A Study on the Source Translator for Generating the Android Game Source from the WIPI Game Source

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, *Corresponding Author: sonbug@dongguk.edu*

Abstract

In the mobile market, the appearance of various smart phone platforms such as Android, iOS(iPhone), Bada and Window Phone has led to game developers to repeatedly develop game contents to suit the different mobile communication companies' platforms in order to service mobile game contents. Furthermore, to use each of the game contents developed on the existing feature phone platform, they need to be recreated based on the smart phone platform. Consequently, large amounts of time and expenses are being used to analyze and convert the sources and resources of the mobile game contents for use on the smart phone platform. Rather than creating new game content, developers are investing twice the amount of time and money required to develop a game for one platform in making existing game content available on other platforms. In this paper, in order to resolve such problems the source translator of the automatic mobile game contents converter system will be implemented in order to convert the WIPI(Wireless Internet Platform for Interoperability) game contents for use on an Android smart platform. This system has enabled contents to be transferred into smart 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, game contents developed for use on feature phones can also be converted and used on smart platforms to increase reusability of game contents and also new game contents creation processes can heighten productivity to consequently provide a more variety of game contents to users.

Keywords: *Source Translator, WIPI-to-Android Mobile Game Converter, WIPI(Wireless Internet Platform for Interoperability), Android, Mobile Platform, Smart Platform, Automatic Mobile Game Contents Converter*

1. Introduction

In the mobile market, the appearance of various smart phone platforms such as Android, iOS(iPhone), Bada and Window Phone has led to game developers to repeatedly develop game contents to suit the different mobile communication companies' platforms in order to service mobile game contents. Furthermore, to use each of the game contents developed on the existing feature phone platform, they need to be recreated based on the smart phone platform. Consequently, large amounts of time and expenses are being used to analyze and convert the sources and resources of the mobile game contents for use on the smart phone platform. Rather than creating new game content, developers are investing twice the amount

of time and money required to develop a game for one platform in making existing game content available on other platforms [1-8].

This research aims to solve these problems by inventing a resource translator for the WIPI-to-Android mobile game contents automatic converter which automatically converts game contents from the existing feature phone platform WIPI to the smart phone platform Android. The WIPI-to-Android game converter consists of a content analyzer, resource converter, source translator, and platform mapping engine. The source translator receives the WIPI Java source code produced by the content analyzer and translates it into Android Java source code that is semantically equivalent and fulfils the same function on the Android platform.

By automatically converting the existing mobile game con-tents used in the WIPI Java feature phone platform to game contents for use in the Android platform, existing game contents can be transplanted into a different platform within a short period of time. As a result, the reusability will be increased, while the labor, time and costs involved in servicing same contents to different mobile communications companies will be saved. A wider range of contents provision to users can be expected as well.

2. Related Studies

2.1. WIPI

WIPI (Wireless Internet Platform for Interoperability) is legislated by KWISF(Korea Wireless Internet Standardization Forum) and a standardized standard chosen by KTTA(Korea Telecommunications Technology Association) as an application program execution environment for mobile communication platforms[9]. 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 depicts the system structure of a WIPI platform.

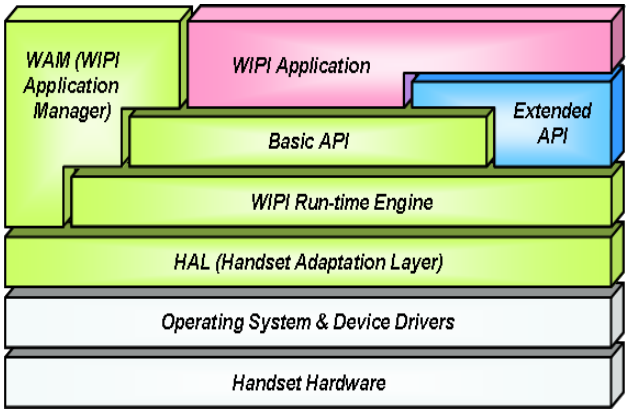

Figure 1. WIPI Platform's System Configuration

WIPI supports the C language and the Java language which were the languages used when developing application programs. 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(Application Programming Inter-face). 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 [9, 10].

2.2. Android

The Android platform, developed by Google is an optimized platform for mobile devices with a perfect combination of an operating system, middleware, and application programs. The Android platform opted the open source policy and consists of the linux kernel, library, run-time, application framework and applications [11]. Figure 2 depicts the Android platform's hierarchical structure and components.

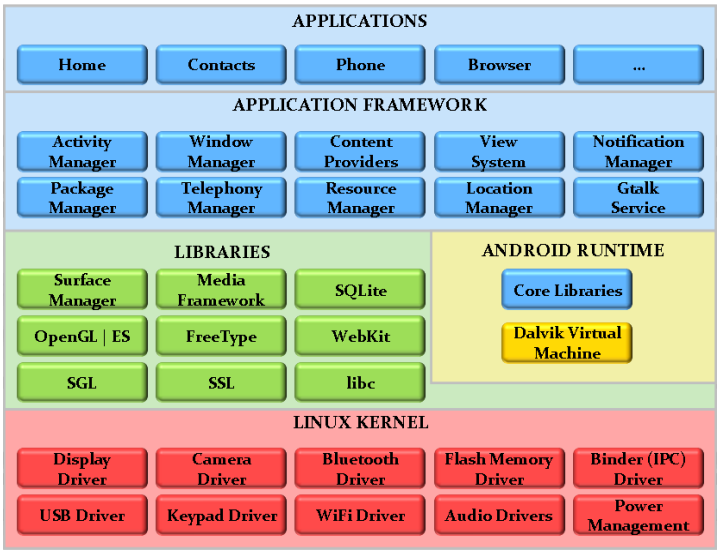

Figure 2. Android Platform's System Configuration

The linux kernel uses core system services such as linux version 2.6, which includes security, memory management, process management, network stack and driver model. The kernel functions as an abstraction layer in between hardware and software. The library consists of C and C++ and provides C system library, media library, 3D library and more. Application framework is a package component consisting of Java and applications can be created using the packages of this framework.

All application programs of the Android platform are created using Java and the applications created are converted into class files through the Java compiler. The class files created are converted one more time into Dalvik Executable Files before being executed, and after conversion they are executed using the Dalvik virtual machine. DEX files are optimized formats for effective mapping, in between storage space and memory. The Dalvik virtual machine has the characteristics of being optimized to limited memory as it is a virtual machine based on register [11-13].

2.2. Existing Mobile Game Converters

To date, despite the very active mobile market, there has been a lack of research on mobile content converters, so there are few examples to which we can refer. Furthermore, converters

for existing content generally only allow conversion of content having a similar programming language environment or do not allow automatic con-version at all. The reality is that programmers must convert content by hand.

An existing mobile game content converter using XML has attempted to convert Java content [14-17]. In addition, the functions of the API used in the source code to be converted were imitated and redefined using wrapper functions. Therefore, there is no need to convert the source code if the same functions are used. The mutual conversion of BREW C and WIPI C [18] and the conversion GVM C into BREW C [19] have been examined; however, these studies were flawed because the source code was not automatically converted, so users had to intervene and convert it manually.

On the other hand, studies of automatic conversion of mobile game content using a compiler writing system [20, 21] have been attempted. A method of increasing the reusability of game content and enhancing productivity by converting the mobile C content of the GVM platform into WIPI C, Java, or MIDP Java has been suggested [1]. In addition, other studies are underway to convert existing mobile game content for use in the growing smart phone market for operating systems such as Android and iOS for example, the WIPI-to-iOS converter, WIPI-to-Android source converter, GNEX-to-iOS converter, GNEX-to-Android converter, Android-to-iOS converter, and iOS-to-Android converter system.

3. The Source Translator of the WIPI-to-Android Game Converter

WIPI-to-Android automatic mobile game contents converter receives WIPI game contents in source form, which it converts into the game contents source form that is run on the Android 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 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 [1-8]. Figure 3 shows a model of the WIPI-to-Android mobile game contents automatic converter system.

The WIPI-to-Android converter receives WIPI Java game contents as an input. It consists of a contents analyzer which classifies resources and source codes, a resource converter which converts the WIPI Java resource format into a format that is usable on Android, a source translator which translates WIPI Java source codes into Android Java source codes, an environment which enable the equivalent display and execution of WIPI Java contents on the Android platform and a platform mapping engine which provides APIs.

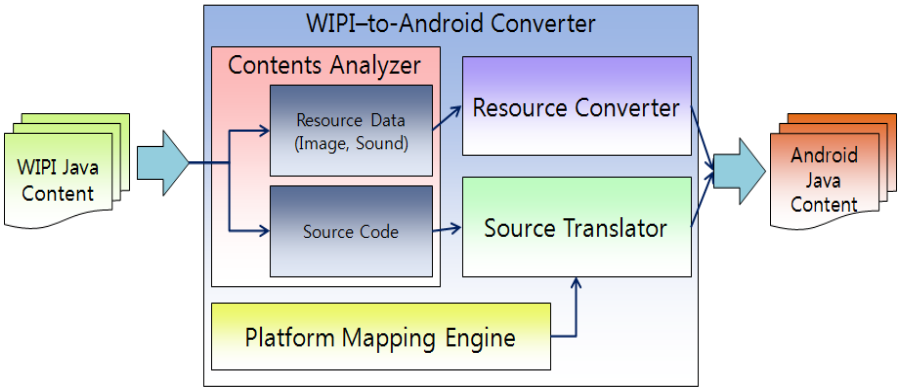

Figure 3. WIPI-to-Android Game Contents Converter

3.1. An Outline of the Source Translator

The source translator receives the WIPI Java source codes that are output by the contents analyzer and translates them into Android Java source codes which are semantically equivalent and execute the execute the same actions as the WIPI Java source codes. Because WIPI Java and Android are both Java based platforms, the characteristics of the language are the same. However, there exists 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. Figure 4 is a depiction of the source translator.

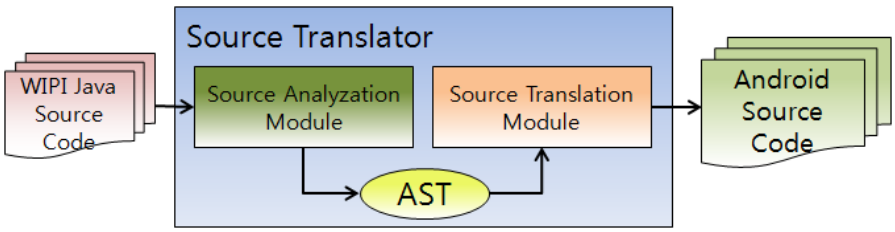

Figure 4. The Source Translator

Source translators can be largely divided into the source analysis module and source translation module. The source analysis modules receive WIPI Java 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 Android Java source codes which are semantically equivalent to the WIPI Java source codes.

3.2. Source Analysis Module

The source analysis module is the first component of a source translator. It receives WIPI Java source code inputs, carries out lexical and syntax analysis, outputs sentence structure as AST and delivers the AST to the source translation module. Figure 5 shows the source analysis module.

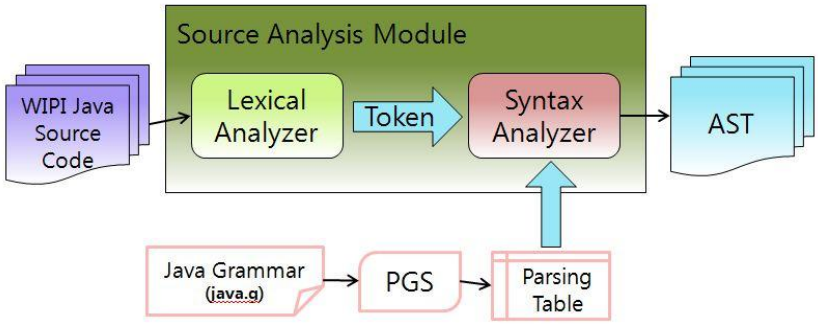

Figure 5. Source Analysis Module

The source analysis module can be largely divided into the lexical analyzer and the syntax analyzer. The lexical analyzer classifies WIPI Java 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. Output Results of the Tokens

Source code	Token

public class ADD
{
  public static int sum;
  public static void
  main(String args[]) {
  int sum;
  sum = 46+12;
  }
}
	Token :::: 1 public : (86, 0)
	Token :::: 1 class : (63, 0)
	Token :::: 1 ADD : (9, ADD)
	Token :::: 2 { : (102, 0)
	Token :::: 3 public : (86, 0)
	Token :::: 3 static : (89, 0)
	Token :::: 3 int : (78, 0)
	Token :::: 3 sum : (9, sum)
	Token :::: 3 ; : (36, 0)
	Token :::: 5 public : (86, 0)
	Token :::: 5 static : (89, 0)
	Token :::: 5 void : (99, 0)
	Token :::: 5 main : (9, main)
	Token :::: 5 (: (21, 0)
	Token :::: 5 String : (9, String)
	Token :::: 5 args : (9, args)
	Token :::: 5 [: (50, 0)
	Token :::: 5] : (51, 0)
	Token :::: 5) : (22, 0)
	Token :::: 6 { : (102, 0)
	Token :::: 7 int : (78, 0)
	Token :::: 7 sum : (9, sum)
	Token :::: 7 ; : (36, 0)
	Token :::: 8 sum : (9, sum)
	Token :::: 8 = : (41, 0)
	Token :::: 8 46 : (15, 46)
	Token :::: 8 + : (25, 0)
	Token :::: 8 12 : (15, 12)
	Token :::: 8 ; : (36, 0)
	Token :::: 9 } : (106, 0)
	Token :::: 11 } : (106, 0)
	Token :::: 0 EOF : (54, 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 6 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.

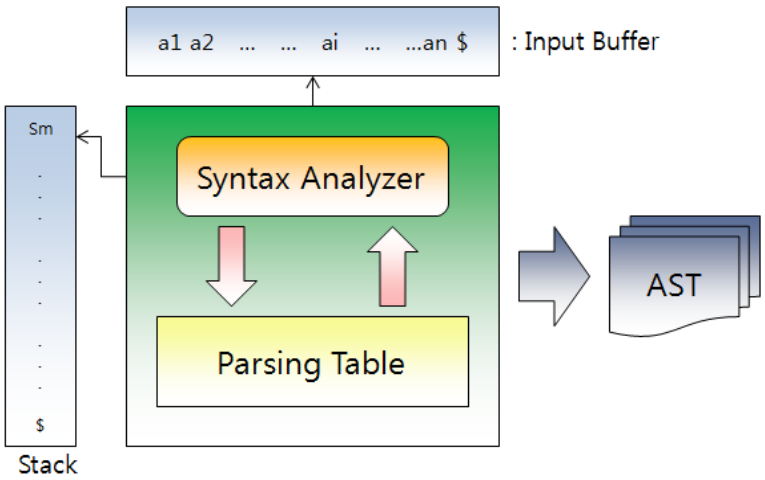

Figure 6. Shift-Reduce Syntax Analyzer

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 theerror action shows that a sentence is wrong because it cannot be revealed in its current symbol state. The shift action actually increases the pointer by one so that it points to the next symbol, while the currently evaluated symbol is pushed onto the stack. The shift action is performed until the handle appears at the top of the stack. When the handle is found, it is reduced according to the relevant creation rules, and then the process is repeated until the first syntax symbol is reached. Table 2 shows the AST information created by the syntax analyzer.

3.3. 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, Android, 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 an Android 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 Android source code.

Table 2. AST Information

Source code	AST

public class ADD {
  public static int sum;
  public static void
  main(String args[]) {
  int sum;
  sum = 46+12;
  }
}
	Nonterminal: PROGRAM
	  Nonterminal: CLASS_DCL
	  Nonterminal: PUBLIC
	  Terminal: ADD
	  Nonterminal: CLASS_BODY
	  Nonterminal: FIELD_DCL
	  Nonterminal: PUBLIC
	  Nonterminal: STATIC
	  Nonterminal: DCL_SPEC
	  Nonterminal: INT_TYPE
	  Nonterminal: VAR_ITEM
	  Nonterminal: SIMPLE_VAR
	  Terminal: sum
	  Nonterminal: METHOD_DCL
	  Nonterminal: METHOD_HEAD
	  Nonterminal: PUBLIC
	  Nonterminal: STATIC
	  Nonterminal: METHOD_ITEM
	  Terminal: main
	  Nonterminal: PARAM_DCL_LIST
	  Nonterminal: PARAM_DCL
	  Nonterminal: DCL_SPEC

Figure 7 shows the execution process of the source translation module.

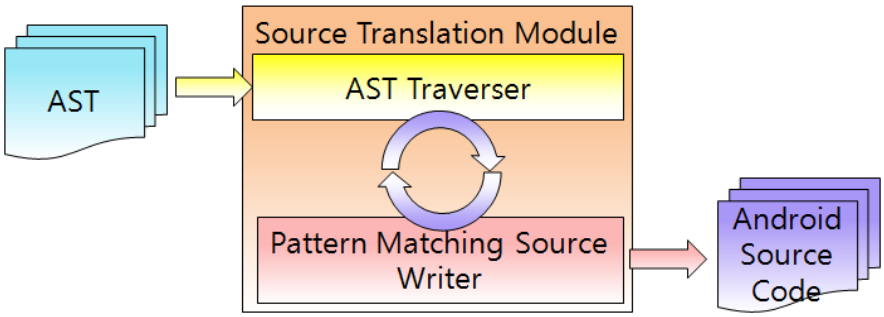

Figure 7. Source Translation Module

Figure 8 shows the screen of the source translator.

The list on the left is the list of WIPI Java files to be translated and the list on the right is a list of the converted Android files. When the conversion button is pressed, the source translator automatically converts the sources.

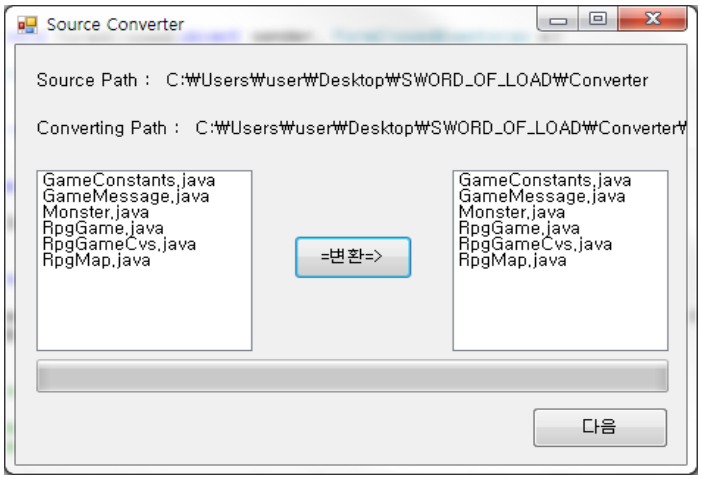

Figure 8. Screen of the Source Translator

Figure 9 shows an example of searching the AST of a WIPI Java program created by the syntax analyzer and converting it into an Android Java program.

AST	WIPI - class

Nonterminal: CLASS_DCL
Terminal: MyView
Nonterminal: EXTENDS
Nonterminal: CLASS_INTERFACE_TYPE
Nonterminal: SIMPLE_NAME
Terminal: GNEX
	class MyView extends GNEX {
	:
	}

	Android - class

	class MyView extends GNEX {
	:
	}

AST	WIPI - member variable
Nonterminal: FIELD_DCL Nonterminal: PUBLIC Nonterminal: STATIC Nonterminal: FINAL Nonterminal: DCL_SPEC Nonterminal: INT_TYPE Nonterminal: VAR_ITEM Nonterminal: SIMPLE_VAR Terminal: N_TIE Terminal: 5	public static final int N_TIE=5;
	Android - member variable
	public static final int N_TIE=5;

AST	WIPI - member function

Nonterminal: METHOD_DCL
Nonterminal: METHOD_HEAD
Nonterminal: PUBLIC
Nonterminal: DCL_SPEC
Nonterminal: INT_TYPE
Nonterminal: METHOD_ITEM
Terminal: Add
Nonterminal: PARAM_DCL_LIST
Nonterminal: PARAM_DCL
Nonterminal: DCL_SPEC
Nonterminal: INT_TYPE
Nonterminal: SIMPLE_VAR
Terminal: a
Nonterminal: PARAM_DCL
Nonterminal: DCL_SPEC
Nonterminal: INT_TYPE
Nonterminal: SIMPLE_VAR
Terminal: b
Nonterminal: METHOD_BODY
Nonterminal: BLOCK (start : 13 end : 15)
Nonterminal: RETURN_ST
Nonterminal: ADD_OP
Nonterminal: SIMPLE_NAME
Terminal: a
Nonterminal: SIMPLE_NAME
Terminal: b
	public int Add(int a, int b)
	{
	Returna+b;
	}

	Android - member function

	public int Add(int a, int b)
	{
	Returna+b;
	}

AST	WIPI Source-array
Nonterminal: FIELD_DCL Nonterminal: DCL_SPEC Nonterminal: ARRAY_TYPE Nonterminal: INT_TYPE Nonterminal: VAR_ITEM Nonterminal: SIMPLE_VAR Terminal: pal2_3 Nonterminal: NEW_ARRAY Nonterminal: INT_TYPE Nonterminal: DIM_EXPRESS Terminal: 17	int[] pal2_0 = new int[17];
	Android Source-array
	int[] pal2_0 = new int[17];

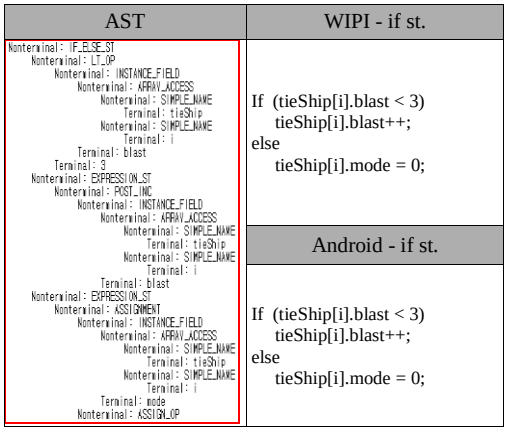

Figure 9. AST and Source Translation

4. Experimental Results and Analysis

In this study, the WIPI-to-Android mobile game contents automatic converter system was used to automatically convert mobile game contents from the feature phone WIPI platform to the smart phone Android platform. The results of the conversion were then compared and the contents converter's performance was measured and analyzed. Each of the platform's emulators used to execute the contents are as in Table 3.

Table 3. Platform's Emulator

Platform	Emulator	Method
WIPI	SKT WIPI Emulator	Native
Android	Android 2.2 Emulator	Native

Figure 10 and Figure 11 show the comparison between the running of games "Aiolos", "Elemental Force" and etc on each emulator to test the overall performance of the game contents which were converted in the WIPI-to-Android converter. It can be confirmed that graphics, image output, sound output and other actions were all executed equivalently.

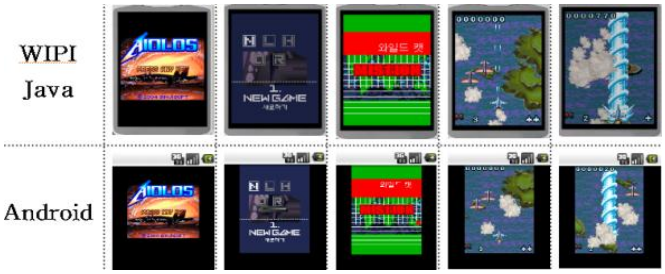

Figure 10. Game Aiolos's Execution Result

Figure 11. Game Elemental Force's Execution Result

5. Conclusions

With the recent appearance of smart phones, the mobile game market is experiencing high growth rates each year, and game content has become killer content in the mobile market. However, differences in mobile platforms have required repeated development or conversion of mobile game content for use on multiple platforms.

The source translator of the WIPI-to-Android mobile game converter created in this research undergo automatic conversion of existing WIPI game contents into the smart phone platform's Android game contents within a short period of time. Therefore increasing the reusability of existing game contents and enabling users to be provided with a more diverse range of contents. Furthermore, the time and money involved in the development and conversion process of converting game contents of feature phone platforms for use in the smart phone platform can be reduced extensively, resulting in increased productivity. Such reduced costs and time can then be invested in developing new game contents, accelerating the development of mobile game contents and contribute towards strengthening the productivity of the mobile industry.

In the future, further research on increasing game contents' execution performance will be carried out, in addition research supplementing and expanding the game contents converter system so that game contents can be run on the various smart phone platforms which are spreading such as iPhone, Android, Windows Phone, bada and more.

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.20100023644).

References

[1] Y. 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. Son, S. Oh and Y. 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. Lee, H. Choi and J. 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. Lee, J. Kim and M. 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.
[5] Y. Lee and Y. 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.

[6] Y. Lee and Y. Son, "A Study on the WIPI-to-Windows Mobile Game Contents Converter using a Resource Converter and a Platform Mapping Engine", Information-an International Interdisciplinary Journal, International Information Institute, vol.16, to be published, **(2013)**.
[7] Y. Son and Y. Lee, "Design and Implementation of an Objective-C Compiler for the Virtual Machine on Smart Phone", Multimedia, Computer Graphics and Broadcasting, Springer, CCIS, vol. 262, **(2011)**, pp. 52-59.
[8] Y. Lee, "Automatic Mobile Contents Converter for Smart Phone Platforms", In Korea Multi-media Society, vol. 15, no. 1, **(2011)**, pp. 54-73.
[9] WIPI (Wireless Internet Platform for Interoperability), KWISF (Korea Wireless Internet Standardization Forum), **(2004)**.
[10] I. G. Kim, K. Kwon and T. T. You, "WIPI Mobile Game Programming", Daelim, **(2005)**.
[11] Goole, Android, http://code.google.com/intl/ko/android/.
[12] S. Hashimi, S. Komatineni and D. MacLean, "Pro Android 3", Apress, **(2011)**.
[13] D. Galles, "Modern Compiler Design", Addison-Wesley, **(2007)**.
[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. 129, **(2005)**.
[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.

Authors

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 2004-2005, 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 2011-2012. 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.