Development of Measurement Software Platform

Yuichi Morinari, Yasuyuki Kato

[Summary] Recently, generic operating systems (OS), such as Embedded Windows and Linux, have become popular and there is more demand for porting applications to help cut costs. However, it is difficult to port software because product features depend on the OS and hardware. We have developed a measurement software platform to improve portability of measurement applications between different products.

1 Introduction

Generally, software platforms are composed of the OS, middleware and hardware control modules needed to run the software and hardware. Application software (hereafter applications) that runs on different hardware can be developed by designing it according to the interface specifications of this software platform.

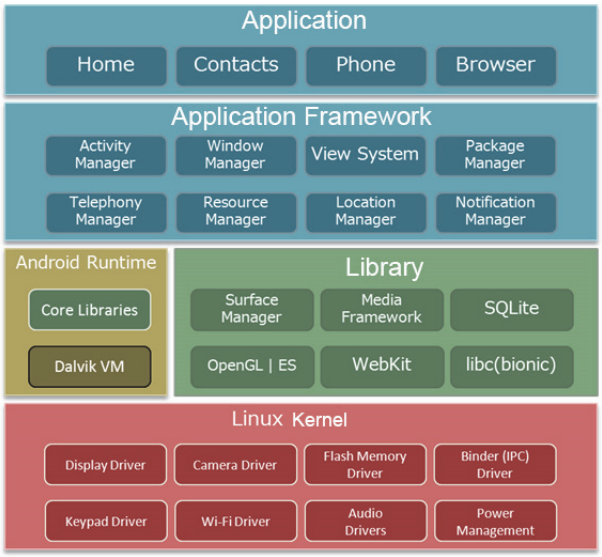

Figure 1 Software Platform Architecture (Android)

Figure 1 shows an example of the software platform architecture using Android OS. Generally, in software platform development, each software layer such as the OS (kernel), libraries, etc., is abstracted and interfaces between layers are defined. Applications developed using interfaces between these abstracted layers have high portability due to the low hardware dependency.

This article explains some examples of how to improve application portability between different hardware and operating systems applied to software for measurement instruments.

2 Development Concept

2.1 Background

In order to measure and analyze carrier wave data on increasingly common wired and wireless networks, high speed processing on measurement instrument is required. In addition, quick adjustment of software platform is also required for the hardware which rapidly evolving performance and specifications such as CPUs, memory, storage, etc. To ensure short Time To Market (TTM) and meet the need for reduced costs in development of measurement instruments, our company has been focusing development efforts on building a software platform supporting the features of commercially available CPU boards and all-purpose operating systems such as Embedded Windows and Linux.

Furthermore, the measurement instruments must be compatible with Windows because development of mobile terminals and various devices requires use of Windows PCs. Additionally, the many measurement instruments on production lines are batch-controlled over a network and required high-speed remote control, stability of long-term operation and low price. We have developed software platform specified for the product line as well as application software embedded in the measurement instruments to respond to the diverse needs measurement instruments. However, such systems still have very poor portability among product lines. And make matter worse, common instrument settings and external remote command for each product line are developed delicately and this hinders reduction of development costs.

Additionally, the recent product mix is driving increasing demand for faster more efficient porting of applications among different product lines. Porting applications among different product lines requires unification of the software

platform interfaces. A common software platform for measurement instruments is currently being developed to solve these problems.

2.2 Basic Development Concept

Development of a software platform for measurement instruments aims to increase application portability between product lines based on the following three concepts:

(1) Separation of OS and applications

(2) Support for Windows OS as well as other general low-cost operating systems, such as Linux

(3) To inclusion common measurement instrument settings and commands

3 Key Points of Design

3.1 Basic Architecture

Figure 2 shows measurement instruments software platform which developed at this time. In order to give no impact on applications by differences in hardware and OS of each product, the platform consist of 4 layers such as application layer, framework/middleware, OS/device driver layer, and hardware layer.

First, differences in the OS (Windows or Linux) are absorbed by the framework/middleware layer and the interface for the application layer does not change.

Second, since support for the various measurement boards is absorbed by the OS/device driver layer, among the hardware and OS are perfectly separated, and the impact on applications are suppressed.

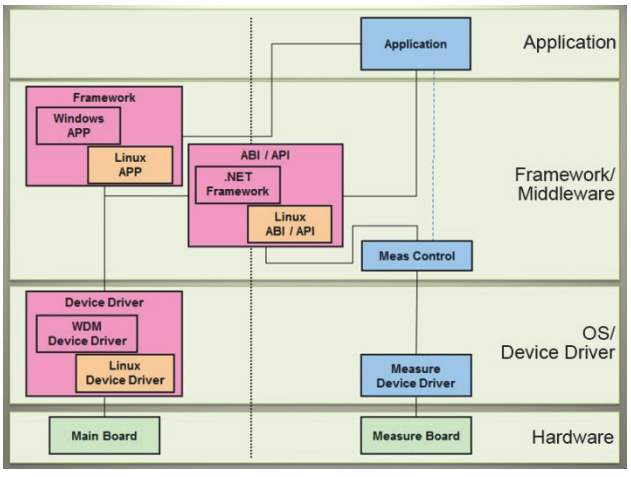

Figure 2 Measurement Software Platform Diagram

3.2 Development Platform for Windows

Generally, there are two embedded OS which is provided as Original Equipment Manufacturers (OEMs), Windows versions: Windows Embedded Compact (WEC), and Windows Embedded Standard (WES). The developed measurement instrument software platform uses WES 2009 due to its high compatibility with PCs compared to WEC and because Anritsu engineers have more experience with WEC.

At platform development for Windows, we focused efforts on abstraction of software from existing products which has same basic architecture. To create a platform with general applicability to a wide range of instruments, it was necessary to precisely examine the basic software architecture and isolate the common functions.

3.2.1 Assuring Optimum Dependency between Modules

Verification of the software configuration using design documents and source code requires a huge amount of time missing and errors of inspection. Consequently, we adopted the Dependency Structure Matrix method (DSM) outlined in Figure. 3 to visualize, quantify, and understand the dependencies in analysis of the developed software architecture. The dependencies (reference and impact degrees) of structural elements, such as subsystems, modules, files, functions, etc., were verified visually using this DSM method. The Lattix (Lattix Corporation, USA) architecture analysis tool was used as an inspection tool offering both fast and problem free inspection.

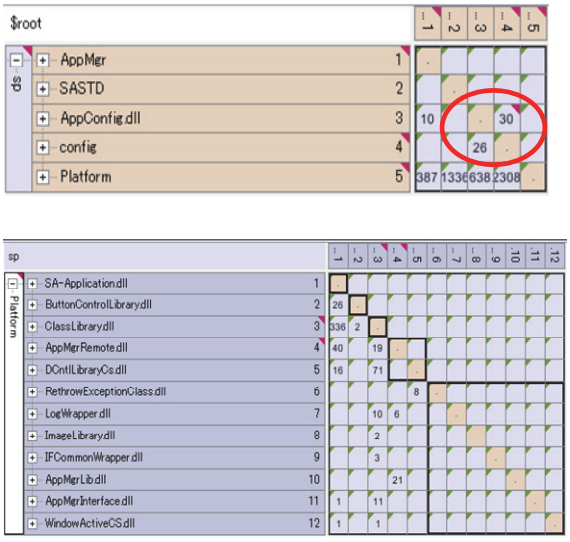

Figure 3 Dependency Structure Matrix (DSM)

3.2.2 Incorporating Common Measurement Functions

To preserve measurement instrument software platform and build-in common measurement functions, the required general measurement functions such as date and time setting, network setting, file operations, power-on run time, license and option management, external remote control via GPIB and VXI-11 protocols, etc., were defined, functions specialized for the each products were excluded from the software to assure independence for the platform modules enclosed by the red lines in Figure 4. The specialized functions were defined as the Product Customize module in Figure 4.

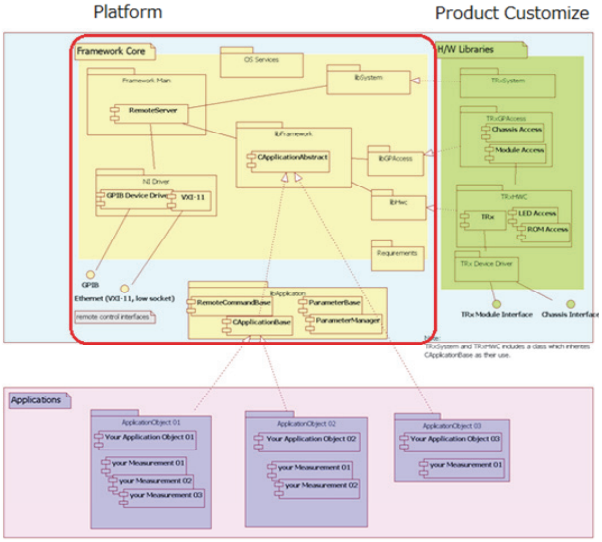

Figure 4 Product Software Diagram

However, these functions had strong dependency on the software modules and simply excluding unwanted functions from the modules resulted in many compile errors, requiring a lot of time to eliminate. Therefore, we use the Lattix tool to remove unwanted functions with dependencies efficiently from modules as a block. By using Lattix tool, dependencies and the impact of module removal could be visually confirmed, and then we could eliminate most of compile errors. After eliminating the most of compile errors, the source code configuration tool Understand (SciTools Corporation, USA) was used to visualize target method and variable referrers, referents, program control flow, structure, class inheritance, factors, variables, etc., to verify and isolate each case one-by-one as shown in Figure 5.

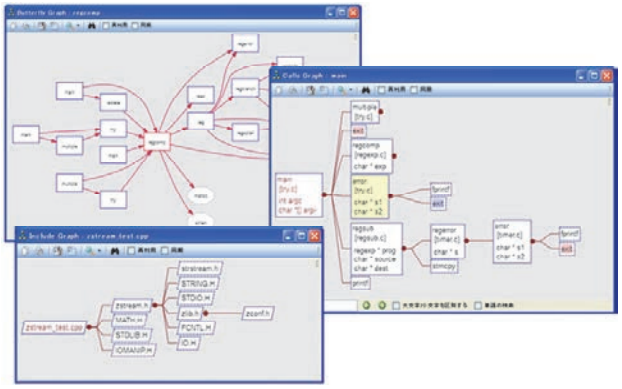

Figure 5 Butterfly Graph, Call Graph, Include Graph

3.3 Development of Linux Platform

Software platforms for measurement instruments often use Linux as well as Embedded Windows for the platform OS to support development and mass production of the latest hardware and middleware at low cost. Although the many available commercial Linux distributions support a variety of applications, for assuring long-term reliability, we adopted a server-type open source distribution for development of this software platform.

There were three key points in using Linux for this measurement instrument software platform. First, risk of infringing export control regulations; second, the need to support the unique GPIB interface of measurement instruments; and third, providing a common architecture for the Windows and Linux versions.

3.3.1 Export Control Countermeasures

Use of open source software is sometimes restricted by export controls. Encryption technologies used in software are restricted and this Linux distribution also risked infringing export controls. Consequently, encoding technologies were removed from the kernel and libraries were developed to create an original non-infringing distribution. This original distribution was configured so that platform users could select whether or not to use the encoding technology.

3.3.2 Incorporating GPIB in Linux Version

Measurement instruments require an external GPIB interface. To support GPIB control, the open source linux-gpib package was used as the device driver and middleware environment for the Linux version of the measurement instrument software platform. This package has been used for many hardware types and has an Application Program Interface (API) for easy porting of existing applications, greatly cutting the time and development costs for porting.

3.3.3 Improving portability between Windows and Linux

To maximize the portability of measurement instrument applications, it is best if the application uses only the platform API. However, when porting Windows applications to the Linux environment, sometimes not only the platform API but also the .NET Framework API must be called, it is necessary to use compatible libraries such as Mono (Xamarin Corporation, USA), or replace source code by using C++ STL (Standard Template Library).

Sometimes, using third-party libraries with which we had no experience in developing the Linux version of the measurement instrument software platform creates a risk of hard-to-solve problems, so we decided to use the C++ STL solution considering future expansion of the platform to support other unique measurement instruments. Using C++ STL required support for the .NET Framework Boxing feature. Since Boxing locations have revisions spreading across a wide range of locations, we designed a new dedicated data class equivalent to Boxing. Moreover, to include all actually usable types at class design, we embedded classes, such as assignment operators with high usage frequency in parts calling the platform API, to achieve high-efficiency substitution.

In addition, some precautions were needed for the memory auto-release feature of .NET Framework. Using STL in the Linux version of the measurement instrument software platform framework required remaining conscious about memory release. To assure memory release, we used the C++test (Parasoft Corporation, USA) software inspection tools to perform both static and flow analyses of memory leak locations. However, these tools caused many mistaken memory leak locations (memory occupied and released by different modules), necessitating manual inspection and review of the source code for each specified item one-by-one.

3.3.4 Post-shipment Updates

Although the software components can be selected for both Embedded Windows and Linux at OS imaging, sometimes the components must be updated either during or after development. To support these updates, we defined the set of packages required by each OS as a standard configuration and included a simple update function.

3.4 Simplifying Remote Command Development using Remote Studio

The measurement instrument software platform also supports the external control commands (remote commands) standardized by IEEE488 and SCPI (Standard Commands for Programmable Instruments) incorporated in all our main measurement instrument. Since there are tens of thousands of remote commands, there have been problems previously with mistakes, omissions and errors in creating the manuals and documentation for the remote command specifications, programming, test cases and instructions.

Analysis the process of the old remote command design, there are many redundantly repetitive operations as well as routine operations. Consequently, we developed the Remote Studio tool to automate creation of specifications, source code, test cases, instruction manuals, etc., based on remote command definition to prevent simple mistakes at the design phase.

Remote Studio provides command specifications and instruction manual format and automates conversion and adjustment work which were done manually. Additionally, Remote Studio generates source code and the test cases which cover a scope of unit test, such as default values, high and low limits, and prefixes, suffixes, etc., permitting addition of original test cases.

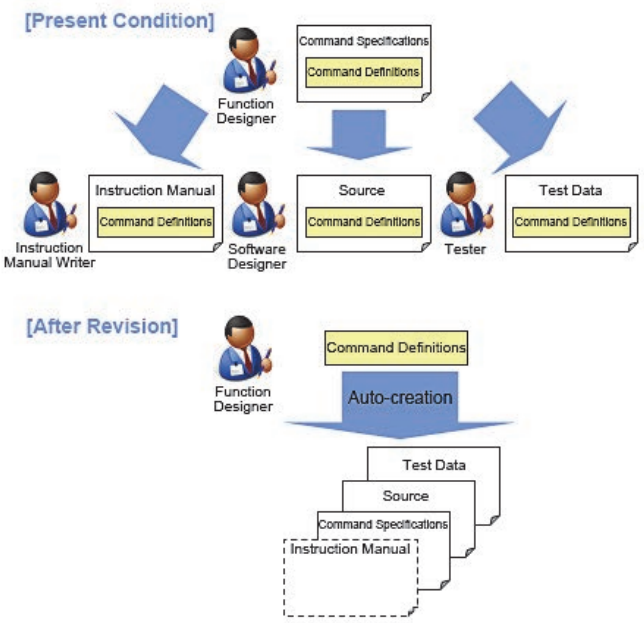

Figure 6 Process of Remote Command Development

Since Remote Studio automates creation of source code and test cases for measurement instrument software platform based on the command explanations standardized by IEEE 488 and SCPI, not only increasing software productivity but also improvement of the quality by preventing misunderstandings of specifications can be expected.

4 Conclusion

We have developed Windows and Linux versions of the measurement instrument software platform. As a result of this development, we have been able to build a software architecture in which applications are independent of the OS and processing system. However, some parts of the program implementation are still dependent on the OS and processing system. Future work involves developing common control of measurement boards, communization of applications and interfaces, concealing of OS system calls, etc. In addition, we aim to extend Remote Studio by fully automating remote control tests, incorporating a function for auto-testing within instruments, incorporating registered command data into a database for use as help guidance, and mechanism for sharing command data with other instruments by create database.

References

1) Android Platform Security Architecture, http://source.android.com/tech/security/#android-platform-security-architecture

Authors

Yuichi Morinari
Anritsu Engineering
General Engineering Promotion Dept.

Yasuyuki Kato
Anritsu Engineering
General Engineering Promotion Dept.

Publicly available