provider.* The developer creates both of these providers to ensure synchronization of data between the device and desktop system.

A number of companies have licensed ActiveSync, making it available on Apple's iPhone and Symbian devices.

9.2 Microsoft Sync Framework

Microsoft Sync Framework (http://msdn.microsoft.com/en-us/sync/default.aspx) is the next generation platform for handling synchronization between systems. It currently is in a release stage called CTP2 (Community Technology Preview), so is not yet available publicly on any devices. However, it is available for download and review. As of CTP2, there is now support for Windows Mobile devices. The Microsoft Sync Framework (MSF) is a generalization of the ActiveSync

architecture. According to Microsoft, it is “a comprehensive synchronization platform enabling collaboration and offline for applications, services and devices with support for any data type, any data store, any transfer protocol, and network topology.”

MSF extends synchronization beyond the mobile device and desktop synchronization scenario. It enables developers to create a *sync-enabled ecosystem* in which devices can roam freely and remained synchronized.

The underlying architecture is similar to ActiveSync but extends beyond it in significant ways. In the MSF, there is the concept of a *participant*. A participant can be either a full participant, a partial participant, or a simple participant.

- Full participant. One where developer can create new applications and new data stores for the device. An example is a smart phone.
- Partial participant. A device that can store data, but which does not support application execution. An example is a thumb drive.
- Simple participant. One that can only provide data upon request, but one that cannot store data. An example is an RSS feed.

All of these participants can take part in the sync ecosystem established by the MSF.

The MSF follows the same client-server architecture as other synchronization systems. The key difference is that the MSF abstracts much of the synchronization tasks into a MSF runtime. The simplified task for the developer is to create the synchronization application, with the MSF runtime embedded in it, along with a provider. A provider is the code which represents, in an abstract way, the data store to be synchronized.

There are several types of common providers that a developer can use:

- ADO.NET Provider – allows sync between ADO.NET enabled databases on a mobile device and ADO.NET enabled databases on servers.
- File Systems Provider – Allows files and folders to be synchronized.
- FeedSync Provider – Allows RSS and Atom feeds to be synchronized.

For information on how to create a synchronization application, see http://msdn.microsoft.com/en-us/library/bb902833(SQL.100).aspx.

The developer can also create their own custom providers as well. For information on how to create a custom provider, see http://msdn.microsoft.com/en-us/library/bb902826(SQL.100).aspx.

MSF provides interfaces for both managed (e.g. NET.C#) and unmanaged (e.g. C++) code.

9.3 SyncML

SyncML (Synchronization Markup Language) is an industry initiative to address the fact that there is no single shared method for synchronizing data between devices and applications. SyncML is now formally managed under the Open Mobile Alliance Data Synchronization and Device Management. See http://www.openmobilealliance.org/Technical/DM.aspx. It is not a new initiative, but is mentioned here for completeness.

The goals of the initiative are to:

1. Synchronize networked data with any mobile device.

2. Synchronize a mobile device with any networked data.

It does this by establishing a common synchronization protocol (SyncML) that application and devices can use to synchronize with each other.

To that end, the SyncML effort parallels the efforts of the Microsoft Sync Framework in that both efforts seek to establish the system in which devices can be synchronized between each other. The difference is that SyncML focus on establishing a common language between different devices and applications while MSF focuses on establishing a common application environment (e.g., the MSF runtime) without necessarily enforcing a common language (e.g., a participant can consume data from an RSS feed or a mobile phone to become synchronized).

The SyncML initiative started in 2000 as a specification for an XML based language called the Synchronization Markup Language (SyncML). SyncML establishes the communications protocol and architecture for synchronizing two systems (see

http://www.openmobilealliance.org/tech/affiliates/syncml/syncmlindex.html#WP). SyncML follows the client-server architecture. The client application, called a Sync Client Agent, runs on the device that initiates the synchronization. The server application, called the Sync Server Agent, runs on the device that allows clients to be synchronized. The Sync Server Agent and the Sync Client Agent communicate with each other using SyncML as the common language. The transport layer between them can be any of the common transport layers such as HTTP, HTTPS or WAP.

SyncML is used in a number of applications today. Wikipedia (http://en.wikipedia.org/wiki/SyncML) provides a comprehensive list of clients and servers available for developers. A sampling of some of these is:

- Funambol (http://funambol.com/) – This is an open source project that provides a server and client solution for synchronization needs and is based on SyncML. It provides client libraries in both Java and C++.
- Synthesis AG (http://synthesis.ch/) – Also provides a server and client solution for data synchronization.
- Sybase One Bridge – Provides synchronization support for a number of devices including SyncML enabled devices.
- Oracle Mobile Data Sync – Provides synchronization support for SyncML enabled devices.

A number of devices support SyncML out of the box (see for example http://files.toffa.com/upload/TSyncML%20Docs/supported_devices.html). Typically, these SyncML compliant devices support synchronization of Contacts, Calendar, Tasks and Email via SyncML.

9.4 Nokia Intellisync

A number of vendors provide enterprise level synchronization systems. These systems typically provide default mechanisms for synchronizing the common set of data such as e-mails, contact lists and calendars.

The Nokia Intellisync Mobile Suite is a suite of applications for managing mobile devices. It includes two synchronization modules:

- *Nokia Intellisync File Sync* – Pushes the most up-to-date information to PCs and devices, and pulls content from remote devices to store locally

- *Nokia Intellisync Application Sync* - Synchronizes data between databases such as Oracle, Sybase and IBM DB2. Synchronizing between a local database store and a remote database store should not require any changes on the application side. It is able to track the changes in the database and synchronize based on the information stored in it.

9.5 Sybase Products

Sybase One Bridge delivers and synchronizes email, contacts, calendar, tasks and notes between mobile devices and computers in an enterprise network. It works with Lotus Domino and Microsoft Exchange. It supports many mobile devices such as Windows Mobile, PalmOS, Symbian and SyncML compliant devices.

Meanwhile, Sybase iAnywhere provides synchronization support for IBM, Microsoft and Oracle databases, the ability to develop sync-based applications using Microsoft .NET or Pearl scripting languages, and support for multiple mobile device platforms including Palm OS, RIM BlackBerry, Symbian and Windows Mobile.

9.6 Oracle Mobile Collaboration 10g

Oracle Mobile Collaboration 10g is a suite of products to enable mobile users to communicate and interact with their company's enterprise applications. It includes two modules that support data synchronization. The first is the Mobile Push Mail module. This module not only pushes email from the company's email systems, but synchronizes email on the phone with the network (e.g. if an email is deleted from the phone it is removed from the system). The second module is the Mobile Data Sync module. This module supports synchronization of calendar and contacts using SyncML as the protocol.

10. Conclusion

This paper has demonstrated the innovation being applied to mobile application development. The various new architectures and tool sets provide developers new options in how they create applications, make applications more efficient and effective, and also put mobile application development within reach of many IT organizations.

AT&T recommends that developers:

- Understand trends in mobile application architectures.
- Understand available architectures and tools.
- Select target device platforms carefully.
- Emphasize application architectures that support all devices of interest.
- Carefully weigh local clients vs. Web-based approaches.
- Consider mobile middleware approaches.

AT&T's developer program, http://developer.att.com provides both overview information and details that can assist developers in evaluating what development options and processes are most appropriate for their application.

11. Acknowledgment

Rysavy Research, LLC contributed to the content of this paper.