2018 International Conference On Business Innovation (ICOBI), 25-26 August 2018, NSBM, Colombo, Sri Lanka

PseudoJ: A Pseudo-code interpreter for transforming Pseudo-code into JAVA

Tharindu Amarasingha
NSBM Green University Town
Pitipana, Homagama, Sri Lanka
t.amarasingha@outlook.com

Rasika Ranaweera
NSBM Green University Town
Pitipana, Homagama, Sri Lanka
ranaweera.r@nsbm.lk

Abstract*—PseudoJ is a prototyping solution, which is aimed at transforming Hand-written pseudo-codes into corresponding JAVA syntaxes. The solution will speed-up the agile development process and makes it easier to prototype a software development. Users of the solution are presented with options to create new workspaces (new projects), create new PSJ (pseudo-code) files and start prototyping. Under a workspace, Major Input and Output panels are created which respectively are a Pseudo-code editor and a JAVA code editor. The solution takes inputs from both editors and generates / strengthen the JAVA code as the output. The transformation is live and users experience code generation at the same time they input while the core process is hidden from the user. PseudoJ at the core, uses a trained-data file (configuration file / a mapping) in order to predict JAVA syntaxes for a given pseudo-code statement. A trained-data file is needed because the code transformation process involves a lot of guesswork. At the heart of the program, Abstract Syntax Tree (AST) of JAVA is used to process JAVA syntax. The core-process is visualized and demonstrated. Regardless of the major input and output options, a mobile input solution is also implemented in order to capture hand-written text from a user, which makes prototyping even easier. The mobile solution uses OCR techniques and parses hand-written text into a digital pseudo-code input. PseudoJ is implement as a plugin for the Eclipse IDE. The goal is to take advantage of built-in code refactoring, compiler and debugger. Utilization and features are compared and contrasted with similar approaches with the demonstrations.

Keywords*—pseudo-code, abstract syntax tree, prototyping, code transformation

I. INTRODUCTION

The PseudoJ project is aimed at generating corresponding JAVA syntaxes (JAVA program) for non-standard Pseudo-codes which involves a lot of guesswork and higher complexity for the project core. Despite that, the PseudoJ core which acts like an interpreter still requires some optimization and it is expected to optimize and change the Interpreter time to time with future versions of the product or depend on the 'user prompt and self-learn method'. The core is finalized in a way that it could be self-optimized; with the usage and help of client, the interpreter optimizes its rules file (configuration file).
The infrastructure is consisting of two major components.

A. Pseudo-code Collectors

The pseudo-code collectors are used to imports Pseudo-code into the code editor for the purpose of parsing. An inbuilt component existing in the main system may function as a collector or any outside programs can also be function as collectors to the main program. According to the current implementation three main collectors are defined. Two of them are inbuilt functionalities, which already resides in the Main program and one functionality is an outside program.

B. Pseudo-code Parser

The parser receives collected Pseudo-code inputs from the Collectors and further analyses the Pseudo-code. Parser consists of a few major components, which are used to filter, process and standardize (brings the Pseudo-code into a standard format which the Parser could understand). The ultimate purpose of the Parser is to predict JAVA syntaxes from the given Pseudo-code. On behalf of that, Pseudo-code parser maintains a trained data file called (rule file) and its' intended job is to optimize the Pseudo-code filtering process (Standardisation process). The final output can be exported to the code editors. The Pseudo-code parser is an independent unit of functionality, which serves as a service. Therefore, as an opinion, the Parser component could be implemented in a separate Web-Service, which enables to have different clients in different platforms and implements more Collectors.

II. LITERATURE REVIEW

A. Abstract Syntax Tree (AST)

According to (Eclipse, 2018), the Abstract Syntax Tree allows modifying a tree model and reflecting these modifications in the source code. The author further specifies that the Abstract Syntax Tree is comparable to the Document Object Model (DOM) in XML. Abstract Syntax Tree is a concept, which is, standardize for syntax organization and mapping. Many frameworks for different platforms are implemented using the AST concept. The current research implements the Abstract Syntax Tree framework available in Eclipse, which is the base for many powerful tools available in Eclipse IDE (Eclipse, 2018). Eclipse IDE provides AST support for JAVA, PHP and many other programming languages. The below diagram demonstrates the PHP flavour of the AST in Eclipse.

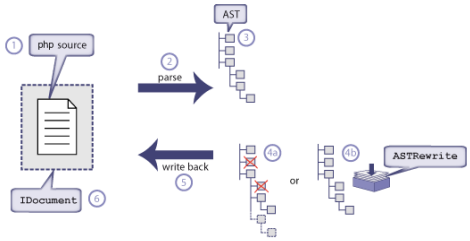

Fig. 1. The AST workflow; Source: (Eclipse, 2018)

AST does not hold pure syntaxes, instead components (Root Objects) related to the syntax component is held. AST is designed (pre-defined) to organise syntax components in a standard way. For an example a While Loop is stored at a parent node and the condition and assignments related to the While loop are stored at child nodes. See below diagram for an AST representation of a While Loop.

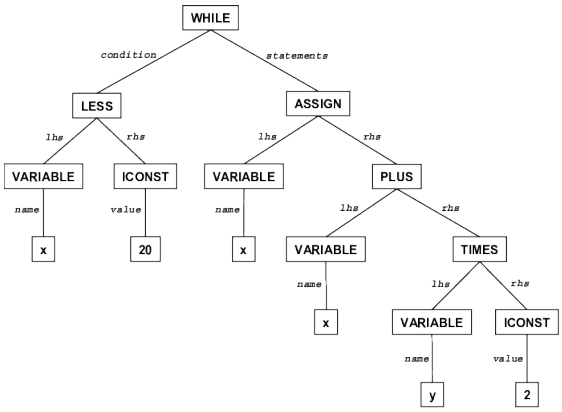

Fig. 2. AST for a While Loop; Source: (Fritzson, 2009)

AST parsers, Rewriters and many forms of modification methods are available in many platforms to modify their AST. Most of the platforms provide support for compilation of AST. Furthermore, there are many complex tools and programs are developed based on AST since manipulation of codes (Syntaxes) and detection of code patterns/ code changes could be implemented using AST. A few examples for such applications are Homology Detection systems, Source code plagiarism detection engines, Faster AST interpreters, Clone detection systems, Automated Quality Assurance systems, Source code pattern detection systems and etc.

Despite that, based on AST, there can be many business logics implemented which manipulates/ detects the source codes in a platform. AST enables easier access, management and organization of the source code in a Tree structure.

The conclusion for the AST is given by an author who has implemented a Source code Plagiarism detection tool using AST. The author (Baojiang Cui, 2010) mentions clearly their Source Code Plagiarism tool CCS is based on AST since AST is a data structure, which holds syntaxes in a syntactic manner.

B. Prompt the User and Self-Learn

The Prompt the User and Self-learn concept is explicitly implemented in the PseudoJ plugin and is not a standard or existing concept. Despite that self-learning of computer programs have (at the time of writing this document), become the trend. There are many variations of these concepts such as Machine Learning, Deep Learning and etc. Self-learning concept involved with the Pseudo-code to JAVA syntax transformation is a basic level implementation of a self-learning program. The key idea here is to,

When the Statement-Resolver component in PseudoJ refers to the trained data set (rules file), if the pseudo-code statement (user input) contains unknown words which cannot be determined by referring the rules file, the program itself prompts user to map the unknown word with a known action / keyword in the rules file.

As the author (Singh, 2016) specifies, a Self-learning program has the ability to learn from the experience / take decisions based on the previous experience.

The same procedure applies here, the PseudoJ system polishes, optimizes its dictionary (rule set) with the experience it gains from its users. As the author further specifies, predictive computing is the self-decision-making approach of computer programs.

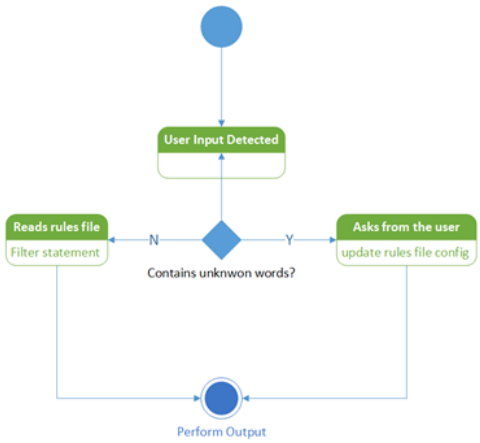

Fig. 3. The basic learning concept

C. Simillar approaches based on simillar technologies

There are number of researches and implementations available which parsers codes. Most of the approaches are based on AST. Despite that, a few researches could be found in the area of Pseudo-code parsing since Pseudo-code is a non-standardised format of code representation.

Another Pseudo-code interpreter:

A better research conducted by (Gimeno, 2017) is also based on an AST to parse Pseudo-codes. (Gimeno, 2017)'s research is about implementing a complete IDE which interprets the Pseudocodes and runs. The research further says the interpreter perform an analysis called Lexical analysis by splitting pseudo-codes and standardizing them by converting them to tokens. The research further performs similar activity to the PseudoJ concept which Parsers the generated tokens in to the output. Nevertheless, the PseudoJ

concept is not using a token generation instead it refers to its' rules file to filter out and standardize the pseudo-code segments.

There are few other approaches also based on AST but not interpreting Pseudo-codes.

A Domain Specific Language implementation which is similar to pseudo-codes, but standardised:

A separate simplified programming language implementation called **Modelica** (Fritzson, 2009) is also based on AST and interprets codes similar to pseudo-codes. But the authors introduce the language as a standardized language, which consist of standard, specific set of syntaxes.

```
while x < 20 loop
        x := x+y*2;
end while;
```

The Modelica language parsers, codes like above which are similar to pseudo-codes. They further specify that the implementation is a **Domain Specific Language** (DSL) as a language, which addresses a specific problem and is not an extension to current languages or a template of a current language.

An Eclipse plugin, which rewrites our JAVA, codes in a proper style:

Author (Arai, 2014) mentions that they have come up with an Eclipse plugin, which learns and rewrites JAVA codes while a user is coding.

The GUI organization and some set of opponents explained by Arai is showing similarities to PseudoJ concept. The approach is also based on AST. As they have come up with an Eclipse plugin, they have used the Eclipse AST.

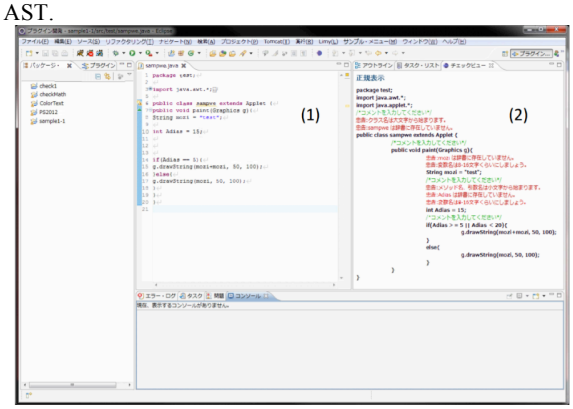

Fig. 4. JAVA code rewriter; source: (Arai, 2014)

The window organization looks similar to the one in the PseudoJ concept. The live code generation also demonstrates key similarities. Nevertheless, they have used an online approach of resolving unknown words, keywords. The program tries to contact a Restful web service implemented in a separate server and resolves unknown words by contacting word dictionaries. The data transferred here uses XML.

Moreover, Arai specifies that they've arranged automated comment outputs that alerts, warns and recommends users with the quality styles of coding. Both the auto-commenting mechanism and JAVA code parsing mechanism is based on AST.

III. METHOD OF APPROACH

The PseudoJ core is an integration of four major component types.

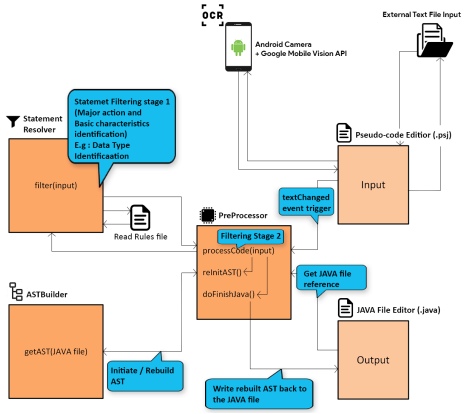

Fig. 5. Core architecture of PseudoJ concept

A. *Pre-Processor*

Pre-Processor consists of three major methods and a set of other helper methods. Respectively processCode (input), reInitAST() and doFinishJava () methods perform collection of code input from Collectors (input methods) and processing, instantiate and obtain an instance of AST and commit processed AST to a JAVA file. Pre-Processor **acts like the central agent** in the whole system since Pre-Processor associates and aggregates few other Helper classes (components) in the system. Pre-Processor maintains every state of the whole process. A few functionalities Pre-Processor further performs in order to maintain the state are,

- Opening and traversing through the rule set. (Pre-defined / trained data set).
- Contacting Statement Resolver to further filter / standardize a Pseudo-code statement.

B. *Statement Resolver*

Acts as a filter. Determines main action of a Pseudo-code statement received from Pre-Processor. Determines and identifies keys (e.g. Type declarations / Variable declarations / Conditions in conditional statements etc.) and related values. Statement-Resolver further associates with the rules file in order to identify keywords, pre-defined words in a statement. By doing that, the prediction process becomes less complex to the Pre-Processor.

C. *AST-Builder*

AST-Builder always obtains the current AST from the Compilation Unit. It contains a static method called

getAST(). Because of the static nature, it always returns the currently obtained AST within the program.

D. I/O Methods

Input/output methods are further described in detail under Section 4.2. Input methods acts as collectors, Pseudo-code importers to the program. The only implemented output method is JAVA syntax output via JAVA code editor.

IV. Working Prototype

With the final implementation of the project, the completion of the PseudoJ core, input / output methods to the program and the mobile application is reported.

A. Pseudo-code Interpretation (Eclipse plugin)

The pseudo-code interpretation processor now recognizes most of the basic level pseudo-codes. A specific set of commonly used keywords is detected in order to predict the JAVA syntaxes at the moment. Hence, the program is now functioning based on a basic set of keywords and will be optimized with the help of prompt the user and Self-learn functionality implemented.

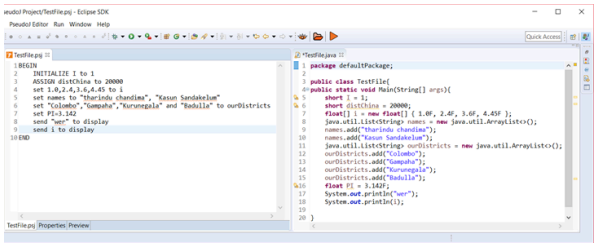

Fig. 6. Core architecture of PseudoJ concept

B. Input Methods

The eclipse plugin hosts three major input methods and a single option to compile (Build file) and run. The eclipse plugin features input methods such as direct pseudo-code text input, External pseudo-code import via. psj files or .txt files and External hard printed pseudo-code input via Android devices.

C. Mobile Application Input with OCR

The Mobile Application is implemented as a basic Android Camera application using Google Camera API and is bound with the Google Mobile Vision API in order to receive automated OCR functionalities. The complete application is a result of hybrid API configuration.

The Android Application is developed using Android Studio with Gradle Build.

V. Business Objectives Achieved

Writing programming syntaxes is not the valuable and major phase of developing a system. The Business logic of a program is the true back-end of a program. Programming languages are there for represent one's logic. At this point, the time taken to write down code lines is unnecessary.

'PseudoJ' is there for fulfill the need. 'PseudoJ' will automate the process at a maximum rate of accuracy possible. Still the operation is far from perfect. But the research can be extended in to a satisfactory level of result. These kinds of applications add value to businesses by making Testers, Unit testers, QA Engineers and developers more productive.

Prediction, Guesswork is always done by humans and was not a job related to software programs since computers are heavily logical. Nevertheless, with the evolution of self-learning concepts, computer programs now operate at a higher level than earlier.

Programs like PseudoJ consists of a frame and PseudoJ will try to fit different pseudo-codes from clients in to that frame. This objective is complex and may cost a few human employees in order to complete the job. The job referred here, most likely will be the prototyping which plays a major role in the modern Agile world.

- PseudoJ is not to auto-generate complete programs with Graphical User Interfaces (GUI). User attention and interaction is needed for large scale programs. Inspection of the auto-generated JAVA syntaxes will not take as much time as re-implementing the complete code from scratch.

- PseudoJ is not for generate exact JAVA syntaxes from a given pseudo code. A regular knowledge in programming is required from the developers to use PseudoJ as the IDE will not generate 100% accurate results. The PseudoJ IDE learns user specific keywords from the user. Learnt keywords and coding behavior will be used to optimize the accuracy of the IDE.

- The IDE's extension OCR scanner will be helpful for developers in order to save even more time. Also, the OCR scanner won't generate exact written characters as it on a paper. In this case, IDE allows users (developers) to manually edit both the generated Pseudo code and the JAVA syntaxes in a code editor.

- Developers will be able to save much time when implementing a complex logic which takes up-to many code lines! Developers and Designers will be able to work together and focus on the logic more. Finally, the overall quality of the product will be higher!

- Inbuilt self-optimizing mechanism will highlight (guess)/ Organize java classes for Objects that should be used inside the development.

- Live code generation:

  Code editors will refresh and react to user's every key press event. Commitments to code editors will result in live java code generation. User-Experience, product Quality will be considered here at its best.

VI. CONCLUSION

PseudoJ research is a prediction tool, which predicts JAVA syntaxes out of Pseudo-codes. Pseudo-codes are used to define program/ business logic in a way that is closer to the human language. However, it's not standardized and is not a DSL (Domain Specific Language). PseudoJ blurs this boundary and makes prototype development easier. PseudoJ basically predicts relevant JAVA syntaxes for a given pseudocode. The program is currently based on JAVA AST provided by Eclipse and could be extended to other platforms also. It can be concluded that programs like PseudoJ makes the development, prototyping process easier, faster. In a business-oriented world, programs, which make the business processes faster, are assets for a business.

REFERENCES

[1] A. Cuhadar, A. C. D., 1994. Scalable Parallel Processing Design for Real Time Handwritten OCR. Colchester, s.n.

[2] Arai, M., 2014. Development and Evaluation of Eclipse Plugin Tool. Vancouver, Canada, s.n.

[3] Baojiang Cui, J. L. T. G. J. W. D. M., 2010. CODE COMPARISON SYSTEM BASED ON. beijing, China, s.n.

[4] Eclipse, 2001. Abstract Syntax Trees. [Online] Available at: https://www.eclipse.org/jdt/core/r2.0/dom%20ast/ast.html [Accessed 02 05 2018].

[5] Eclipse, 2018. Abstract Syntax Tree - PHP Development Tools. [Online] Available at: https://www.eclipse.org/pdt/articles/ast/PHP_AST.html [Accessed 28 04 2018].

[6] Eclipse, 2018. PDE. [Online] Available at: https://www.eclipse.org/pde/ [Accessed 28 04 2018].

[7] Fritzson, P. &. P. P. &. S. M. &. P. A., 2009. Towards a Text Generation Template Language for Modelica. Como, Italy, Modelica.

[8] Gimeno, C. L. D., 2017. Pseudocode Interpreter (Pseudocode. Iloilo City, Philippines, Asia Pacific Journal of Multidisciplinary Research.

[9] Google, 2018. Android Debugging Bridge (ADB). [Online] Available at: https://developer.android.com/studio/command-line/adb [Accessed 03 05 2018].

[10] Google, 2018. Google Mobile Vision. [Online] Available at: https://developers.google.com/vision/ [Accessed 30 04 2018].

[11] Google, 2018. Optical character recognition (OCR). [Online] Available at: https://cloud.google.com/vision/docs/ocr [Accessed 01 05 2018].

[12] Google, 2018. SDK Platform tools release notes. [Online] Available at: https://developer.android.com/studio/releases/platform-tools [Accessed 02 05 2018].

[13] Singh, A., 2016. The rise of self-learning software. [Online] Available at: https://www.recode.net/2016/6/29/12045632/self-learning-software-enterprise-predictive-big-data-net-intelligence [Accessed 01 05 2018].