

White Paper

Integrating Operating Systems with Freescale's Cellular Software Platform

Document Number: INTGFSLCELPLATWP
Rev 0
1/2007



Overview

Smartphones, once a small portion of the overall cell phone market, are growing more popular as cellular data rates and phone computing power increase. Each new generation of phones accommodates more sophisticated end-user and network applications, and phone software overall becomes exponentially more complex. Addressing the smartphone opportunity comes with a challenge to develop, integrate and test a lot of software without compromising quality. Freescale believes that open operating systems are the most effective tools that can help developers, network operators and manufacturers meet this challenge.

Freescale's Cellular Platform Architecture (CPA) is designed to be a reusable, OS-independent interface to adapt a cellular platform to different operating systems. CPA can take advantage of unique technical benefits in our Mobile eXtreme Convergence (MXC) cellular platform, including clean separation of the communications and applications stacks, for faster and less expensive software development and more efficient approaches to 3G standards.

Contents

1	Introduction	1
2	The Business of Phone Software	1
2.1	Network Operator Perspective	1
2.1.1	Operators Need Open Operating Systems	2
2.2	Handset OEM Perspective	2
3	Technical Concerns	3
4	MXC + CPA = Reuse = Flexibility + Quality	4
5	Building Blocks = Components + Frameworks + Design Patterns	6
5.1	Components	6
5.2	Frameworks	6
5.3	Design Patterns	7
6	Summary	7

1 Introduction

Developing software for cellular phones today is extremely complex. You can view this complexity from two perspectives: the radio frequency (RF) hardware and software that make up the physical communications layer, or the applications, middleware and higher-level communication protocols. This overview examines the challenges of cellular from the second perspective.

Many factors contribute to increasingly complex cell phone software, including.

- **Burgeoning feature sets** as end users demand richer multimedia content and phones become all-in-one devices that support cameras, music and video players, personal information management, Internet end e-mail access, streaming TV and emerging standards such as IP Multimedia Subsystem (IMS) and firmware-over-the-air (FOTA)
- **Multiple operating system environments**, each with their own technical requirements and customized for different tiers, OEMs and operators
- **Network infrastructure compatibility** to comply with older, new and emerging communications standards
- **Complex protocol standards** with many alternate means of accomplishing the same function
- **Extremely low power consumption** constraints as battery life remains constant, but hours of use by end users increase and cell phones require more power to run more processor-intensive applications
- **High communication data rates** and scalable bandwidth put a processing burden on phones
- **Hyper-competitive delivery schedules**, driven by market demand, that force operators to release new products every six months or so
- **Product cost constraints** will not allow the price of end products to rise above what the market will bear in a very competitive, price-cutting environment
- **Very complex test environments** that must accommodate GCF conformance testing, separate IOT tests for each major network, tests for each major operator, and field testing for complex use cases such as urban canyons, large lakes or mountains

When examining software complexity from the perspective of applications and the operating systems that host them, four issues in particular stand out: Demand for new features, rapidly increasing data rates, support for multiple operating systems and the difficulty of testing in a cellular environment.

Freescale's Cellular Platform Access (CPA) interface was designed to meet these demands. Platforms with a high degree of reuse are the only means of addressing market and software complexity while maintaining quality. Freescale's software architecture is specifically designed to support multiple open operating systems and to minimize the cost, time and effort needed to implement the OS adaptation layer.

2 The Business of Phone Software

Freescale's CPA was designed to address both the technical needs of developers and the business needs of OEMs and operators. The cost and time of developing and testing phone software is an important business factor that can determine whether OEMs and operators can compete and survive.

2.1 Network Operator Perspective

In the cellular market value chain, network operators' primary commodity is still voice calls. Since the market for cellular services is nearly saturated, it is almost impossible to increase revenue by acquiring new subscribers. In addition, the cost of customer churn—subscribers buying services from a different network operator in order to acquire the latest phone at a price subsidized by the network operator—is significant. In this saturated voice market, operators must turn to data services, increasingly at broadband rates, to increase their average revenue per user (ARPU) and reduce subscriber churn costs.

2.1.1 Operators Need Open Operating Systems

To provide these additional features and services, network operators need handsets that support higher data rates. An example of a new broadband service is push-to-show, where a cell phone transmits real-time video and audio images so that another person can see and hear the same event remotely. Operators must also deploy unique software applications to create the services which differentiate them from their competitors. To do this, operators are increasingly requiring open operating systems on handsets to control development and support costs, and to prevent the need to port or redevelop these applications for each handset manufacturer's proprietary operating system and middleware. Skydeck's blog puts it this way:

*We believe that five years from now it will be very difficult to buy a phone that does not have an open application development environment, in the developed world at least. Why... It's because open platforms are the only cost-effective way for carriers to create the services that they want to sell to customers, while still being able to differentiate.*¹

Nomura International's *Mobile Software* report from September 2006 concurs.

*In order to have greater control over the user experience operators need to become more aware of handset and software issues, as it is only through these can they deliver the services to revitalise their revenue stream. We see mobile operators becoming much more active in the mobile software space over the next 12-24 months.*²

Each open operating system has its own strengths and weaknesses. Microsoft Windows is well-suited for enterprise environments. Pulling new technology from the desktop, Windows Mobile is capable of hosting familiar desktop applications such as the Internet Explorer browser. Symbian is well-suited to consumer electronics. Linux is highly flexible and can draw upon a large open source community. RTOS application frameworks such as the former TTPCOM AJAR are more cost-efficient, and more suited to mid- and low-tier devices.

Likewise, each operating system has its own drawbacks. Linux, Microsoft Windows and Symbian, due to their large memory requirements and intensive computational needs, are best for more expensive smartphones. Windows is controlled by Microsoft. Linux, while flexible and open from a platform perspective, is notoriously fragmented from a phone application development viewpoint. Symbian can be difficult to program for and there is limited documentation and example software.

2.2 Handset OEM Perspective

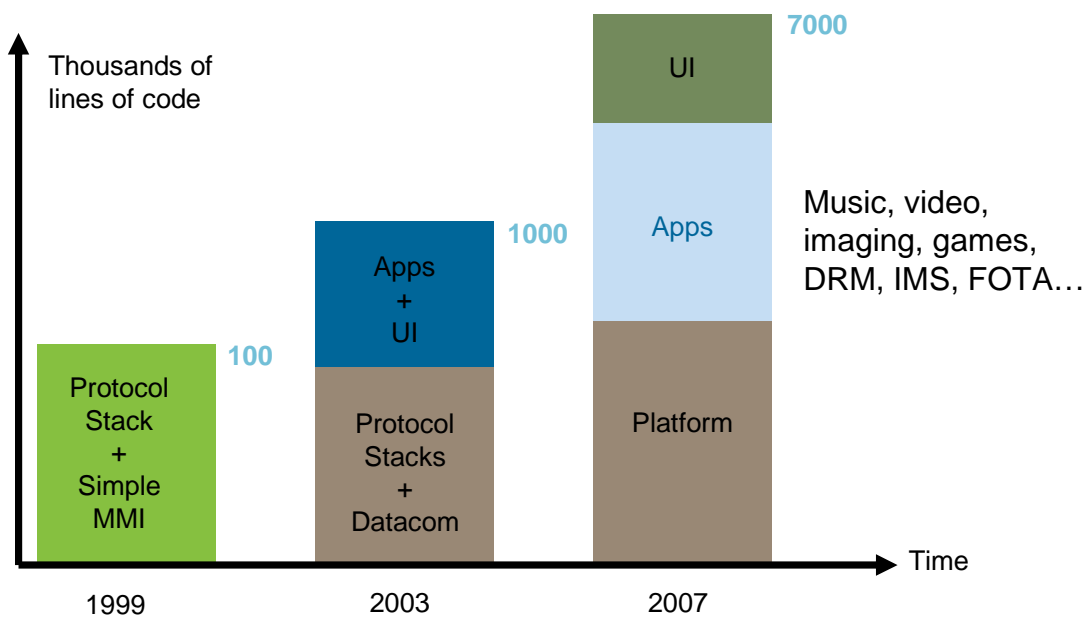
In order to prevent a single company from dominating the cellular handset software market, the largest OEMs and network operators want significant control over the OS and application framework, which handset applications are based on. Operators want handset offerings that are consistent with their competitors. For handset OEMs, the operating systems and application frameworks that they use to develop handsets is a significant investment decision.

The combination of business strategies employed by network operators and handset OEMs drives the need for features to be developed on many different operating system environments. One thing is certain: the diversity of operating system environments and the demand for differentiating features have caused the number of lines of software that the OEMs must develop and support to reach extraordinary levels, as illustrated in Figure 2.1.

¹ *Open Moto*, May 17 2007, <http://skydeck.com/blog/technology/open-moto/>

² Nomura International London, European Equity Research, *Mobile Software*, September 2006

Figure 2.1 Lines of Code Increase with Higher-Bandwidth Applications



3 Technical Concerns

Ensuring product quality for this amount of software is a daunting task regardless of the environment in which the software is deployed. Cellular, however, presents a unique set of software testing challenges. Testing software in cellular handsets is difficult for several reasons.

- It is the nature of cellular that there are protocols between the handset and many network infrastructure components, including the cellular base station, mobility and authentication infrastructure, telephony switches, SMS message service centers, email and browser gateways, etc.
- Because network components and handsets are supplied by many different vendors whose equipment must work together, interoperability between different vendors equipment is a primary concern.
- RF communication is notoriously variable in its characteristics when it encounters natural and man-made topography, population density and other factors that affect real-world handset performance.
- Many companies and governments have a variety of incentives, financial and other, to affect the 3GPP protocol standards. Frequently this results in specifications with several different ways of accomplishing similar functions.
- There is no mathematically precise way of describing the very complex set of behaviors required for cellular handsets and infrastructure to interact. Writing the 3GPP protocol specifications in natural language inherently leads to some ambiguity.
- Networks and users are truly global; therefore, to conduct real-world testing a company must be physically present at many locations around the world.

All of these factors conspire to make testing handsets extremely important—and extremely costly. For this reason, handset manufacturers attempt to minimize the number of lines of code changed as each new feature is implemented. Addressing risk due to software change in this way, however, can lead to a software architecture that is excessively rigid.

*A PLA is a common architecture for a set of closely related products. As the set of products and their features changes, the PLA must evolve as well. A typical problem in managing such evolution is that the overall structure of the PLA slowly but surely degrades. This is caused by the fact that a set of individual, localized changes does not necessarily result in the best structure for the overall PLA.*³

Indeed, this is a documented problem in handset software.

*Most of the proprietary OSs have been evolved far beyond what they were originally intended for and are consequently not easy to upgrade and keep adding features to. The fact that most have not really been evolved into proper platforms means that adding new features has to be done on a case by case basis and there is very little ability to reuse the effort that goes into the implementation.*⁴

Operators need new applications to differentiate themselves from their competitors. Many of these applications will use very high data rates to deliver rich multimedia content to subscribers. These applications will be implemented on a variety of operating systems to meet end user needs and to support handset OEM strategies. The software company SysOpen Digia PLC's 2006 first quarter interim financial report summarized the situation:

*In the case of many, a challenge to rapid market growth is expected to arise in the form of the requirement for variations in platform according to market segment and operator, while maintaining the integrity, flexibility and high quality of the product platform.*⁵

4 MXC + CPA = Reuse = Flexibility + Quality

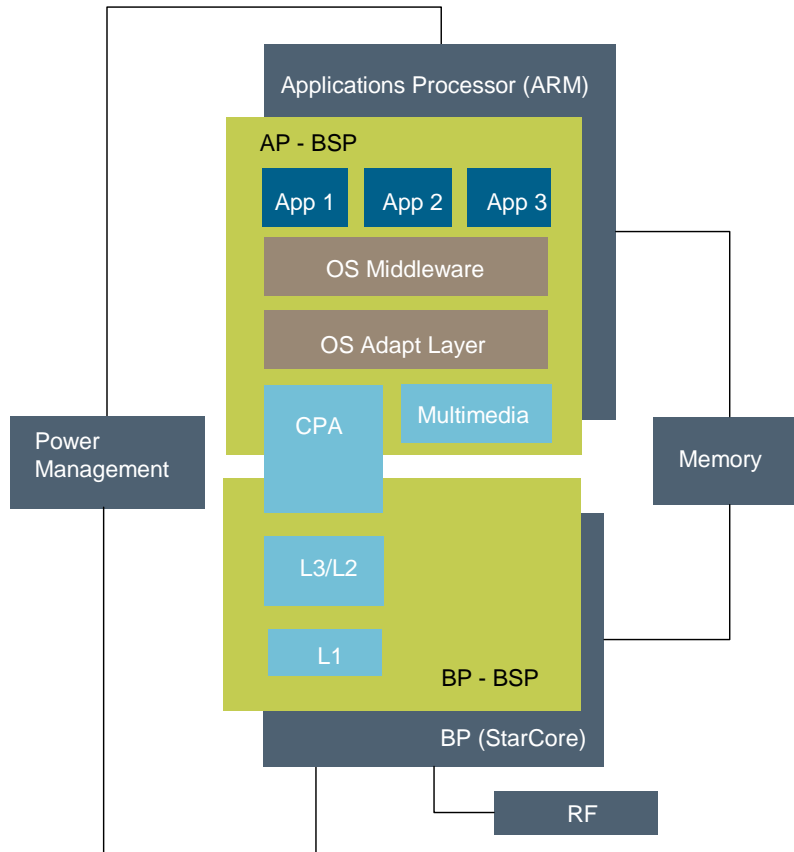
Freescale has carefully designed CPA to directly address these needs—rapid implementation of new features, efficient support of high communication data rates, support for multiple operating system environments—all while maintaining the high quality of handsets built on top of the CPA framework. To better understand CPA's function, Figure 4.1 shows CPA in context with the rest of the components in Freescale's Mobile eXtreme Convergence (MXC) cellular platform. The figure also shows the “variable” software components that operating system vendors and handset OEMs supply to create a complete handset.

³ *Refactoring Product Line Architectures* by Matt Critchlow, Kevin Dodd, Jessica Chou, and André van der Hoek, Department of Informatics, School of Information and Computer Science, University of California, Irvine

⁴ Nomura International London, European Equity Research, Mobile Software, September 2006

⁵ *SysOpen Digia Plc's First Quarter Interim Financial Report 2006*, press release, <http://www.kauppalehti.fi/4/i/eng/stocks/share/bulletin.jsp?id=200604278411&stoid=SYS1V&comid=SYS#alku>

Figure 4.1 Components of Freescale's Mobile eXtreme Convergence (MXC) Cellular Architecture



One notable and unique feature of the MXC hardware architecture is that it consolidates all of the real-time sensitive network communications software on the StarCore DSP. This hardware/software partitioning provides several benefits. Most open operating systems cannot support the very low latencies that the 3GPP air-interface standards require. Even if open operating systems could support the timing requirements, they still would require significant porting and retesting as described above. The MXC architecture allows one external random access memory to contain code and data for both the application and baseband processor, resulting in reduced part count, more efficient memory use and overall lower handset bill of materials.

By consolidating the entire 3GPP protocol stack on a single processor, the MXC architecture eliminates the need to port that software to each operating system environment. To further enhance software reuse between operating system platforms, the CPA architecture divides the problem of adapting the modem software to the operating system and application framework into two parts: the CPA component, which is OS-independent and highly portable, and the OS adaptation layer.

CPA was designed with three guiding principles.

- To provide a reusable, OS-independent interface for adapting the cellular platform to each operating system. The common OS-independent layer of software allows test frameworks and extensive suites of test scripts to be reused across each OS implementation.
- To minimize the code required to adapt each OS to the CPA interface by studying the interfaces presented by each OS and change CPA wherever required to make the adaptation simpler.
- To research and apply software architecture best practices to ensure maximum reuse between operating systems and maximum flexibility to add new features.

The remainder of this overview explains how these principles were applied in the development of CPA.

5 Building Blocks = Components + Frameworks + Design Patterns

Before describing the CPA architecture in detail, it is useful to describe the three building blocks of the CPA architecture: software components, software frameworks and design patterns. The term “software components” can have context-specific definitions. To avoid confusion, it is important to define software components within the context of this overview. Much is written about commercial off-the-shelf (COTS) software components. COTS components are not what we refer to here; rather, software components as discussed here are families of related software building blocks. Software components in this sense are fundamental to product line architectures (PLAs), of which CPA itself is a “macro” component.

5.1 Components

As implied by the previous statement, software components can come in two types. Macro components are depicted in Figure 4.1 above. The other type of component which we refer to most often in this overview is the design component. Design components are defined as cohesive, portable, and independently testable units of software construction. Design components enable black-box reuse to rapidly build the macro component variants required for each specific platform targeted at different tiers and customers.

Design components are implemented in terms of a component model. The Freescale component model has two important characteristics: an OS abstraction layer and a processor- and compiler-independent messaging mechanism. The Freescale model is OS-agnostic, lightweight and portable across open operating systems and real time operating systems. (Examples of other component models are Microsoft's COM and Sun Microsystems' Java Beans. However, because those models are tied to specific languages and operating systems, neither one fits the context of Freescale's model).

5.2 Frameworks

The second major building blocks of CPA are frameworks. In the article *Applying Patterns and Frameworks to Develop Object-Oriented Communication Software*, Douglas C. Schmidt describes the purpose of software frameworks.

Frameworks define “semi-complete” applications that embody domain-specific object structures and functionality: Class libraries provide a relatively small granularity of reuse.... In contrast, components in a framework collaborate to provide a customizable architectural skeleton for a family of related applications. Complete applications can be composed by inheriting from and/or instantiating framework components...this reduces the amount of application-specific code since much of the domain-specific processing is factored into the generic components in the framework.⁶

The goal of CPA development is to minimize the cost, calendar time and effort to implement the OS adaptation layer. Since the protocol standards and the underlying hardware do not vary between instantiations of the Freescale platform created for each operating system and IC set, using the framework structure allows us to divide the software into OS-specific and OS-independent, but domain-specific, components.

One defining characteristic of frameworks is that instead of the calling hierarchy originating with applications which call lower layer interfaces, frameworks contain their own control logic and require that applications implement “call backs” which are called by the software framework. These call backs are written to a specification determined by the framework. This “inversion of control flow” is sometimes referred to as the “Hollywood principle,” or “don't call us, we'll call you.” The advantage of this arrangement for CPA is that the CPA framework can provide a significant portion of the software required to interface the cellular modem as an OS-independent piece, thus minimizing the OS-dependent code which is contained in the call backs and the resulting much simpler OS adaptation layer.

When you use a framework, you reuse the main body and write the code it calls. You'll have to write operations with particular names and calling conventions, but that reduces the design

⁶ “Applying Patterns and Frameworks to Develop Object-Oriented Communication Software” by Douglas C. Schmidt. *Handbook of Programming Languages*, Volume I, edited by Peter Salus, MacMillan Computer Publishing, 1997

*decisions you have to make. Not only can you build applications faster as a result, but the applications have similar structures. They are easier to maintain, and they seem more consistent to their users.*⁷

Frameworks do have drawbacks. First, frameworks are difficult to define.

*If applications are hard to design, and toolkits are harder, then frameworks are the hardest of all. A framework designer gambles that one architecture will work for all applications in the domain. Any substantive changes to the framework's design would reduce its benefits considerably, since the framework's main contribution to an application is the architecture it defines. Therefore it's imperative to design the framework to be as flexible as possible ... That makes loose coupling all the more important; otherwise even a minor change to the framework will have major repercussions.*⁸

To ease the definition of the CPA framework, we have built our framework around a component model as mentioned above. One advantage of using the component model is that the messaging between components provides the loosest possible coupling between the components in the framework and between the CPA servers, which is discussed below.

The second major drawback to frameworks is that they can be difficult to document.

*Since frameworks are reusable designs, not just code, they are more abstract than most software, which makes documenting them difficult. ... The principal audience of framework documentation is someone who wants to use the framework to solve typical problems, not someone building a software cathedral. Patterns seem to be well suited for this audience.*⁹

5.3 Design Patterns

In order to document the CPA frameworks in this overview, we have chosen to do so partially by describing the design patterns used to implement the frameworks. The other advantage to using design patterns is that some of the code in the CPA frameworks actually exists as OS kernel mode drivers. Kernel mode drivers are very difficult to make portable across operating systems without significantly increasing their complexity and negatively impacting efficiency. For high-bandwidth data communications implemented on cost- and power-constrained devices, this inefficiency is prohibitive. In addition, it has been our experience that the code required to abstract the OS kernel mode services is often as large and as complex as the "portable" part of the code. Design patterns allow us to reuse the design if not all of the source code in these instances.

6 Summary

Freescale's CPA uses a framework paradigm for three main reasons:

1. To maximize code reuse between platforms that require different OS and application frameworks
2. To maximize flexibility so that new features have minimal effect on existing code
3. To minimize time to market because less code must be written between product instantiations and as a result fewer defects will be found. Given the difficulty of finding and fixing software defects this can have a profound impact on time to market.

The Freescale MXC hardware architecture with its ability to consolidate all of the important modem functionality on the powerful StarCore DSP and innovative technology to allow both processor cores to share a single external memory part creates a uniquely cost effective hardware solution for 3G cellular phones. The CPA software architecture helps OEMs bring powerful handsets to market quickly with the flexibility to tailor the handsets to their individual business strategies.

⁷ *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, John Vilissides., Addison-Wesley, 1995, ISBN 0-201-63361-2

⁸ Ibid.

⁹ *Documenting Frameworks using Patterns*, Ralph E. Johnson. Conference on Object Oriented Programming Systems, Languages and Applications, Vancouver, British Columbia, Canada, 1992, pages 63 – 76, ISBN 0-201-53372-3

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

www.freescale.com/support

USA/Europe or Locations Not Listed:

Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright license granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.