MOBILE OS ARCHITECTURE TRENDS

Contributors

Xiao-Feng Li
Software and Services Group,
Intel Corporation

Yong Wang
Software and Services Group,
Intel Corporation

Jackie Wu
Software and Services Group,
Intel Corporation

Kerry Jiang
Software and Services Group,
Intel Corporation

Bing Wei Liu
Software and Services Group,
Intel Corporation

The world is flat, because it becomes increasingly mobile, fast, connected, and secure. People expect to move around easily with their mobile devices, keeping close communications with their partners and family, enjoying the versatile usage models and infinite contents, and without worrying about the device and data management. These all put requirements on the mobile devices, of which the mobile OS is the soul. Based on our years of experience in mobile OS design and an extensive survey of current industry situation, we believe there are several commonalities in future mobile OS architecture, such as user experience, power management, security design, cloud support, and openness design. We develop an analysis model to guide our investigation. In this article, we describe our investigations in the trends of mobile OS architecture over the next decade by focusing on the major commonalities. Based on the findings, we also review the characteristics of today's mobile operating systems from the perspective of architecture trends.

Introduction

Mobile OS design has experienced a three-phase evolution: from the PC-based OS to an embedded OS to the current smartphone-oriented OS in the past decade. Throughout the process, mobile OS architecture has gone from complex to simple to something in-between. The evolution process is naturally driven by the technology advancements in hardware, software, and the Internet:

- *Hardware.* The industry has been reducing the factor size of microprocessors and peripherals to design actual mobile devices. Before the form factor size was reduced enough, the mobile device could not achieve both small size and processing capability at the same time. We had either a PC-sized laptop computer or a much weaker personal data assistant (PDA) in phone size. Mobile operating systems for PDAs usually did not have full multitasking or 3D graphics support. Features like sensors, such as accelerometers, and capacitor-based touchscreens were not available in the past mobile operating systems.
- *Software.* With a laptop computer, the software is mainly focused on the user's productivity, where support for keyboard and mouse that have precise inputs are essential. The software for a personal data assistant, as its name implies, helps the user to manage personal data such as contacts information, e-mail, and so on. The mobile operating systems were not designed for good responsiveness or smoothness with a rich user interface (UI) including both touchscreen and other sensors.

- *Internet.* Along with Internet development, especially after Web 2.0, there is abundant information in the network waiting to be searched, organized, mined, and brought to users. People are increasingly living with the Internet instead of just browsing the Web. More and more people are involved in the development, including information contribution, application development, and social interactions. The mobile operating systems cannot be self-contained, but have to be open systems.

The usage model of past mobile devices is limited.

The usage model of past mobile devices is limited. A user mostly runs the device applications for data management and local gaming, only occasionally browses Internet static Web pages or accesses specific services like e-mail. In other words, the possible usages of the device are predefined with the preinstalled applications when the user purchases it. This is largely changed in new mobile devices, where the device is virtually a portal to various usage models. All the involved parties such as service providers, application developers, and other device users continuously contribute and interact through the device with its owner. Figure 1 shows the high-level usage model difference between the past and new mobile devices.

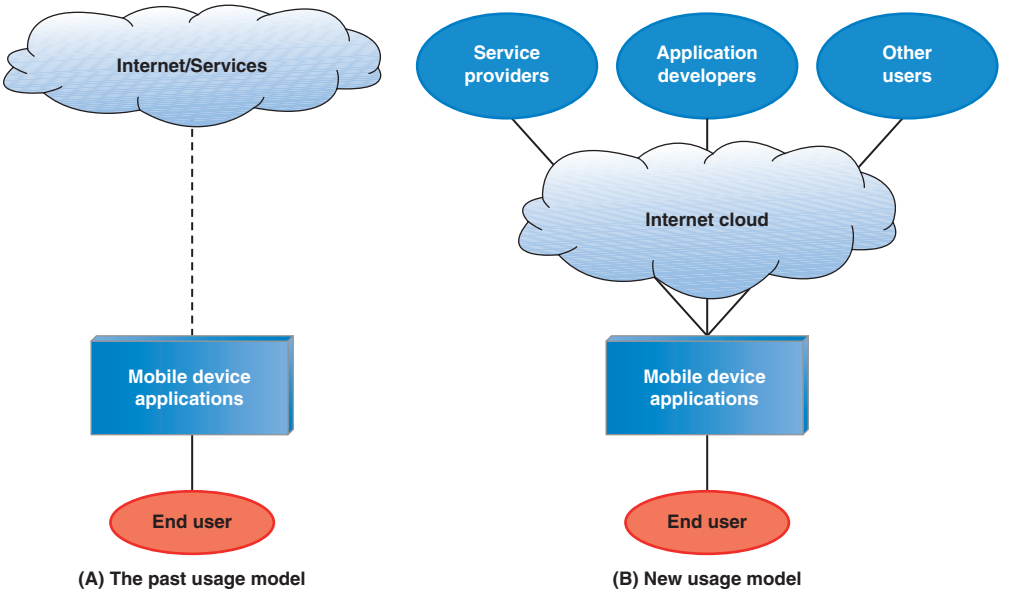

Figure 1: High-level usage models of mobile devices (Source: Intel Corporation, 2012)

The representatives of current mobile operating systems include Apple's iOS* 5.0, Google Android* 4.0, Microsoft Windows* Phone 7.0, and a few others. In terms of their usage models, they share more similarities than differences:

- All of them have a documented software development kit (SDK) with well-defined APIs that enable the common developers to develop applications for these systems.

- All of them have online application stores for the developers to publish and for the users to download applications, such as Apple App Store, Google Play, and Windows Phone Marketplace.
- All of them have a certain level of multitasking and 3D graphics support. Touchscreens and sensors are just no-brainers. Much effort have been spent to make the user interactions smooth and responsive.
- Browsing experience is far beyond static Web pages. HTML5 is becoming the default so as to run Web-based applications.
- All of the operating systems support device-based payment. Together with enterprise applications and private information, system security is always a key concern to the device users.
- As mobile operating systems, one of key design differences from non-mobile operating systems is the focus on battery life. The systems try best to reduce the active power consumption of the device components and put them into idle whenever possible.

"System security is always a key concern to the device users."

The similarities of the current mobile operating systems reflect the advancement trend in hardware, software, and the Internet. Anticipating the trend of mobile operating systems, we believe those areas are the major focuses of the next generation of mobile OS design, including user experience, battery life, cloud readiness, security, and openness. They are actually conflicting targets to a large extent:

- *User experience and battery life.* To achieve best responsiveness and smoothness, the system expects all the hardware resource available to exploit their best capacity. At the same time, to sustain the battery life as a mobile device, the hardware components should be idle whenever possible.
- *Security and openness.* One does not want to expose all of one's system functionalities to external entities, because that puts the system under security threat. On the other hand, without exposing enough system APIs, it is impossible for the developers to create innovative usages.
- *Cloud readiness.* As more and more services and applications are offered from the cloud, it is natural to consider a thin client device model that trusts the cloud and that offloads computations to the cloud. But as of today, the thin client model still has technical challenges in user experience and security.

"As of today, the thin client model still has technical challenges in user experience and security."

In this article, we try to investigate the various aspects of mobile OS design and present our opinions about the future of mobile OS architecture.

The article is arranged as follows. Based on the framework laid out in this section, we use separate sections to discuss the respective topics in text below. They are arranged in user experience, power management, security, openness, and cloud readiness. Finally we have discussions and a summary.

User Experience (UX)

Traditional performance is inadequate to characterize modern client devices. *Performance* is more about the steady execution state of the software stack and

is usually reported with a final score of the total throughput in the processor or other subsystems. *User experience* is more about the dynamic state transitions of the system triggered by user inputs. The quality of the user experience is determined by such things as the user perceivable responsiveness, smoothness, coherence, and accuracy. Traditional performance could measure every link of the chain of the user interaction, while it does not evaluate the full chain of the user interaction as a whole. Thus the traditional performance optimization methodology cannot simply apply to the user experience optimization. It is time to invest in device user interaction optimizations so as to bring the end user a pleasant experience.

"The traditional performance optimization methodology cannot simply apply to the user experience optimization."

User Interactions with Mobile Devices

In a recent performance measurement with a few market Android devices, we found there was a device X behaving uniformly worse than another device Y with common benchmarks in graphics, media, and browsing. But the user perceivable experience with the device X was better than device Y. The root reason we identified was that traditional benchmarks or benchmarks designed in traditional ways did not really characterize user interactions, but measured the computing capability (such as executed instructions) or the throughput (such as processed disk reads) of the system and the subsystems.

Take video evaluation as an example. Traditional benchmarks only measure video playback performance with some metrics like FPS (frames-per-second), or frame drop rate. This methodology has at least two problems in evaluating user experience. The first problem is that video playback is only part of the user interactions in playing video. A typical life cycle of user interaction usually includes at least the following links: "launch player" → "start playing" → "seek progress" → "video playback" → "back to home screen." Yet good performance in video playback cannot characterize the real user experience in playing video. User interaction evaluation is a superset of traditional performance evaluation.

"Good performance in video playback cannot characterize the real user experience."

The other problem is, using FPS as the key metric to evaluate the smoothness of the user interactions cannot always reflect good user experience. For example, when we flung a picture in the Gallery3D application, the device Y had obvious stuttering during the picture scrolling, but the FPS value of device Y was higher than that of device X. In order to quantify the difference of the two devices, we collected the data of every frame during a picture fling operation in the Gallery3D application on both device X and device Y, as shown in Figure 2 and Figure 3 respectively. Every frame's data is given in a vertical bar, where the x-axis is the time when the frame is drawn, and the height of the bar is the time it takes the system to draw the frame. From the figures, we can see that device X obviously has a lower FPS value than device Y, but with smaller maximal frame time, less frames longer than 30 ms, and smaller frame time variance. This means that, to characterize the user experience of the picture fling operation, those metrics like maximal frame time and frame time variance should also be considered.

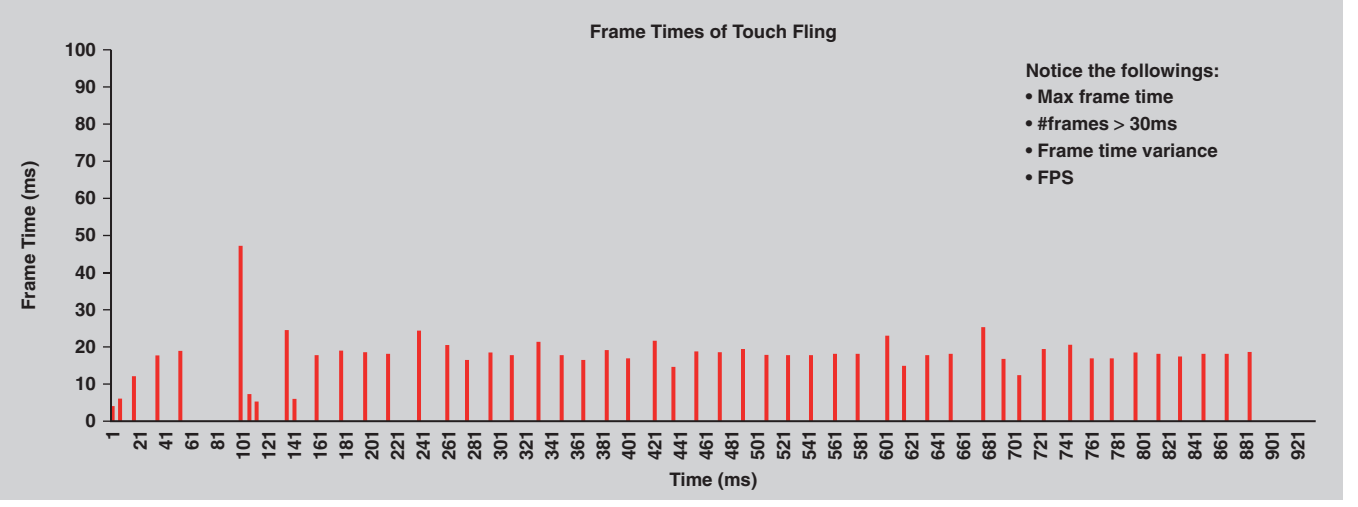

Figure 2: Frame times of a fling operation in Gallery3D application on device X (Source: Intel Corporation, 2011)

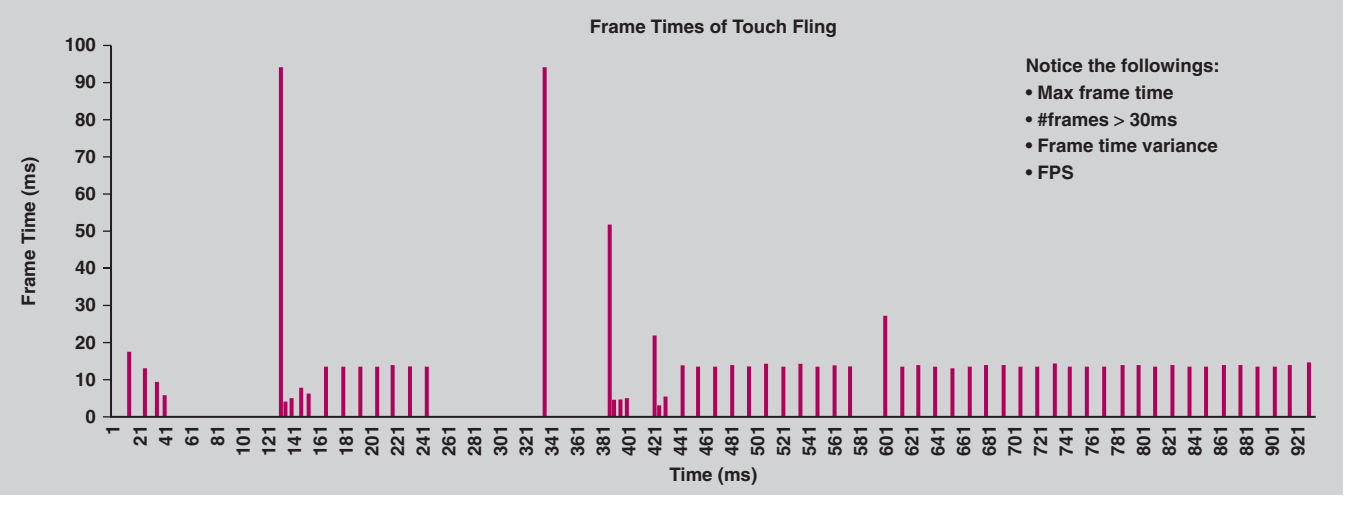

Figure 3: Frame times of a fling operation in Gallery3D application on device Y (Source: Intel Corporation, 2011)

As a comparison, Figure 4 shows the frame data of a fling operation after we optimized the device Y. Apparently all the metrics have been improved and the frame time distribution became much more uniform.

User experience is more about dynamic state transitions of the system triggered by user inputs. A good user experience is achieved with things such as user perceivable responsiveness, smoothness, coherence, and accuracy. Traditional performance could measure every link of the chain of the user interaction without evaluating the full chain of the user interaction as a whole.

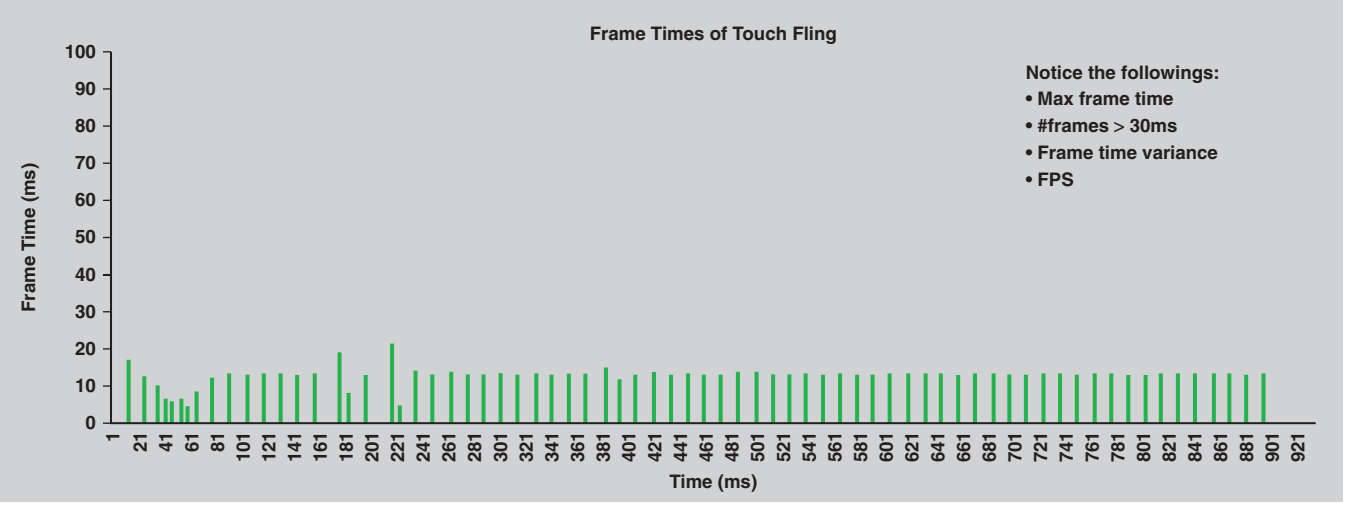

Figure 4: Frame times of a fling operation in Gallery3D application on device Y after optimization (Source: Intel Corporation, 2011)

Another important note is that user experience is a subjective process; just consider the experience when watching a movie or appreciating music. Current academic research uses various methodologies such as eyeball tracking, heartbeat monitoring, or just polling to understand user experience. For our software engineering purpose, in order to analyze and optimize the user interactions systematically, we categorize the interaction scenarios into four kinds:

- *Inputs to the device from the user, sensor, network, and so on.* This category evaluates if the inputs can trigger the device to action accurately or fuzzily as expected. For touchscreen inputs, it measures the touch speed, pressure, range, and so forth.
- *Device response to the inputs.* This category evaluates how responsive the device is to the inputs.
- *System state transition.* This category especially evaluates how smooth graphics transition on the screen. It can be a follow-up of the device response to some input.
- *Continuous control of the device.* People operating the device not only give a single input, but sometimes also control the graphic objects in the screen, such as to control a game jet-plane, or to drag an application icon. The category is to evaluate the controllability of the device.

Among them, "inputs to the device" and "control of the device" are related to the user experience aspect of *how a user controls a device*. "Device response to the inputs" and "system state transition" are related to the aspect of *how the device reacts to the user*. We can map a user interaction life cycle into scenarios that fall into the categories above; then for each scenario, we can identify the key metrics in the software stack to measure and optimize.

User Interaction Optimizations

As we have described in last subsection, there is no clear-cut and objective way to measure the user experience. We set up following criteria in our measurement of the user interactions:

- *Perceivable.* The metric has to be perceivable by a human being. Otherwise, it is irrelevant to the user experience.
- *Measureable.* The metric should be measurable by different teams. It should not depend on certain special infrastructure that can only be measured by certain teams.
- *Repeatable.* The measured result should be repeatable in different measurements. Large deviations in the measurement mean that it is a bad metric.
- *Comparable.* The measured data should be comparable across different systems. Software engineers can use the metric to compare the different systems.
- *Reasonable.* The metric should help reason the causality of software stack behavior. In other words, the metric should be mapped to the software behavior and it should be possible to be computed based on software stack execution.
- *Verifiable.* The metric can be used to verify an optimization. The measured result before and after the optimization should reflect the change of the user experience.
- *Automatable.* For software engineering purposes, we expect the metric can be measured largely unattended. This is especially useful in regression tests or pre-commit tests. This criterion is not strictly required though, because it is not directly related to user experience analysis and optimization.

Guided by the measurement criteria, we focus on the following complementary aspects of the user experience, how a user controls a device and how a device reacts to a user. How a user controls a device has mainly two measurement areas:

"How a user controls a device has mainly two measurement areas."

- *Accuracy/fuzziness.* It evaluates what accuracy, fuzziness, resolution, and range are supported by the system for inputs from the touch screen, sensors, and other sources. For example, how many pressure levels are supported by the system, how the sampled touch events' coordinates are close to the fingertip move track on the screen, how many fingers can be sampled at the same time, and so on.
- *Coherence.* It evaluates the drag lag distance between the fingertip and the dragged graphic object in the screen. It also evaluates the coherence between the user operations and the sensor-controlled objects, such as the angle degree difference between the tilting controlled water flow and the device oblique angle.

How a device reacts to a user also has two measurement areas:

- *Responsiveness.* It evaluates the time between an input being delivered to the device and device showing visible response. It also includes the time spent to finish an action.

- *Smoothness.* This area evaluates graphic transition smoothness with the maximal frame time, frame time variance, FPS, frame drop rate, and so forth. As we have discussed, FPS alone cannot accurately reflect the user experience regarding smoothness.

For these four measurement areas, once we identify a concrete metric to use, we need to understand how this metric is related to a "good" user experience. Since user experience is a subjective topic that highly depends on human being's physiological status and personal taste, there is not always scientific conclusion about what value of a metric constitutes a "good" user experience. For those cases, we just adopt the industry experience values. The Table 1 gives some examples of the industry experience values.

	Best	Good	Acceptable
Response delay	≤100 ms	≤200 ms	≤500 ms
Graphics animation	≥120 fps	≥60 fps	≥30 fps
Video playback	≥60 fps	≥30 fps	≥20 fps

Table 1: Example Industry Experience Values for User Experience (Source: Intel Corporation, 2011)

Due to human nature, there are two notes for software engineers to pay attention to the user experience optimizations.

The value of a metric usually has a range for "good" user experience. A "better" value than the range does not necessarily bring "better" user experience. Anything beyond the range limit could be invisible to the user.

The values here are only rough guidelines for common cases with typical people. For example, a seasoned game player may not be satisfied with the 120 fps animation. On the other hand, a well-designed cartoon may bring perfect smoothness with 20 fps animation.

"A seasoned game player may not be satisfied with the 120 fps animation. A well-designed cartoon may bring perfect smoothness with 20 fps animation."

Now we can set up our methodology for user experience optimization. It can be summarized into following steps.

Step 1: Receive the user experience *sightings* from the users or identify the interaction issues with manual operations. This can be assisted by high-speed camera or other available techniques.

Step 2: Define the software stack scenarios and metrics that transform a user experience issue into a software symptom.

Step 3: Develop a software *workload* to reproduce the issue in a measureable and repeatable way. The workload reports the metric values that reflect the user experience issue.

Step 4: Use the workload and related *tools* to analyze and optimize the software stack. The workload also verifies the optimization.

Step 5: Get *feedback* from the users and try more applications with the optimization to confirm the user experience improvement.

Based on this methodology, we have developed an Android Workload Suite (AWS)[33] that includes almost all the typical use cases of an Android device. We have also developed a toolkit called UXtune[34] that assists user interaction analysis in the software stack. Different from the traditional performance tuning tools, UXtune correlates the user-visible events and the system low-level events. As the next step, we are porting the work from Android to other systems.

"The UX optimization is somehow similar to parallel application optimization."

Mobile OS Design for User Experience

Based on our experience with Android, we found the UX optimization is somehow similar to parallel application optimization, only with more complexities, for the following four reasons:

- UX involves multiple hardware components, and multiple software processes, and their interactions;
- UX on a client device has to consider the power consumption, because UX also includes the battery life and device temperature.
- UX has precise timing requirements, such as smoothness where the user expects no frame time variance. Neither faster nor slower is acceptable; hitting the exact time point is required. This point is more like a real-time requirement.
- UX has some subjective factors one has to consider in mobile OS design, such as whether some animation is just a hint or essential to UX, and whether the system can drop some frames to get better response.

One critical lesson learned from our experience is to understand the critical path of certain operations. Different from parallel application tuning, mobile OS design does not always have strict explicit synchronization between the involved hardware components and software threads. For example:

"Every application uses an event loop to handle requests."

- Every application uses an event loop to handle requests. When a thread A has a request to another thread B, it may not directly invoke the function, but instead posts a message to thread B. The message is then queued in the event loop of thread B waiting for handling. It is then out of the caller's control how fast the event could be handled if there are multiple events in the queue.
- Another example is when a thread A executes a sequence of operations and then posts a message to another thread B for follow-up actions and response to the user. Not all the sequence of the operations by thread A must be done in order. For example, it could post the message to thread B as early as possible so that thread B can respond to the user earlier.

The major point regarding power is that, with user experience, faster is not necessarily better—contrary to traditional performance optimizations. When the system already reaches the user-perceivable best responsiveness, next step optimization is the slower the better within the best perceivable range. For example, if a game has 60 fps without problem, then the mobile OS should try to get both the CPU utilization and CPU frequency as low as possible. We

have to always distinguish the two cases (faster-better and slower-better) clearly. The optimization techniques for the two cases can be quite different.

When multiple cores and GPUs are introduced, the two cases above become more obvious. More cores help "faster-better" scenarios usually, but hurt battery life for "slower-better" scenarios. A mobile OS has to turn on and off the cores with a smart policy, because the on/off latency could be longer than a touch operation (usually in a range of 100 ms to 500 ms).

"A mobile OS has to turn on and off the cores with a smart policy."

For parallel application performance tuning, people found "execution replay" useful in debugging. It is usually one multithreaded application reply, that is, within one process. For UX, the interactions cross processes through IPC, and between CPU, GPU, and display controller (DC), are a whole-system-wide collaboration. The traditional replay technique cannot help.

Power Management (PM)

Power management has always been a key challenge to mobile OS designers and will be even more so moving forward. Power demands are increasing rapidly on mobile devices as more and more power hungry applications are developed for mobile platforms. However, battery capacity growth could never keep up in the meantime due to both the slow development in battery technologies and the fact that people want more sleek and compact form factors that could fit into a pocket. Power management is becoming an increasingly complex problem on mobile devices and a holistic approach needs to be employed to address it.

"Power management is becoming an increasingly complex problem on mobile devices."

Processor Power Management

Mobile operating systems have been making steady progress in the power management area in the last decade. Initially the focus of mobile OS power management work had been on processor power management since the processor had long been the most significant consumer of total platform power. Modern processors support dynamic frequency and voltage scaling such as Enhanced Intel SpeedStep® Technology. Such processor capabilities enabled mobile operating systems to adjust processor frequency and/or voltage dynamically at runtime based on the demand of computing power required by the workload that is currently running on the processor. This saves a significant amount of processor power while it is active since consumed power is proportional to the square of core voltage and frequency. The cpufreq subsystem in the Linux kernel is an example of managing processor power while it is active. In addition to dynamic frequency and voltage scaling, modern processors typically support multiple processor idle states with varying amounts of power consumed in those idle states. The deeper the idle state, the more power could be saved although at the expense of longer entry and exit latency. A mobile OS could direct the processor to enter an appropriate idle state based on predicted idleness and QoS constraints posed by other subsystems and user space. Linux's cpuidle subsystem is an example of power managing a processor while it is idle.

Device Power Management

The focus of mobile OS power management work has shifted to device power management. In particular, a mechanism has been introduced to manage the power of I/O devices at runtime. Runtime power management for I/O devices could automatically put I/O devices into whatever appropriate low power states they support when the corresponding devices are detected as idle at runtime. In addition to managing the power of I/O devices while they are idle, there are some technology innovations to save I/O device power while they are active. For example, modern GPUs are starting to support dynamic voltage and frequency scaling similar to that found on CPUs. GPU dynamic voltage and frequency scaling could reduce power consumption by as much as 50 percent for mobile 3D graphics in some cases. In addition, I/O devices are becoming smarter in the sense that they can work on their own without CPU intervention. For example, technologies like panel self-refresh could save a significant amount of power while the image is static in the cases like when a user is reading a book on a mobile device. The display panel could keep rendering from its local memory in this case and many hardware components that traditionally must be working while rendering display could be shut down, including CPU, memory, display engine and display port.

"Modern GPUs are starting to support dynamic voltage and frequency scaling similar to that found on CPUs."

Mobile OS Power Management Cases

Android gained momentum and became popular before the infrastructure of device runtime power management was introduced into the Linux kernel and it came up with another approach called *opportunistic suspend* in order to achieve the goal of extending battery life on Android devices. Without device runtime power management capabilities, Android tries to suspend the system aggressively whenever there is no interesting work going on. This is indicated by no one holding a *wakelock*.

Windows 8 introduced a new system power state called *connected standby*. Unlike traditional S3 standby, which halts all system activities, the system is still running though in an extremely low power state and this enables users to stay up to date with the latest information such as their e-mails. Windows 8 connected standby is based on processor idle power management and device runtime power management technologies.

"Software hygiene is the most challenging problem for both suspend- and idle-based power management approaches."

Software hygiene is the most challenging problem for both suspend- and idle-based power management approaches, and the battery life depends very much on application behaviors on such systems. A recent study says that free Android applications waste 75 percent of power on ads by holding a *wakelock* in the background, thus block system suspend. This is also true for Windows 8, where one power-unfriendly application staying busy for no good reason will prevent the entire system from entering connected standby power state. Some people expect the system power management to be more robust even in the face of such power-unfriendly applications by introducing more capable mechanisms, while others think that putting such a mechanism in place will only lead to the proliferation of more ill-behaved applications.

Openness

Another major trend of mobile operating systems is the openness. In this context, the openness of mobile operating systems means the level of opportunity and freedom that people have to use, contribute to, customize, and innovate for the mobile OS for their purposes. There is already work[6] that has studied the openness from the developers' perspective. The work has identified the facets that decide the developers' perception of the platform openness. Here we study the trend of openness from the ecosystem perspective, As we believe the openness matters to enable and foster the mobile ecosystem.

"Another major trend of mobile operating systems is the openness."

Openness to Players of the Mobile Ecosystem

The players of the mobile ecosystem include manufacturers (OEMs) who make and sell mobile devices, the service providers (operators) who provide network connection and other value-added services, consumers (end users), the ISVs who develop commercial applications, and developers from communities who develop applications and even contribute on mobile OS development and evolution if the mobile OS is open-sourced.

The openness of mobile operating systems implies different things to different players in the mobile ecosystem. For operators, the openness of the mobile OS determines how easy their services can be ported, migrated, deployed, and run smoothly across the devices. For mobile device manufacturers, the openness determines how much they can customize the mobile OS itself to run across platforms and differentiate their devices from others, and even more importantly, the openness determines how easily they can build devices with a consistent user experience. For ISVs and community developers, the openness determines how easily they can develop new applications with their creative ideas and how their investment on application development can be maximized through programming once running across devices. For the end users or consumers of mobile devices, the openness means how easily they can get more applications, like the rich applications downloaded from the stores, without worrying too much about inconsistency of user experience and incompatibility of applications across devices. The openness may also give people the chance to participate in the development and evolution of the mobile OS itself during the life cycle of the mobile OS.

"The openness determines how easily they can build devices with a consistent user experience."

Evolution of Mobile OS Openness

A couple of years ago, most mobile devices were feature phones, and people mostly used the phones for voice calls, as a phonebook, and for text messages. For consumers, the applications they could play were limited to the applications built in devices when the devices were shipped out from the factory. For application developers or third-party ISVs, they didn't have access to any level of source code without a contract with the OS owner. Such a mobile OS was a purely closed system and was typically owned by a mobile device maker. For operators, they had to work closely with OS developers to enable their services, because only the people who developed the mobile OS knew how to develop applications.

Later, as the mobile phone started to transition from feature phone to smart phone, people expected the smart phone to be able to do more, like browsing the Web and playing music/video, rather than just making calls, storing contact information, and sending text messages. To encourage more developers to develop more applications to meet people's needs, the creators of mobile operating systems simply provided sets of APIs and related tools like SDKs, so that people could develop all kinds of applications for mobile devices. With such openness, application developers gained the freedom to develop applications for mobile operating systems, so it became possible for consumers to buy and install more applications rather than being limited to the pre-built applications. Because application developers and consumers benefitted from such openness, it became almost a "must-have" for most mobile operating systems to provide a set of APIs and an SDK. The iOS from Apple is one of the great examples of a mobile OS providing such a level of openness. In recent years more and more mobile operating systems have made all source code public, in addition to providing APIs and SDKs. Anyone could have the chance to view all source code, contribute to the code, evolve, and customize the mobile OS itself. Compared with the level of openness of just providing APIs and SDK, the open-source mobile OS can provide some additional freedom. For mobile device makers, they may have the freedom to build their own mobile OS based on the open source OS to run on their platforms across devices. For operators, they can easily build and deploy their services across devices running the open source OS and its variants. For developers, the open source mobile OS provides everything they need to easily build their applications. Eventually the end users of mobile devices can benefit from this level of openness, as they have more choices of applications and more devices to choose to run the applications. Lastly, everyone has the freedom to participate in evolving and shaping the open source mobile OS, which is very attractive to the talents from communities around the world. Android OS is another great example of an open source mobile OS. Its great success and segment share growth in the smart phone market during the past few years have indicating to the industry just how successful it has been and how fast it has been growing as an open source mobile OS.

"Anyone could have the chance to view all source code, contribute to the code, evolve, and customize the mobile OS itself."

As a summary, the openness of future mobile operating systems is one of the key factors to make mobile platforms friendly to the mobile ecosystem, especially to be attractive to application developers and consumers. Mobile OS openness is a requirement of computing continuum, which expects most software to be built once and running everywhere with a consistent user experience to end users.

"Mobile OS openness is a requirement of computing continuum."

Cloud Readiness

The cloud has been widely used by mobile users and most of the cloud services are presented as Web sites and accessed by the browser running on the mobile browsers. More and more cloud services have been provided through web applications, which are installed from an application store and run like native applications on the mobile client. Either with a browser or standalone web application, the following areas should be considered in mobile OS design.

HTML5 Capability

HTML5 capability is essential for web applications to integrate cloud services well and to provide a good user experience.

"HTML5 capability is essential for web applications to integrate cloud services."

Web engines being used: we listed the web layout engines and JavaScript engines being used on the mainstream mobile browsers. Chrome* has the potential to be the default browser of Android and bring its success from desktop to mobile. The Webkit is used in iOS and Android.

The HTML5 test web site[9] scores HTML5 support of browsers on various mobile devices. We can see in Table 2 that iOS, Android, and Windows Phone are all improving their browser's capability to support HTML5. Google made Chrome work on Android 4.0 and showed its ambition to have the lead browser in mobile operating systems. Tizen, the new participant in the mobile OS campaign even got the highest score on its development device released in the first Tizen Developer Conference. We can easily see the intense race of HTML5 support between mobile operating systems.

	Layout engine	JavaScript engine	Score + Bonus
Safari on iOS 5.1	Webkit	Squirrel Fish Extreme (SFX)	324 + 9
Android browser on Android 4.0	Webkit	V8	273 + 3
IE on Windows Phone 8	Trident	Chakra	300 + 6
Tizen 1.0	Webkit	SFX	400 + 15 *
Chrome Beta on Android 4.0	Webkit	V8	364 + 10
Opera Mobile 12	Presto	Carakan	369 + 11
Firefox Mobile 10	Gecko	SpiderMonkey	325 + 9

Table 2: Web Layout Engines, JavaScript Engines, and Their Score from html5test.com
(Source: [9] accessed on May 1, 2012. * The score of Tizen 1.0 is from the latest Tizen development device, tested by Intel Corporation.)

The third-party browsers are in difficult situation and need a strategy to have their own host mobile OS. Opera and Firefox are in such a situation. Due to the fact that they have less control of mobile OS development, they won't be able to win easily if the built-in browser is capable enough for HTML5 support. Firefox has been looking for Boot To Gecko as its host mobile OS. A video on YouTube[10] also shows the preview of Opera OS on Asus EeePC .

The mobile OS vendors view HTML5 support as more and more important and are making it a core competency. The browser vendors are also looking for the possibility to make it default in the mobile OS.

"The web application can be installed and run even offline on the devices."

Web Applications

The web application defines the client side applications developed with web technologies. It provides rich features by providing APIs for client side development. The web application can be installed and run even offline on the devices. The web application can access local devices and resources as a native application and can be sold in an application store, which benefits much from the cloud service delivery and billing.

The web application is more than a URL accessed from a browser. The related capabilities are being defined in several working groups in W3C. The Web Applications Working Group is the central place related to those works.[11]

To enable web applications, the mobile OS needs to provide a web application platform, which includes web runtime, web framework, and development tools:

- *The web runtime provides the core capability to run a web application.* It is derived from the browser but is more integrated with the OS runtime. The web runtime provides an HTML5 engine and the APIs to access local devices. It also provides the capabilities to integrate well with the OS runtime, which includes the application life cycle management (install/update/uninstall and launch), OS integration (desktop integration, security policy, OS services access.)
- *The web framework provides rich libraries for web application development.* Examples are jQueryMobile and Sencha. Such frameworks are widely used in iOS and Android.
- *Development tools need to be flexible.* The development tools are very diverse and can be chosen according to the web application developer's preferences. Development tools can be a very concentrated SDK suite like the recently released Tizen SDK 1.0,[12] browser-based like appMobi XDK[13], or just a set of tools like RIB and Web Simulator, released on 01.org.[14][15]

As a trend, the mobile OS has to provide a capable and high performance web runtime, a rich web framework, and flexible development tools.

Security is always an important topic for cloud computing. For HTML5 cloud integration in mobile operating systems, the following features must be present:

- *Sandbox support in web runtime.* The web applications should run in separate processes and be managed by the web runtime. Sandbox in browser or web runtime has been supported on most modern mobile operating systems.
- *JavaScript code protection.* JavaScript is a scripting language, so the best way to protect the code is still to run the code on the server side.

"In reality, different mobile operating systems provide different HTML5 support."

Cross-Platform Capability

HTML5 is well known for its cross-platform capability. But in reality, different mobile operating systems provide different HTML5 support, and the

standardization of HTML5 is still ongoing. PhoneGap has been developed to address the cross-platform HTML5 support by providing its own device APIs. Apple iOS, Android, and Windows Phone all have supported PhoneGap. It is evolving and following W3C. Other Web API providers are also trying to make them into the HTML5 standard in W3C.

The trend is to have a unified HTML5 standard but that is not easy. Mobile OS vendors will implement their ideas in their own way before they go to standard. Apple, Google, and Microsoft are all active participants in the W3C standard definition. For other mobile OS vendors, either following W3C or joining the definition is the trend to make their mobile operating systems survive.

Performance

The performance is complained about most by mobile application developers when they start to build applications with HTML5. The optimization for mobile devices is the most important work to do for HTML5 to really succeed in the mobile area. We consider the following to be the most important areas to do work in optimization for mobile operating systems:

"The optimization for mobile devices is the most important work to do for HTML5."

- *Hardware acceleration.* The graphics and video should be accelerated by hardware. WebGL has been enabled on more and more platforms.
- *Multithreading support.* Web Worker should be supported as a key feature in HTML5.
- *JavaScript engine optimization.* JIT (Just In Time) has been enabled in both SFX and V8.
- *Native or hybrid application support.* The capability to reuse existing native libraries will be another approach for web applications. Android NDK provides such a capability and it is widely used in Android applications. But on web applications, it has not been widely used. The NaCL by Chromium is an option to support that.

Cloud Integration

Besides the powerful capabilities provided by HTML5, the seamless integration between the cloud and client is even more important. It is not only for web applications but also for native applications. Important reasons for integration include:

"Apple has made iCloud deeply integrated into iOS 5."

- *Cloud storage seamless integration.* The client should be using the cloud storage just as it uses its native storage. That requires the cloud storage client to be tightly integrated into the mobile OS. Apple has made iCloud deeply integrated into iOS 5. The Google Drive is integrated with Google web services.
- *Cloud APIs accessibility.* On the cloud client side, the mobile operating systems should provide capable and easy-to-use libraries for both web and native applications to access cloud client APIs, which are normally RESTful, SOAP, or Query APIs.

"The synchronization and notification are key features in mobile operating systems."

- *Account management.* With cloud integration, multiple clients with a single account share the cloud and the synchronization should be managed. Accounts should be tightly integrated in the mobile OS and account APIs should be provided for applications to securely access the corresponding cloud services and the local resources. The synchronization and notification are key features in mobile operating systems to enlarge the usability of cloud integration.

Discussion and Summary

In this section, we first discuss the major mobile operating systems in the market today, and then summarize this article.

Apple iOS

Apple has been the leader in mobile OS design. Its iPhone* and iPad* have prevailed across the world in only a few years. Both products feature the Apple iOS.

User experience: iOS provides good performance and is normally set as the benchmark for other mobile operating systems. Apple is continuously enhancing the UX performance. The iPhone 4S has much better performance boost than its previous generations, especially the Internet and browser.[1] With more new features added, it adds more performance requirements. An unofficial study [1] showed the UX performance drops after the upgrade from iOS 4.x to 5.x on iPhone 3GS.

Power management: iOS power optimization seems not able to catch up with the increasing demands on power for new features. Arieso, a mobile network management company, estimates that iPhone 4S users consume twice as much data as the previous iPhone model due to increasing use of online services like the virtual personal assistant Siri, which definitely consumes much more power.[3]

Openness: iOS is perceived as a closed mobile OS. Research work[6] defines a concept of perceived platform openness (PPO), where a platform's openness degree is decided by its developers' perception.

Cloud readiness: iOS 5.0 with HTML5 support makes it a good cloud client and the iCloud has been integrated by default as storage.

Google Android

Android is currently a popular operating system for mobile devices and is developed by the Open Handset Alliance led by Google. The goal of the Android Open Source Project is to create a successful real-world product that improves the mobile experience for end users.[16]

User experience: The Android user experience team defined a set of design principles[17] with three overarching goals: "Enchant me," "Simplify my Life," and "Make me amazing."[18] State-of-the-art Android and iOS devices achieved similar results in a set of battery life benchmark tests.[19]

Power management: Android aggressively suspends devices to save power whenever nothing blocks suspend by holding a *wakelock*.[20] However, Android allows third-party applications to run in the background, which might hold such a *wakelock* for no good reason and thus suck power quietly.

"Android aggressively suspends devices to save power."

Security: Each application runs in a sandbox environment to enforce security in Android and this is done by assigning each application a unique user ID and running that application as that user in a separate process.[21]

Openness: Google releases the Android code as open source under Apache license and the Android Open Source Project is where Android development and maintenance happen. However, Google typically partners with a selected manufacturer to make a flagship device for each new version of Android and only makes the new code publicly available after that device has been released. Fragmentation has become more and more a big concern in the Android ecosystem. Android maintains the Android compatibility program and offers the compatibility test suites to guarantee applications developed for Android run on every Android device.

Cloud readiness: Although Google has a huge lead in the cloud area, it has not put together a comprehensive solution as Apple does with iCloud yet.

Microsoft Windows Phone

Microsoft has released its latest redesigned mobile OS called Windows Phone. Based on their design change between Windows Mobile 6.5 and Windows Phone 7, some characteristics of the newer OS are exposed.

User experience: With touchscreen-based user interaction replacing the previous stylus input, Microsoft decided to break the application compatibility between Windows Phone and Windows Mobile.[23] Similar to Android's AppWidget design, Windows Phone invents the concept of *Live Tiles* for the home screen.[28]

Power management: Similar to its design security, Windows Phone's design for battery life can largely benefit from its Windows CE and Windows Mobile experience. One special consideration is that Windows Phone chooses black as the main default color theme, because black pixels do not emit any light, hence saving power for the OLED screen.[32]

Security: Windows Phone's design is shifted from the original Windows Mobile's enterprise-oriented design to an end-user–oriented one. The security experience accumulated for the enterprise product should be still useful.[31]

Openness: Before Windows Phone was available in the market, Microsoft released its SDK to enable the developers to program for the new OS.[24] Windows Phone Marketplace has provided its services to 35 different countries/regions.[25] The current programming languages are C# and Visual Basic. These are not a surprise to any Windows developers, so the language learning curve is expected to be flat.

"Windows Phone Marketplace has provided its services to 35 different countries/regions."

Cloud readiness: Windows Phone is approaching cloud readiness at a fast pace. Windows Phone 8 integrates Internet Explorer 10 that is claimed to have full HTML5 support and supports parallel page loading in multiple tabs. [29] Besides that, Skype is deeply integrated into the OS.[26] One new concept in Windows Phone are *hubs,* which aggregate various similar service features into one hub. This is supposed to greatly improve the phone's user experience with cloud services.[27] Furthermore, the software framework design of Windows Phone includes two parts: Screen and Cloud. The Cloud part is especially designed for "Developer Portal Services" and "Cloud Service."

In this article, we have investigated the major aspects of mobile OS design based on the analysis model we have developed, including user experience, battery life, cloud readiness, security, and openness. These should be the areas of focus for next-generation mobile OS design.

The future mobile OS also depends on the available hardware design. We believe a successful mobile system is a result of co-design between software and hardware, together with the progress of the Internet.

References

(The paper references the web sites accessed on May 1, 2012.)

[1] ***http://www.youtube.com/watch?v=ng33wXDkyRM, http://www.iphonedownloadbolg.com.***

[2] ***http://www.anandtech.com/show/4951/iphone-4s-preliminary-benchmarks-800mhz-a5-slightly-slower-gpu-than-ipad-2***

[3] ***http://www.arieso.com/news-article.html?id=89***

[4] ***http://www.apple.com/iphone/business/docs/iOS_Security_Mar12.pdf***

[5] ***http://www.techrepublic.com/blog/security/comparing-android-and-ios-security-how-they-rate/5774***

[6] ***http://www.user.tu-berlin.de/komm/CD/paper/090322.pdf***

[7] ***http://www.apple.com/icloud/***

[8] ***http://www.networkworld.com/community/blog/apples-ios-5-and-cloud***

[9] ***http://html5test.com/results/mobile.html***

[10] ***http://www.youtube.com/watch?v=mWSPNDD0tek***

[11] ***http://www.w3.org/2012/webapps/charter/Overview.html***

[12] ***https://developer.tizen.org/sdk***

[13] ***https://chrome.google.com/webstore/detail/onmkoldigcfmebcinpmineoadckalllb***

[14] ***https://01.org/projects/rapid-interface-builder-rib***

[15] *https://01.org/projects/web-simulator*

[16] *https://www.mylookout.com/mobile-threat-report*

[17] *http://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2011-mobile-threats-report.pdf*

[18] *http://webupon.com/security/information-systems-attacker-motivations/*

[19] *http://source.android.com/tech/security/index.html*

[20] *http://www.w3.org/2011/webappsec/*

[21] *http://www.trustedcomputinggroup.org*

[22] *http://www.globalplatform.org/specifications.asp*

[23] *http://channel9.msdn.com/Events/TechEd/NorthAmerica/2010/WPH201*

[24] *http://www.engadget.com/2010/09/16/microsoft-demoes-twitter-and-netflix-apps-for-windows-phone-7-r/*

[25] *http://windowsteamblog.com/windows_phone/b/windowsphone/archive/2011/07/06/windows-phone-around-the-world-language-support-in-mango.aspx*

[26] *http://wmpoweruser.com/microsoft-reps-claiming-windows-phone-8-definitely-coming-to-second-gen-handsets-probably-to-first-gen/*

[27] *http://www.engadget.com/2010/03/18/windows-phone-7-series-the-complete-guide/*

[28] *http://www.engadget.com/2010/02/15/windows-phone-7-is-official-and-microsoft-is-playing-to/*

[29] *http://pocketnow.com/windows-phone/exclusive-windows-phone-7-web-browser-comparison*

[30] *http://arstechnica.com/microsoft/news/2011/04/windows-phone-7-mango-one-heck-of-an-upgrade.ars*

[31] *http://arstechnica.com/microsoft/news/2010/03/windows-phone-7-series-in-the-enterprise-not-all-good-news.ars*

[32] *http://www.neowin.net/news/interview-windows-phone-7-battery-life-copypaste-multitasking-and-more*

[33] *http://software.intel.com/en-us/articles/aws-android-workload-suite-for-user-interaction-measurement/*

[34] *http://software.intel.com/en-us/articles/quantify-and-optimize-the-user-interactions-with-android-devices/*

Author Biographies

Xiao-Feng Li (xiao-feng.li@intel.com): Architect of System Software Optimization Center, Intel Corporation. Xiao-Feng has been working with Intel for 12 years, with extensive technical experience in parallel system, compiler design and runtime technologies, where he has authored about 20 academic papers and 10 U.S. patents. Two years ago, Xiao-Feng initiated the evaluation and optimization efforts for best Android user experience on Intel platforms. Before he joined Intel, Xiao-Feng was a technical manager in Nokia Research Center. Xiao-Feng holds a PhD degree in computer science, and is a member of Apache Software Foundation. His personal homepage can be found at http://people.apache.org/~xli.

Yong Wang (yong.y.wang@intel.com): Senior software engineer from Intel's Open Source Technology Center. He has been with Intel for 7 years working on a variety of projects, including virtualization, manageability, OSV enabling, etc. Most recently Yong has been working on power management for a wide range of mobile operating systems such as Moblin, Meego, Tizen and Android. Yong graduated from Beijing University of Aeronautics and Astronautics and holds a master degree of computer science. He enjoys sports and reading in his spare time.

Weihua Jackie Wu (jackie.wu@intel.com): Engineering Manager in Intel's Open Source Technology Center leading a team on Mobile OS and HTML5 tools development. Before that, as a research engineer, Jackie was focused on wireless network and energy efficient communications. Prior to joining Intel in 2004, Jackie was in Chinese Academy of Sciences developing embedded operating system and smartphone products. Jackie received his M.S. and B.S. in Engineering Mechanics from Beijing University of Aeronautics and Astronautics in 2002 and 1999 respectively. He has two US patent applications.

Kerry Jiang (kerry.jiang@intel.com): Software Manager in System Optimization Technology Center (SOTC), Intel Corporation. Kerry has been working with Intel for eight years, four years in open source mobile OS software stack on IA, which includes Android optimization and MeeGo SDK. Before joining Intel, Kerry worked in Motorola on mobile platform and 2G wireless base-station software developments. Kerry holds a master degree in electronics engineering.

Bingwei Liu (bing.wei.liu@intel.com): Engineering manager of Open Source Technology Center, Intel Corporation. Bingwei has been working in Intel for 11 years, with abundant experience in Linux OS, open source software and system engineering. His working scope spans from enterprise to client platforms and now is focused on Mobile OS. Bingwei holds a master degree of computer science.