

## Vehicle Information Systems Integration Framework

HAN-CHUN YEH, SU-YING CHANG, YUAN-SHENG CHU,  
CHIEN-WEI CHEN AND CHI-SHENG SHIH  
*Graduate Institute of Networking and Multimedia  
Department of Computer Science and Information Engineering  
National Taiwan University  
Taipei, 106 Taiwan*

Embedded software systems are usually custom designed and are close systems in terms of software design and deployment. One example is the Vehicle Information System (VIS), in which several components gather vehicle information such as RPM and fuel tank level, and are tightly coupled with the embedded hardware. Several projects have been started by commercial software companies or passionate open source developers to meet the needs for entertainment, gathering vehicle information, and traffic information in vehicle. We design and implement a software integration framework for such systems and a vehicle safety box component for VIS. The framework allows the users and developers to integrate independently developed software components into one system and provides a unified GUI. In addition, it eliminates the efforts for re-designing or re-compiling the system for software upgrades.

**Keywords:** component-based software engineering, user interface, framework, software reliability, UML, ITS, GPS

### 1. INTRODUCTION

Thanks to the low electronic hardware cost and open standards for electronic hardware specification, computers can now be installed almost everywhere. Computers are not stacked on the rack or on the office desk anymore. One example is Vehicle Information System (VIS). When we spend more and more time in vehicle, computers in vehicle have become an emerging era. Lots of attempts have tried to install computers on vehicles (including car, bus, and train) to reduce energy consumption, enhance horse-power, and provide entertainments and navigation services for drivers and passengers.

In last few decades, enormous efforts have been made to develop the software to enhance our work performance, provide the entertainment, and organize our daily activities. However, almost all (if not all) the efforts for software development focus on the keyboard or mouse-based computing environment. No matter we are driving a car or sit in vehicle, both keyboard and other pointing devices are not the good choices as input devices. As a result, although there are not much difference in terms of hardware design between automotive computers and portable computers, it requires enormous efforts to redesign existing desktop applications for VIS. In this project, we are concerned with how to easy the burden for porting open source desktop applications to automotive computers. The objectives of this project are three folds: (1) passenger friendly user interfaces, (2) software integration framework, and (3) vehicle safety box. We design a touch-screen based unified GUI which allows the driver and passengers to easily interact with

---

Received September 6, 2005; accepted March 10, 2007.  
Communicated by Jane W. S. Liu.

VIS. The unified GUI uses eye-catching buttons and can be easily operated with fingers. Based on the interfaces, we propose an integration framework, which makes use of the unified template to integrate JAVA-based stand-alone applications and multimedia codec library programmed in C into parts of a VIS. Several desktop applications such as Car Companion [1], RSS View [2] and movie and music library [3] written in C are ported into our VIS as an example. Furthermore, we develop a safety component for vehicles called vehicle-box (V-Box). The V-Box is designed to keep track of the vehicle trajectory all the time, which is analogous to the black box on avionic navigation systems. We can reconstruct the vehicle state by scene reconstruction systems which are only used by the authorized organization such as the police and government. Our scene reconstruction systems could analyze the information recorded by GPS and rebuild the scenario when the vehicle is in the abnormal situation such as car accidents, urgent brakes, and high speeds.

The remainder of this paper is organized as the following. Section 2 presents the efforts been made to develop Vehicle Information System and related works on software engineering. Section 3 presents the new features developed in the system and section 4 presents system architecture and design. Section 5 presents the design of V-Box and section 6 discusses the process of quality assurance. In section 7, we discuss the experience with integrating other open source software component and the lesson learned in this project. Finally, section 8 concludes the paper and discusses the future development for this project.

## 2. RELATED WORK

Commercial automotive PCs products are available on market and can be classified into hardware-oriented and software-oriented products. The major differences between automotive PCs and general PCs are the size of displays and input devices. Most of automotive PCs are equipped a touch screen as I/O devices. In addition, due to the less computation power required by applications of automotive PCs, the automotive PCs usually require less computation than general PCs do. Many manufacturers of automotive PCs such as Xenarc [4], Auto Vision Inc. [5], and Lilliput [6] focus on reducing the size of the systems. In this project, we choose the automotive PC designed by LOGISYS Corporation [7] as our implementation platform. (Later, we will show that our VIS is hardware independent and can be installed on other platforms.) As for software products, there also are many companies or open source projects devoted to developing automotive PC software. One of the popular projects is InDashPC [8]. InDashPC is dedicated to provide the full-line of services for those who want to explore state-of-the-art computing technology in the car. It not only provides an integrated application, called PyCar, to play back multimedia files but also integrates peripheral devices to connect the sensor and control in the vehicle to the VIS. For example, the Car2PC-TOY is a device that connects the automotive PC to car stereos. Another example is G-NET [9]. G-NET delivers a total solution for automotive PCs including both software and hardware. Its DIGITAL DASH is a commercial application designed to run on Windows XP/2000 and gives users one click access to applications on the automotive PC. It features large buttons that link to already integrated tools such as multimedia player, Sirius Satellite Radio [10], and Google Maps Navigation [11], *etc.* Besides, Vehicle Information System from HITACHI [12] also offers solutions for safety, convenience, and entertainment services in vehicles.

Compared with our VIS, great majority of automotive PC software are supported by Windows based OS. The commercial products are able to obtain various and detailed vehicle information such as engine RPM (Revolutions Per Minute), and acceleration by the support of vehicle manufactures. Hence, most of them charge the drivers an extraordinary price. If there are third-party open source applications able to get particular hardware information of vehicles, they can be easily integrated into our VIS. Furthermore, the current commercial automotive PC software is not allowed to be modified, while we propose an open integration framework to let users customize these favorite applications. Open source components of our VIS can be added by users to enrich functionalities of automotive PC and can also be removed when users do not need them anymore.

Our VIS is related to Intelligent Transportation Systems (ITS) [13]. ITS is a program focusing on intelligent vehicles, intelligent infrastructure and creation of an intelligent transportation system through integration with and between these two components, and it is sponsored by the US Department of Transportation. There are 16 categories of applications, and these applications are divided into Intelligent Infrastructure and Intelligent Vehicles. As for the Tracking System in our VIS which tries to keep vehicle moving data being recorded, there is IEEE 1616 Standard for Motor Vehicle Event Data Recorders (MVEDRs) [14] which targets the same purpose. In IEEE 1616 Standard, 86 kinds of data elements must be recorded, and those elements include date, time, vehicle velocity, *etc.*

### 3. NOVEL FEATURES AND TECHNICAL INNOVATIONS

The system provides three novel features, compared to other existing vehicle information systems: unified touch-based GUI framework, vehicle-box and integration framework. In the following, we present the new features.

#### 3.1 Unified Touch-Based GUI Framework

Existing open source software is usually developed for desktop or laptop computers. Text messages and graphic images are popularly used to interact with the users. However, they are too small to interact with drivers and passengers in the vehicle environment. Least but not the last, it could be difficult for the driver or passengers to use keyboard and mouse as input devices when the vehicle bumps up and down.

We design a unified user interface framework to enhance the usability of the system. First, we design the user interface with large icons and buttons for the drivers and passengers to use the system with dim light and bumping driving. Second, we choose touched-based are rather than voice commands as the input device for the following reasons. Although the modern speech recognition technology makes it possible to use voice commands as an input manner for applications, the background noises including outside and inside the vehicle make it challenge to provide reliable speech recognition results, especially when the radio is on or passengers are chatting. In order to reduce the input error, we adopt touch screens as input devices. Touch-based interface has been adopted in Windows-based Tablet PCs, but not popular in Linux-based environment. The unified GUI uses eye-catching buttons and is operated with fingers. It is a straightforward manner and does not require any training. Therefore, the users could control the system with

finger tips for most of the times. When the users are asked to input texts, the system automatically displays the on-screen keyboard to emulate a physical one. However, on TablePC, the users are expected to use pen and applications are designed with small icons and buttons.

### 3.2 Vehicle-Box (V-Box)

The V-Box is designed to keep track of the vehicle and driving information, which is analogous to the flight data recorder or cockpit voice recorder used in aircraft. The purposes of V-Box are twofold. First of all, we can re-construct vehicle states. For instance, when a car accident occurs, the car's location, speed and orientation are recorded in the automotive PC. Afterward, the authorized organization can rebuild accident scene using the recorded vehicle trajectory information. It can help the police to determine the responsibility of car accidents. Second, the record can also be used to bargain for the lower car insurance rate for good drivers.

Because the VIS is equipped with a GPS receiver, the system obtains vehicle's location, velocity and orientation periodically. If the VIS detects a dramatic change on the velocity and determines that this is an urgent braking event, it will store the vehicle information into secondary storage devices for further use. Similarly, when the VIS determines this is an event of high speed or drifting, it will also mark this irregular information. When the VIS does not have GPS receiver, the V-Box will be disabled.

### 3.3 Integration Framework

To integrate independently developed software components, we adopt Component-Based Software Engineering (CBSE) [15] approach. Component-based software engineering is a new subdiscipline of software engineering and its major goals [16] are the following:

- To provide support for the development of systems as assemblies of components;
- To support the development of components as reusable entities;
- To facilitate the maintenance and upgrading of systems by customizing and replacing their components.

In CBSE, software systems are built by assembling components already developed and prepared for integration. CBSE has many advantages including more effective management of complexity, reduced time to market, increased productivity, improved quality, a greater degree of consistency, and a wider range of usability [17].

Traditional embedded software development has the advantage of supporting the customer's system design requirements. If the system design requirements including hardware specification and functional requirements are unique, it leads to highly integrated system and better system performance. The disadvantages of custom design are the high system development cost and the long time to market.

The VIS integration framework can be viewed as a circuit board on which other (open source) software components can be plugged into the empty slots. In the integration framework, the slot interfaces are defined by templates. Each template defines the command interfaces, features, and requirements of the software requirements. As long as

the software components conform to the interface definition, the integration framework can integrate the new components to the system.

Existing open source software components are integrated into our VIS through the converters. The converter wraps the functionalities and input/output messages of each open source component into our predefined application interface, and integrates the individual graphical window of each open source component into our well-designed unified GUI to avoid displaying multiple graphical window on screen. For instance, the application interface for the multimedia category must consist of play, stop, pause, forward, and backward functionality no matter what kinds of codec libraries that the users/developers choose. Another example is that the graphical window of the RSS View [2] is integrated into our VIS and does not pop up another individual graphics windows or message boxes on screen separately. Hence, we do not implement all the components such as RSS View and media player, but integrate these components into one system to demonstrate the capability of our integration framework.

VIS designed with our integration framework can evolve when the operating environment changes, user requirements are modified, and errors (which inevitably occur in such systems) come to light. Comparing to custom designed embedded software, there are several major features with our VIS integration framework:

- Object-oriented languages have a better structure for facilitating CBSE than traditional languages. These OO languages have a foothold in the mainstream software development community, with increasing numbers of libraries of classes available.
- We select Java as our implementation language. Its OO structure makes it ideal for creating component libraries, and building Java applet and applications from them. In addition, Java contains sufficient information for interface and inheritance details to be determined from a Java component even when the component's source is not available. JavaBeans further increases the ease with which Java components can be interchanged and assembled.

## 4. SYSTEM ARCHITECTURE

Software architecture is the key design of our VIS integration framework. The system architecture consists of three parts: VIS components, templates and converters, and VIS platform as illustrated in Fig. 1. VIS components shown on the right-hand side are the integrated or custom designed software components in the VIS. Templates and converters shown in the middle connect the VIS components to the system. The VIS platform shown on the left-hand side integrates all the components. We discuss the design for each part in the following.

### 4.1 VIS Components

The VIS consists of various VIS components such as RSS View, Car Companion, Chart Plotter, and xine multimedia codec library. The components can be commercial software components or open source software components. The developers of the components have to follow certain pre-defined component interface to design the software components, or define the component interfaces for their software components.

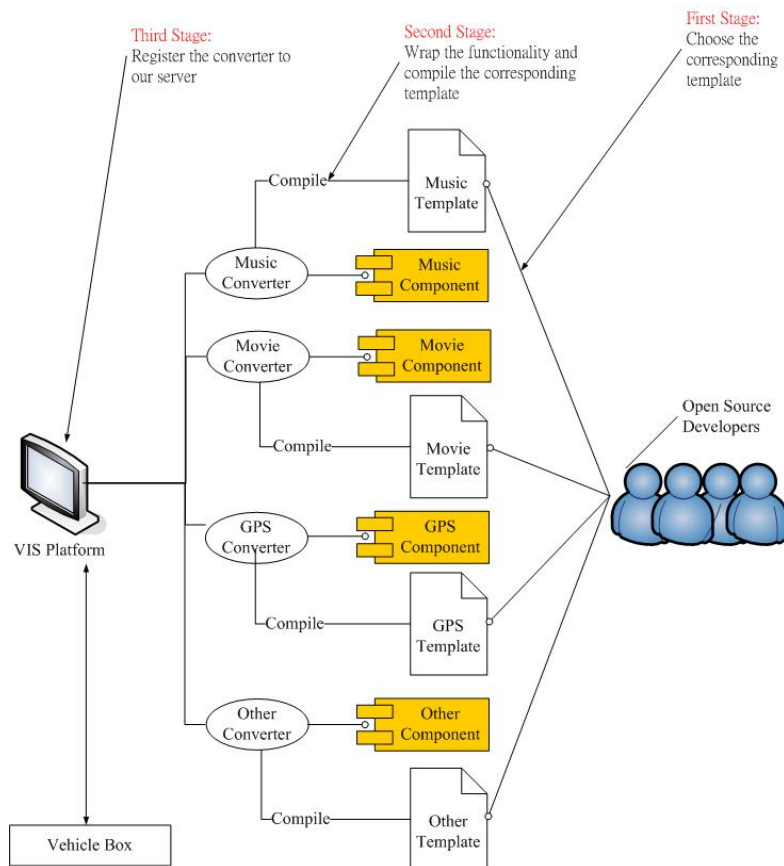


Fig. 1. The automatic tab generation scheme.

To ease the burden of integrating software components, we design a three-stage integration scheme called *Automatic Tab Generation scheme*, as shown in Fig. 1. In the first stage, the developer selects one pre-defined component interface template or defines a new component interface. In the second stage, the template is compiled as a converter. In the third stage, the converter is responsible for registering the components to the VIS, and sending/receiving commands to/from the components. After a component has been registered to a VIS, a tab on the unified GUI will be generated and can be used to trigger the component.

#### 4.2 Templates and Converters

The templates and converters (compiled templates and executable files) are two major components in the integration framework. One template defines the component interfaces for one type of software components. The interfaces define the services provided by one type of software component and the resource requirements of the software components. With the interface, the VIS can access the services provided by the components by sending commands through the interfaces. Unified Modeling Language [19] is

selected to define the interface. We designed several application-specific templates as examples. Fig. 2 shows the example templates for music, movies, and internet components. Note that the developers can define new templates for different components. It is also important to note that a template offers no implementation of any of its operations. Instead, it merely names a collection of operations and provides the descriptions and the protocols of these operations. The Application Interface illustrated in Fig. 2 not only provides a common way for the VIS to command the integrated components but also forms a standard interface for developers to implement their corresponding operations. The template tells the open source developers everything that they need to know in order to integrate their open source components into the VIS. Ideally, open source components should be black boxes, to enable our VIS to use them without knowing the details of their implementation. After open source developers wrap their functionalities and input/output messages of each open source component in the application-specific template, we call these implemented/compiled templates converters which have capability of communicating with our VIS through socket. This separation makes it possible to (1) replace the implementation part without changing the template, and improve the system performance without rebuilding the system; and (2) add new functions (and implementations) without changing the existing implementation, and improve the template adaptability.

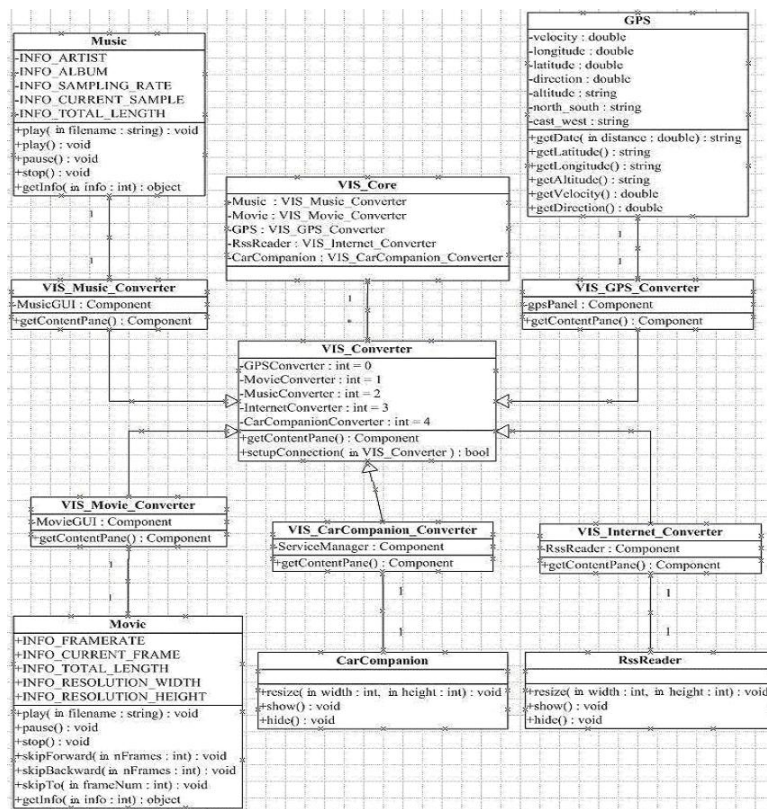


Fig. 2. The UML-class diagram of application interface.

### 4.3 VIS Platform

VIS platform hosts the software components and interacts with the users. The users insert and remove software components via the VIS platform at any time. After a component is integrated into the VIS, a new tab will be generated on the unified GUI, and there is no need to recompile the system. Hence, the passengers or drivers could add and delete any their favorite open source components through our unified GUI as shown in Fig. 5.

## 5. VEHICLE BOX DESIGN

Vehicle box (called V-box for short) is one of the major components. It keeps track of the vehicle status and allows the authorized users to reconstruct the vehicle trajectory and status later. It consists of two major components: tracking system and scene reconstruction system. The tracking system is responsible for recording the vehicle status; the scene reconstruction system reconstructs the vehicle trajectory according to the recorded vehicle status and is executed on desktop computers.

### 5.1 Tracking System

The logged data for each sample consists of date, event type, speed, longitude, latitude, and direction. An example of logged data is “2006/05/15 23:12:43#3#48.9#E 121.00127#N 23.00033#90”. All these information are obtained from GPS. The sampling rate for the tracking system depends on the sampling rate of the GPS sensor on the vehicle. Its typical sampling rates range from several times per second to multiple seconds. Lifetime is a time period how long the recorded data are stored on the secondary storage and is computed as the following.

$$\text{Lifetime} = \text{DiskQuota} / \text{ConsumedRate} \quad (1)$$

where DiskQuota is the size of the disk space dedicated for storing the vehicle moving records and ConsumedRate is the rate that the disk space is consumed by storing the record. By changing ConsumedRate, we can determine the LifeTime for each recorded data.

For example, given 1 GB of DiskQuota, equivalent to 1,073,741,824 bytes and ConsumedRate is 60 bytes of disk space per second, Lifetime of every record is 17,895,697 seconds. In other words, the record created earlier than 7 months ago would be erased when more than 1 GB of disk space are consumed.

In V-Box, one important issue is to keep the recorded data when accident occurs or the system fails. In Tracking System, we define three vital events to flush the data in memory to secondary storage: urgent break events, speeding events, and drifting events. Drifting stands for a sudden and severe change of the vehicle direction. Since most of accidents happened when urgent break, speeding, or drifting occurs. The data are flushed into the secondary storage to protect the data. In addition, the system marks the occurrence of three events for later analysis. Note that how to define these events reasonably by itself is a research issue and subjective. In our design, functions of detecting the events are listed below.

$$UB(v_t, v_{t-1}) = \begin{cases} 1, & \text{if } v_t < \text{UBLS, and } |(v_t, v_{t-1})| > \text{UBvariation} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

$$HS(v_t, v_{t-1}) = \begin{cases} 1, & \text{if } v_t \text{ HSLB} \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

$$Drift(v_t, a_t, a_{t-1}) = \begin{cases} 1, & |(a_t - a_{t-1})| > \text{Driftvariation} \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where  $v_t$  is the vehicle velocity at time  $t$ ,  $v_{t-1}$  is the vehicle velocity at time  $t - 1$ ,  $a_t$  is the vehicle direction at time  $t$ , and  $a_{t-1}$  is the vehicle direction at time  $t - 1$ . UBLB is the velocity lower bound of an urgent event, and HSLB is the velocity lower bound of a high speed event. UBvariation is the minimum difference of velocity of an urgent break event, and Driftvariation is the minimum difference of directions of a drifting event.

Our goal is to propose a basic framework of V-Box. As for the different recording policies, number of events, and definition of events, they go beyond the scope of focus of this project. These policies can be improved according to various requirements. Furthermore, to achieve the flexibility, these parameters of policies can be modified through a readable configure file.

## 5.2 Scene Reconstruction System

The second component of the V-Box is the scene reconstruction system. The system is an independent software and is executed on desktop computers but automotive computers. It imports the recorded data from one or several tracking systems and reconstructs the vehicle trajectory of each vehicle. Because the recorded data are obtained from GPS, all the data are synchronized to the same clock system. Fig. 3 shows the screenshot of the scene reconstruction system. The time-line is shown on top of the screen. The user can scroll the time-line to view the reconstructed scenes. On the top-left, two buttons are shown to import the recorded vehicle data. The users can import at most the data for two vehicles to re-construct the scene. The recorded data are stored in a raw data format and are not encrypted. However, any encryption algorithms can be implemented to protect the privacy without great efforts. In the central part of screen, the users can view the relative location of the vehicle. Two buttons on the right allow the user to play back and stop to play the vehicle status along the time-line.

## 6. QUALITY ASSURANCE

The goal of this project is to design a framework to allow effort-less software integration. Hence, how to assure the reliability of the VIS platform and the integrated software components are one of the important design issues. We design and implement a black-box quality assurance mechanism using Monkey Program testing process. The mechanism tests not only the quality of the VIS platform but also the integrated software components. Monkey Program is a software testing process where the objective is to

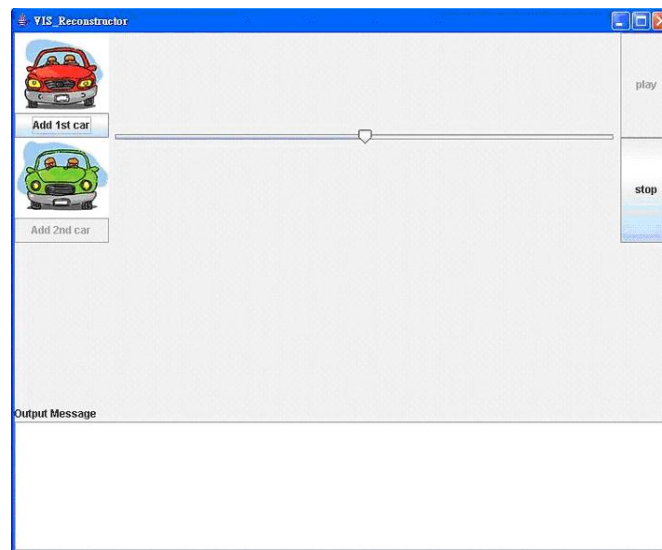


Fig. 3. The screenshot for our reconstructor of the vehicle-box.

measure the reliability of the integrated open source components. When the failures are discovered, the underlying faults causing these failures are repaired so that the reliability of our VIS should improve in the testing process. The steps of Monkey Program involved in the testing process are the following.

- Determine the operational profile of the software. The operational profile is the usage pattern of the software.
- Select or generate a set of test motion corresponding to the operational profile, apply these test cases to the VIS, and record the amount of execution time between the observed system responses.
- After a number of system responses are observed, the software reliability can be computed. We compute the required reliability metric based on the number of failures detected and the time between these failures.

The operational profile of the software reflects how it will be used in practice. However, when a software system is new and innovative, it is more difficult to anticipate how it will be used. The VIS may be used by a range of users with different expectations, backgrounds and experience. There are no usage historical usage databases, so it is often impossible to develop a trustworthy operational profile. Hence, the operational profile of our interactive VIS now is random sequence of clicked motion. This is why we named our testing process Monkey Program because it is like that a naughty monkey roughly plays your any integrated open source components to discover failures. We hope that increasing the number of test cases may decrease the uncertainly of our VIS.

When invoked, the Monkey Program will execute the VIS core program. Then the testing developer can load their applications and begin testing. The Monkey Program may generate random clicked motions to test the robustness and reliability of the application. When exited, it prints out the failure rate observed during the testing process. The

application developers can aim for a desired low failure rate and try to improve the failures reported by the Monkey Program during testing. Developers can view the Monkey Program as the “debug mode” of the VIS framework; it can be invoked as a separated program for the purpose of testing. After testing, the usual VIS core program can be used without the overhead incurred by the Monkey Program.

**Table 1. Testing results by monkey program.**

Test components	# of failure/hour
Music (xine library)	196.432251
Movie (xine library)	0.0
Car companion	0.0
RSS view	0.0

During the design and testing for the project, we use the Monkey Program to find out several bugs and correct them. Table 1 shows one of the testing results. In this example, the Monkey Program tests four integrated components: Music, Movie, Car Companion, and RSS View. The results show that the integrated Music component has 196 failures per hour. After we checked the Music component and integrated framework, we found that the failures are caused by unified GUI not by the Music component. The revised unified GUI passed the testing later.

## 7. PLATFORM AND DEVELOPMENT ENVIRONMENT

The integration framework is implemented by Java. Hence, it can be installed on any platform which supports Java Virtual Machine (JVM). However, if the integrated software components depend on other libraries which are not supported on the platforms, certain functions may not work properly on those platforms.

We choose a Linux-based platform, which helps us to integrate a lot of applications efficiently and seamlessly, to implement the systems. Although selected applications could be implemented by different programming languages such as C, C++, Java, *etc*, we select Java as our primary programming language for two reasons. First of all, a great majority of open-source software is written in Java and it will be easier for us to integrate this kind of software into our system. Second is the higher portability for Java is better. In addition, to efficiently implement our system, we adopt ANT, a Java-based build tool, and OpenSVN, a version control tool to make our teamwork more smooth and time-saving

The hardware development environment is described as below,

- **Car PC:** We bought a mini car pc (Logisys M134) for implementation. The specification for Logisys M134 is listed in Table 2.
- **7-inch touched screen:** It can sense the touch or drag operations, and can accept the power supply from cigar-lighter.
- **USB2.0 GPS receiver (Holux GR-213).**
- **USB2.0 54Mbps wireless card (ASUS WL-167G).**

**Table 2. Specification for M134.**

CPU	VIA C3 E-Series 800MHz
Motherboard	VIA CLE266 + VT8235 Chipset Motherboard
SDRAM	256MB DDR400
HardDisk	2.5 inch 20GB 5400RPM

The open source software components we used or referenced in this product are listed as follows,

- **Car Companion 1.3.2 [1] (Integrated):** A tool to track the maintenance that needs to be performed on the user's car. It is written in Java and released under GPL.
- **Communication with computer via RS232 port [20]:** A simple Java-based program that can redirect data stream from RS232 port to the GPS-related application.
- **ChartPlotter [21] (Integrated):** This chart plotter program can show a map with a mark at your position.
- **NMEAParserDemo Project [22]:** This is a .Net-based NMEA 0183 parser program.
- **GPSylon [23]:** This project is a GPS total solution, which consist of NMEA 0183 parser and navigation, *etc.*
- **mpg123 [24]:** A fast real-time MPEG Audio Player for Linux and Unix systems.
- **ogg vorbis [25]:** A completely open, patent-free, professional audio encoding and streaming technology.
- **xine library [3] (Integrated):** An open source high-performance, portable and reusable multimedia playback engine.
- **mplayer [26]:** A movie player which runs on many systems and plays a wide range of video file formats.
- **RssView 1.2.0 [2] (Integrated):** RSS Viewer is a comfortable and portable viewer for RSS/RDF-compatible newsfeeds. It is written completely in Java and licensed under the GNU General Public License.

Figs. 4 to 5 show the screenshot for the integrated software components.

## 8. SUMMARY

In this paper we presented a software integration framework for vehicle information systems. The framework is highly modularized, consisting merely a minimal core program with an ATG register server on its own, and application components can be independently "hot-plugged" into it by the means of the ATG scheme described in section 4. To facilitate the integration process, we distributed several common "templates" which define both the component interfaces and behaviors of applications for vehicle information systems. We demonstrated our concept worked by implementing "converters" for chosen real-life open source application components so as to integrate them into our framework. The most important component in the implemented system is the V-Box, which acts like a black box on airplanes, and keeps tracking driving information in order to re-construct the scene afterwards. In our implementation we also provided a unified GUI specially designed for touch-based screen usage, for the sake of convenience of

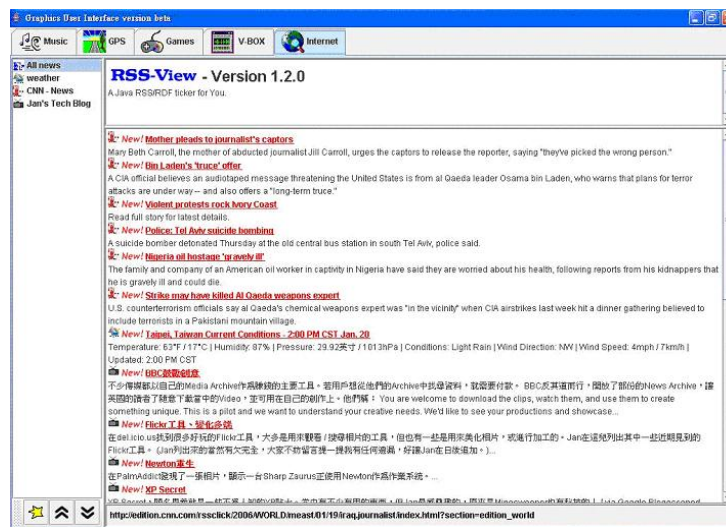


Fig. 4. The screenshot for integrated RSS view.



Fig. 5. The screenshot for integrated chart plotter.

interaction in the vehicle environment. Moreover, the reliability of the framework has to be considered. If a component plugged into our VIS is buggy, the whole system should not be brought down altogether. At the same time, when testing the system, the system should have the ability to monitor and report which component has crashed.

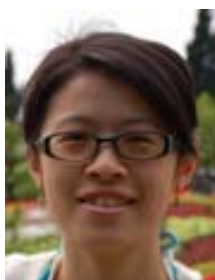
Finally, considering the privacy and security issues arose by the V-Box, we suggested applying encryption to the driving record database and only the authorities are allowed to access the encrypted database. Additional efforts may be spent to ensure that the privacy of the driver is not invaded and the security of database is not broken.

## REFERENCES

1. Car Companion, <http://sourceforge.net/projects/car-companion>.
2. RSS View, <http://rssview.sourceforge.net/>.
3. Xine Library, <http://xinehq.de/index.php>.
4. Xenarc Technologies, <http://www.xenarc.com/>.
5. Auto Vision Inc., <http://www.autovision.com.tw/>.
6. Lilliput Electronics Co., Ltd., <http://lilliputz.en.ecplaza.net/>.
7. Logisys Corporation, <http://www.logisysus.com/>.
8. InDashPC, <http://indashpc.org/>.
9. G-Net Inc., <http://www.gnetcanada.com/>.
10. Sirius Satellite Radio, <http://www.sirius.com/>.
11. Google Maps, <http://maps.google.com/>.
12. HITACHI – Vehicle Information Systems, <http://www.hitachi.co.jp/Div/apd/en/products/vis/index.html>.
13. Intelligent Transportation System, <http://www.its.dot.gov/index.htm>.
14. IEEE 1616 Standard, <http://grouper.ieee.org/groups/1616/home.htm>.
15. I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*, Artech House, 2002.
16. G. T. Heineman and W. T. Councill, *Component-Based Software Systems, Putting the Pieces Together*, Reading, Addison-Wesley, MA, 2001.
17. A. W. Brown, *Large-Scale Component-Based Development*, Prentice Hall, NJ, 2000.
18. J. Cheesman and J. Daniels, *UML Components – A Simple Process for Specifying Component-Based Software*, Addison-Wesley, MA, 2000.
19. NMEA0183Parser, <http://www.teilo.net/software/NMEA0183Parser/>.
20. ChartPlotter, <http://www.eit.se/chartplotter/>.
21. NMEAParserDemo Project, <http://www.visualgps.net/SiteMap.htm>.
22. GPSylon, <http://gpsmap.sourceforge.net/>.
23. mpg123, <http://sourceforge.net/projects/mpg123>.
24. ogg vorbis, <http://www.vorbis.com/>.
25. mplayer, <http://www.mplayerhq.hu>.



**Han-Chun Yeh (葉瀚駿)** received the Bachelor degree in Computer Science from National Chi Nan University, Taiwan, in 2004. He is currently pursuing the Master degree in Computer Science and Information Engineering at National Taiwan University. His research interests include medication authoring tools, web services, and workflow.



**Su-Ying Chang (張素熒)** received her Bachelor degree in Computer Science from National Tsing Hua University, Taiwan in 2004. She is a graduate in the Department of Computer Science and Information Engineering at National Taiwan University. Her current research interest is in the field of intelligent sensor networks.



**Yuan-Sheng Chu (朱原陞)** received the Bachelor degree in computer science from National Tsing Hua University, Taiwan. He is a graduate student in the Department of Computer Science and Information Engineering at National Taiwan University. Her current research interest is the issues on flash memory storage systems.



**Chien-Wei Chen (陳健偉)** received his B.S. degree in Computer Science from National Chengchi University in 2005. He is a M.S. student in the Department of Computer Science and Information Engineering, National Taiwan University, since 2005. His research interests include energy-efficient real-time systems and scheduling algorithms.



**Chi-Sheng Shih (施吉昇)** has been an assistant professor at the Department of Computer Science and Information Engineering at National Taiwan University since February 2004. In 2003, he received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. His main research interests are real-time systems and database systems. Specifically, his main research interests focus on object-relational database query optimization, real-time operating systems, real-time scheduling theory, embedded software, and software/hardware co-design for system-on-a-chip. He received the Best Student Paper Award from the IEEE Real-Time System Symposium in 2004 and Best Paper Award from the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications in 2005.