

Mobile Handset Design: Realizing Flexible, Low Cost, Higher Security, Device Management Using OpenOS and Real-Time Virtualization™

The rapidly-evolving mobile handset market presents both a major market opportunity and a major design challenge. The market opportunity is to transform the handset from a mobile voice and basic data terminal into the consumer's primary mobile communications, information, financial transaction and entertainment center, with a commensurate added value.

This transformation is well under way, with many handsets combining voice and simple data communications with:

- Communication capabilities such as secure inter-device wireless communications and Internet access.
- Information capabilities, such as secure remote mail access and office data access.
- Entertainment capabilities, such as MP3 audio playback, digital camera, and video recording and playback, with secure intellectual property rights protection.

In the coming years, the fastest growing segment of the mobile handset market will be that which delivers this functionality – smart phones. However, it is an inescapable dynamic of consumer markets that the functionality of today's high-end product must be implemented in tomorrow's mid-range product in order to maintain sales volume, growth and market share. Consequently, the industry faces the design challenge of migrating smart phone functionality to the mid-range handset – the feature phone. And it must do so, with the competitive development costs, the competitive bill of materials (BOM) and the short design, implementation and validation time necessary to succeed in fast-moving and demanding consumer markets.

This whitepaper describes the design challenge in more detail, and compares and contrasts various implementation approaches. It then discusses how the VirtualLogix real-time virtualization solution may be used to enable the implementation of smart phone software functionality on an existing feature phone hardware platform, with minimal effort and cost. It also describes how a virtualization-based solution opens the phone software architecture – both smart phone and feature phone – to ease the addition of new functionality, such as service availability, security and device management.



Mobile Handset Design Challenges

Market Segmentation

The mobile handset market is conventionally partitioned into three major cost/feature segments: low cost phones, feature phones and smart phones.

Low cost phones typically provide only voice communication and short messaging services (SMS). They typically use a DSP for signal processing and a RISC CPU to execute the upper protocol stack layers.

At the other end of the spectrum, smart phones offer advanced services similar to those provided by PDAs. Such phones typically use multiple processors: a DSP for signal processing, a RISC CPU with an RTOS to execute the upper protocol stack layers and some applications, and a more powerful RISC CPU with an OpenOS – such as Windows CE, Symbian OS or Linux – to execute a richer set of native applications that make the phone a smart phone.

In the middle of the spectrum, feature phones offer a trade-off between low cost phones and smart phones. They typically offer fewer services than smart phones on less expensive hardware and with less battery drain. Typical hardware configurations use a DSP for signal processing and a RISC CPU (more capable than those used in low cost phones) to run an RTOS that executes the upper protocol stack layers and some applications, including Java- and Flash-based applications.

However, as noted in the introduction, the design challenge is equip the feature phone with smart phone applications and functionality, and to run native OpenOS applications instead of those running on the feature phone RTOS.

Implementation Challenges

The feature phone design challenge is to offer smart phone services on less capable hardware. Consequently, the processing capabilities of the platform (CPU, coprocessors and hardware accelerators) must be optimally selected to support real-time protocol stacks and some end-user applications such as browsers, email clients, calendar applications and so forth.

In addition, mobile handsets, whether feature or smart phones, must be IP-connected using the common communications technologies, including cellular radio protocols – both narrow- and broadband – Wi-Fi, Bluetooth, and InfraRed, as well as USB and serial line. Such connectivity enables the delivery of more end-user services, such as TV, video and audio streaming, video and audio download and playback, network gaming, photo transfer, mobile data synchronization with PC or laptop, and e-mail. Location Based Services will soon enrich this set of services.

The on-time delivery of a new phone model mandates extensive design reuse, especially of legacy software. This software includes functions – such as radio protocol stacks with their real-time constraints, and multimedia services – that are difficult to validate and certify over the full range of operating conditions.

However, end-user services such as those enabled by IP protocols are better delivered using OpenOS' and open environments such as Linux, Symbian or Windows. These environments enable the reuse of high-level applications such as email clients, browsers, audio, video players, and Java applications.

Linux, in particular, is rapidly becoming an OpenOS of choice. It provides the latest evolutions of IP family protocols and up-to-date drivers for complex interfaces, such as USB, significantly easing the implementation of further connectivity and new features in a mobile handset.

In summary, the implementation requirements are: extensive legacy software reuse, the adoption of additional “new” OS suites to execute new applications, and the delivery of new features on a hardware-constrained phone – and all within the time to market constraints.

The optimum implementation strategy must therefore enable the un-modified reuse of multiple software stacks executing on a single processor – known as “resource consolidation”.

Evolution Readiness

Obviously, time to market constraints will not abate as mobile handsets continue to evolve with richer feature sets – and mobile handset software stacks must be ready to support these evolutions. Hence a robust, but flexible stack architecture is essential to the timely delivery of evolving, but validated, software stacks.

Moreover, mobile handsets are and will be increasingly open, enabling the download of software to satisfy more personalized user requirements. Personalized requirements might well include – in addition to the phone’s standard feature set – business oriented Java applications; “one-touch” transactions such as train control gate “pass”, theater and train ticket purchase, and more complex financial transactions such as bank and brokerage account access.

The handset will also be open to access by multiple external agents, such as employers and banks, each with its own security policy, and each wishing to control “its share” of the phone.

To avoid problems with the administration, management and security of the evolving functionality and of the increasing complexity of external agent requirements, such software is best executed in independent run-time environments.

Open OS’ and Security

Security will be a serious issue for mobile handsets. There are already more than 180 known viruses for mobile phones – still far behind the number of known PC viruses, but the number is growing fast. Consortia such as OMA, OMTP, LiPS Forum, and TCG are already attacking the issue.

Security cannot be an afterthought. The frequent, emergency post-development security upgrades and patches common in the PC world would be unacceptable to the average user of a mobile handset. Indeed, such an approach to security could stall the proliferation of smart and feature phones. Consequently, security must be designed-in from day one of the handset’s life cycle.

Environments that allow software downloads are obviously more susceptible to malware attacks than closed ones, and therefore must be protected and isolated to limit or prevent attacks. For example, operator stacks must be protected against any kind of attack, so that the phone continues to work and the customer is not billed for calls that he did not explicitly make. A second example: enterprise data must not be compromised by an attack from an infected downloaded game. Another example: the digital rights of a downloaded video should be stored securely to prevent unauthorized access. A final example: personal data must be protected to prevent spyware monitoring.

Most often, security is implemented by the sub-systems that are subject to attack. For instance, scanning for viruses in incoming and outgoing mail is usually executed by a program running in the same environment as the user's email client. However, it is dangerous to rely upon self-security, especially in a multi-use scenario with multiple external agents such as service operators, banks, and other enterprises. The mobile handset user is simply a roaming agent in a wireless IT world – and security is every bit as important as it is in the fixed-wire IT world.

Security must be achieved by means of clear partitioning and robust isolation. As an example, the NTT Docomo Intel specifications (http://www.nttdocomo.co.jp/english/binary/pdf/corporate/technology/osti/OSTI_Arch_R1_00.pdf) clearly calls for the phone to support two independent environments. Isolating the assets of each party (operator, content provider, bank, user, etc.) ensures that if one asset is compromised, the others remain secure. In addition, robust isolation enables security checks to be conducted from a part of the system other than that containing the attacked asset. Consequently, handset functionality must be firewalled, just as IT networks and home LANs are firewalled.

Handset Management

The increasingly complex nature of mobile phone software will necessitate its management from various remote sources. Mobile phones are already managed over the air by the operators. For instance, it is not uncommon to be asked to power cycle the phone to activate new configuration data remotely installed in the SIM card. Indeed, the SIM card is considered to be a trusted execution environment that could be used as the standard means to manage and control many or all of the additional services offered by smart phones and feature phones. Of course, this use of the SIM card must be accompanied by the deployment of trusted application extensions, such as a secure user interface, in order to maintain security integrity.

Device management can be partitioned into three main categories: closed device management, open device management and local device management.

Closed device management encompasses management of the physical device and of the software that controls operator services, such as voice communications, SMS, and MMS. It thus manages the basic phone functions for which the phone manufacturer and the phone service provider are responsible. Examples of such management operations may include, but are not limited to:

- Availability: monitor hardware and software events; safely log errors for improvement purposes; retrieve and reset logs; restart failed systems and components.
- Upgrade: enable both operators and handset designers to patch, modify or upgrade their part of the system, safely and securely, with fall-back mechanisms.

Open device management encompasses management of all other phone software components that have been installed post-purchase by the user or an external agent. Since the high end mobile phone is used for sensitive and confidential business, banking or personal applications, open device management may be the responsibility of different stakeholders, such as the enterprise, bank and the user.

In both management categories, management facilities must provide control data, software backup, software refresh, provisioning, and software removal and installation. Any and all such management operations must be executed safely and securely with authenticated management servers, and all software and critical data transfers must be signed.

Local device management is required in addition to the closed and open device management categories. It enables the mobile phone to operate under a broad range of conditions in a wide variety of environments. For example, local management dynamically adapts the system and/or application by stopping or suspending an application stack or component execution to give higher priority to another application. It also adapts phone behavior to changing security policies as it moves from one working environment to another. Such local management is critical to making the phone truly mobile.

Implementation Strategy

So, what are the requirements to address the foregoing challenges?

First, a solution should enable the reuse of legacy real-time systems, telephony protocol stacks, together with OpenOS protocol stacks (IP protocol) and native OpenOS applications. Such a consolidation enables the simultaneous execution of both software environments on a single processor to deliver both sets of services without the necessity for more powerful and more expensive processing capacity. Also, an ideal consolidated solution would deliver native performance levels at no extra porting cost.

Second, a consolidation solution would enable evolutions of both systems. In particular, it should enable the use of newer versions of the legacy OS and the OpenOS – with new or upgraded protocol stacks, drivers and applications – to deliver up-to-date services.

Third, a solution should enable the execution of security controls and policies, robustly isolated from both the legacy RTOS environment and from the OpenOS environment. This ensures that security services and secure services remain protected in the event that other – more open – assets are breached. In fact, security services should be designed in a modular fashion and are subject to evolution over time, requiring that their execution environments be sufficiently flexible to easily support such evolutions.

Fourth, the device management services should execute separately from the legacy real-time system and OpenOS. Such an approach enables to more easily provision and configure the resources of the mobile handset between each party in multi-tenants configurations. It also enables the secure and controlled upgrade of functions and services, with fall-back mechanisms in case of any misbehavior on the part of updated software.

Implementation Alternatives

Various potential implementation alternatives to implementing smart phone software functionality will now be considered.

Two processors: Obviously, two processors could execute two different software environments without software modification. Unfortunately, this approach violates one of the most important requirements: bringing smart-phone functionality to feature phones without increasing the processing power of the feature phone. Single processor, single OS: Another approach merges the legacy and OpenOS environments in a single environment. However, in practice, this conceptually simple approach requires a huge amount of effort to port, test and revalidate the software, thus significantly increasing time to market. Moreover, a OpenOS may not possess all of the properties of a legacy RTOS, especially real-time quality of service (QoS). On the other hand, the use of a legacy RTOS would make it difficult to use existing open implementations of protocol stacks such as TCP/IP and application software. Once the merge is complete, developers would then face evolution problems such as how to keep the merged environment up-to-date with the evolution of its individual components in their native environments. As an example, the TCP/IP family protocol is still evolving and becoming ever richer. Such evolutions are easier to implement when using an unmodified OpenOS such as Linux.

Hybrid OS: This approach partially solves the main problems of the single OS approach just described. Hybrid OS' usually provide a real time environment within a Linux kernel. However, this conceptually-simple approach lacks the requisite flexibility. The real time environment usually commences execution after Linux has started, delaying critical real time applications. Moreover, the systems are not mutually isolated. Consequently, any misbehaviour of one system could seriously impact the behaviour of the overall platform, jeopardizing both availability and security. Moreover, such hybrid OS' approaches impose their own real time framework and interfaces, which are generally different from those of any legacy system. Hence, additional porting effort is required. Finally, the approach is limited to two environments: it is quite difficult to extend the approach to support additional execution environments, for example, for security or device management applications.

Paravirtualization: Paravirtualization enables the execution of multiple OS environments on a single processor, each environment being isolated within a dedicated partition. Physical resources such as memory and I/O devices are dedicated to partitions. The number of partitions can be optimized to the needs of the overall system, thus permitting the isolation of given security and device management functions in dedicated partitions. System environments can be reused in their partitions without much effort. Only parts of the OS' hardware abstraction layer (HAL) require some adaptation, at relatively low effort and cost.

Transparent Virtualization: An even better solution would be to use transparent virtualization, which in addition to all of the benefits of the paravirtualization approach, does not require modification of the operating systems. Depending upon the processor capabilities, transparent virtualization may require some complex dynamic binary translation mechanism, such as that implemented by VMware on x86 machines, or may be implemented through hardware-based mechanisms, such as those found on Intel VT-enabled processors. Unfortunately, embedded processors in mobile phones do not provide support for efficient transparent virtualization. Hence paravirtualization appears to be the best trade-off choice.

The Solution: VirtualLogix VLX Real-Time Virtualization

VirtualLogix develops real-time virtualization products and solutions for the embedded and real-time markets. With processors supporting hardware virtualization, VirtualLogix real-time virtualization which enables execution of the unmodified binary OS – but at significantly lower porting effort and cost than traditional transparent virtualization.

Real-Time Virtualization

As stated, real-time virtualization enables real-time environments to execute in parallel with OpenOS and other execution environments on a common hardware platform or processor. The virtualization software layer that supports these multiple environments maintains the system's real time properties. It also occupies the small memory footprint required by real-time embedded applications. The most important benefits of real-time virtualization are:

- **Consolidation of RTOS and OpenOS:** One or more OpenOS such as Linux, Symbian, or Windows may execute in parallel with a RTOS without disrupting the behaviour of the supported applications. Guest OS' require only some adaptations in their Hardware Abstraction Layer (HAL) to run within the virtualized environment.
- **Binary Legacy Reuse:** Most of the guest OS' is reused without modification. Native device drivers, protocol stacks and system modules can be reused in a straight-forward fashion. Legacy applications remain unmodified and run in the new context. This ensures minimum development cost and a short time to market.

- **Device Assignment:** Devices can be dedicated to a given partition or efficiently and securely shared by different guest operating systems.
- **Increased Security:** The real-time virtualization solution offers robust security via the hardware-enforced memory isolation of partitions, which isolates each OS from the others, and prevents cross-corruption. In addition, specific partitions may be added and used to execute secure applications in small certifiable environments protected from the larger and open OpenOS or RTOS executing in other partitions.
- **Increased Availability:** Guest systems may be transparently monitored and, depending upon the configuration, may be automatically rebooted in case of failure. The failure of a guest system has no impact on the other systems, which continue to deliver their service. Moreover, restart of a failed system is triggered by software, bypassing any hardware mechanisms, enabling a faster return to service. Hence, overall system reliability is increased.

The VirtualLogix Real-Time Virtualization relies on a foundation layer known as a hypervisor or a Virtual Machine Monitor (VMM). VirtualLogix VMM is modular and configurable:

- VLX Virtualizer is the core component of VLX VMM providing the basic features required by virtualization –memory partitioning, CPU virtualization and scheduling.
- The VLX Isolator component implements robust isolation. An OS, whether RTOS or OpenOS, executing in an isolated partition cannot access anything outside that partition. Hence, it cannot read or corrupt other partitions, and its I/O accesses are also controlled.
- The VLX Core BSP component provides virtualization of a few basic devices such as clock, timers, and PIC (Programmable Interrupt Controller). The manner in which other devices are handled will be described later.

If multiple guest OS' execute on the same processor, VirtualLogix VLX schedules the guest OS' in a way that maintains the RTOS' real-time properties.

Resource Partitioning

Physical memory is partitioned by the VLX Virtualizer, and each memory partition is allocated to a given guest OS. Thus, each guest OS may use its own native memory management mechanisms and policies without interfering with other guest OS'. This approach allows OS' that make use of the MMU, for example, Linux and Windows CE, and those that do not, for example, Nucleus and VxWorks, to run cooperatively on a single processor.

I/O devices used by a single guest OS are assigned exclusively to that guest OS. Native device drivers can thus be reused with no modification.

Resource Virtualization

Hardware resources required by several guest OS', such as the CPU, MMU, and FPU are virtualized so that they can be shared between all guest OS'. Such virtualization is performed by the VLX Virtualizer. The CPU is shared among guest OS' based on scheduling policies that ensure that real-time guest OS' (and their applications) are assigned the highest priorities and can thus maintain their real-time

guarantees. Once a guest OS has been granted CPU access, the OS uses its native policies to schedule its own applications and services.

The MMU is virtualized so that each guest OS may use it for its own purposes. As discussed above, MMU use by one guest OS is independent of that by another guest OS.

OS Isolation

When VLX Isolator is configured, memory partitions are protected and securely isolated from each other so that no OS can read or write into the memory partition of another OS. The VLX Isolator also controls I/O access performed by the guest OS.

Flexible OS Scheduling

VLX Virtualizer proposes different OS scheduling policies which can be configured. The usual policy is close to a FIFO scheduling policy within an OS: guest OS' are given priorities. The RTOS is assigned the highest priority, ensuring that it will preempt any other guest OS as soon as needed. Remember, it is the RTOS where time critical tasks are running.

Other OS' sharing a given priority level may be scheduled in a FIFO manner or on a fair share basis, depending upon the configuration.

Optionally, VLX also provides a unique scheduling feature that enables interleaved scheduling of activities from different guest OS'. For example, this may be used to execute:

- High-priority threads in the RTOS first
- High-priority threads in a OpenOS, in a second step
- Then lower priority threads in the RTOS
- And finally low priority threads in the OpenOS.

Device Virtualization

Executing different OS' simultaneously on the same processor is only one part of the problem. The guest OS' also need to share other hardware resources beside the CPU, the MMU and the physical main memory, namely, core devices and I/O peripherals:

- Core devices are used by all guest OS': clock, timers and Programmable Interrupt Controller. A UART used for OS' console and for debugging also falls into this category.
- I/O peripherals are I/O devices such as network interface controllers, disk controllers, and serial lines.

Core Devices

The VLX Core BSP component virtualizes core devices so that a guest OS' can use them seamlessly and concurrently. The Core BSP component physically manages these devices and cooperates with front-end device drivers configured in the guest OS'. Core BSP responds to requests sent by the front-end drivers and also propagates virtual interrupts to such guest OS' when appropriate.

The VLX Core BSP component is itself optional in the system configuration. The services it provides may be implemented differently, for example by extending a real-time OS to virtualize core devices for another guest OS'.

I/O Peripherals

I/O devices such as screen, keypad, USB, audio device, flash memory, camera, Bluetooth interface, and modem may be further classified depending upon their use by the various guest OS'.

Dedicated Peripherals

I/O peripherals that are used by a single environment, for example a Bluetooth interface exclusively accessed by a Linux system, are dedicated to that system. In such a configuration, the guest OS may use a native driver to manage the peripheral. Interrupts generated by the peripheral are virtualized and propagated to the guest OS by virtue of the Core BSP component previously described.

Shared Peripherals

Some I/O peripherals may be accessed by applications from both OS'. For example, the screen may be accessed by real-time applications dedicated to voice calls, and by a Linux system for browsing the file system on the flash. In such configurations, VLX imposes no predefined scheme and offers full flexibility. One of the guest OS' owns the peripheral and manages it through a real native device driver, as if it were a dedicated device. The guest OS owning the device also runs a so-called back-end driver that receives and mediates requests from other guest OS' that want access to that device. The back-end driver relies on the real native device driver to perform real I/O operations. The other guest OS' no longer use their native real device drivers, but instead use a "front-end" device driver, providing identical services to the guest OS. Such a front-end device driver communicates with its peer entity, the back-end driver. This scheme is commonly known as a split-driver model. However, contrary to some other approaches, VLX does not impose the choice of the guest OS that owns the I/O device.

Shared I/O peripherals require back-end and front-end drivers. VLX already provides support for a set of shared peripherals including LCD screen, touch screens, keypad, audio device, modem, and power management.

Virtual Peripherals

It is possible to configure a guest OS' with purely virtual devices that are not associated with actual hardware. For example, one may configure a virtual Ethernet network connecting two or more guest OS', enabling them to further communicate with whatever TCP/IP application they wish. VLX delivers a set of such virtual peripherals, including Ethernet and UART.

Communication

A typical OS supports multiple processes, provides them with services such as synchronization and inter-process communication, as well as scheduling policies, shared access to system and network interfaces, and memory management.

Similarly, the VLX virtualization solution provides each guest OS with synchronization - a virtual cross-interrupt mechanism - and inter-OS communication mechanisms. Usually, the basic low-level communication mechanisms are not exported to guest OS' applications, but are used by virtual and shared peripherals. Stacking the most appropriate virtual peripheral driver on top of the low-level communication mechanisms insures that the communication is well tuned to the application's needs

while hiding the low-level complexity from the application. This enables applications running in an OpenOS guest to communicate efficiently with applications running on the legacy OS or RTOS, using the most appropriate communication paradigm. Remember that such applications cooperate within the mobile handset to deliver an integrated service to the user.

Additional features

VLX-based systems can be configured with additional features executing in a dedicated partition, separated from legacy software as well as from the OpenOS environment. The dedicated partition operates an execution environment known as VLX Executive. Several services can be configured in this partition, such as monitoring or watchdog mechanism. More services will be added.

Monitoring

The Monitoring service is typically used during development. It captures scheduling-related events on the platform, such as hardware interrupts and VLX Virtualizer scheduling decisions. These events are collected locally and are then retrieved and analyzed by a tool running on a host system connected to the embedded device.

The Monitoring tool is running as a plug-in component in the VLX Developer IDE, based upon Eclipse. The various scheduling events are displayed graphically.

Watchdog

The Watchdog service can be configured to control the aliveness of other partitions and guest OS'. This service appears as a watchdog device in the guest OS' and requires a dedicated device driver to be configured in that OS'. If a guest OS fails to re-arm such a virtual watchdog in time, either because it is wedged, dead, or infinitely looping, the Watchdog service stops and restarts the failing guest OS. More complex escalation policies can be implemented, if needed.

VLX Developer

VLX Developer is the Eclipse-based IDE tool delivered with VLX. It enables system developers to generate the guest OS' and to build and configure the overall system comprising the VLX components, and the guest OS' to be deployed on the platform.

Mobile Phone Use Cases

The combination of highly integrated chipset solutions with Linux mandates new software architectures and technical choices that must be validated against critical features such as real-time support, robustness and efficiency.

VirtualLogix VLX for Mobile Handsets enables new mobile phone integrated architectures to execute modem and other applications on the same processor core by sharing hardware resources. With VirtualLogix VLX, the existing modem protocol stack, validated on its own RTOS, can be reused without modification, while Linux with its drivers, IP protocol stacks and applications can be adopted at minimal cost and effort.

The VirtualLogix VLX high-performance virtualization solution preserves RTOS native behavior while providing the advanced features needed for mobile phones, such as fine grain scheduling, power management, and optimized cross-OS communication.

For example, VirtualLogix VLX enables Philips Semiconductors to reuse its mobile phone stack over several handset generations, saving man-years of engineering effort and releasing the end products to the market faster.

Consolidation Use Case with VLX

The Use Case is that of the consolidation of two environments - that had previously executed independently on their own processors and OS' - to execute concurrently on a single processor. The graphic shows the consolidation of a RTOS and a Linux OS on a typical feature phone platform.

In order to achieve the optimum consolidation, the functions delivered by the Linux environment and those delivered by RTOS must be defined, as must those that are delivered co-operatively. Such decisions are usually intuitively obvious and mostly dictated by the legacy and Linux configurations being consolidated.

Assignment of I/O Peripherals

As part of system configuration, it must be determined how I/O peripherals are to be managed, that is, which peripherals are used exclusively by the RTOS or by Linux. For peripherals that are accessed by both, it must be determined which guest OS owns the device and virtualizes it for the other guest OS.

For example, the microphone, speaker, SIM card, I2C, GPIO, battery charger, physical watchdog and the radio modem are dedicated to the RTOS. The keypad, display, image/video, USB and Wi-Fi peripherals are dedicated to the Linux OS.

The audio, interrupt controller, timers, power management and UART peripherals are owned and virtualized by the RTOS, and can be used concurrently by the Linux OS applications. The Bluetooth peripheral is owned and virtualized by Linux, and can be used by the RTOS applications.

Benefit of Multiple Independent Application Stacks

Consolidation enables reuse of application stacks - OS, device drivers, middleware and applications - with minimal development effort, while avoiding the use of a dedicated processor for each OS.

In a mobile phone, this enables the reuse of the legacy RTOS with the modem driver and the radio protocol layers without the necessity for modifying them for the OpenOS, thus eliminating redesign, test, and validation time and effort.

The virtualization of the modem enables Linux applications to set up calls for their own needs simply by using the features delivered by the RTOS. Similarly, Linux provides connectivity over Wi-Fi and Bluetooth. Such connectivity can be used by RTOS applications without having to develop such connectivity management in the real time environment.

Security Use Cases with VLX

VLX enables can enhance a consolidated system beyond simply executing an OpenOS concurrently with a RTOS. VLX can also expand the set of services delivered by a mobile handset running a non-consolidated system. Security is one of the areas where virtualization adds significant value.

Digital Rights Management Example

Secure and effective digital rights management (DRM) will be critical in the transformation of the handset from a mobile voice and basic data terminal into the consumer's primary mobile communications, information, financial transaction and entertainment center.

Taking the example of video playback on a mobile phone, virtualization enables the establishment of a dedicated partition of a DRM-based player, external to the OpenOS partition. By isolating the OpenOS with VLX Isolator, the DRM player is guaranteed to remain uncorrupted by any malicious software downloaded to the OpenOS environment.

When the mobile phone user downloads a DRM-encrypted video, the OpenOS is used with some secured connection (https). As the video is encrypted it can be stored where it best fits the user's expectation. In order to play the video, one has to use trusted DRM player which is the only one to have the appropriate keys to decrypt the video. The secure isolation established between the DRM layer and the OpenOS insures that the DRM player cannot be corrupted by any piece of software or system modules which could be loaded and run on the OpenOS side. Moreover, the DRM player is also able to grab access to the display screen for the duration of the video insuring the video output cannot be caught after decryption.

Call control example

The framework enabled by virtualization may also be used to control the behavior of applications running within the OpenOS environment. Applications downloaded by the user must be prevented to send SMS or establish communications without explicit permission from the user. Virtualization permits to insure that the modem and the communication stack are not under the direct control of the OpenOS. Hence establishing a communication from the OpenOS can be intercepted by trusted software running outside of the OpenOS. Such an interception permits to verify whether the user agrees to set up such a communication or not. This verification can be performed without worrying whether the OpenOS is corrupted or not since it is performed outside of such a OpenOS.

Management Use Cases with VLX

Availability through Monitoring

Let's assume we now have a mobile handset with a OpenOS running in a partition and a legacy RTOS with the protocol stacks running in a different partition. If the environment supported by the OpenOS is open to let the user downloads applications or module, then the OpenOS environment may be at risk and face failure, dead-lock or live-lock situations. The virtualization architecture permits to monitor the liveness of one or more environments. If a guest system appears to have failed or blocked, the monitoring service can take appropriate actions, such as: actually stopping the suspected system, alerting other partitions so that they for example grab the control of the screen, and restarting the crashed system. This kind of response to one system's failure insures that the device is still usable for basic services such as giving a phone call even though the OpenOS is being restarted. It also insures that the service delivered by the handset will be fully functional after the OpenOS has restarted without requiring the user to power off and on his device.

Upgrade

Because the guest OS' running on the virtualization layer are independent of each other, upgrading software in the mobile handset is simple. Once a new guest OS image has been downloaded onto the

platform, using the new environment is simply a matter of stopping the running guest OS and restarting it with the new image. Checks must be performed to guarantee that the new system functions correctly. In case of failure, the monitoring services should fall back to the previous image, so that the handset reverts to a stable and usable state.

Summary

The VirtualLogix VLX virtualization solution enables mobile handset designers to enhance feature phones with smart phone functionality. It does so by supporting the simultaneous execution of different operating systems and their respective environments on a single processor while preserving their original features and properties and delivering the following benefits:

- Implement smart phone capabilities on feature phone hardware platforms.
- Consolidate multiple unmodified software stacks - both real-time and OpenOS - while maintaining their QoS properties, *on a single processor* by simply configuring the system.
- Execute legacy, real-time and general purpose software without complex and lengthy redevelopment and integration, thus minimizing time to market.
- Increase security for the user, operator and content providers.
- Enable higher availability of software components to maintain core services even in the event of contextual software failures of richer components.

VirtualLogix

292 Gibraltar Drive, Bldg B2
Sunnyvale, CA 94089, USA
Tel +1 408 954 7355
Fax +1 408 432 7235