Challenges for Mobile Middleware Platform: Issues for Embedded Open Source Software Integration

Toshihiko Yamakami

ACCESS
Toshihiko.Yamakami@access-company.com

Abstract. Linux is penetrating into mobile software as the basis for the mobile middleware platform. It accelerates the increasing visibility of open source software (OSS) components in the mobile middleware platform. Despite multiple challenges in mobile embedded software engineering, it is crucial to promote open source-aware development of the mobile software platform. The author presents the challenges to open source software integration in embedded software development. The author discusses the open source-aware software development and identifies a path to transition for moving toward it.

1 Introduction

Linux has penetrated into a wide range of digital appliances, e.g. mobile handsets, digital TVs, game consoles and HD recorders. It facilitates the reuse of PC-based rich user experience data service software with high speed network capabilities in an embedded software environment. As Linux-based software is widely adopted in digital appliances, the original weak points of Linux in an embedded environment have been addressed e.g. real time processing and battery life capabilities. The author presents these challenges, and suggests a path towards a more open source-aware development model. Then, the author describes how open source-aware development can be deployed from the perspectives of organization and design.

2 Purpose and Related Work

The purpose of this research is to identify a path toward open source-aware software development evolution and its implications for software development management.
OSS has been increasing its visibility in embedded softawre [5] [3] [1]. There are several examples of emerging foundation engineering utilizing OSS in the mobile platform software [4] [6] [2]. This has caught industry attention worldwide.
The originality of this paper lies in its discussion of impacts from OSS-based foundation collaboration in the mobile industry.

3 Open Source-related Landscape

3.1 Driving Forces for Embedded Open Source and Issues

There are multiple factors that drive embedded software engineering:
– Market pressure to shorten time to market,
– Market pressure to reduce software development and maintenance cost, and
– Device convergence.

As the many more devices become network-enabled, the size of network and application software grows, which impacts the cost structure of digital appliances. Managing the costs of maintaining this large-scale code base is a critical concern for embedded software development.

Nowadays, advanced mobile handsets consist of (a) Linux kernel, (b) middleware platform, and (c) application code. Each is 5–10 million lines of code.

This is causing a transition from constraint-based specialized software development to OSS-based large-scale heterogeneous software development.

The amount of OSS code is increasing dramatically. Almost 10 million lines of code are used for just the middleware of advanced mobile handsets, excluding the kernel itself and higher level applications. The ratio of OSS code in relation to them is increasing. It is estimated that 80–95 % of the Linux-based mobile middleware platform is originated from OSS. The ratio is expected to increase in the future.

However, there are several constraints in the mobile middleware platform from (a) embedded software, (b) mobile-specific service development, and (c) OSS-based software management.

The combination of these issues provides a significant challenge for today's embedded software development.

3.2 Issues from Embedded Software Engineering

Mobile handsets require the inclusion of special hardware management for telephony, which is device-specific.

– Customization for hardware integration: Embedded software depends on some hardware components. These components need hardware-dependent coding.
– Stability for embedded integration: Generally, open source software is based on bazaar-style development. Embedded software has inheritent constraints on software updates after factory shipment. Since the software is written into ROM (read-only memory), it cannot be amended. Embedded software development needs to address this challenge.
– Real-time processing requirements: Mobile phone requires real-time processing of incoming calls. OSS generally does not set performance as the top priority. Device development needs to address device-specific performance requirements.
– Battery life duration requirement: Mobile phone and other battery-driven devices require special consideration to the duration for battery life.

3.3 Issues from Mobile-specific Service Development

Mobile data services are hot issues in mobile business development. This leads to multiple issues that need in mobile middleware platform development.

- Wireless carrier customization: For differentiation, wireless carriers would like to deploy carrier-specific services. Embedded software development needs to address this carrier-specific customization.
- Synchronization for handset launch schedule with OSS roadmaps: Mobile handsets have some seasonal cycle for shipment depending on each country for marketing which is irrelevant to OSS roadmaps.

3.4 Issues from OSS Management

There are issues surrounding OSS management in general.

- Design and Customization
  - Coordination among multiple OSS components: Each OSS component has a different release cycle, major version updates and roadmap. In order to facilitate product commercialization, coordination and version selection are crucial.
  - Coordination among multiple dependencies among OSS component sets: When a large number of OSS components are used, different OSS components have different dependencies on some common OSS components.
- Miscellaneous Non-coding Tasks requiring Resources
  - Evaluation to choose components and versions: There are an emerging number of OSS components with different version releases. Commercialization takes care of evaluation and version selection.
  - Synchronization for software development with upstream OSS project roadmaps: When there is visibility of an upcoming a major version up of major OSS component, commercialization needs to synchronize a future development plans and future major version updates.
- Community Coordination and Management
  - Granting back to the OSS community: As a good citizen of the OSS community, general patches and modifications need to be granted back to the original OSS community.
  - Constraint to disclose handset commercialization schedule: During interactions with the OSS community, there are some non-disclosable trade secrets.
- Legal issues
  - GPL contamination: GPL code needs to be carefully managed in order not to disclose any proprietary software in wrong use of GPL code.
  - Export compliance: Some of encryption modules need to be managed in compliance to appropriate import/export compliance procedures.
  - Patent protection: As well as any proprietary software, OSS modules also need to address patent protection.

4 An Open source-aware Software Development

The management of these issues is leading to a new software design and maintenance paradigm. Open source-oriented embedded software engineering requires the following special expertise and coordination:

– Interface design between OSS, customized, and proprietary components
– Professional service for code and packaging management, and evaluation

A three-stage model of software development transition towards open source-aware development is depicted in Fig. 1. Most of the proprietary software development depends on the assumption of in-house development. This influences the organization, design process, and source control schemes. Considering the fact that the major part of the software development is shifting towards an open source-based one, at least for the middleware level, it is crucial to manage the transition. The transition starts with proprietary development. In the proprietary development, the entire code base is owned by the company. It does not require per-module code and license management.

During OSS penetration, some of the modules may be replaced by OSS codes. However, this is a replacement-oriented process, not one that wholly replaces the development process.

When OSS penetration reaches a certain point, the whole development process needs to be revisited. Code and license management is done on a per-module basis. The OSS professional service needs to be deployed in the development process.

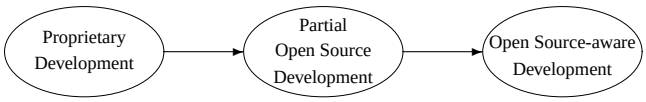

Fig. 1. A three-stage model of software development transition towards open source-aware development.

The software development team organization with open source awareness is illustrated in Fig. 2.

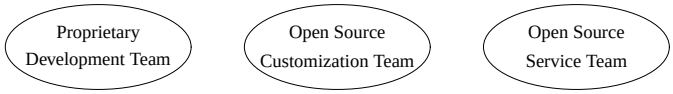

Fig. 2. Open source-aware development organization.

The software design process with open source awareness is depicted in Fig. 3. It is difficult to isolate customized OSS components from proprietary components post-process. Therefore, a clear separation of design is necessary during the software design process.

Multipel OSS components and their customization need to be integrated with awareness of code control, version control and license control. When there are conflicting OSS dependencies, it requires minor adjustments, e.g. back-porting et al. Also, the total code base including proprietary and OSS components needs

to be integrated with awareness of code control, version control and license control.

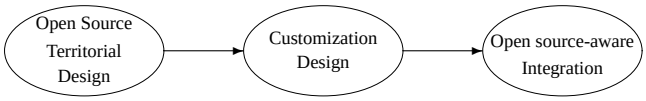

Fig. 3. Open source centric design principle.

5 Transition Management

Software vendors are aware of the importance of OSS. However, the challenges posted by less visibility and higher diversity of projects and codes make it difficult for software vendors to make a best-fit blending of multiple OSS components.

Embedded software needs to cope with the time-dimensional issues of software management. There is a lot of hardware-dependent code in the embedded software. This makes the transition from one software framework to another one difficult.

Also, this hardware-dependency puts an unavoidable portion of OSS components in need of case-by-case customization. Transition management needs to address this type of customization in the case of embedded software engineering.

In addition to the requirements to address multiple license terms in different OSS components, each customization needs to be carefully separated from other code to ensure the proper license management and grant back to the upstream OSS versions.

This puts the fundamental heterogeneity into embedded software engineering. Transition management needs to give considerations to organization, process, architecture, and source code management ahead of any transition management.

6 Conclusion

Since the major portion of the large-scale software platform consists of OSS components, the software development process needs to revisit this reality in the long run. The mobile industry is one of the areas where this transition is becoming increasingly visible. The author describes the challenges for large-scale software project with a large number of software components. The author proposes an open source-aware software development process. This will bring the procedural and organizational impacts on embedded software development. In the past, embedded software development focused on code-size-aware hardware-specific coding. There is ongoing radical transition of software development towards tens of millions of lines of code in an embedded environment.

Increasing involvement of OSS components leads to open source-aware development. In-depth analysis reveals the new heterogeneity caused by multiple different OSS components in an integrated embedded software context. This heterogeneity needs to be addressed through the advanced design of the entire software development process and its transitions.

References

1. Barr, M., Massa, A.: Programming Embedded Systems: With C and GNU Development Tools (2nd Ed.). O'Reilly Media, Inc. (2006)
2. Google: Android - an open handset alliance project. http://code.google.com/android/ (2007)
3. Hollabaugh, C.: Embedded Linux: Hardware, Software, and Interfacing. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
4. LiMo Foundation: LiMo Foundation Home page. http://www.limofoundation.org/ (2007)
5. Massa, A.J.: Embedded software development with eCos. Prentice Hall (2002)
6. Symbian Foundation: Symbian foundation web page. available at: http://www.symbianfoundation.org/ (2008)