Middleware provides various transparencies (HW, OS, location, fault, ..) for apps.

APIs for: RPC, messaging, transactions, session management, storage, directories, trading, etc.
The Hourglass

- diverse applications
- divergence
- transport layer (TCP/IP)
- convergence
- diverse physical layers
Mobile Platforms

- Collections of central services and libraries with both reactive and proactive functions

- APIs typically logically centralized

- Distributed between elements of the environment
  - Multi-tier client-server
  - Peer-to-peer
  - Hybrids

- The platform running on the mobile terminal and the characteristics of the device determine how service is rendered for the end user
Wireless and Cloud

- Wireless hop is the limiting factor
  - Bandwidth, connectivity, reachability, tail energy, costs

- Server side scalability can be achieved by using traditional solutions:
  - clusters, caching, geographical distribution, load balancing, data centers

- Cloud computing
  - Integration, offloading
  - Web apps vs. native apps
Mobile Service Development

- The mobile landscape is fragmented
  - Heterogeneous device base
  - Many different wireless technologies
- The situation is challenging for the developer
  - Many APIs
    - Open vs. private APIs
  - Many middleware platforms
  - APIs evolve over time
- Current challenge of the industry pertains to improving the development processes
Mobile middleware aims to support the development, deployment, and execution of distributed applications in the heterogeneous and dynamic mobile environment.

The goals for mobile middleware include adaptability support, fault-tolerance, heterogeneity, scalability, and context-awareness.

The industry solution to these challenges has been to create middleware platforms.

A platform collects frequently used services and APIs under a coherent unified framework.
Platforms

- **2009**
  - Java Micro Edition (Java ME)
  - iOS
  - Symbian and Series 60
  - Windows Mobile
  - Linux Maemo (MeeGo)
  - Android
  - BREW
  - WAP

- **2012**
  - iOS
  - Android
  - Windows Phone 8
  - HTML5 web apps
Application Trends

- WP8
  - Native apps, cloud integration

- iOS
  - Native apps, cloud integration
  - Potential for Web apps

- Android
  - Native apps, cloud integration

- WebOS
  - Web apps with HTML5
  - Obsolete (open source)

- Blackberry
  - Native and Web apps
Challenges

- Fragmentation is a major problem
  - device-level fragmentation
  - standard fragmentation
  - implementation fragmentation

- Energy consumption
  - Modelling: where is the energy going
  - Optimization: how to improve things

- Security is also a problem
  - Sandboxed environments and privileged operations require certification
  - Certification is difficult for developers
  - Current trend is towards application stores and more lightweight certification processes
  - No malware for iOS, plenty for Android
Examples

- Classical examples
  - WAP
  - Java ME
  - Symbian
  - MAEMO / MeeGo

- Current Platforms
  - Windows Phone 8
  - iPhone
  - Android
  - Web apps
WAE

- **Wireless Application Environment (WAE)**
  - A suite of protocols and specifications for optimizing data transfer for wireless communication

- **WAP stack**
  - Focus on binary transmission
  - **WSP (Wireless Session Protocol)**
    - HTTP replacement, “compressed”
  - **WTP (Wireless Transaction Protocol)**
    - Request/response, more efficient than TCP
  - **WTLS (Wireless Transport Layer Security)**
    - Based on TLS, may not be end-to-end with a gateway
  - **WDP (Wireless Datagram Protocol)**
    - UDP replacement
Web Access with Gateway

Client → encoded request → Encoders

Gateway

Gateway

Server

Protocol Gateways

HTTP Server

CGI,...
Web Access

- Data transformation
  - WAP gateway performs data transformation between WML (or XHTML) and HTML

- Data compression
  - Technique are used for dealing with images and other graphics

- Adaptability
  - User profile and device characteristics are stored in the WAP gateway

- Security
  - Secure Enterprise Proxy (SEP) using 128-bit encryption in WAP 1.2

- Service discovery and mobility support
  - WAP’s “walled garden” – WAP gateways are provided by ISP such as AOL
WAE: current status

- WAP Forum now in OMA (Open Mobile Alliance)
- WAP 2.0, is a re-engineering of WAP using a cut-down version of XHTML with end-to-end HTTP
- Gateway and custom protocol suite is optional.
- WAP used by many handsets
  - 1.2 version introduced WAP Push (typically using an SMS message)
- Typically versatile networking stacks with also IPv6 support
Java Micro Edition (Java ME)

- Java for consumer electronics and embedded devices
- A virtual machine and a set of APIs
- Configurations and profiles
  - Configurations
    - two-low level APIs and optimized VMs
      - CDC, CLDC
  - Profiles
    - API specification on top of a configuration for complete runtime
    - CLDC: MIDP
    - CDC: Foundation, Personal Basis, Personal
    - Profiles defined using Java Community Process (JCP)
Important JSRs

- 75 File Connection and PIM
- 82 Bluetooth
- 120 Wireless Messaging API (WMA)
- 135 Mobile Media API (MMAPI) Audio, video, multimedia
- 172 Web Services
- 177 Security and Trust Services
- 179 Location API
- 180 SIP API
- 184 Mobile 3D Graphics
- 185 Java Technology for the Wireless Industry (JTWI) General
- 205 Wireless Messaging 2.0 (WMA)
- 211 Content Handler API
- 226 SVG 1.0
- 229 Payment API
- 234 Advanced Multimedia Supplements (AMMS) MMAPI extensions
- 238 Mobile Internationalization API
- 239 Java Bindings for the OpenGL ES API
- **248 Mobile Service Architecture General**
  - Collects useful specifications
- 256 Mobile Sensor API
- 287 SVG 2.0
MIDP 3.0

- MIDP 3 specified in JSR 271 will specify the 3rd generation mobile APIs.
  - AMS (Application Management System)
  - Multitasking
  - Provisioning and OTA
  - Shared libraries
  - Security and access control
  - Service framework
  - Inter-MIDlet communication
  - User Interface improvements

- A key design goal of MIDP3 is backward compatibility with MIDP2 content

- Approved in Dec, 2009. Not supported by current phones.
Symbian

- OS for handheld devices with limited resources
- User interface framework
- APIs (C++)
- Tools
- Operating System
  - Pre-emptive, multitasking, multithreading, memory protection
  - Event-based, active objects
  - Memory conservation, reliability, CPU optimizations
Software Components

- **Kernel**
  - Manages and controls access to hw
  - Hw-supported privileges, kernel mode

- **Application**
  - Program with a user interface
  - Runs in user mode in its own process

- **Server**
  - Program without a user interface
  - Manages resources, provides interface to clients
  - File server, window server, comms, ..

- **Engine**
  - Application part that manipulates data, typically separate DLL
Key layers

- The Symbian OS System Model contains the following layers:
  - UI Framework Layer.
  - Application Services Layer.
  - Java ME.
  - OS Services Layer: generic OS services, communications services, multimedia and graphics services, connectivity services.
  - Base Services Layer.
  - Kernel Services and Hardware Interface Layer.
MAEMO and MeeGo

- Open Source development platform for Nokia Internet Tablets and other Linux-based devices
- Previously MAEMO, integrated with Intel’s Moblin to create MeeGo
- MeeGo 1.1 for Atom and ARM
- Nokia is no longer developing MeeGo
- www.meego.com
MeeGo

- Versatile platform for mobile computing
- Linux-based, Qt is the key development environment
- The MeeGo includes a set of components called the content framework to gather and offer user metadata to application developers
MeeGo

- Hardware Adaptation Software
- MeeGo Kernel
- Comms Services
- Internet Services
- Visual Services
- Media Services
- Data Mgmt
- Device Services
- Personal Services
- Handheld UX
- Netbook UX
- Other UX
- MeeGo UI Toolkit (Qt)
- GTK/Clutter
- UX
- Middleware
- OS Base
- Hardware Adaptation Software
Qt

- Qt is a cross-platform application framework
  - Rapid creation of GUIs
- For Linux and Symbian application development
- The Qt API is implemented in C++ and most Qt developers use C++ (bindings for other languages)
- Extensions for using mobile functionality from within Qt code
  - access points, alarms, audio, calendar, camera, contacts, installer, landmarks, location, media, messaging, profile, resource access, sensor, settings, system information, telephony, vibration, other utilities etc.
Windows Mobile

- Windows Mobile 6 was released by Microsoft at the 3GSM World Congress 2007 and it came in three flavours
  - standard version for smartphones
  - a version for PDAs with phone functionality
  - a classic version for PDAs without phone features.

- Based on the Windows CE 5.0 operating system and has been designed to integrate with Windows Live and Exchange products.

- Software development for the platform is typically done using Visual C++ or .NET Compact Framework.
The .NET Compact Framework CLR is made up of the following three components:
- class libraries
- execution engine
- platform adaptation layer

The purpose of the class libraries is to provide a basic set of classes, interfaces, and value types
- the foundation for developing applications in .NET.

The execution engine is the core component of the CLR. It provides the fundamental services needed for executing managed code.
- The execution engine includes components such as a JIT compiler, a class and module loader, and a garbage collector.

The PAL layer maps calls from the execution engine to the functions of the underlying operating system.
CLR

Source: http://en.wikipedia.org/wiki/Common_Language_Runtime
WP7 and WP8

- Windows Mobile 7 was announced in 2010
- Limited APIs for third party applications
- WP store for applications
- Development: C# and XNA, Silverlight 4, VB, on-going API work
- WP7 used CE kernel, WP8 uses NT kernel
Development

- Simple applications with Silverlight
  - XML-based UI declaration and C# code
  - Executed in .NET CLR
  - UI widgets, event based input
  - Similar to Android Java and XML

- Games with XNA (WP7)
  - C# code executed in .NET CLR
  - Direct 3D
  - Xbox

- SDK is free of charge
  - Microsoft Windows Phone Developer Tools
  - Windows 7 required
  - Add-on for Visual Basic
Android

- Mobile OS and application platform from Google
- Open Handset Alliance
- Linux kernel
- Open Source
- Uses Java to build applications (Java SE class library parts from Apache Harmony project)
- Optimized virtual machine called "Dalvik"
  - Runs .dex files (derived from .class or .jar)
  - Relies on underlying system for process isolation, memory mng, and threading
- Independent of Sun and JCP
- Java APIs for basic comms, location, SQLite, OpenGL, SyncML
Android History

- 2005
  - Google acquires startup Android Inc.
  - Work on Dalvik VM starts
- 2007
  - Open Handset Alliance announced
  - Early work on Android SDK
- 2008
  - SDK 1.0 released
  - Android released open source (Apache License)
- 2009
  - SDK 1.5-2.1
- 2010
  - Nexus One
  - SDK 2.2 (Froyo)
    - Flash support, tethering
  - SDK 2.3 (Gingerbread)
    - UI update, system-wide copy-paste
- 2011
  - SDK 3.0-3.2 (Honeycomb) for tablets only
    - New UI for tablets, support multi-core processors
  - SDK 4.0.x (Ice Cream Sandwich), 4.1/4.2 (Jelly Bean)
    - Changes to the UI, Voice input, NFC
Android II

- Android includes a set of C/C++ libraries used by various components of the Android system. The capabilities of these libraries are exposed to developers through the Android application framework APIs.
- The core libraries include:
  - System C library, a BSD-derived implementation of the standard C system library (libc), adapted for embedded Linux-based devices.
  - Media Libraries based on PacketVideo’s OpenCORE.
  - Surface Manager that manages access to the display subsystem and seamlessly renders 2D and 3D graphic layers from multiple applications.
  - LibWebCore, a web browser engine which powers both the Android browser and an embeddable web view.
  - SGL, the underlying 2D graphics engine.
  - 3D libraries, an implementation based on OpenGL ES 1.0 APIs.
  - FreeType, bitmap and vector font rendering.
  - SQLite, a lightweight relational database engine available to all applications through the framework API.
Android Fragmentation

- For example: 7 API levels in active use starting from Android 1.6 to 3.0
- APIs are forward compatible
- Device features vary
- App must specify requirements
Application model

- Application components classes
  - Activity: UI view
  - Service: background component
  - Content provider: shared app data
  - Broadcast receiver: reacting to system events

- Component-level garbage collection
Android SDKs

- Java SDK
- Native SDK
- Eclipse IDE for Linux, Mac, Windows
- Visual UI builder

More information:
developer.android.com/sdk/index.html
http://developer.android.com/guide/developing/building/index.html#detailed-build
Android: Key Components

- **AndroidManifest.xml.** This XML document contains the configuration that tells the system how the top-level components will be processed.

- **Activities.** An activity is an object that has a life cycle and performs some work. An activity can involve user interaction. Typically one of the activities associated with an application is the entry point for that application.

- **Views.** A view is an object that knows how to render itself to the screen.

- **Intents.** An intent is a message object that represents an intention to perform some action.
  - In Android terminology, an application has an intent to view a Web page, and generates an Intent instance in order to view the Web page using a URL. The Android system then decides how to implement the intent. In this case, a browser would be used to load and display the Web page.

- **Services.** A service is code that runs in the background. The service exposes methods for components. Other components bind to a service and then invoke methods provided by using remote procedure calls.

- **Notifications.** A notification is a small icon that is visible in the status bar. Users can interact with this icon to receive information.

- **ContentProviders.** A ContentProvider provides access to data on the device.
Manifest

- AndroidManifest.xml defines the app
  - Activities and components
  - Device features
  - Intent filters, actions to associate with
  - Permissions needed / required
  - Minimum API support
When the first of an application's components needs to be run, Android starts a Linux process for it with a single thread of execution.

- Can spawn additional threads
  - Thread class, Looper, Handler, ...

- RPC for interprocess communications
  - Java-based IDL: AIDL
Activity states

- An activity has four main states:
  - Active. An activity is active when it is in the foreground of the screen and at the top of the activity stack.
  - Paused. An activity is paused when it has lost focus, but is still visible. A paused activity is alive, but can be destroyed by the system if memory needs to be freed.
  - Stopped. An activity is stopped when it is obscured by another activity. The stopped activity retains its state, but it is no longer visible and can be destroyed by the system when memory is needed.
  - Destroyed/Inactive.

- If an activity is paused or stopped, the system can remove the activity from memory. This can happen in two ways, the system can ask the application to finish or simply destroy the process.
User navigates back to your Activity

Activity starts

onCreate()

onStart()

Activity is running

onResume()

New Activity is started

onPause()

onStop()

Your Activity is no longer visible

onDestroy()

Activity is shut down

onRestart()

Your Activity comes to the foreground

onRestart()

Your Activity comes to the foreground

Process is killed

Other applications need memory
Activity Manager

- An activity represents app UI (typically single screen)
  - Stack of activities (foreground, background)
- Each user interface screen is represented by an Activity class.
- Each activity has its own **life cycle**.
- Activity uses Intent object to jump between them.
Intent and Intent filters

- Intent activates activities, services, and broadcast receivers.
- Components can register to receive intents (also broadcast)
- Examples
  - An incoming phone call
  - Change in battery level
  - App specific update
- Intent can be used in explicit way or implicit way.
  - Explicit specifies the component (Java class)
  - The implicit way depends on parameters: Action, Data (url and MIME type), Category
    - Register via IntentFilters in AndroidManifest
    - BroadcastReceiver can be registered in app as well
Content manager

- Manages data
- Client+server architecture.
- APIs for query, delete, update and insert rows
- **Content Resolver** provides API interface for applications.
- **Content Provider** is the server managing the DB tables and database content for different applications.
  - Data sharing across apps
Content manager

- URI identifies the data or the table

- A: Standard prefix indicating that the data is controlled by a content provider.

- B: The authority part of the URI; it identifies the content provider.

- C: The path that the content provider uses to determine what kind of data is being requested.

- D: The ID of the specific record being requested.
Security and permissions

- Security between applications and the system is enforced at the process level through standard Linux facilities
- Inter-component communications security enforced with a reference monitor (in Android mw)
  - Manifest file, intent permissions
- Application cannot disrupt other applications, except by explicitly declaring the permissions for it
- Each Android package is given its own unique Linux user ID
Net and Telephony

- Standard Java networking APIs

- Telephony APIs (android.telephony)
  - Send / receive SMS
  - Network info and parameters
  - Current device state (connected, idle, active)
Media Framework

- Android use OpenCore as core component of Media framework
- OpenCore supports MP3, AAC, AAC+, 3GPP, MPEG-4 and JPEG,
Media Framework
Media Framework

- Example:
  - `MediaPlayer mp = new MediaPlayer();`
  - `mp.setDataSource(PATH_TO_FILE);`
  - `mp.prepare();`
  - `mp.start();`
Media Framework

- OpenCore lib has a C/S Architecture.
- MediaPlayer invoke JNI to manipulate client.
- The client request to the server to control hardwares.
Media Framework

Applications
- Application

Application Framework
- MediaPlayer
- JNI

Libraries
- MediaPlayer
- Binder IPC
- AudioFlinger
- Dynamic load
- libaudio.so
- ALSA

Linux Kernel
- Kernel Driver
## Summary

<table>
<thead>
<tr>
<th>Feature</th>
<th>Android/Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>Java, native code with JNI and C/C++</td>
</tr>
<tr>
<td>Network scanning</td>
<td>Yes</td>
</tr>
<tr>
<td>Network interface control</td>
<td>Limited</td>
</tr>
<tr>
<td>Background processing</td>
<td>Yes (services)</td>
</tr>
<tr>
<td>Energy and power monitoring and control</td>
<td>Yes</td>
</tr>
<tr>
<td>Memory management</td>
<td>Yes</td>
</tr>
<tr>
<td>Persistent storage</td>
<td>Yes</td>
</tr>
<tr>
<td>Location information</td>
<td>Yes</td>
</tr>
<tr>
<td>HTML 5</td>
<td>Yes, support depends on version</td>
</tr>
<tr>
<td>SIP API support</td>
<td>Limited</td>
</tr>
<tr>
<td>Open Source</td>
<td>Yes</td>
</tr>
<tr>
<td>3rd party application installation</td>
<td>Certificate, Android store</td>
</tr>
<tr>
<td>Level of fragmentation</td>
<td>Some fragmentation</td>
</tr>
</tbody>
</table>
iOS

- iOS is a mobile operating system developed by Apple Inc. for their iPhone, iPod touch, and iPad products.

- The OS is derived from Max OS X and uses the Darwin foundation.

- Darwin is built around XNU, a hybrid kernel that combines the Mach 3 microkernel, various elements of Berkeley Software Distribution (BSD) Unix, and an object-oriented device driver API (I/O Kit).
iOS development

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

- Objective C and Cocoa Touch are used by developers
  - Objective C is a strict superset of ANSI C
  - Single inheritance, dynamic runtime, no garbage collection on iOS devices

- An universal application determines the device type and then uses the available features based on conditional statements.
Layers

- CoreOS
  - Threading (POSIX), Networking, File system, service discovery (Bonjour, DNS), memory management, math support
  - External accessory (Bluetooth), Security Framework, Accelerate framework

- Core Services
  - SQLite, iCloud storage
  - Grand Central Dispatch (GCD): management of concurrent tasks
  - In-app purchase, XML support
  - Object-oriented abstractions for networking
  - Many frameworks: media, location, telephony, store kit, system configuration
Layers: Cocoa Touch

- UIKit framework
  - Storyboards, multi-touch, camera, sensors, ...
- Document support (UIDocument for data management)
- Multitasking
- Printing and file sharing, P2P services
- Data protection
- Push notification
- Gesture recognizers
- Game Kit
- Advertisement framework
The iPhone OS’s user interface is based on *multi-touch gestures*. Interface control elements consist of sliders, switches, and buttons.

Interaction with the OS includes gestures such as swiping, tapping, pinching, and reverse pinching.

Additionally, using internal accelerometers, rotating the device on its y-axis alters the screen orientation in some applications.
GUI ("Aqua")

API
- Carbon
- Quick-time
- BSD
- Classic
- Cocoa touch
- Application Services
  - Quartz
  - OpenGL
  - PrintCore
  - ...
- Core Services
  - Core foundation
  - Core services
  - non-GUI API...

Core OS ("Darwin")
- System utilities
- Kernel ("xnu")
  - File systems
  - Networking NKE
  - POSIX
  - 1 / 0 kit
  - Drivers

Hardware
Versions

- iOS 1 Web apps
- iOS 2 introduced the App Store
- iOS 3 single tasking, new features
- iOS 4 multitasking, FaceTime videoconferencing, iBooks, iAd, in-app purchases (apps can run in the background)
- iOS 5 cloud integration
iPhone OS 4

- The iPhone OS 4.0 was announced in April 2010 and it supports multitasking for 3rd party applications.
- The key design principle is to offer APIs for specific background operations in order to be able to optimize overall system performance.
- For example VoIP applications will be able to receive calls in the background.
- Third party push servers are supported for sending notifications to applications.
iOS4 multitasking

- The new iPhone multitasking-specific APIs includes
  - support for background audio play
  - VoIP
  - location services
  - task completion, and
  - fast application switching

- If you know the task then you can optimize scheduling
Application model

- **Mac OS X model**
  - Traditional main() starts UI event loop
  - View controller reacts to events
  - View objects manage visible objects
  - View controller unserializes views from a .nib file (UI builder)
  - All code is native
- **Patterns**
  - Model-View-Controller
  - Delegate
    - Implement application specific logic
Structure

Model
Data Model Objects
Todo item, calendar, …

Controller
UI application
Event loop

Application delegate
App events

Controller objects
A view controller handles events

View
UIWindow
Views and UI Objects
A view draws content
**XCode**

- Xcode is Apple’s IDE for iOS development
- Interface Builder
  - XML view definitions, .nib files
- Emulator for testing
- Pointer: developer.apple.com
iCloud

- A cloud storage service
  - Documents and app data stored in the iCloud account
  - iCloud key-value data storage

- App must handle exceptions (partial downloads)
## Summary

<table>
<thead>
<tr>
<th>Feature</th>
<th>iPhone OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>Objective-C</td>
</tr>
<tr>
<td>Network scanning</td>
<td>No</td>
</tr>
<tr>
<td>Network interface control</td>
<td>No</td>
</tr>
<tr>
<td>Background processing</td>
<td>No (Yes for 4.0)</td>
</tr>
<tr>
<td>Energy and power monitoring and control</td>
<td>Monitoring since 3.0</td>
</tr>
<tr>
<td>Memory management</td>
<td>Yes</td>
</tr>
<tr>
<td>Persistent storage</td>
<td>Yes</td>
</tr>
<tr>
<td>Location information</td>
<td>Yes</td>
</tr>
<tr>
<td>HTML 5</td>
<td>Yes</td>
</tr>
<tr>
<td>SIP API support</td>
<td>Limited</td>
</tr>
<tr>
<td>Open Source</td>
<td>No</td>
</tr>
<tr>
<td>3rd party application installation</td>
<td>Certificate, Apple AppStore</td>
</tr>
<tr>
<td>Level of fragmentation</td>
<td>Minor fragmentation</td>
</tr>
<tr>
<td></td>
<td>Android Linux</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><strong>Development</strong></td>
<td>Java, native code with JNI and C/C++</td>
</tr>
<tr>
<td><strong>Network scanning</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Network interface control</strong></td>
<td>Limited</td>
</tr>
<tr>
<td><strong>Background processing</strong></td>
<td>Yes (services)</td>
</tr>
<tr>
<td><strong>Energy and power monitoring and control</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Memory management</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Persistent storage</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Location information</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>HTML 5</strong></td>
<td>Yes, support depends on version</td>
</tr>
<tr>
<td><strong>SIP API support</strong></td>
<td>Limited</td>
</tr>
<tr>
<td><strong>Open Source</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Level of fragmentation</strong></td>
<td>Some fragmentation</td>
</tr>
</tbody>
</table>
Discussion

- The current state is fragmented
- Difficult to achieve portability
- Certain patterns are pervasive (model view control and others)
- Solutions?
Web Apps

- Emerging as an alternative to native applications
- Hybrid usage: Web content to native application interfaces
- Web content can partially solved portability issues

Survey: Android Programmers Shifting Toward Web Apps
  - CNet (03/20/12) Stephen Shankland
HTML5

- HTML 5 is the next version of HTML
  - The first public working draft of the specification available in January 2008 and completion expected around 2012

- Improvements
  - Web Socket API, advanced forms, offline application API, and client-side persistent storage (key/value and SQL).

- HTML 5 support divides the platforms.
  - The iPhone platform has a very good support for HTML 5
  - Also Windows Phone and Android support it

http://mobilehtml5.org/
JavaScript access

- The Open Mobile Terminal Platform (OMTP) group defines requirements and specifications that aim towards simpler and more interoperable mobile APIs.
- BONDI defines requirements governing Device Capability access by JavaScript APIs to promote interoperability and security of implementations.
- The 1.1 release of BONDI is compliant with the W3C Widgets: Packaging and Compliance specification.
BONDI Architecture

- Website
- Browser
- Widget
- Widget User Agent

- Web Engine (WebKit, ...)
- JavaScript Extension
- JavaScript API and Access Control
  - Application Invocation, Network, Messaging, Communication Log, Media Gallery, Media Recording, Personal Information, Persistent Data, Location, User Interaction, Device Status, System Events, Policy Management, API Management, Extensions
- Device Capability Access Control
- Operating System
Native

Pros:
* Performance
* Native UI
* App Store distribution.
* Notifications

Cons:
* Single platform.
* Limited user support.

Hybrid

Pros:
* Cross platform
* Native calls
* Offline mode.
* Large user support.
* Mobile device processing leveraged.
* App Store distribution
* Notifications

Cons:
* Some native features may be absent.
* Not suited for very high performance requirement.

Web

Pros:
* Cross platform
* Centralized distribution.

Cons:
* No native API
* No offline mode.
* Processing cannot be leveraged to mobile device.

Source: http://www.developer.nokia.com/Community/Wiki/Cross_Platform_Mobile_Architecture
Libraries/frameworks for Portability

- **Apache Cordova (former PhoneGap)**
  - Use JavaScript to access a native API, like geolocation, camera, storage, etc. using a single API call irrespective of the mobile platform

- **Sench Touch**
  - An HTML5 framework and native api wrappers

- **Xamarin**
  - Xamarin compiler bundles the .NET runtime and outputs a native executable, packaged as an iOS or Android app
System bus

- An asynchronous system-wide event bus is a basic solution for interconnecting various on-device components.

- There is no single standard for this.
  - Android and Java ME use Java-specific events (Android Intent filtering)
  - MeeGo uses D-BUS
  - Palm's WebOS W3C Events

- One particular trend is to utilize URI-based conventions for naming system resources and services.
  - This is extensively used in Android, WebOS and the BONDI architecture. Can be used for iOS as well (URL handlers)
Summary

- Three current smartphone platforms
  - iOS, Android, WP8

- Fragmentation is a current problem
  - Device, standard, implementation

- Closed vs. Open platforms
  - Trust in apps, API access, privileges

- Cross-platform development
  - HTML5, native, or hybrid
  - Middleware support
  - Hybrid is a viable alternative