Printed in Japan

ISSN 1343-4500 (print)
ISSN 1344-8994 (electronic)

# INFORMATION

## An International Interdisciplinary Journal

Volume 20 Number 2(A), February 2017

Published by International Information Institute
www.information-iii.org

***Medicine and Life Sciences***

INFORMATION
Volume 20, Number 2(A), pp.963-968
ISSN 1343-4500 eISSN 1344-8994


# xCodeParser based on Abstract Syntax Tree Metamodel (ASTM) for SW Visualization

Hyun Seung Son* and R. Young Chul Kim**

** SE Lab., Dept. of Computer and Information Communication, Hongik University*
*Sejong, 30016, Korea*
*E-mail: {son*, bob**}@selab.hongik.ac.kr*

**Abstract**

The software visualization is a process to recover the architecture from program code, which changes representation to graph from text through reverse engineering. To perform the process, we need to have three tools as the parser, database, and visualizer. The parser in these tools performs a most important role due to syntax and semantic analysis of a program code such as C, C++, or Java. The parser generates Abstract Syntax Tree (AST) that is preparation to analyze the statements and expressions in functions and classes of code. But the AST is dependent on the parser. The existing AST are not compatible with other. If we achieve SW visualization using the existing specific parser, we must implement a visualizer separately for each parser or program language. To solve this problem, OMG proposes the standard named Abstract Syntax Tree Metamodel (ASTM). This means to define metamodel of the AST, which does not depend on any parser or program language. Therefore, we can represent several languages with just an ASTM. This paper introduce to implement the parser named *xCodeParser* that uses ASTM for multi-language. We show the design and implementation of *xCodeParser*. Then we present a case study to suggest a whole procedure for SW visualization with the ASTM.

**Key Words**: Abstract Syntax Tree Metamodel (ASTM), Metamodel, Parser, Software Visualization, Reverse Engineering

## 1. Introduction

Korea's most small sized companies and ventures used to develop the software code without any design due to lack of time and cost. They just want to release SW product quickly, but it may spend more cost at the maintenance stage as a result, and low quality of SW product is made. Therefore, to make high quality software, the companies need method to show inside of the developing code of the complex software.

The software visualization is able to recover the architecture from a program code through reverse engineering [1]. For the SW visualization, NIPA Software Engineering Center support the tool-chains based on open source software such as Source Navigator [2], Graphviz [3], SQLite [4], Jenkins [5], and etc. But it is require diverse tools such as parser, database, and visualizer basically [6]. To recover an architecture in NIPA's software visualization, they use Source Navigator (SN). The original goal of the SN shows call graphs

from code, and the tools generate databases files as a result of parsing the code between the processes. The idea of NIPA use the databases file without general parser. But, the idea have some weakness as follows: 1) it not show inside of function, 2) it not read comments in program code, 3) it cannot be more than work to except for the provided features. Therefore, we need the general parser to generate the Abstract Syntax Tree (AST) for using advanced reverse engineering techniques. Generally, the parser generates an AST to analyze the statements and expressions in functions and classes of code. But the existing ASTs are not compatible with other AST that dependent on the specific parser. In this case, it can be faced with difficulties to waste cost and time when you use other languages, because it should be developed separately for each language.

The OMG's Abstract Syntax Tree Metamodel (ASTM) [7] is defined by industry companies to solve the problems of the previous. ASTM is metamodel of abstract syntax tree that the main purpose easily exchanges the metadata for program code such as C, C++, C#, Java, Ada, VB/.Net, COBOL, FORTRAN, Jovial, and so on. OMG's ASTM has defined and complicated with 193 elements of metamodel but just specifications without any implementation. We try use the ASTM in previous researches [8].

In this paper, we introduce the parser for multi-programing language using ASTM named *xCodeParser* and show the design and implementation of *xCodeParser*. Then we present case study to suggest a whole procedure for SW visualization with the ASTM. This paper is organized as follows. Chapter 2 mentions the procedure of SW Visualization. Chapter 3 describes a case study to show an example of Six-leg Robot Simulator using SW Visualization. Last chapter mentions the conclusion and future work.

## 2. A Procedure of SW Visualization

To develop SW visualization tools basically, it is required the parser and visualizer. The figure 1 shows a whole structure of *xCodeParser* for SW visualization.

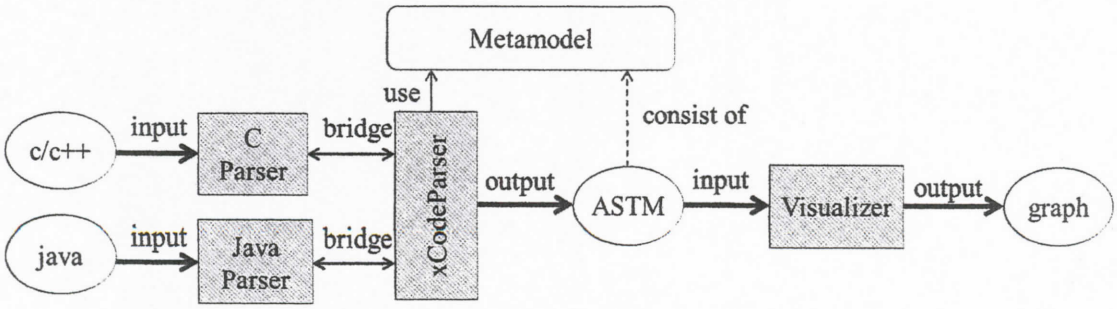



Fig. 1. The strategy of *xCodeParser* for SW visualization

In the parser part, ASTM is generated via parser from a program code such as C, C++, or Java. In the visualizer part, the graph is generated from the ASTM. Through this process, we

can reverse the architecture from the program code. Our *xCodeParser* reuse the existing parser as C/C++ Development Tooling (CDT) [9] and Java Development Tools (JDT) [10] because better than to develop a new parser. The CDT is a tool in Eclipse platform to develop C/C++ application. It supports to create the project, to build the program, to edit the C/C++ code, to analyze the static code, and to debug & refactor functions. The JDT is a tool to develop Java application. It supports the same function like the CDT.

## 2.1 A design of Specialized ASTM

The ASTM of OMG's standard is the metamodel of abstract syntax tree (AST). The main goal of ASTM is easily able to exchange the metadata of a detailed software structure between metadata repositories in the software development, the tools for software modernization, platform, or the heterogeneous environment distributed. Especially, The ASTM defined a specification about modeling elements to represent the AST that can share between the various tools from other vendors. Also, the ASTM can represent a number of programming languages such as C, C++, C#, Java, Ada, VB/.Net, COBOL, FORTRAN, Jovial, and etc.

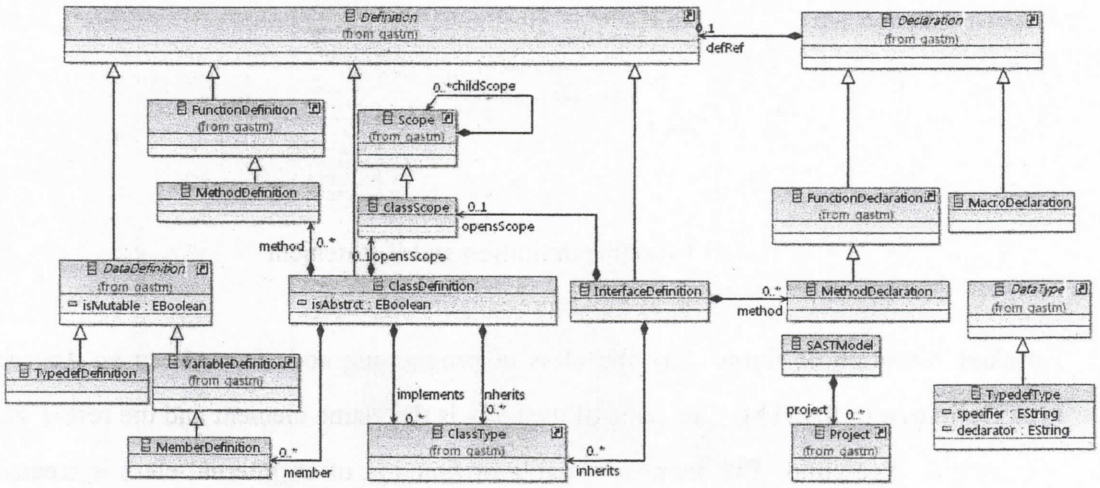
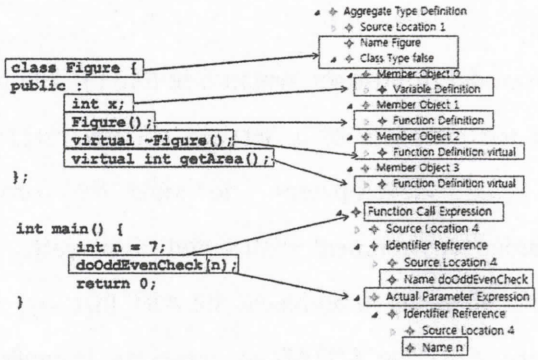



Fig. 2. A design of Specialized ASTM (SASTM)

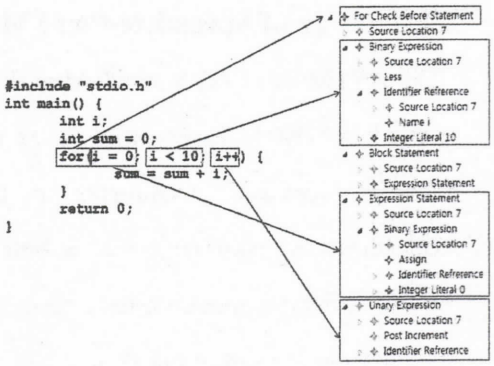
But, The OMG's ASTM just have a specification of metamodel that do not implement and the metamodel structures of complex form is made. We redesign the specialized ASTM (SASTM) to extend metamodel of the existing ASTM. The SASTM to accept the standard document were implemented all of the 193 metamodel structures and it was added 10 structures necessary for the SW Visualization as shown in figure 2. The top model of SASTM is a SASTModel. The SASTModel have multiple project that below added all code syntax. The SASTM include the new type such as Class Definition, Typedef Type, Class Scope, Interface Definition because ASTM has not it.

## 2.2 A design of Specialized ASTM

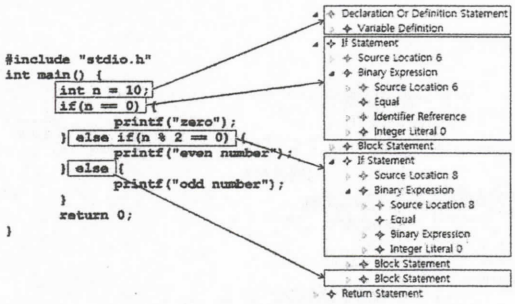
The *xCodeParser* can generate the SASTM files from input code of programing language via code analyzer. The generated SASTM metamodel based XMI file can be interoperable through the existing tools based on Model Driven Development (MDA). The figure 3 address mapping relationship between code and SASTM of class & variable definition, function call, and if & for statement.




(a) class definition and function call statement




(b) for statement




(c) variable definition and if statement

Fig. 3. The representation of code

In class definition of figure 3(a), the class of programing code is made to an Aggregate Type Definition of SASTM. The name of the class is the Name element and the rest is stored in the Class Type child. The member variable or function of an internal class is created to Member Object and the child variable or function is made to each Variable Definition or Function Definition. In figure 3(b), for statement is stored divided into three parts: 1) Initbody, 2) Condition, and 3) IterationBody. The "i=0", "i < 10" , or "i++" of for statement in code is represented to each Binary Expression of Initbody, Binary Expression of Condition, or Unary Expression of IterationBody. In figure 3(c), the variable statement is change to a Variable Definition of Declaration or Definition Statement child unlike the variable in class definition above. The condition part of if statement represent various expressions according to the form of code. The equal relationship is made to the Binary Expression, and the else if and else statement is changed to sub-structure of first if statement in Block Statement.

## 4. A Case Study

To test the xCodeParser, we use the code of Six-leg Robot Simulator (SRS) [5] that is developed by ourselves. The SRS is simulator to develop for easily controlling the action of multi-jointed robot as shown figure 4. This simulator using 3D rendering is developed by C++ based on Microsoft foundation class (MFC), Open Dynamic Engine (ODE).

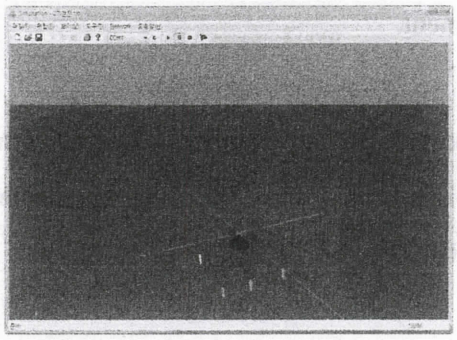
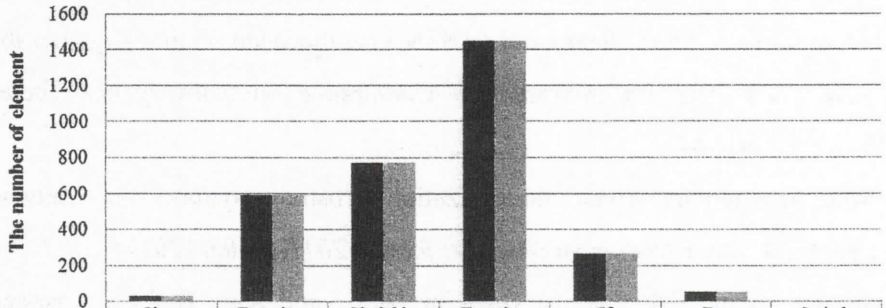


Fig. 4. The execution result of Six-leg Robot Simulator

To validate the transformation of *xCodePaser* exactly, we perform the translation from 45 header and source code files in SRS. As shown figure 5, we compare the number of each elements (such as class, function, variable, function call, if, for, switch) and the number of each elements of generated ASTM files from *xCodePasrer*. From the compared result, we can see correspondence exactly and our *xCodeParser* is validated to be able to transform the ASTM from program code.




| | Class Definition | Function Definition | Variable Definition | Function Call | If Statement | For Statement | Switch Statement |
|---|---|---|---|---|---|---|---|
| Program Code | 29 | 592 | 772 | 1449 | 261 | 53 | 3 |
| xCodeParser | 29 | 592 | 772 | 1449 | 261 | 53 | 3 |

Fig. 5. The comparison of the number between program code and xCodeParser

## 5. Conclusions

Using the existing analyzer tools, if we need the functions don't support in tools, it have various problems. Therefore, we require the Abstract Syntax Tree (AST) to correspond the one-to-one code for software visualization. But, the existing AST have a problem to develop new tools when using other AST, as the each program code or the compiler is different.

On the other hand, the Abstract Syntax Tree Metamodel (ASTM) is useful to convert from

the various program codes which is good for interoperability. But it has a complex structure of 193 elements with just specifications. We proposed the parser for multi-programing language named *xCodeParser* based on ASTM, showed the specification of SASTM to extend the ASTM, and implemented the *xCodeParser*. Also, to apply a case study, we validated the xCodeParser. Our future study will extend the existing *xCodeParser* for high quality software development.

## 6. Acknowledgments

This work was supported by the Human Resource Training Program for Regional Innovation and Creativity through the Ministry of Education and National Research Foundation of Korea (NRF-2015H1C1A1035548).

## References

[1] Lee, S.E. and et al., SW development quality management manual (SW Visualization). *National IT Industry Promotion Agency (NIPA)*, (2013).

[2] Source Navigator, http://sourcenav.sourceforge.net/

[3] Graphviz, http://www.graphviz.org/

[4] SQLite, https://sqlite.org/

[5] Jenkins, https://jenkins.io/

[6] Son, H.S., Moon, S.Y., Kim, R.Y.C., and Lee, S.E., Replacing Source navigator with Abstract Syntax Tree Metamodel (ASTM) on the open source oriented tool chains SW Visualization. *The 5th International Conference on Convergence Technology (ICCT 2015)*, pp. 366-367.

[7] OMG, Architecture-driven Modernization: Abstract Syntax Tree Metamodel (ASTM) Version 1.0. *OMG Document Number: formal/2011-01-05*, (2011).

[8] Son, H.S., Kim, Y.S., Park, Y.B., Kim, W.Y., and Kim, R.Y.C., Abstract Syntax Tree Metamodel for SW Visualization, *International Conference on Convergence Technology (ICCT 2014)*, pp. 157-158.

[9] CDT, http://www.eclipse.org/cdt/

[10] JDT, http://www.eclipse.org/jdt/

**Corresponding author: Prof. R. Young Chul Kim

Department of Computer and Information Communication,

Hongik University Sejong Campus,

2639, Sejong-ro, Jochiwon, Sejong, 30016, Korea

E-mail: bob@hongik.ac.kr