International Journal of Smart Home
Vol.8, No.2 (2014), pp.23-32
http://dx.doi.org/10.14257/ijsh.2014.8.2.04

Design and Implementation of the Smart Virtual Machine on iOS Platform for the Mobile Game Portability

Yunsik Son[1], JaeHyun Kim[2] and YangSun Lee[3*]

[1]Dept. of Computer Engineering, Dongguk University
26 3-Ga Phil-Dong, Jung-Gu, Seoul 100-715, Korea
[2]Dept. of of Computer Engineering, Seokyeong University
16-1 Jungneung-Dong, Sungbuk-Ku, Seoul 136-704, Korea
[1]sonbug@dongguk.edu, [2]statsr@skuniv.ac.kr, [3]yslee@skuniv.ac.kr

Abstract

Development of computing environments and mobile devices lead to rapid growth of smart device, mobile OS and application market. Especially, the mobile platform sector with major runners of Apple and Google, compose the core technology for mobile applications. iOS and Android represent mobile platforms each supporting individual execution environment, development tools and development methodology. Applications and contents developed in each platform inevitably have exclusive properties in the other platforms. Therefore, generally it is not possible to execute a mobile application on a different platform. In order to provide service to multiple platforms by portability, additional costs and development period is required. Particularly, the lifespan and development duration of mobile applications are becoming shorter at this point in time. Therefore, fast development and deployment is required and has been standing out as a weakness. Smart Cross Platform is a virtual machine based solution developed to resolve this weakness. In this paper, Smart Cross Platform's content execution component, Smart Virtual Machine based on an independent neutral language was designed and implemented to be run in iOS. In the Smart Virtual Machine, the programmer can make a program not being restricted to the development language of a particular platform environment, making it is easier to port previous contents or provide service of a single program to multiple platforms.

Keywords: *Smart Device, Virtual Machine, Intermediated Code, Contents Reusability*

1. Introduction

Current smart devices, Google's Android platform based on Java, Apple's iOS using Objective-C have been released and have constructed a dominant environment [1]. Also recently, Microsoft has announced a platform Windows Phone using C# as the main language [2]. These previous smart device platforms have independent application development environment - develop language, tools - and individual characteristics per platform. Each application must go through a defined development process in order to be executed in a certain platform. The object code also has to be created separately depending on the platform. Therefore, in order to provide service to different platforms even if the content is the same, depending on the object device, separate development environments must be used to re-develop which shows a weakness in efficiency.

[*] Corresponding Author

The Smart Cross Platform was developed as a solution to resolve this weakness [3, 4]. It supports various programming languages and is not restricted to a certain platform, aiming to execute contents independent of platforms. To do this, the smart cross platform provides a compiler which creates intermediate language as object code and has a virtual machine as an important component which interprets intermediate language on a stack base[5-7].

The paper is organized as follows. In Chapter 2, Smart cross platform, SVM to load iOS, intermediate language used in SVM and execution file formats are examined. In Chapter 3, the structure of SVM and optimization techniques to enhance the execution performance are examined. In Chapter 4, results of contents run using the SVM implemented shown. Lastly in Chapter 5, the conclusion of the paper and future direction of research are discussed.

2. Researches

2.1. Smart Cross Platform

The Smart Cross Platform was developed by our research team as a platform based on virtual machines for smart devices. It is composed of three parts; compiler, assembler and virtual machine. It supports C/C++ and Java programming languages for contents development [6, 7]. Contents made of each language are converted to an intermediate code by the compiler. SIL is used as intermediate code which can accommodate procedural language and object orientated language. Intermediate code is converted to execution format by the assembler and executed in the SVM. Figure 1 shows smart cross platform's model.

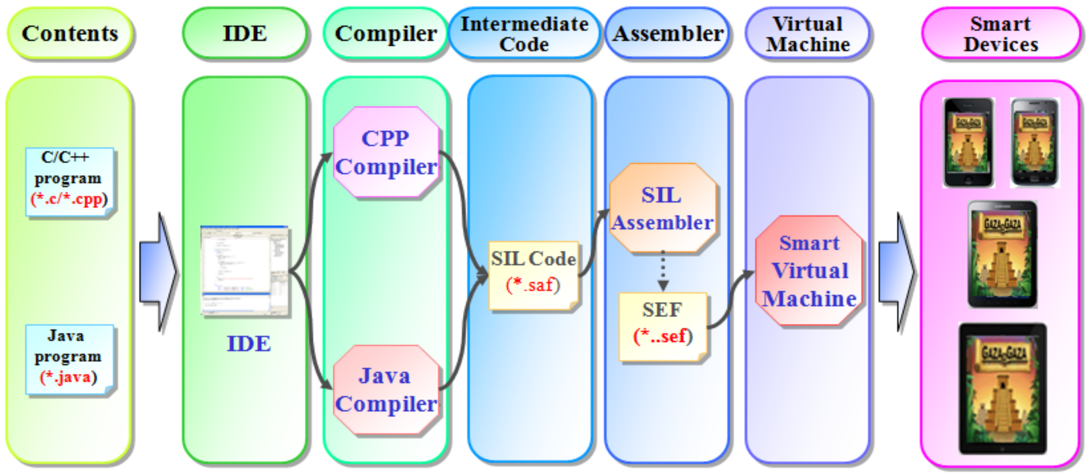


Figure 1. System Configuration of the Smart Cross Platform

Smart cross platform compiles an application program and is composed of three parts. A compiler creating SAF format file composed of SIL(Smart Intermediate Language), an assembler converting SAF(Smart Assembly Format) file to a SEF(Smart Executable Format), and a virtual machine executing SEF format file entered [4, 8]. The SVM system is designed as a hierarchical model minimizing burden created during the process of retargeting other devices and operating environment.

SIL from compiling and translating process is converted to SEF, an execution format, by the assembler and SVM executes the program entered by SEF.

2.2. iOS

iOS[1] is an operating system built into Apple's smart phone iPhone, digital media player iPod touch and tablet computer iPad. iOS acts as a middle role for the hardware and application shown on the screen. The application is not directly connected to the hardware, but connected through the interface. iOS technique implementation is a bundle of classes composed of four layers: Core OS layer, Core service layer, media layer and Cocoa Touch layer.

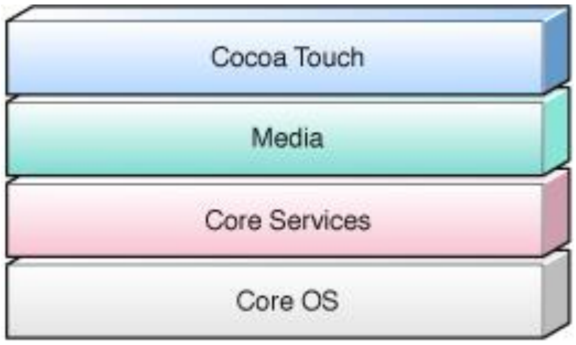


Figure 2. iOS Architecture Layer

The Core OS and Core Services layers act as fundamental interfaces of iOS providing file accessibility, low level data type, Bonjour service, network socket, core foundation, CFNetwork, SQLite, POSIX thread, UNIX sockets and etc. The media layer provides 2D/3D drawing, audio and video related APIs and includes OpenGL ES, Quartz, and Core Audio, Core Animation libraries. The Cocoa Touch layer is a fundamental infrastructure for applications providing file management, network operations, access to the user's contact, photo information, accelerometers and etc. Also provides UIKit framework, windows, views, controls, and the controllers class to programmers.

2.3. SIL & SEF

SIL and SEF are each intermediate language and execution format for SVM. First, SIL is an intermediate language for virtual machine code method. As a standardized model of virtual machine codes for general smart phones and embedded systems, it is designed for language, hardware and platform independency. In order for SIL to accommodate various programming languages, it is defined based on analysis of previous virtual machine codes such as .NET IL [9, 10], byte code [11, 12] and *etc*. To accommodate all object orientated language and procedural language, it has a set of stack based operation code. The operation code is not subordinated to a specific hardware or source language, having an abstract form. In order to make debugging which is assembly language level, simple, a consistent naming rule was applied and defined with mnemonics with high legibility. Figure 3 shows the SIL's operation code categories. It is mainly grouped into: 5 main operation code for executing commands, other operation code for stack management and short form operation code for optimization. SIL has meta code other than operation code which perform specific work such as class and method declaration.

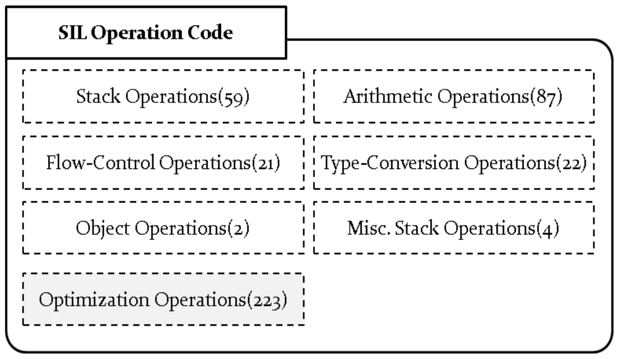

SIL Operation Code
Stack Operations(59)
Arithmetic Operations(87)
Flow-Control Operations(21)
Type-Conversion Operations(22)
Object Operations(2)
Misc. Stack Operations(4)
Optimization Operations(223)

Figure 3. Category of Operation Codes

Programs composed of SIL code are converted to SEF by the assembler. SEF, an execution file format in SVM, is composed of the header domain which expresses the SEF file's composition, program segment and debugging segment which expresses debugging information. The program segment is divided into three segments expressing code and data [13]. Figure 4 below shows a simple diagram of the SEF structure.

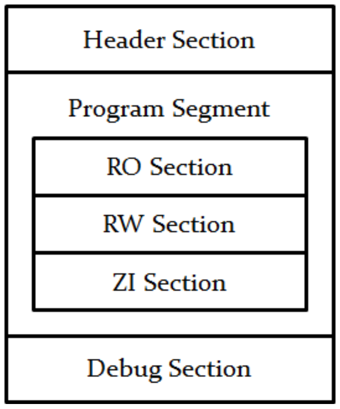


Figure 4. Structure of SEF

The header domain expresses the detailed composition information of the program segment, records information of the program's entry point. It also reads the SEF file's header domain information and estimates the total memory size required and makes it easy to approach the detailed entry point.

The program domain is a format which is loaded and executed in the SVM memory and is composed only of code and data. The SEF program domain separates symbol table and debugging information unnecessary for execution and is designed to minimize the memory and speed required to load SVM memory. Depending on the nature of the elements composing the program domain, it is divided into RO, RW and ZI domain. The RO section stores code and literal data with read-only characteristic. The RW section stores data of global variables which have initial values in the source code with read-write characteristic. The ZI domain stands for global variables which do not have initial values in the source code.

The debugging domain expresses debugging information for application programs saved as SEF. It is not loaded in SVM's execution memory and is used by IDE(Integrated Development Environment) or debugging tools. Depending on the SVM compile option, the debugging domain in SEF exists optionally and does not affect execution.

3. Smart Virtual Machine for iOS Platform

In this chapter, iOS platform with SVM loaded will be introduced. First, SVM system's composition and each module's function are introduced. Next, performance enhance technique applied to interpretation routine is shown.

3.1. Implementation of the iOS SVM

A Smart Virtual Machine uses the execution file created by the smart cross platform's core, compiler and assembler. The file is entered and the contents are executed and the results are given as an output. The Smart Virtual Machine is composed of SEF loader, interpreter, internal library, runtime environment. The runtime environment is composed of memory manager, thread scheduler and event handler. Figure 5 shows the structure of the virtual machine.

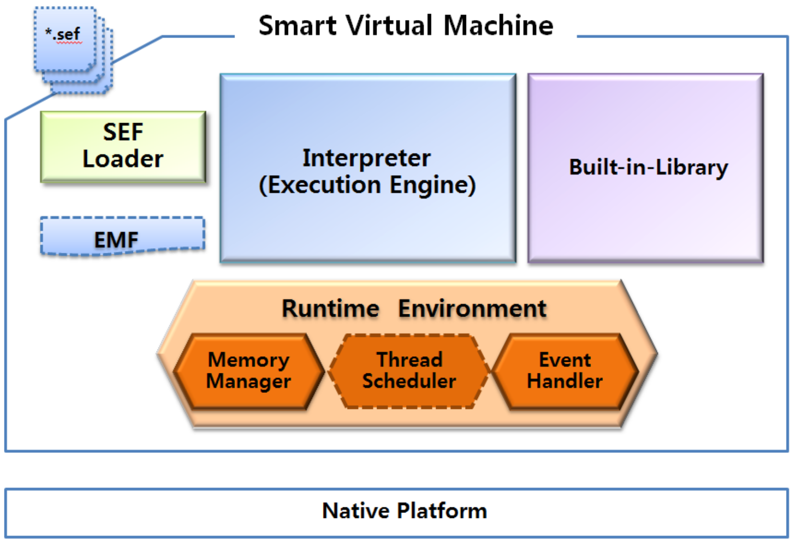


Figure 5. System Architecture of SVM

First, the SEF loader validates integrity of the content and loads to the memory converts to an internal data structure EMF(Executable Memory Format). EMF is managed in a double linked list of meta data and SIL command. Next, the interpreter is the core module of the virtual machine which actually calculates and executes the SIL code. It has been implemented by making actual process operations for each SIL code and refers to meta data saved by the loader and executes commands. Data during processing is stored in stacks and heaps and if an error occurs during processing, an exception raises. The exception prints a corresponding message by the exception handler and exits.

A total of 198 instructions are supported by the smart virtual machine and are divided into stack instructions, calculation instructions, flow control instructions, type conversion instructions and object instructions. In the internal library, an API set of ANSI C library and graphic, sound, file, network, character string, font, math and *etc*. A total of 16 groups, 425 API are provided for language in order to execute contents. Lastly, the runtime environment is composed of memory manager, thread scheduler and event handler. When the interpreter executes the SIL command, it receives support from the program's runtime environment. First, in the memory manager of the runtime environment, depending on each command, a stack calculation process takes place. Constant pool information from global domain, object information from heap domain, variable information from local domain and are divided into activation record, operation stack and display vector and processed. The event handler catches events during the process of content execution and in order to process these, handlers from within the program are called and event-driven content action is supported.

3.2. Optimization Technique for Interpretation Routine

Next, the optimization for SVM interpretation routine is as follows. Looking from the content portability aspect, the SVM has a big advantage since it supports various programming languages and uses virtual machine based execution method. However, it executes software interpretation using the commands in the contents which has a very low performance compared to the general native execution method. Therefore, to enhance performance, optimization technique is very important and in this paper, interpretation routine optimization was applied [14, 15]. Figure 6 shows the model of interpretation routine model applied.

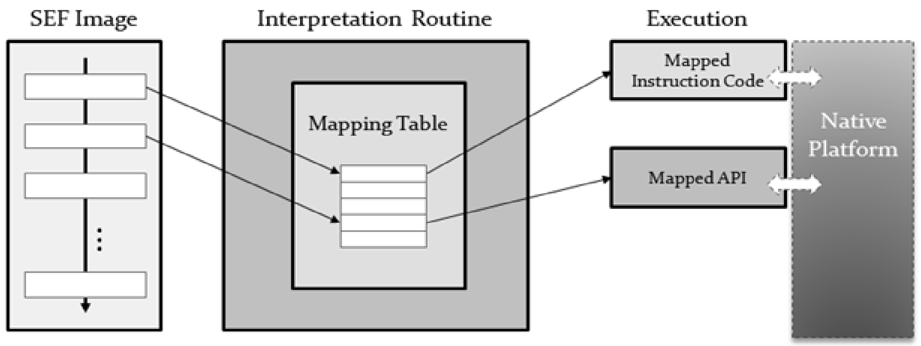


Figure 6. Optimization Model for Interpretation

The stacked based fetch-decode-dispatch method generally used for executing VM has a very slow performance. This is due to the lookup routine when it loads the command and uses a lookup routine to match commands or execute API. The lookup routine is a Hotspot which requires a high calculation time. To resolve this problem, in the optimized model, the corresponding routine execution was modeled and mapping tabulated to reduce cost for executing the command calling API. Each decoded command was used as an entry value of the mapping table from statically calculated results and the mapped result values were earned. As a result of applying the optimized model, the execution time of SVM reduced by 23~37% compared to not applying the optimized model.

4. Experimental Results and Analysis

In this chapter, we used game contents implemented with C++ on a smart cross platform to convert into SEF files and show results of executing them on implemented iOS SVM. The games tested were “Gaza Gaza”, “Zoro” and “Aiolos”. First the conversion process for Gaza content is shown. Table 1 shows part of the source program of “Gaza Gaza” made of C++ language.

Table 1. Source Code Fragment of the “Gaza Gaza”

```
…
class GazaPlay {
int V_Bar1_4[24][10];
   int H_Bar9_12[28][MAX_H];
…
public:
  GazaPlay(int w = 0, int h = 0)
     : wide(w), high(h) {}
  void MoveLeft();
   void MoveRight();
  void MoveUp();
   void MoveDown();

};
…
```

```
void GazaPlay::MoveLeft()
{
   if( pm_x > 0 ) {
      if( st_V_Bar[pm_y][pm_x] == 0 ) {
         layerS = PS_LEFT;
         pm_x -= 1;
         pm_c = PS_MOVE_FRAME;
      }
      else {
         if( st_V_Bar[pm_y][pm_x-1] == 0 ) {
            st_V_Bar[pm_y][pm_x] = 0;
            st_V_Bar[pm_y][pm_x-1] = 1;
            pm_x -= 1;
            PlayerS = PS_LEFT;
            pm_c = PS_MOVE_FRAME;
         }
         …
```

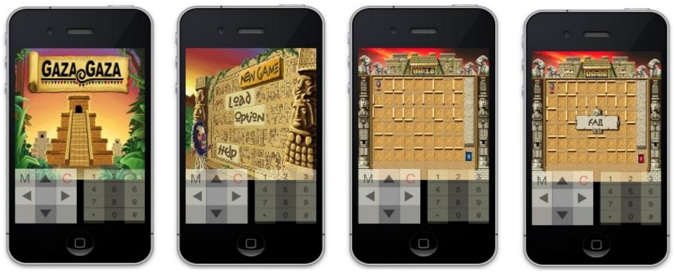
Next, Smart cross platform’s C++ to SIL compiler was used to convert the contents into SIL code and reconvert it into SEF using an assembler. Table 2 shows the translated results.

Table 5. SIL Code & SEF File for the “Gaza Gaza”

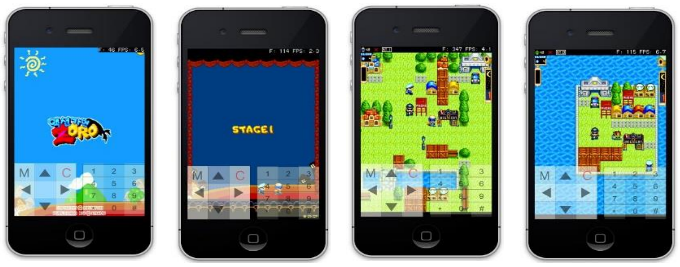
```
%FunctionStart
   .func_name &GazaPlay::MoveLeft$2
   .func_type 2
   .param_count 0
   .opcode_start
      proc 4 1 1
      str.p 1 0
      lod.p 1 0
      ldc.p 456
      add.p
      ldi.i
      ldc.i 0
      gt.i
      fjp ##93
      lod.p 1 0
      ldc.p 4236
      add.p
      lod.p 1 0
      …
```

```
000006a0h: 27 00 00 00 A0 DD 02 00 A2 00 BA 00 00 00 A3 00
000006b0h: 09 00 33 00 00 00 09 00 A9 01 00 00 27 00 00 00
000006c0h: 88 EA 02 00 A2 00 BA 00 00 00 17 00 01 00 00 00
000006d0h: 00 00 0C 00 64 19 00 00 3E 00 09 00 00 00 00 00
000006e0h: 34 00 17 00 01 00 00 00 00 00 0C 00 64 19 00 00
000006f0h: 3E 00 1F 00 09 00 17 00 00 00 6F 00 93 00 D4 06
00000700h: 00 00 A3 00 09 00 47 00 00 00 17 00 01 00 00 00
00000710h: 00 00 0C 00 64 19 00 00 3E 00 1F 00 09 00 16 00
00000720h: 00 00 47 00 3B 00 09 00 A7 01 00 00 27 00 00 00
00000730h: 24 ED 02 00 A2 00 BA 00 00 00 17 00 01 00 00 00
00000740h: 00 00 0C 00 64 19 00 00 3E 00 1F 00 03 00 09 00
00000750h: 01 00 00 00 3B 00 17 00 01 00 00 00 00 00 0C 00
00000760h: 64 19 00 00 3E 00 05 00 34 00 94 00 46 06 00 00
00000770h: A3 00 09 00 39 02 00 00 09 00 A9 01 00 00 27 00
00000780h: 00 00 B0 0A 03 00 A2 00 BA 00 00 00 A3 00 09 00
00000790h: 80 02 00 00 09 00 61 00 00 00 27 00 00 00 18 86
000007a0h: 02 00 09 00 01 00 00 00 A2 00 BB 00 00 00 A3 00
000007b0h: 09 00 4D 02 00 00 09 00 69 00 00 00 27 00 00 00
000007c0h: 1C 0D 03 00 A2 00 BA 00 00 00 A3 00 09 00 4D 02
000007d0h: 00 00 09 00 E2 00 00 00 27 00 00 00 28 24 03 00
000007e0h: A2 00 BA 00 00 00 A3 00 09 00 4D 02 00 00 09 00
000007f0h: 25 01 00 00 27 00 00 00 14 32 03 00 A2 00 BA 00
00000800h: 00 00 A3 00 09 00 4D 02 00 00 09 00 4A 01 00 00
00000810h: 27 00 00 00 C8 39 03 00 A2 00 BA 00 00 00 A3 00
00000820h: 09 00 39 02 00 00 09 00 61 00 00 00 27 00 00 00
```

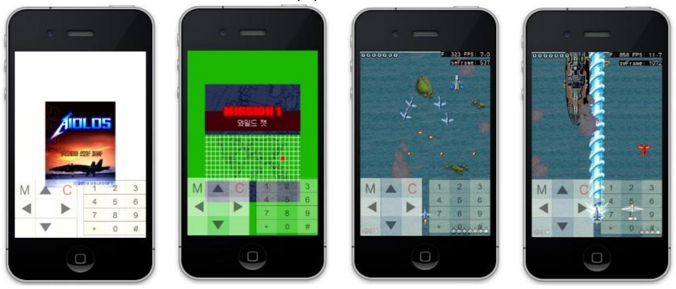
Next, SEF implemented was uploaded to SVM and the execution results are shown. The result screen shows “Gaza Gaza”, “Zoro” and “Ailos” executed and running in the emulator. The compilation process for each process is the same as the process introduced earlier for “Gaza”.



(a) Gaza Gaza



(b) Zoro



(c) Aiolos

Figure 7. Execution Results of the Mobile Game Contents on the iOS SVM (iOS Emulator Environment)

Moving forward from executing in the emulator, tests were carried out to check the implemented SVM worked on the hardware. This was carried out on an iPod Touch with iOS installed. The implemented SVM was loaded and the contents were executed. Figure 8 shows a screen of "Gaza Gaza" working on the iPod Touch.

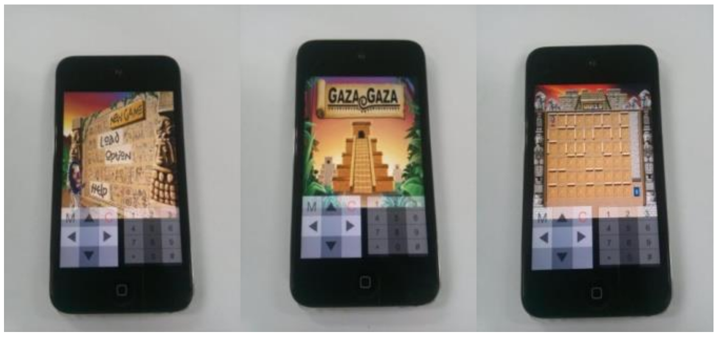


Figure 8. Execution Result of the "Gaza Gaza" Game Content on the iOS SVM (ported on the iPod Touch)

5. Conclusions and Further Researches

A virtual machine has a characteristic of being able to use application programs without changing it even if the operating system is changed. Especially, it is a core technology which executes various contents in the latest mobile, embedded and smart systems. In the case of mobile, embedded and smart systems, for embedded systems, due to diversity and frequent changes in processors and operating systems, it is suitable to use virtual machines. For downloadable solutions, virtual machine applications seem to be the valid method.

In the current paper, a virtual machine was designed and implemented on an iOS platform to download and execute numerous applications loaded on smart devices. As a result, an environment where various game contents can be executed on smart devices without restriction of iOS platform and programming language was provided.

In this research, contents created using C++ was executed on an SVM in iOS and the results were shown. All languages supported by the smart cross platform are executable. Therefore, an environment to develop contents easily in iOS without language restriction was provided.

In future research, development of compiler to further secure language provided by the smart cross platform, optimization of code to enhance execution speed of virtual machine, develop security method to minimize bugs and security problems from initial period of content development will be carried out.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(No.20110006884) and in part by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and future Planning(No.2013067205)

References

[1] Apple, iOS Reference Library, iOS Technology Overview, http://developer.apple.com/devcenter/ios.
[2] Goole, Android-An Open Handset Alliance Project, http://code.google.com/intl/ko/android/.
[3] Y. S. Son and Y. S. Lee, "Design and Implementation of the Virtual Machine for Smart Devices", Proc. of the 2011 Fall Conference, Korea Multimedia Society, Seoul, Korea, **(2011)**.
[4] Y. S. Son and Y. S. Lee, "A Study on the Smart Virtual Machine for Smart Devices", Information: An International Interdisciplinary Journal, vol. 16, **(2013)**, pp. 1465.
[5] Y. S. Lee and Y. S. Son, "A Study on the Smart Virtual Machine for Executing Virtual Machine Codes on Smart Platforms", International Journal of Smart Home, vol. 6, no. 4, **(2012)**.
[6] Y. S. Son and Y. S. Lee, "An Objective-C Compiler to Generate Platform-Independent Codes in Smart Device Environments", Information: An International Interdisciplinary Journal, vol. 16, **(2012)**, pp. 1457.
[7] Y. S. Son and Y. S. Lee, "Design and Implementation of an Objective-C Compiler for the Virtual Machine on Smart Phone", CCIS, vol. 262, no. 52, **(2011)**.
[8] S. L. Yun, D. G. Nam, S. M. Oh and J. S. Kim, "Virtual Machine Code for Embedded Systems", International Conference on CIMCA, Osaka, Japan, **(2004)**.
[9] J. Gough, "Compiling for the .NET Common Language Runtime(CLR)", Prentice-Hall, **(2002)**.
[10] S. Lindin, "Inside Microsoft .NET IL Assembler", Microsoft Press, (**2002**).
[11] J. Meyer and T. Downing, "JAVA Virtual Machine", O'REYLLY, **(1997)**.
[12] J. Engel, "Programming for the Java Virtual Machine", Addison-Wesley, **(1999)**.
[13] Y. S. Lee, Y. K. Kim and H. J. Kwon, "Design and Implementation of the Decompiler for Virtual Machine Code of the C++ Compiler in the Ubiquitous Game Platform", LNCS, vol. 4413, no. 511, **(2007)**.
[14] Y. S. Lee, "The Virtual Machine Technology for Embedded Systems", Journal of the Korea Multimedia Society, vol. 6, no. 36, **(2002)**.
[15] Y. S. Son and Y. S. Lee, "A Study on Optimization Techniques for the Smart Virtual Machine Platform", LNCS, vol. 7709, no. 167, **(2012)**.

Authors

Yunsik Son, he received the B.S. degree from the Dept. of Computer Science, Dongguk University, Seoul, Korea, in 2004, and M.S. and Ph.D. degrees from the Dept. of Computer Engineering, Dongguk University, Seoul, Korea in 2006 and 2009, respectively. Currently, he is a Researcher of the Dept. of Computer Science and Engineering, Dongguk University, Seoul, Korea. His research areas include smart system solutions, secure software, programming languages, compiler construction, and mobile/embedded systems.

JaeHyun Kim, he received the B.S. degree from the Dept. of Mathematics, Hanyang University, Seoul, Korea, in 1986, and M.S. and Ph.D. degrees from Dept. of Statistics, Dongguk University, Seoul, Korea in 1989 and 1996, respectively. He was a chairman of Dept. of Internet Information 2002-2007. Currently, he is a member of the Korean Data & Information Science Society and a Professor of Dept. of Computer Engineering, Seokyeong University, Seoul, Korea. His research areas include mobile programming, cloud system and data analysis.

YangSun Lee, he received the B.S. degree from the Dept. of Computer Science, Dongguk University, Seoul, Korea, in 1985, and M.S. and Ph.D. degrees from Dept. of Computer Engineering, Dongguk University, Seoul, Korea in 1987 and 2003, respectively. He was a Manager of the Computer Center, Seokyeong University from 1996-2000, a Director of Korea Multimedia Society from 2004-2013, a General Director of Korea Multimedia Society from 2005-2006 and a Vice President of Korea Multimedia Society in 2009. Also, he was a Director of Korea Information Processing Society from 2006-2010 and a President of a Society for the Study of Game at Korea Information Processing Society from 2006-2010. And, he was a Director of Smart Developer Association from 2011-2013. Currently, he is a Professor of Dept. of Computer Engineering, Seokyeong University, Seoul, Korea. His research areas include smart system solutions, programming languages, and embedded systems.