3

The Role of WAC in the Mobile Apps Ecosystem

Zeiss Joachim, Davies Marcin and Pospischil Günther
FTW. Telecommunications Research Center Vienna
Austria

1. Introduction

The Wholesale Applications Community (WAC) was founded to change the overall market for mobile applications. WAC intends to achieve this by introducing open standardized technologies based on W3C widgets and/or OneAPI definitions. In addition, WAC provides complimentary commercial models. This will allow developers to deploy an application across multiple devices and across multiple operators. Developers should not need to negotiate with each of them. WAC provides the commercial prerequisites.
WAC's objective is to commercialize products for its member companies. Open Web standards are utilized in support of this commercialization effort as long as such adoption does not impact the required time to market. The WAC widget specification is therefore based on W3C and OMTP standards to the greatest extent possible.
As already mentioned, WAC widgets utilize Web technologies. If a developer understands HTML and CSS to design a good looking website and has used JavaScript to create functionality and services for a user, he/she will also be able to write mobile apps in WAC. The widget packaging format is based on the W3C Widget Packaging specification and introduces some extensions to meet WAC requirements, such as specifications for billing. WAC widgets can optionally utilize a comprehensive handset API. A code-signing security system ensures that widgets can only access APIs that are suitable to their level of trust.
This chapter gives an understanding of WAC and compares it to existing mobile app eco-systems and technologies. We investigate the details of WAC regarding its API definition, runtime implementation, technologies used, and SDK-based development. We answer the questions on what makes the difference about developing WAC apps and how developers and users can both benefit. Finally, recommendations are given to industry on how to use WAC and how to further develop WAC in upcoming releases of the standard.

2. Mobile app development overview

The current mobile platform space suffers a lot of fragmentation. Developing software for mobiles generally can happen in a native way (like Objective C for iOS) or by using a virtual platform that offers (theoretical) portability.
Today, the most relevant native development approaches are:

- iOS using Objective-C, Xcode and Interface Builder
- Native Qt libraries and SDK from Nokia: may survive only for embedded systems but not for app development, they are still too complicated although much better than the basic Symbian APIs

- Java based virtual machine programming in Android based on the Dalvic VM, Linux kernel and Eclipse Plugin SDK

NET or Silverlight type of development for Windows Phone 7 which didn't gain momentum yet

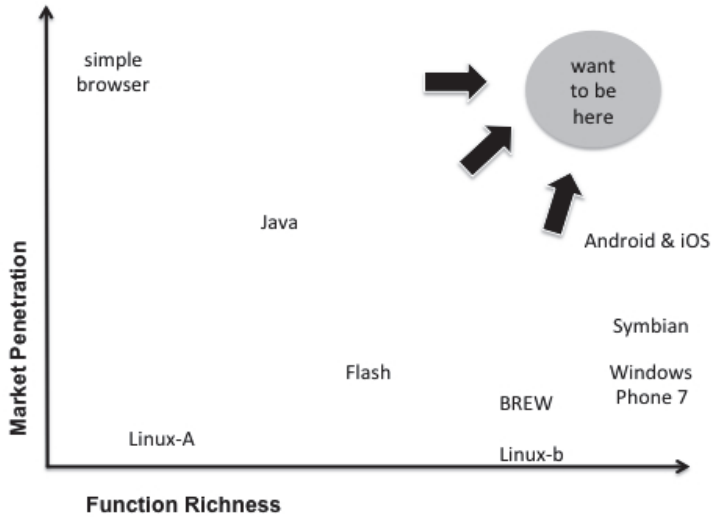


Fig. 1. Mobile Platform space (source: Fraunhofer Fokus)[1]

Virtual development approaches include:

- Web based development, WebOS: the scene and stage based paradigm in conjunction with a W3C DOM compliant model and Javascript development is promising and innovative. The Mojo framework provides access to device functionality, but native development is offered as well
- Java ME, has become a runtime for apps on middle to low-end mobile devices
- Flash: its future on mobile devices is unclear and technology might become obsolete in turn of HTML5 supporting browsers and web runtimes
- Desktop based web development (widgets): Yahoo! widgets on Windows (former Konfabulator runtime) and Dashboard on Mac OS X show the importance of including graphics to coding activities
- Other widget systems: Standardization as with W3C widgets, Opera Widgets, etc.

2.1 Frameworks

WAC needs to compete with the following frameworks for native or web-based mobile app design and implementation:

- COCOA and Quartz for iOS, or OpenGL ES
- Android Java APIs
- W3C widget based runtimes
- Generic Javasscript libraries: jQuery, ExtJs, jqTouch, SenchaTouch

[1] Linux-A and Linux-B denote different Linux Variants/Distributions

- HTML5 supporting runtimes
- WebGL supporting runtimes
- PhoneGap or Titanium apps (see Section 3)

For the iPhone, Apple offers a series of well known frameworks grown and proved in years of software design for Desktop computers and notebooks running the OS X operating system. Cocoa APIs for operating system task interaction and Quartz for GUI implementation have been adapted to the needs of mobile devices (iPhone and iPad), especially for multi touch events (a feature WAC does not support as of today). Although Objective-C as the used programming language is not so common, the frameworks are very well designed, easy to use, and the MVC (Model View Controller) software design pattern has been excellently implemented. The latter is one of the main reasons of the huge success of iPhone apps. Also the backwards compatibility to the C language and the support for Open GL allows rather easy porting of existing game software to the iOS platform.

Android decided for using Java as the main implementation language including most of the standard SDK APIs known for desktop computers. In addition, apps benefit from garbage collection and a virtual machine called Dalvik has been designed to fulfil the particular needs and to cope with performance restrictions of mobile devices. The use of Java attracts a huge designer community familiar with Java programming on the desktop and encourages the switch from designing Java programs on the PC to implementing apps for Android phones and tablets.

WAC is based on the W3C widget standard with all its good and bad implications. Being standardized, it can be easily implemented for all runtime programs and sandboxes already supporting W3C, e.g. the Opera browser or widget runtimes. However, there is more to be done — libraries like jqtouch or SenchaTouch need to be supported or integrated to allow for a state of the art user experience known from native apps. HTML5 is partially, WebGL not supported at all for WAC runtimes. This means that game development and sophisticated graphical user interfaces are hard or even impossible to implement with WAC.

2.2 App stores

Currently industry experiences a tremendous growth of the mobile application market. While mobile applications were distributed via individual channels in the past, developers and users now clearly turned to centralized catalogues of software offerings for their devices that are managed and provided by trusted entities.

The WAC store and distribution platform may compete with the following prominent stores:

- AppStore form Apple

The Apple AppStore is designed to meet the various application requirements of iPhone and iPod users. Being a storehouse with mobile applications that can be downloaded and installed, the AppStore also recommends related applications that can be rated by users. Apps are tested and certified by Apple before they are included in the AppStore catalogue. Except for developer testing, there is no (official) alternative way to install apps on the iPhone. Thereby Apple ensures a consistent user experience and avoids security issues. As of July 2011 there are 425,000 apps available with 15 billion downloads since the AppStore's inception. Due to its success, Apple recently started an Application store for desktop computers as well.

- Android market

The Google Android Market allows users to browse through and download third party applications for Android mobile phones. The store has grown from 2,300 applications in 2003 to 80,000 applications in 2010 with over 1 billion downloads in total. As there is no centralized testing of apps, Android systems currently suffer a little from poor quality apps and the fact that developers can make money mostly out of in-app advertisements but not with selling them in the store as such.

- OVI from Nokia

Although the destiny of Symbian remains unclear, the download rate has reached 5 million per day in July 2011. It seems to be successful for emerging markets like India, China or Turkey.

- Blackberry App World

Developed by Research In Motion (RIM), the Blackberry App world is all about BlackBerry and the various applications. It was previously known as the BlackBerry Application Storefront. Although RIM initially announced that the store would only be available in the United States, United Kingdom, and Canada, it is available across 70 countries today. The store has around 25,000 applications in its catalogue and over 2 million applications are downloaded daily. The store is offered in multiple languages (English, French, Italian, German, etc.) and allows the user to download free and paid applications. Unlike other application stores it offers latest news as well as a support system to take care of one's questions and concerns.

2.3 Software development kits

The following SDKs may be considered for Web-based app development:

- Eclipse plugins, e.g. for Android and JIL, WAC
- Online SDKs such as ARES for WebOS
- Standalone target specific SDKs (Xcode, Interface Builder, Visual Studio)
- Widget specific SDKs (template based with graphical tools) like Dashcode
- WebDesign SDKs such as DreamWeaver

These SDKs are considered in the following section regarding the differentiation of WAC from its competitors.

3. Differentiation of WAC

All technologies integrated in WAC have been used before by other initiatives, e.g. the usage of web design features by WebOS, offering widgets as apps by Opera, Apple or Yahoo. Application stores are already out there as well.

So what makes the difference?

Well, it's the combination of all of that, the device and network operator independence, the billing methods and in that sense the possibility to integrate network operator assets like network based GPS, call control, in-app billing or identity management. This is what the WAC consortium should focus on. Unfortunately, WAC 3.0 supporting OneAPI standards is not finished yet.

Properties of the WAC API and runtime are:

- Based on W3C widget standards
- Detailed certification based app functionality management (maybe a bit too complex / over engineered)

- Integration of remote APIs and Telco assets (based on GSMA OneAPI specifications)
- Operator independence
- Device/Manufacturer independence
- Entirely web based, i.e. apps run in a HTML rendering engine supporting CSS and Javascript, based on W3C Standardization; WAC specific JavaScript APIs are additionally available.
- Integration of Telco assets, remote APIs: e.g. for identity management

Two of the main competitors in web-based mobile cross-platform development are PhoneGap and Titanium mobile, which are discussed briefly in the following sections.

3.1 PhoneGap

PhoneGap (for details please see reference PhoneGap) is an open-source mobile framework that supports the following important platforms:

- iOS
- Android
- Blackberry
- Palm webOS
- Symbian WRT

As in WAC, developers write code in HTML/CSS/JS and deploy it to their target platform(s) (HTML5/CSS3 supported). However, no native code is produced, the final app is wrapped into a native "web view'"object (like in the desktop projects Fluid or Prism).

The currently supported API features of the PhoneGap platform can be found under reference (PhoneGap).

3.2 Appcelerator titanium mobile

Titanium mobile from Appcelerator is another open-source mobile framework for iOS/Android (Blackberry RIM to come). Developers use web technology (HTML5/CSS3 support) but Titanium's plug-in architecture also allows coding modules natively (Objective C or Java) to extend the app with native functionality. Titanium offers an integrated IDE and one of the biggest advantages is that native code is generated, so apps are faster and can make use of native UI controls. Another useful feature is that apps can also be ported to the desktop with the Titanium desktop edition.

3.3 Comparison with WAC

Table 1 compares PhoneGap and Titanium Mobile with WAC in terms of platform support, SDKs, UI controls, documentation & community, and whether a runtime is needed. It should be noted that native UI controls can be „emulated" in PhoneGap and WAC by using Javascript libraries such as SenchaTouch. Finally, the performance of all three is most likely best with Titanium (as native code is produced), however no hands-on test was performed.

Although WAC claims to be platform independent in the future, right now only Android based runtimes exist. Other platforms like PhoneGap or Titanium mobile are already there and well accepted by the web design community. Additionally, as WAC claims its own application store it is currently unclear if WAC apps could be offered on iOS devices,

because of licensing issues. Developing under PhoneGap or Titanium mobile does not create this issue as these SDKs integrate into the platform specific stores.

	WAC	PhoneGap	Appcelerator Titanium
Platforms	n/a (depending on runtime support)	5 (6)	2 (3)
SDKs	Eclipse-based	Platform-specific	Integrated
Runtime needed	Yes	No	No
Native UI controls	No	No	Yes
Documentation & Community	Fair	Good	Good

Table 1. Comparison of WAC, PhoneGap and Titanium Mobile

4. Analysis of WAC APIs and SDKs

At the time of writing, there are three different SDKs for WAC widgets. One is provided directly by the WAC consortium, the two others are from Obigo (see reference Obigo) and Aplix (see reference Aplix). All three runtimes are similar tools that are based on the Eclipse platform and run on Android devices. In our opinion Eclipse may not be the best platform though. Being certainly a very powerful platform, it might be too complex for relatively simple widget projects and for standard web developers (which should also be attracted by the WAC initiative). A simpler and more tailored, graphical editor like for the ARES project (see reference ARES) might be better suited.

4.1 WAC standardization timeline

WAC 1.0 (December 2010)

- Based on subset of JIL 1.2.2
- Supports following W3C standards:
  - HTML 4.01, xHTML 1.1, CSS 2.1, SVG Tiny 1.2, Widget Packaging and Configuration, MediaQueries
  - Javascript
- Widget Security: based on W3C Widgets 1.0 (Digital Signatures)
- Handset APIs:
  - Accelerometer, Address Book, Application Launcher, Audio, Camera, Messaging (sending only) and Location

WAC 2.0 (aka "Waikiki", Jan 2011)

- Supports following W3C standards:
  - HTML 5 content parsing plus input element, canvas element, canvas 2D context, audio element, video element, contenteditable attribute
  - Javascript 1.5 (and JSON)

- CSS 2.1 and parts of CSS 3 (transforms, transitions, ...)
- DOM, XMLHttpRequest
- Support for popular libraries, e.g. jQuery
- Improved Security:
- Multi-level authentication/signatures: AppStores, Clients, Runtimes, Widgets...
- Policy-based access control
- Handset API enhancements:
- more methods per API, orientation, filesystem, calendar, tasks...

WAC 3.0 planned features (Sep 2011)

- Extended range of developer tools
- Billing support (+ in-widget billing)
- User identification
- Network APIs according to OneAPI specifications
- Advertising
- Feature phones

Technical details on WAC 3.0 are not available at the time of document writing (July 2011). Regarding user identification, Aepona presented a Message flow including prototype at the Mobile World Congress 2011 in Barcelona (based on OneAPI), but did not clearly identify the API, as the GSMA OneAPI suggests OAUTH.
Media transport or streaming is out of scope of WAC, SQLite is not supported and also ciphering is left to the developer to implement (not considering the possibility of communication with https, which is supported by the runtime). Cooperative multitasking is a must for WAC 2.0 runtimes but limited to the capabilities of the host system.

4.2 Creating a WAC widget

In the following we would like to go through the necessary steps to create a small widget that displays a camera preview window and offers taking pictures by pressing a button.
A simple WAC widget basically consists of three files, namely a HTML file that is the entry point of the widget and defines the layout, a Javascript file defining the logic, and, finally, a CSS file that specifies the visual properties of the widget.
We would like to start first with editing the HTML file (shown in Figure 2):

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>CameraApp</title>
<link href="CameraApp.css" rel="stylesheet" type="text/css"/>
<script type="text/javascript" src="CameraApp.js"></script>
</head>

<body onload="preview();" onUnload="destroy();">
    <object id="previewWindow" width="400" height="400"></object></td></tr></table>
    <div style="background-image:url(Default.png)"
                onclick="takePicture()" id="start" class="cameraButton"></div>
</body>

</html>
```

Fig. 2. HTML code for a WAC widget

The head of the file points to the CSS and Javascript files we will edit later. In the body we define the preview() method to be loaded upon runtime. We also define a preview windows and a camera button that launches a takePicture() method.
Figure 4 shows the Javascript code for the widget. The preview() method uses the WAC API to create a camera object. The window of that object is then assigned to the preview window defined in the HTML file before. The takePicture() method finally captures the image and stores it on the device. It makes use of a helper function generateFileName() that creates a unique filename based on the current DateTime. This was needed because the picture could not be stored if a file with the same name already exists.

```
#front {
    position: absolute;
    background-image: url("Default.png");
    top: 0px;
    left: 0px;
    width: 235px;
    height: 158px;
}

.cameraButton {
    position: relative;
    top: 300px;
    left: 5%;
    width: 235px;
    height: 158px;
}
```

Fig. 3. CSS code for a WAC widget

Finally, the CSS file (Figure 3) is relatively straightforward and just defines the styles for the button and the background. The SDK would now pack these three files together with content information in a zip file with filename extension wgt. The WAC app is finished and ready to be deployed (however, it should be noted, that a Publisher ID/certificate is needed for distribution in application stores).

4.3 Comparison with native App development frameworks

4.3.1 iOS

The primary programming language is Objective-C, the primary SDK is Xcode in combination with Interface builder. The SDK is free, but development is only possible on Mac Computers. iOS was designed to meet the needs of mobile environment, where users' needs are different than for a desktop system.
The iOS SDK contains the code, information, and tools needed to develop, test, run, debug, and tune applications for iOS. Xcode tools provide the basic editing, compilation, and debugging environment for code writing. Xcode is also the launching point for testing applications on an iOS device, and in iOS Simulator, a platform that mimics the basic iOS environment but runs on a local Macintosh computer.
In the meantime, iOS has become a mature mobile runtime environment for apps. This also applies to Xcode, which is very well tailored to the needs of developing function rich, highly appealing and simple to use mobile applications for the user. The possibilities of graphical user interface editing while separating the functional implementation in Objective-C gives some advantages compared to WAC widgets. This is because Xcode addresses the true

nature and power of Objective-C and COCOA UI libraries whereas the WAC SDKs try to mimic code writing development where they should address rather graphical oriented widget design.

```
var prevWindow;
var mCamera;

function preview() {
    prevWindow = document.getElementById("previewWindow");
    mCamera =  Widget.Multimedia.Camera;
    mCamera.setWindow(prevWindow);
}

function destroy() {
    mCamera.setWindow(null);
}

function generateFileName() {
    var currentTime = new Date();
    var day = currentTime.getDate();
    var month = currentTime.getMonth() + 1;
    var year = currentTime.getFullYear();
    var hours = currentTime.getHours();
    var minutes = currentTime.getMinutes();
    var seconds = currentTime.getSeconds();

    if (month < 10){
        month = "0" + month;
        }
    if (day < 10) {
        day = "0" + day;
        }
    if (minutes < 10) {
        minutes = "0" + minutes;
        }
    if (seconds < 10) {
        seconds = "0" + seconds;
        }
    fileName = year + "-" + month + "-" + day + " " + hours + "." + minutes + "." + seconds;
    return fileName;
}

function takePicture() {
    try{
    mCamera.onCameraCaptured= function(fullpath) {
    alert("Image path is:"+fullpath);
    };
    mCamera.captureImage("/sdcard/DCIM/Camera/" + generateFileName() + ".jpg",false);
    }
    catch(e)
    {
    alert("inside catch"+e.message);
    }
}
```

Fig. 4. Javascript code for a WAC widget

4.3.2 Android

The architecture of Android is based on the Linux kernel 2.6. It is responsible for memory management, process management and network communications. It also provides the hardware abstraction layer for the rest of the software and device drivers for the system.
Other important components are based on the architecture developed by Sun Microsystems (now Oracle), namely, the Java technology-based virtual machine Dalvik and its Android

Java class libraries. In order to program Android applications, the development system (m3-rc20a, published in November 2007) contains 1448 Java classes and 394 interfaces, of which 511 classes and 128 interfaces are Android-specific.
Applications for the Android platform are written exclusively in Java, taking advantage of the wide spread expertise on Java within the designer community. For speed-critical tasks Android apps may take advantage of many C or C++ written, native libraries under the hood. The catalogue includes codecs for media playback, a web browser based on WebKit, a database (SQLite) and an OpenGL based 3D graphics library.
In order to develop programs for Android, a recent Java SDK and also the Android SDK is required (e.g. Eclipse). First, the source code written in Java is translated with a normal Java compiler and then adapted by a cross-assembler for the Dalvik VM. For this reason, programs can in principle be created with any Java development environment.

4.3.3 WebOS

WebOS is a multitasking operating system for smart phones with a Linux-based kernel. Multiple applications may open and run simultaneously and can be browsed via a live preview, even videos may run in this preview mode. As iOS and Android, it is operated by finger gestures on a touch screen.
PIM data is not only stored on the device, but always synchronized with Internet services like, e.g. Gmail. Using a technology called Synergy, all information, e.g. from different calendar systems such as Exchange and Google Calendar, is summarized in a single application. Synergy links contacts, calendar events and e-mails from various sources (including the above-mentioned Exchange, GMail, Hotmail, Yahoo, Facebook).
With the HP app catalogue, the manufacturer offers (similar to iPhones and Android phones) an online service that allows to download and/or buy applications (Apps) for WebOS phones.
Palm HP offers several ways to develop applications for WebOS:

- SDK

The Mojo Application Framework SDK allows applications to be developed with HTML5, CSS and JavaScript. The SDK needs to be installed on a desktop computer.

- Ares

Ares is a new development environment for HP WebOS, which is now released in version 1.0. With Ares it is possible to develop applications directly in the browser. An installation of the SDK is no longer necessary. To test the applications in Ares, an emulator is integrated. Ares works with current Web browsers such as Firefox, Safari and Chrome.

- Plug-in Development Kit

Since March 2010 a "Plug-In Development Kit" offers the possibility of using C or C++ code in applications, which should ease porting of external applications from other platforms.
WebOs is a very modern and well thought through approach, which - in contrast to WAC – is taking full advantage of the introduced web technologies. It uses HTML5, CSS and Javascript to interact and synchronize with cloud and Web2.0 services whereas WAC simply offers HTML GUI rendering facilities, which are not even the strength of HTML. WAC should follow WebOS as it does with its Mojo application framework.

5. Recommendations for industry

The stakeholders to be kept in mind while offering application frameworks, SDKs, app stores and runtimes are mainly:

- Application developers
- Mobile device users
- Network operators

WAC has well considered the needs of network operators and partially the need of potential users, but so far it does not optimally attract and support app developers. As WAC is not primarily a platform for coding experts it should focus on the needs of web designers: a graphical toolset, a simple to use Javascript library for multi touch UIs, dpi resolution based GUIs, etc.

The registration process for becoming a WAC designer is currently rather cumbersome and introduces administrational hurdles not known for designers who signed up as an app developer at Apple or Google Android. With the App Market you simply sign up with your Google account and pay a minimal amount for the permission to submit apps. There is no need to fax or mail paper documents as required for WAC. Admittedly Apple or Google Android apps are limited with respect to using network APIs, hence some overhead on WAC-side for this additional functionality is understandable. Still the processes should be simplified.

In short, we summarize the key aspects that should be considered for the future WAC roadmap:

- Provide an SDK including a graphical toolset, like Dreamweaver or the like
- Integrate a framework á la SenchaTouch
- Offer an online SDK including development life-cycle support, like it is done for WebOS
- And in addition to the bullet above: Offer network functionality testing via network sandbox once WAC 3.0 is out
- Introduce user controlled device functionality access policies: In addition to certificate/configuration based functionality and privacy control interactive means of allowing or rejecting access to user data should be enabled
- Support for SQLite
- Support a dpi based solution for screen resolution scaling should be introduced. If this is not possible an automatic CSS adaptation by the device runtime should be applied. Currently it is not possible for WAC widgets to automatically adapt to different screen resolutions on different devices. In the worst case this means that for each resolution format a different widget needs to be written, which would be inacceptable for developers. How should this be represented in the WAC store?

And last but not least, in the times of cloud computing it should be considered to integrate the WAC APIs to any browser runtime so that the widgets (and their date) can be stored in the cloud and downloaded and executed on demand just as it is done for plain web sites or Web 2.0 services. This “running in the cloud” mechanism could be the main differentiator to prefer WAC widgets over native app development like its done on iOS, Android or Blackberry.

Finally, WAC runtimes need to be offered for all major platforms not only for Android. Simply because it is so easy to develop and deploy such software on Android phones, other platforms such as Blackberry and WebOS (and iOS?) should be supported as well.

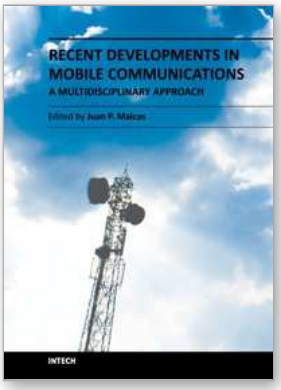
The WAC activity is a promising start to enter the application market place. The operator and device agnostic approach together with the opportunity to offer Telco assets in applications might be very attractive to developers, users and network operators.
In our opinion the WAC activity in its current form (April 2011), however, will need some tuning to become successful. Currently it is not as innovative as its competitors (AppStore and Android Market) and does not fully leverage additional opportunities a Telco operator could give to users and developers.
The WAC 3.0 specification (due in September 2011), however, that supports network APIs for widget development may be an important differentiator from its competitors. Additional focus should be put on: A more tailored and user-friendly web-design type of SDK, a simpler sign up process for developers, more supported platforms or even the deployment of WAC APIs via mobile browsers and better support for state of the art user interface technologies.

6. Acknowledgment

The authors would like to thank the Telecommunications Research Center Vienna (FTW), partners of the APSINT project managed by FTW, and the COMET (Competence Centers for Excellent Technologies) programme of the Austrian Government for supporting the APSINT project.

7. References

APSINT http://www.ftw.at/research-innovation/projects/apsint
WAC-Dev http://www.wacapps.net/
WAC open collaboration http://www.wacapps.net/
WebOS http://developer.palm.com/
ARES https://ares.palm.com/Ares/about.html
PhoneGap http://www.phonegap.com
Titanium Mobile http://www.appcelerator.com/products/titanium-mobile-application-development/
Obigo http://www.obigo.com
Widget SDK fro Aplix http://widgetsdk.org/



Recent Developments in Mobile Communications - A Multidisciplinary Approach

Edited by Dr Juan P. Maícas

ISBN 978-953-307-910-3
Hard cover, 272 pages
Publisher InTech
Published online 16, December, 2011
Published in print edition December, 2011

Recent Developments in Mobile Communications - A Multidisciplinary Approach offers a multidisciplinary perspective on the mobile telecommunications industry. The aim of the chapters is to offer both comprehensive and up-to-date surveys of recent developments and the state-of-the-art of various economical and technical aspects of mobile telecommunications markets. The economy-oriented section offers a variety of chapters dealing with different topics within the field. An overview is given on the effects of privatization on mobile service providers' performance; application of the LAM model to market segmentation; the details of WAC; the current state of the telecommunication market; a potential framework for the analysis of the composition of both ecosystems and value networks using tussles and control points; the return of quality investments applied to the mobile telecommunications industry; the current state in the networks effects literature. The other section of the book approaches the field from the technical side. Some of the topics dealt with are antenna parameters for mobile communication systems; emerging wireless technologies that can be employed in RVC communication; ad hoc networks in mobile communications; DoA-based Switching (DoAS); Coordinated MultiPoint transmission and reception (CoMP); conventional and unconventional CACs; and water quality dynamic monitoring systems based on web-server-embedded technology.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Zeiss Joachim, Davies Marcin and Pospischil Günther (2011). The Role of WAC in the Mobile Apps Ecosystem, Recent Developments in Mobile Communications - A Multidisciplinary Approach, Dr Juan P. Maícas (Ed.), ISBN: 978-953-307-910-3, InTech, Available from: http://www.intechopen.com/books/recent-developments-in-mobile-communications-a-multidisciplinary-approach/the-role-of-wac-in-the-mobile-apps-ecosystem

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821