Design and Implementation of the WIPI-to-Windows Mobile Automatic Game Content Converter System

YangSun Lee[1] and YunSik Son[2*]

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

Abstract

Mobile communication companies choose different mobile platforms from each other so that developers have to create contents for each of the platforms according to their different characteristics or undergo a converting process to provide the game contents to consumers. However, existing mobile game contents require large amounts of time and expenses when converting to be used on different platforms. This is one of the reasons why a variety of contents are not provided on such platforms.

In this paper, in order to resolve such problems the game contents of the existing mobile platform, WIPI (Wireless Internet Platform for Interoperability), will be analyzed. Then a resource converter, a source translator and a platform mapping engine will be implemented in order to convert the game contents for use on a smart phone platform, Windows Mobile. A mobile game contents converter system has enabled contents to be transferred into smart phone platforms within a short time, so that the time and money it takes to launch services for different mobile communication companies can be reduced. Furthermore, contents developed for use on feature phones can also be converted and used on smart phone platforms to increase reusability of contents and also new contents creation processes can heighten productivity to consequently provide a more diverse range of mobile game contents to users.

Keywords: *Mobile Game Contents Converter, Contents Analyzer, Resource Converter, Source Translator, Platform Mapping Engine, WIPI(Wireless Internet Platform for Interoperability), Windows Mobile, WIPI-to- Windows Mobile Game Converter*

1. Introduction

Due to the use of different mobile platforms for each of the mobile communications companies, mobile contents developers must repeat development process to create different versions of games that match the different characteristics of the different smart phone platforms if they aspire to service their games. This has led to the need for developers to convert contents that have been already developed for use on smart phone platforms. However, large amounts of time and costs occur from analyzing one mobile game content's sources and resources and then converting (porting and retargeting) it. The time and money that could be used to create new game contents are being used to service an existing product on different platforms.

In this paper, to resolve this problem, the WIPI-to-Windows Mobile automatic mobile game converter system was designed to automatically translate game content from the WIPI(Wireless Internet Platform for Interoperability) platform for feature phones

to the Windows Mobile platform for smart phones. The WIPI-to- Windows Mobile converter consists of a content analyzer, resource converter, source translator, and platform mapping engine. The content analyzer analyzes the content that is input, and produces an output in which the resource data and source code stored within the content are separated. The resource converter is a system which converts the text or binary resource data from the game to be converted into image, sound and user data so that it can be used on the target platform's file system. The platform mapping engine is a system which provides API functions which allow the previous platform's execution environment to be recreated using the target platform's wrapper functions.

By automatically converting the existing mobile game contents to the smart phone game contents, the existing game contents can be ported quickly to a different platform and the human resources, time and expenses used to service the contents to different mobile communication companies can be saved [1-11].

2. Related Studies

2.1. WIPI

WIPI(Wireless Internet Platform for Interoperability) is legislated by KWISF(Korea Wireless Internet Standardization Forum) and a standard chosen by KTTA(Korea Telecommunications Technology Association) as an application program execution environment for mobile communication platforms. Because mobile communication companies use different platforms each, contents developing companies feel a great burden from having to repeat development of contents, users' rights of using are restricted and cell phone manufactures feel burdened to develop new phones. Thus a need for standardization arose and as a result, the Korean standard was set for wireless internet platforms. Figure 1 shows the structure of a WIPI platform.

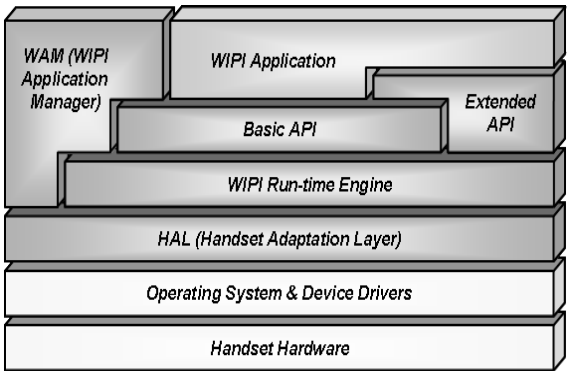

Figure 1. System Configuration of the WIPI Platform

WIPI supports the C language and the Java language which were the programming languages used when developing applications. In the case of Java, bytecode programs are recompiled using an AOTC (Ahead Of Time Compiler) and then executed in a native way for each cell phone. The WIPI standards can be largely divided into the HAL (Handset Adaptation Layer) and the basic API. HAL is a standardized hardware abstraction layer to increase transferability. Also, since it is hardware-independent, it can be executed with no connection with the native system. Only using the standardized HAL and API, a WIPI runtime engine can be implemented and a basic API – for both the C language and the Java language - can be created over it [12].

2.2. Windows Mobile

Windows Mobile(brand name changed to Windows Phone) is a mobile operating system invented by Microsoft Corporation. It is an embedded operating system based on Windows CE. The Windows Mobile 6 version is a platform for mobile devices which was created using Windows CE 5.0 as its base. Figure 2 shows the structure of a Windows Mobile System.

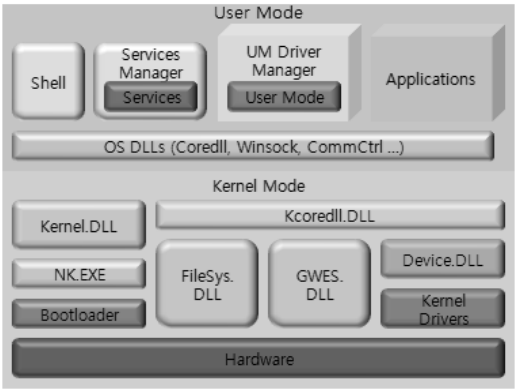

Figure 2. System Configuration of the Windows Mobile Platform

Windows Mobile 6.5 version is the result of applying a Windows desktop line to a windows mobile device. In this version, a considerable number of UIs were changed for use with a touch screen, the classic pocket PC version supported previously and the resolution version that was seldom used were deleted, and a reinforced simpler version of internet explorer mobile 6 (compared to Windows mobile 6.1.4) is built in. Windows mobile is the basis to Windows embedded CE 5.2 and supports .NET compact framework. The Windows mobile platform offers higher security and diverse APIS such as Bluetooth and POOM (Pocket Outlook Object Model). It also includes a wide range of programming models such as the native code (C/C++), a managed code (C#), mobile web development, multithreading and other device supports. The development environment is similar to that of Windows, allowing development time and money to be reduced [13].

2.3. Existing Mobile Game Contents Converter

Until now, despite the invigoration of the mobile market, there has been a lack of research for mobile contents converters which has led to few examples to refer to. Furthermore, converters for existing contents generally only allow conversion of contents that have similar programming language environments or don't allow automatic conversion at all. The reality is that programmers have to undergo the converting process by hand.

There has been a study on an existing mobile game contents converter using XML that attempted to convert Java contents [14-17]. In addition, the functions of the API used in the source codes to be converted are imitated and redefined using wrapper functions. Therefore there is no need to convert the source codes while the same functions are used. There was a study on the mutual conversion of BREW C and WIPI C [18] or converting GVM C into BREW C [19], however it was flawed because the source codes were not automatically converted, the users had to intervene and convert it manually.

On the other hand, studies on automatic conversion of mobile game contents using the compiler writing system [1-11, 20-22] have been attempted. Studies have suggested a method to increase the reusability of game contents and enhance productivity by converting mobile C contents of the GVM platform into WIPI C, Java or MIDP Java [1]. Also other studies are

underway to convert existing mobile game contents for use in the rising smart phone market for operating systems such as Android and iOS [2-3] such as the WIPI-to-iOS converter, the WIPI-to-Android converter, the GNEX-to-iOS converter, the GNEX-to-Android converter, the Android-to-iOS converter, and the iOS-to-Android converter system.

3. The WIPI-to-Windows Mobile Game Converter System

The WIPI-to-Windows Mobile automatic mobile game content converter receives WIPI game contents in source form, which it converts into the game contents source form that is run on the Windows Mobile platform. For automatic conversion on the source level, first the source code must be converted into a source code of the subject platform that executes the same action. Other resource data such as images, sound and etc. must also be converted into a form that can be used on the new platform. In addition, the API library must be provided to maintain equivalent programming and event environments. Figure 3 shows a model of the WIPI-to-Windows Mobile game contents automatic converter system.

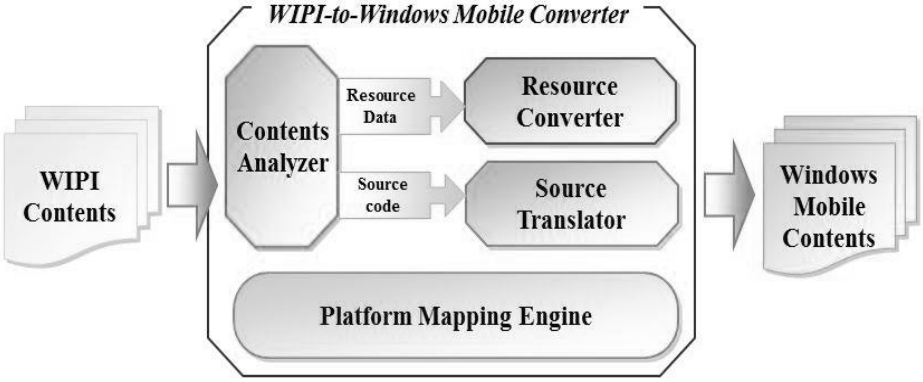

Figure 3. WIPI-to-Windows Mobile Game Content Converter System

The WIPI-to-Windows Mobile converter consists of a contents analyzer which classifies resources and source codes, a resource converter which converts the WIPI C resource format into a format that is usable on Windows Mobile, a source translator which translates WIPI C source codes into Windows Mobile CPP source codes, an environment which enable the equivalent display and execution of WIPI C contents on the Windows Mobile platform and a platform mapping engine which provides APIs [1-8].

3.1. Content Analyzer

A WIPI content analyzer receives WIPI contents' project files, separates the sources and resources of the content and creates a list of them. In the separation process of a WIPI content analyzer, there is the content analysis step which separates resource codes, source codes and other files, there is the source file categorization step which separates the source files and creates a list, there is the resource management file check step where the resource managing files of WIPI are analyzed, there is the image file categorization step where the image files are classified and listed up and finally there is the sound file categorization step where sound files are classified and listed up[4-7]. Figure 4 is a diagram of a WIPI content converter system. Figure 5 shows the results of experimenting with the WIPI content analyzer.

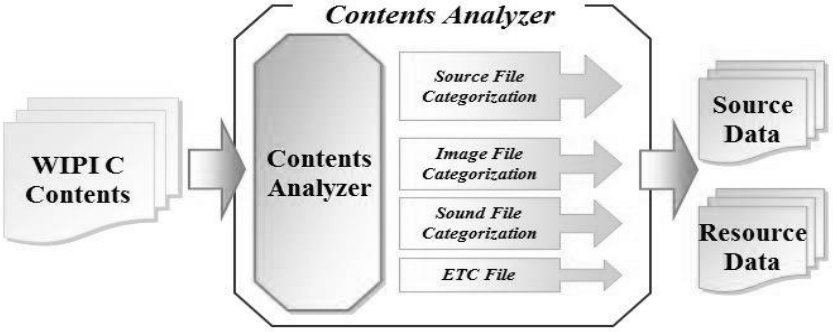

Figure 4. WIPI Content Analyzer System

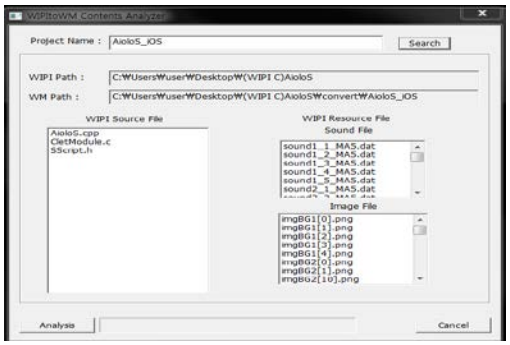

Figure 5. Result of the WIPI Content Analyzer

3.2. Resource Converter

The WIPI-to-Windows Mobile resource converter is a file converting system that converts the resource data in text or binary form from the WIPI input contents into forms - image data, sound data and user data that can be used in the target platform, Windows Mobile. For this to be executed, the image file forms (VDI, BMP, GIF and *etc*.), the sound forms (MMF and *etc*.) and user data for different platforms must be researched and then converted into forms usable in the target platform[4-7]. Figure 6 shows a model of the resource converter system.

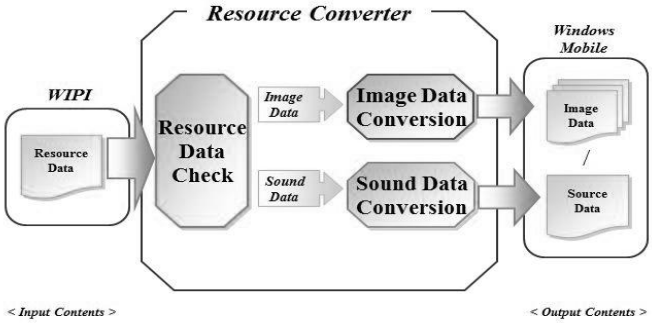

Figure 6. Model of the Resource Converter System

In WIPI C, all resources are managed using a file with an xrf extension, but since this is a structure that is only used in WIPI C, in order to be used on different platforms, a resource management file generator needs to be created so that resource management files can be made according to the different platforms. A resource management file is created in XML grammar and the types of resources are classified into ‘Token’, ‘Image’ and ‘Ami’. Here, what is

needed in the converter is classified into the 'Image' category, a list of image files and sound files. Generation of resource managing files requires writing up lists of the necessary items stated above separately so that when automatic conversion takes place, the necessary resource management file can be created.

In this paper, the Windows Mobile resource management file wmr(windows mobile resource) was regulated and used as the only extension. The resource management file structure for Windows Mobile consists of the images name, ID and route. To image routes, "storage cards" are added because when handling images they are loaded additionally from the memory so additional resource management files are created. Also, after resource management files are created, they are added to the Windows Mobile project composition just like image and sound resources. Figure 7 shows the executing result of a WIPI-to-Windows Mobile resource converter.

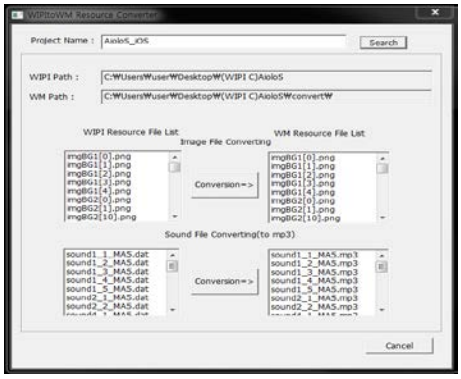

Figure 7. Result of a WIPI-to-Windows Mobile Resource Converter

3.3. Source Translator

The source translator receives the WIPI C source codes that are output by the contents analyzer and translates them into Windows Mobile source codes which are semantically equivalent and execute the execute the same actions as the WIPI C source codes. Because WIPI C and Windows Mobile are both C based platforms, the characteristics of the language are the same. However, there exist some parts which have been differently altered to suit each of the platform's virtual machines.

Source translators have been created so that they can overcome the differences of the platforms and automatically translate the game source programs using compiler writing technology. Compiler technology analyzes programming grammar and syntax and provides a method for automatic translation into another language [1-6, 8, 20-22]. Figure 8 is a depiction of the source translator.

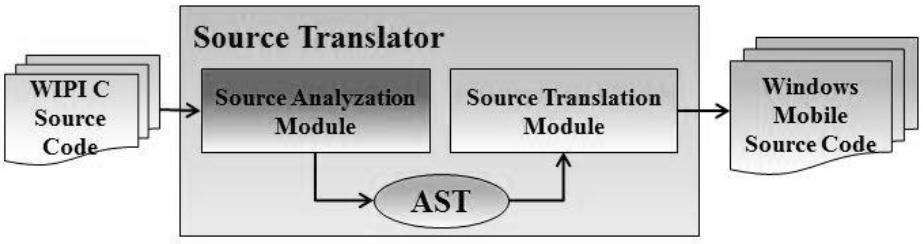

Figure 8. The Source Translator System

Source translators can be largely divided into the source analysis module and source translation module. The source analysis modules receive WIPI C source code inputs and carries out lexical and syntax analysis to create an Abstract Syntax Tree (AST). The source translation module searches the AST and creates Windows Mobile source codes which are semantically equivalent to the WIPI C source codes.

3.3.1. Source Analysis Module

WIPI The source analysis module is the first component of a source translator. It receives WIPI C source code inputs, carries out lexical and syntax analysis, outputs sentence structure as AST and delivers the AST to the source translation module. Figure 9 shows the source analysis module.

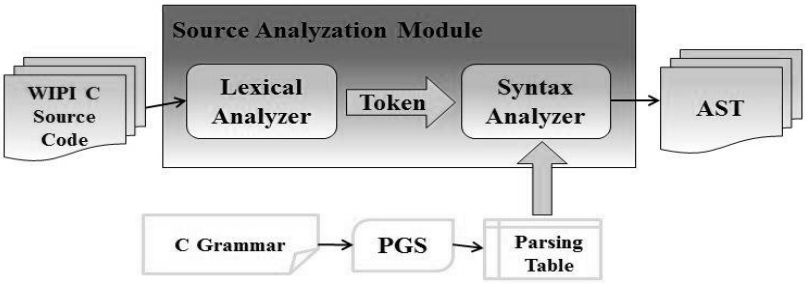

Figure 9. Source Analysis Module

The source analysis module can be largely divided into the lexical analyzer and the syntax analyzer. The lexical analyzer classifies WIPI C source codes that it receives into tokens, the smallest unit with grammatical meaning. The lexical analyzer analyzes the source codes and delivers the results to the syntax analyzer. The token information transferred between the two analyzers are composed of token numbers and token values. Table 1 shows the output results of the tokens analyzed by the lexical analyzer.

Table 1. Tokens' Result of the Lexical Analyzer

Source code	Token

void MoveCloud() {
int i;
for (i=0; i<MAX_CLOUD; i++) {
if (gCloudS[i] == FS_ACTIVATE) {
gCloudY[i] += gCloudY[i];
if (gCloudY[i] > swHeight+60) {
gCloudS[i] = FS_GHOST;
gCloudCount--;
}
}
}
}
	token : void (110, 0)
	token : %ident (10, MoveCloud)
	token : ((21, 0)
	token :) (22, 0)
	token : { (113, 0)
	token : int (94, 0)
	token : %ident (10, i)
	token : ; (38, 0)
	token : for (89, 0)
	token : ((21, 0)
	token : %ident (10, i)
	token : = (43, 0)
	token : %integer (11, 0)
	token : ; (38, 0)
	token : %ident (10, i)
	token : < (39, 0)
	token : %ident (10, MAX_CLOUD)
	token : ; (38, 0)
	token : %ident (10, i)
	token : ++ (26, 0)
	token :) (22, 0)
	token : { (113, 0)
	token : if (91, 0)
	token : ((21, 0)
	token : %ident (10, gCloudS)
	token : [(69, 0)
	token : %ident (10, i)
	token :] (70, 0)
	token : == (44, 0)
	token : %ident (10, FS_ACTIVATE)
	token :) (22, 0)
	token : { (113, 0)
	token : %ident (10, gCloudY)
	token : [(69, 0)
	token : %ident (10, i)
	token :] (70, 0)
	token : += (27, 0)

The syntax analyzer uses the token information obtained from the lexical analyzer and the parsing table created by the Parser Generating System (PGS) to analyze the syntax of the program. The results of the syntax analyze output error messages about wrong programs, and for correct syntax, results are created in the form of a syntax tree. This tree is the Abstract

Syntax Tree (AST) which is used in the source translation module. Depending on the stack's top and the current input symbol, the syntax analyzer refers to the parsing table and makes a parsing action.

The four parsing actions of the syntax analyzer include shift, reduce, accept, and error. Depending on the top of the stack and the currently evaluated symbol, it refers to the parsing table and makes a decision. Figure 10 shows the process followed by the analyzer. Because this process is a continuous action of shifting and reducing, as described below, it is called a 'shift-reduce' syntax analyzer.

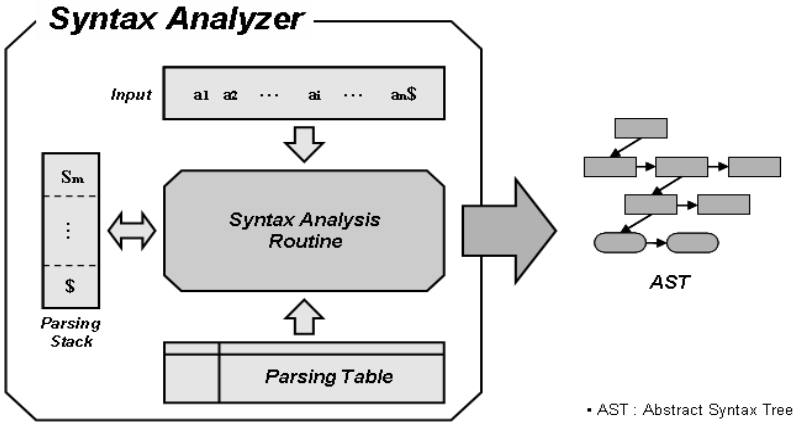

Figure 10. Shift-Reduce Syntax Analyzer

The shift action signifies the transfer of the currently evaluated symbol to the stack. The reduce action abbreviates the handle at the top of the stack according to the creation rules. In addition, the accept action indicates that the given string is grammatically correct, and the error action shows that a sentence is wrong because it cannot be revealed in its current symbol state. Table 2 shows the AST information created by the syntax analyzer.

Table 2. Source Code and AST Information

Source code	AST

void MoveCloud() {
int i;
for (i=0; i<MAX_CLOUD; i++) {
if (gCloudS[i] == FS_ACTIVATE) {
gCloudY[i] += gCloudY[i];
if (gCloudY[i] > swHeight+60) {
gCloudS[i] = FS_GHOST;
gCloudCount--;
}
}
}
}
	Nonterminal: FUN_DEF
	Nonterminal: DCL_SPEC
	Nonterminal: VOID
	Nonterminal: FUNC_DCL
	Nonterminal: SIMPLE_VAR
	Terminal(Type:id / Value:MoveCloud)
	Nonterminal: COMPOUND_ST
	Nonterminal: DCL
	Nonterminal: DCL_SPEC
	Nonterminal: INT
	Nonterminal: DCL_ITEM
	Nonterminal: SIMPLE_VAR
	Terminal(Type:id / Value:i)
	Nonterminal: FOR_ST
	Nonterminal: INIT_PART
	Nonterminal: ASSIGN_OP
	Terminal(Type:id / Value:i)
	Terminal(Type:int / Value:0)
	Nonterminal: TEST_PART
	Nonterminal: LT
	Terminal(Type:id / Value:i)
	Terminal(Type:id / Value:MAX_CLOUD)
	Nonterminal: MODIFY_PART
	Nonterminal: POST_INC
	Terminal(Type:id / Value:i)
	Nonterminal: COMPOUND_ST
	Nonterminal: IF_ST
	Nonterminal: EQ
	Nonterminal: INDEX
	Terminal(Type:id / Value:gCloudS)
	Terminal(Type:id / Value:i)
	Terminal(Type:id / Value:FS_ACTIVATE)
	Nonterminal: COMPOUND_ST
	Nonterminal: ADD_ASSIGN_OP
	Nonterminal: INDEX
	Terminal(Type:id / Value:gCloudY)
	Terminal(Type:id / Value:i)
	Nonterminal: INDEX
	Terminal(Type:id / Value:gCloudDY)
	Terminal(Type:id / Value:i)

3.3.2. Source Translation Module

The source translation module receives the AST as an input from the source analysis module, searches each of the nodes of the tree and creates source codes that will execute in the same manner in the target platform, Windows Mobile, as they did in WIPI. Since this module has been designed to analyze ASTs which are expressed with consistency, it is possible to match it with all program structures that can be created.

The source translation module which receives AST as an input, begins a successive search from the tree's root. During the search process, if a significant node appears, the pattern matching source writer receives the node and translates it into the Windows Mobile source code. When the entire AST search process is finished, the pattern matching source writer analyzes the nodes until now and creates each of the translated source codes into one file, this is the Windows Mobile source code. Figure 11 shows the execution process of the source translation module.

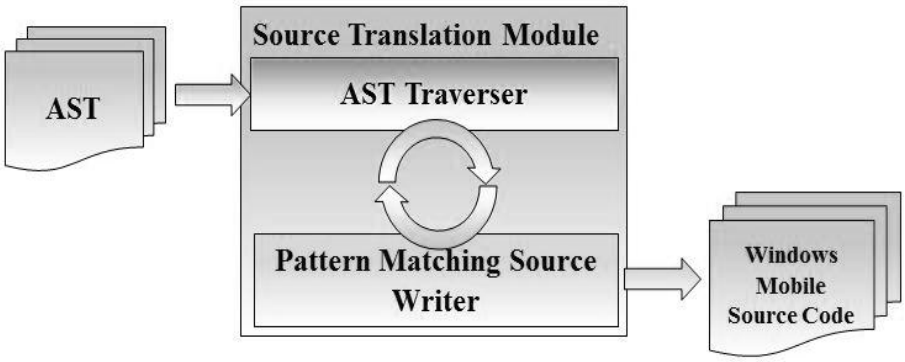

Figure 11. Source Translation Module

Figure 12 shows the execution result of the source translator. The list on the left is the list of WIPI C files to be translated and the list on the right is a list of the converted Windows Mobile files. When the conversion button is pressed, the source translator automatically converts the sources.

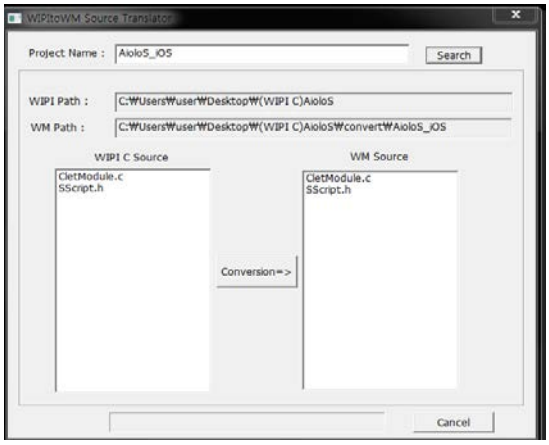

Figure 12. Executing Result of the Source Translator

Figure 13 shows an example of traversing the AST of a WIPI C program created by the syntax analyzer and converting it into the Windows Mobile program.

AST	WIPI Source.

Nonterminal: FUN_DEF
Nonterminal: DCL_SPEC
Nonterminal: VOID
Nonterminal: FUNC_DCL
Nonterminal: SIMPLE_VAR
Terminal(Type:id / Value:MoveCloud)
Nonterminal: COMPUND_ST
Nonterminal: DCL
Nonterminal: DCL_SPEC
Nonterminal: INT
Nonterminal: DCL_ITEM
Nonterminal: SIMPLE_VAR
Terminal(Type:id / Value:i)
Nonterminal: FOR_ST
Nonterminal: INIT_PART
Nonterminal: ASSIGN_OP
Terminal(Type:id / Value:i)
Terminal(Type:int / Value:0)
Nonterminal: TEST_PART
Nonterminal: LT
Terminal(Type:id / Value:i)
Terminal(Type:int / Value:3)
Nonterminal: MODIFY_PART
Nonterminal: POST_INC
Terminal(Type:id / Value:i)
Nonterminal: COMPUND_ST
Nonterminal: IF_ST
Nonterminal: EQ
Nonterminal: INDEX
Terminal(Type:id / Value:gCloudS)
Terminal(Type:id / Value:i)
Terminal(Type:id / Value:FS_ACTIVATE)
Nonterminal: COMPUND_ST
Nonterminal: ADD_ASSIGN_OP
Nonterminal: INDEX
Terminal(Type:id / Value:gCloudY)
Terminal(Type:id / Value:i)
Nonterminal: INDEX
Terminal(Type:id / Value:gCloudDY)
Terminal(Type:id / Value:i)
Nonterminal: IF_ST
Nonterminal: GT
Nonterminal: INDEX
Terminal(Type:id / Value:gCloudY)
Terminal(Type:id / Value:i)
Nonterminal: ADD
Terminal(Type:id / Value:swHeight)
Terminal(Type:int / Value:60)
Nonterminal: COMPUND_ST
Nonterminal: ASSIGN_OP
Nonterminal: INDEX
Terminal(Type:id / Value:gCloudS)
Terminal(Type:id / Value:i)
Terminal(Type:id / Value:FS_GHOST)
Nonterminal: POST_DEC
Terminal(Type:id / Value:gCloudCount)
	void MoveCloud() {
	int i;
	for (i=0; i<MAX_CLOUD; i++) {
	if (gCloudS[i] == FS_ACTIVATE) {
	gCloudY[i] += gCloudY[i];
	if (gCloudY[i] > swHeight+60) {
	gCloudS[i] = FS_GHOST;
	gCloudCount--;
	}
	}
	}
	}
	Windows Mobile Source.
	void MoveCloud() {
	int i;
	for (i=0; i<MAX_CLOUD; i++) {
	if (gCloudS[i] == FS_ACTIVATE) {
	gCloudY[i] += gCloudY[i];
	if (gCloudY[i] > swHeight+60) {
	gCloudS[i] = FS_GHOST;
	gCloudCount--;
	}
	}
	}
	}

AST	WIPI Source.

Nonterminal: FUN_DEF
Nonterminal: DCL_SPEC
Nonterminal: VOID
Nonterminal: FUNC_DCL
Nonterminal: SIMPLE_VAR
Terminal(Type:id / Value:InitFlight)
Nonterminal: COMPUND_ST
Nonterminal: ASSIGN_OP
Terminal(Type:id / Value:gFlightS)
Terminal(Type:id / Value:FS_ACTIVATE)
Nonterminal: ASSIGN_OP
Terminal(Type:id / Value:gFlightX)
Terminal(Type:id / Value:gCX)
Nonterminal: ASSIGN_OP
Terminal(Type:id / Value:gFlightY)
Nonterminal: ADD
Terminal(Type:id / Value:swHeight)
Terminal(Type:int / Value:40)
Nonterminal: ASSIGN_OP
Terminal(Type:id / Value:gFlightC)
Terminal(Type:int / Value:0)
Nonterminal: ASSIGN_OP
Terminal(Type:id / Value:gFlightStep)
Terminal(Type:int / Value:0)
Nonterminal: ASSIGN_OP
Terminal(Type:id / Value:gFlightShooting
Terminal(Type:id / Value:FALSE)
Nonterminal: IF_ST
Nonterminal: EQ
Terminal(Type:id / Value:gStage)
Terminal(Type:int / Value:0)
Nonterminal: ASSIGN_OP
Terminal(Type:id / Value:gFlightMT)
Terminal(Type:int / Value:0)
	void InitFlight(){
	gFlightS = FS_ACTIVATE;
	gFlightX = gCX;
	gFlightY = swHeight + 40;
	gFlightC = 0;
	gFlightStep = 0;
	gFlightShooting = FALSE;
	if (gStage == 0)
	gFlightMT = 0;
	}
	Windows Mobile Source.
	void InitFlight(){
	gFlightS = FS_ACTIVATE;
	gFlightX = gCX;
	gFlightY = swHeight + 40;
	gFlightC = 0;
	gFlightStep = 0;
	gFlightShooting = FALSE;
	if (gStage == 0)
	gFlightMT = 0;
	}

Figure 13. AST and Source Translation

3.4. Platform Mapping Engine

Platform mapping engines convert APIs such as displays, graphics, sound outputs, system variables and event handlers used in WIPI contents' source codes into forms that are usable in Windows Mobile, the target platform. For this, identical execution environments to WIPI's are built and based on these environments, wrapper functions are used to execute WIPI's APIs, system variables and *etc*. in the same form and thus implement WIPI's APIs as Windows Mobile's APIs. By doing so, the translated source codes for Windows Mobile contents do not need additional adjustments before implementation. Also it enables simplified understanding and source code modification as identical forms of APIs used in WIPI are used [1-6]. Figure 14 is a model of a platform mapping engine.

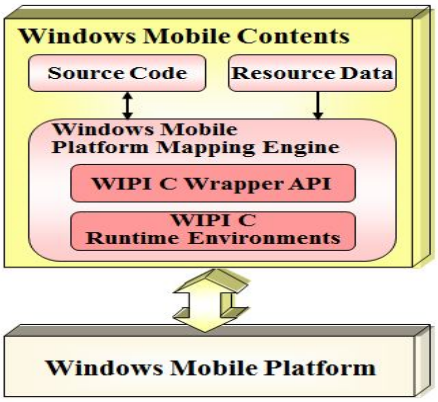

Figure 14. Model of a Platform Mapping Engine

If graphic library functions from WIPI's API are used to create image data or texts, they create not an actual LCD frame buffer but a virtual LCD frame buffer. In this case, internal virtual LCD buffers are put out so that actual LCDs cannot be created. To generate actual LCDs, a library function MC_grpFlushLcd must be used.

3.3.1. Project File Generation

The project files which make up Windows Mobile, are managed by the "Microsoft Visual Studio Solution" and actual sources are managed by the "VC++ Project" file. In order to run actual sources, there is a composition which must be included in the project. This composition is Windows Mobile's basic headers, Register Class set-up, Procedure registration and WINAPI WinMain function. Through the platform map-ping engine, headers for Windows Mobile and wrapper APIs for WIPI C source are added to the basic headers. RegisterClass's registration related contents are needed to differentiate different contents within one Windows Mobile phone. Also, Procedure is responsible for actions such as draw, event and etc. for contents and handling them. The WinMain function is responsible for starting the contents so it takes care all of the actions above in order.

3.3.2. Event Environments

In WIPI C, "handleCletEvent", an event handler is registered and used. Each event is defined as a certain type and when an event occurs, it is automatically called. Also in order to provide additional information about the event to the event handler, two parameter variables are used. The platform mapping engine converts events that occurred in Windows Mobile into a WIPI C event form. Then, events are transferred to event handlers, which have been defined

for specific translated source codes, so that they can be handled. Event handlers have been set to be called when events for WIPICAPP or WIPICKNL sources – WIPI C's APIs defined – occur. In this paper, only the timer event and key input event among WIPI C's events were handled and implemented.

3.3.3. Graphics Environments and Image Output

WIPI basically provides graphics environments using a frame buffer. Frame buffers provide main LCD frame buffers and assistant LCD frame buffers. And Virtual LCD Frame Buffers are used internally to increase speed of generating outputs and to ensure smooth output generation. Figure 15 shows the WIPI's graphic output method.

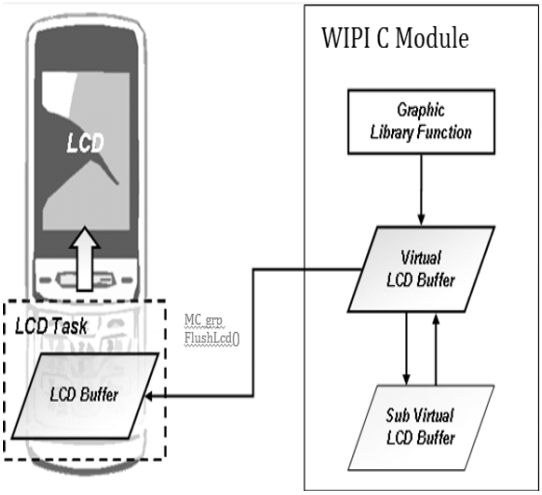

Figure 15. WIPI's Graphic Output Method

Windows Mobile's graphic functions have been made so that actual LCD buffers are generated, so when they are used, the outputs appear on the screen right away. For platform mapping engines support graphic output functions in the same way as WIPIs do, they use Windows Mobile's API to generate a virtual LCD buffer. Also graphic functions identical to WIPI C's graphic library functions have been designed to create the virtual LCD buffer's images, figures and texts. The function MC_grpFlushLcd function has been designed to use virtual LCD buffer's information so that deliver it to actual LCD buffers and consequently generate LCDs in the same way as WIPI C.

WIPI C's image output process is divided into three steps; image information collection, image loading and image output. Image information collection is the process of bringing the resource information of an image from the resource managing file by using a kernel library. Based on the information collected, the images memory is kept. If the image's memory has been kept normally, the image's data can be automatically loaded by format type using an image loading function. Then the image is generated only using the image's ID through a WIPI image generating function in the desired image output location. In the same method as this, Windows Mobile's image handling structure is converted into a mapping engine.

To generate images using Windows Mobile, there is the method of using a graphic context that uses GDI – a basic API – and the method of using Direct X which uses a graphic acceleration device. In this paper, Direct X was used as the image generating method because this method enhances the contents speed after conversion to Windows Mobile form. The

platform mapping engine was implemented using Direct X, identical to WIPI's image generating library function.

3.3.4. Library Functions (APIs)

Windows Mobile's APIs were used in the implementation process so that they would carry out the same actions as the WIPI library functions. The WIPI library function is defined within the WIPICHEADER file, and since each header is inherited through translation of a translator, the WIPI library function within the source codes translated into the C++ language can be used in the same form as the original functions, thus requiring no additional conversion for use in the target platform. Table 3 is a list of the WIPI library functions supported.

Table 3. WIPI's Library Functions Supported

Category	Supported APIs
Kernel(9)	MC_knlPrintk, MC_knlGetResourceID, MC_knlCalloc, MC_knlGetResource, MC_knlDefTimer, MC_knlSetTimer, MC_knlUnsetTimer, MC_knlSprintk, MC_knlCurrentTime
Graphic(16)	MC_grpGetPixelFromRGB, MC_grpGetPixelFromRGB, MC_grpSetContext, MC_grpFillRect, MC_grpGetScreenFrameBuffer, MC_grpInitContext, MC_grpFlushLcd, MC_grpRepaint, MC_grpDestroyImage, MC_grpDrawImage, MC_grpCreateImage, MC_grpDrawRect, MC_grpPutPixel, MC_grpCreateOffScreenFrameBuffer, MC_grpCopyFrameBuffer, MC_grpDrawImageRegion
Media(5)	MC_mdaClipCreate, MC_mdaClipPutData, MC_mdaPlay, MC_mdaStop, MC_mdaSetVolume
Mathermatics(9)	MC_mathAbs, MC_mathRand, MC_mathSin100, MC_mathCos100, MC_mathTan100, MC_mathArcSin100, MC_mathArcCos100, MC_mathArcTan100, MC_mathSrand

4. Experimental Results and Analysis

The Using the resource converter and platform mapping engine proposed in this paper, a WIPI-to-Windows Mobile game contents converter was designed. Using this, feature phone WIPI game contents were converted into smart phone Windows Mobile game contents and the results were compared, the contents converter's performance was measured and analyzed.

As can be seen in the screens shown in Figure 16, WIPI contents have been converted using the WIPI-to-Windows Mobile contents converter and can be run on Windows Mobile just like it would be run on WIPI. Contents execution speed was measured using FPS (Frame Per Second) as the measure. The converted Windows Mobile Contents (A) in Figure 17 is slower if it has more frames than the original WIPI Contents (A) however if they have a similar number of frames like contents B, there is no difference in execution speed. As you all know from the experiment above, the reason behind such speed differences seems to occur from the speed difference of the WIPI API supported by the platform mapping engine and each individual emulators.

The sizes of contents were compared by comparing the sizes of the emulator execution file. As is visible in Figure 18, the execution file for Windows Mobile contents is smaller than that for WIPI contents. However in the case of Windows Mobile, resource data are separately

included in the memory and if the volume of resource data is included, the size is similar to that of WIPI contents.

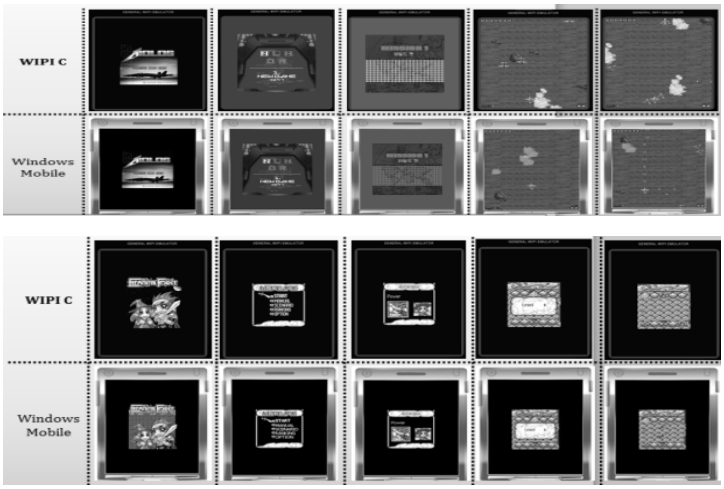

Figure 16. Execution Results of Game Contents A and B

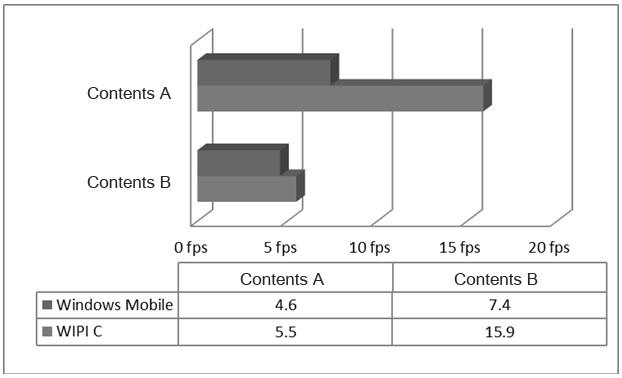

Figure 17. Results of Game Contents Execution Speed

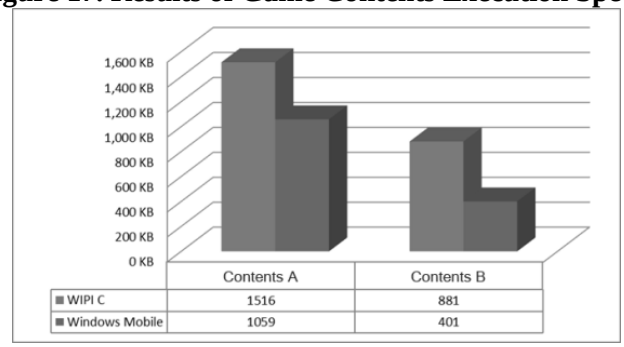

Figure 18. Size Comparison of Game Contents

5. Conclusions

A Recently, the appearance of smart phones caused the mobile contents market to experience high annual growth rates and the killer contents of the mobile market have become smart phone contents. However, the mobile platforms are different according to the mobile communications company or cell phone manufacturer they are associated

with. So to service one mobile game contents, the contents have to be duplicated or converted which consumes additional time and expenses.

The mobile contents converter developed in this paper using a resource converter and a platform mapping engine is one way to solve the converting problems of a mobile game content. Through a converter such as this, the job of converting contents can be carried out automatically within a short period of time. This will shorten the time invested in converting WIPI game contents for feature phones into Windows Mobile game contents for smart phones along with reducing expenses and enhancing productivity.

For future enhancement of mobile game contents converter, a source code translation and APIs provision would become possible for the specific platform and device used. Also, the study will be extended to create contents game converters for the rapidly growing smart phone platforms such as Android, iOS(iPhone), Windows Phone 7, bada and *etc.* by supplementing the converters' systems and functions.

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. 201000236 44).

References

[1] Y. S. Lee, "Design and Implementation of the GNEX C-to-WIPI Java Converter for Automatic Mobile Contents Translation", Journal of Korea Multimedia Society, vol. 13, no. 4, **(2010)**, pp. 609-617.
[2] Y. S. Son, S. M. Oh and Y. S. Lee, "Design and Implementation of the GNEX C-to-Android Java Converter using a Source-Level Contents Translator", Journal of Korea Multimedia Society, vol. 13, no. 7, **(2010)**, pp. 1051-1061.
[3] Y. S. Lee, H. J. Choi and J. S. Kim, "Design and Implementation of the GNEX-to-iPhone Converter for Smart Phone Game Contents", Journal of Korea Multimedia Society, vol. 14, no. 4, **(2011)**, pp. 577-584.
[4] Y. S. Lee and Y. S. Son, "A Platform Mapping Engine for the WIPI-to-Windows Mobile Contents Converter", Multimedia, Computer Graphics and Broadcasting, Springer, CCIS, vol. 262, **(2011)**, pp. 69-78.
[5] Y. S. Lee, "Automatic Mobile Contents Converter for Smart Phone Platforms", Korea Multimedia Society, vol. 15, no. 1, **(2011)**, pp. 54-73.
[6] Y. S. Lee and Y. S. Son, "A Study on the Source Translator of the WIPI-to-Android Mobile Game Converter", Information-an International Interdisciplinary Journal, International Information Institute, vol. 16, no. 1, **(2013)**, pp. 739-746.
[7] Y. S. Lee, J. S. Kim and M. J. Kim, "Development of the Contents Analyzer and the Resource Converter for Automatic Mobile Contents Converter", Journal of Korea Multimedia Society, vol. 14, no. 5, **(2011)**, pp. 681-690.
[8] Y. S. Lee and Y. S. Son, "A Study on the Source Translator for Generating the Android Game Source from the WIPI Game Source", International Journal of Multimedia and Ubiquitous Engineering, SERSC, vol. 7, no. 4, **(2012)**, pp. 95-106.
[9] Y. S. Son and Y. S. Lee, "Automatic UI Generation Technique for Mobile Applications on Touch-Screen based Smart Phones", International Journal of Smart Home, SERSC, vol. 6, no. 4, **(2012)**, pp. 67-80.
[10] Y. S. Son, Y. S. Lee and S. M. Oh, "Design and Implementation of the Compiler with Secure Coding Rules for Developing Secure Mobile Applications in Memory Usages", International Journal of Smart Home, SERSC, vol. 6, no. 4, **(2012)**, pp. 153-168.
[11] H. M. Jang and S. P. Hong, "A Study on the OSMU (One-Source Multi-Use) Management for Smart Devices", International Journal of Smart Home, SERSC, vol. 7, no. 1, **(2013)**, pp. 1-8.
[12] WIPI (Wireless Internet Platform for Interoperability), KWISF (Korea Wireless Internet Standardization Forum), **(2004)**.
[13] Microsoft, Windows Mobile MSDN, http://msdn.microsoft.com/en-us/library/bb158486%28v=MSDN.10%29.aspx, **(2010)**.
[14] S. H. Kim, "Design and Implementation of a Mobile Contents Conversion System based on XML using J2ME MIDP", Master's Thesis, Hannam University, **(2003)**.

[15] Y. S. Kim and D. C. Jang, "A Design for Mobile Contents Converting Using XML Parser Extraction", Journal of Korea Multimedia Society, vol. 6, **(2003)**, pp. 267-276.
[16] S. I. Yun, "Integrated Conversion System for Wired and Wireless Platform based on Mobile Environment", Ph.D Thesis, Hannam University, **(2003)**.
[17] Y. S. Kim and S. Y. Oh, "A Study on Mobile Contents Converting Design of Web Engineering", Journal of Korea Information Processing Society, vol. 12-D, no. 1, **(2005),** pp. 129-134.
[18] Y. J. Lee, "A Method of C Language based Solution Transformation between WIPI and BREW Platform", Master's Thesis, Chungnam National University, **(2007)**.
[19] C. U. Hong, J. H. Jo, H. H. Jo, D. G. Hong and Y. S. Lee, "GVM-to-BREW Translator System for Automatic Translation of Mobile Game Contents", Game Journal of Korea Information Processing Society, vol. 2, no. 1, **(2005)**, pp. 49-64.
[20] Y. S. Lee, "Design and Implementation of the MSIL-to-Bytecode Translator to Execute .NET Programs in JVM platform", Journal of Korea Multimedia Society, vol. 7, no. 7, **(2004)**, pp. 976-984.
[21] Y. S. Lee and S. W. Na, "Java Bytecode-to-.NET MSIL Translator for Construction of Platform Independent Information Systems", Knowledge-Based Intelligent Information & Engineering Systems, LNAI 3215, Springer, vol. 3, **(2004)**, pp. 726-732.
[22] D. Galles, "Modern Compiler Design", Addison-Wesley, **(2007)**.

Authors

YangSun Lee 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, 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, and a President of a Society for the Study of Game at Korea Information Processing Society from 2006. And, he was a Director of Smart Developer Association from 2011-2012. Currently, he is a Professor of Dept. of Computer Engineering, Seokyeong University, Seoul, Korea. His research areas include smart cross platforms, automatic mobile content converter system, programming languages, compiler construction, and mobile/embedded systems.

Yunsik Son 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, HCI, programming languages, compiler construction, and mobile/embedded systems.