Analysis and Design of Object-oriented Program Understanding System

Nor Fazlida Mohd Sani[†], Abdullah Mohd. Zin[††] and Sufian Idris [††]

[†]Universiti Putra Malaysia, 43400 Serdang, Selangor, MALAYSIA
[††]Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor, MALAYSIA

Summary
Programming is the most difficult task to most computer sciences students. They always faces problem to write and to understand the program codes. There are lots of exercises is been given to students for practice on writing a program code. But those students still faces problem on understanding the programming codes while written it. The purpose of this paper is to present the model and the developed system that can help students to understand the programming codes. It is a program understanding system for an object-oriented programming language using the plan base approach named CONCEIVER++. In this paper will describe the design and implementation of two main modules of this system, the understanding and editor modules.

Key words:
Program Understanding, Object-oriented Programming, Plan-base, UML, Java.

1. Introduction

Program understanding is an activity that enables the programmer to know the meaning for programming codes. It is an important activity in maintaining available system, debugging a programming code and one of activity in reverse engineering. Romero [1] suggested that program understanding is an intermediate skill to programmers. Understanding on computer program is a complex cognitive activity, therefore realization or development of system or tool is very beneficial to novice programmers and the experienced. Those who involve in programming activities which is difficult are the programmers. Knowledge and experienced of programmer in programming covered writing capability, reading and understanding of a program code. Understanding of a program code is ability and also a difficult task especially for novice programmer. The important skill for any programmers to be developed is the ability to read an available program code which being coded by other programmers [2].

Research in program understanding is still being study until now and the common approach used for supporting program understanding is with completing the program code through abstraction [1]. Abstraction approach is used which directly deal with the source code or system that to be comprehend. It can help to reduced complexity and minimizing the numbers of details in certain program codes [3], also helps a lot in the process of understanding purposes. Another concrete reasons using abstraction is the reliability of the understanding or inference result is true based on the source program [4].

Most of the program understanding algorithm with this approach were using library of programming plan with multi-heuristics strategies to find the existence of plans in the source program. This statement has emphasized in former researches such as Quicili et al. [5], Woods and Yang [6], and Kozaczynski and Ning [7]. The available program understanding system using the plan based approach usually are developed for non-object-oriented programming languages, there are such as Programmer's Apprentice [8], PROUST [9], Talus [10], PAT [11]), CONCEIVER [13] and BUG-DOCTOR [13]. But the object-oriented languages has widely used in education and industry recently. Obviously in local and foreign countries, this new paradigm programming language has been used in teaching of computer programming and has been proved by Bruhn and Burton [14], Gerailt [15], and Madden and Chambers [16]. Therefore, an object-oriented program understanding system is needed specifically for teaching of programming.

Problems arise on the existing program understanding system is incompatible of the usage of teaching a programming process at university. Most of the available programs understanding systems are specially developed for maintaining a system in an organization. A program understanding system can help students on learning programming, reading and understanding certain program codes. Currently, learning of programming at university has widely used the object-oriented programming languages to learn programming at many universities, local or abroad. Program understanding system for object-oriented programming language such as Java does not

Manuscript received January 5, 2009
Manuscript revised January 20, 2009

exist was the main reason why this research is being carried out.

Most the available program understanding systems do not provide utility for add new plan into plan base for their system. Add plan utility will be focused in this research because if students are given with new programming problem, lecturers need to add new plan to the plan base. Now therefore, one module name plan editor will be developed for CONCEIVER++ to accommodate lecturers or users in order to create and add new plan that can solve the problem of the new statement of codes.

The purpose of this paper is to present the design and implementation of CONCEIVER++, which contain two main modules, which are understanding and editor module. The remainder of this paper is organized as follows. Section 2 is a short survey of related works on automated program understanding, knowledge based program understanding, and plan base with its formalism. Section 3 discusses the methodology use to produce the UML sequence diagram of the system that is object-oriented analysis and design. The design of understanding module is explained in section 4, while for editor module in section 5. Section 6 is devoted to the result or implementation of the system. Final section concludes the current findings and discusses some potential future works.

2. Related Works

2.1 Automated Program Understanding

Plan is use for the purpose of understanding must be formatted in a standard form. The standard form is vital so the derived plan formed in the same format and easily accessed using specified recognition algorithm. The language design that will be used for creating plans is called plan formalism. This formalism must be designed to ensure each plan that will be created is in even formed. There are several plan formalisms used by previous researchers. One of is a Plan Calculus in Programmer's Apprentice system is for Lisp language [17]. There was a plan formalism to recognize COBOL programming language by Kozaczynski et al. [18]. This formalism is use in development of program understanding system named Concept Recognizer.

Lots of researchers' groups have focused their efforts in developing tool and technique for automating program understanding. Different program understanding systems are tends to apply different representative framework and heuristics in recognition algorithm. Example, *Concept Recognizer* by Kozaczynski and Ning [7] used top-down library based approach for plan recognition. This system recognizing plan using heuristic approach, specific rules and constrains instruction using representation of component and constraint of plan. Source code will be transformed into abstract syntax tree. The plan recognition algorithm starts by collecting all patterns from the library, then matching all components to the program, come out with a set of potential plans with all components matched. After that, the constraint part of the set of plans will be implemented. Limpiyakorn [19] says that plan representation in Concept Recognizer is simple and unambiguous, also algorithm used is successful to recognize plan in COBOL programs.

Representation of abstraction emitting information that not needed such as syntax tree omits format variations, while control flow graph omitting variation for control statements. Representation replacing codes with abstract model such as event for Quicili [20, 21] represents syntax tree entity. Abstraction was needed for recognition because it will simplify searching area for program representation. In addition, abstract representations have multiple use if there any missing information. Syntax tree and control flow graph will retain the same execution.

2.2 Knowledge based program understanding

PROUST produced by Johnson and Soloway [9] was programming tutor or debugger which match students code with true and known solution. PROUST works as top-bottom, use goal tree solution and analysis with synthesis. It's cannot understands code which do not have problem explanation. Some heuristics is used for instructions, comparison and evaluation during recognition. Transformation will reduces the differences because variation in execution and bugs. Example, different between a=b+c has been declared in plan and for a=c+b that has been written in code will not been recognized as error.

Recognition plan for PROUST is not many as programming knowledge for certain programs, solution or algorithm. It is an explanation of program problems which is written as goal list that the program need to execute. The goal lists have to match the plan with code. A recognition component contains Pascal instances that are language-dependent instances that not tolerate with syntax variations, sub-goal and constraint to be checked compare to the Pascal program representation and the goal that fulfilled earlier. Plan match program statements if the instance being integrated, including sub-patterns, matching syntax tree for statement and its sub-goals and sub-goals completed. If it is failed to matched, PROUST will refer to its knowledge base for standard error so that it

be as reasons that caused by unmatched between student's program and their own solution. This system demonstrates recognition for whole program is possible with small and limited students' code.

Mei Hong et al. [22] have developed an understanding system for C++ language called BDCom-C++. This system analyzed C++ program in static with bottom-up inference, extracting programs information based on conceptual model created by using Enhanced Entity Relationship (EER) model and keep the information in the database. In return to the different requirements, it may ease the functional understanding and structure of C++ program, creating an object-oriented design document, which is Object-oriented Design (OOD) for C++ program in reversed, or supporting the usability of component extraction from the existing program code.

2.3 Plan base and formalism

Plan base is the important component for any program understanding system, usually with plan based approach. This is because of the plan base is the library of inference knowledge for each program code that will be identified by the system in the future. Therefore, in this section is focusing the discussion about plan base approach and plan formalism that has commonly used in existing understanding system nowadays.

The terms 'plan' always been used by the Artificial Intelligent (AI) and Program Analysis community. According to Ning [11], in AI, plan usually referred as sequence of action to achieve the goal. Mean while, for program analysis research literature, this term is used for referring to different subjects such as:

i. Abstract representation for fragment of code
Rich [23] states that, the real 'plan' is for data and control flow representation together with input/output set and intermediate test/operation specification. Ning [11] suggested that, plan is for representation aspect, which as how to represents the language independent information in form of plans.

ii. Programming heuristics
Here, plan stressed as implementation aspects, by which how implementation (instantiation, coding) of problem solving algorithm by using a plan [11].

iii. Programming abstraction concept
Letovsky [24] explain that plan is the conceptual part (example is opposite to primitive or physical part, such as variable, statement, loop) for represents the idea of design of language independent. Ning [11] describe a more semantics orientation on plan definition, that is anything that express the abstract concept or higher level.

iv. Knowledge for identify programming concept
Programming plan is procedural or strategic knowledge for mean of code [9]. It shows plan as knowledge which is needed to understand the programming abstract concept [11].

An experienced programmer is keep on redeveloping lots of hierarchy for program design by recognizing from the data structure and algorithm which is commonly used and typically know how to do the higher level abstraction. The common computerize structure that being used called as cliché. Cliché is a frequently appears pattern in program codes, such as algorithm, data structure or pattern specific domains. Plan is the representation of cliché. The objective of plan recognition is identifying cliché by using the plan. There were three approach of plan recognition, which is top-down, bottom-up, and hybrid which combining the top-down and bottom-up approach [25].

Plan base approach is consists of programming plan that represents specific programming code or domain of application. The developed plan for a plan base is based on formalism that standardized the form of production of the plan.

3. Methodology

This research phase is to produce a model of program understanding system which will be developed using the software development methodology called Unified Approach (UA).

This methodology has been chosen because of the system will be developed by using an object-oriented language and it also can show the system model clearly, also show overall objects and classes that has been developed. Bahrami [26] explained that UA methodology create standardization and unification around the working environment using the Unified Modeling Language (UML) for explaining, modeling and documenting processes of system development. Hence, results from this research in producing a model for the suggested system, CONCEIVER++ will be showed by using the UML notation.

UML has become the universal language for modeling a system. The purpose is been used for representing various type and different model purposes same as the programming languages or natural languages that can be

use in any ways. UML has become standard notation for modeling the object-oriented system. It is an expansionary notation and still being developed. UA use UML to explain and model the analysis and design phase for system development.

In this research of producing the object-oriented program understanding system model which is CONCEIVER++, some of the involved UA processes are the object-oriented analysis (OOA) and object-oriented design (OOD). UA give authorization iterative development which allows going forward and backward between the design and modeling or analysis phase. It makes that tracing become easier and different than the linear process of Waterfall which don't have any form of tracing [26].

3.1 Object-oriented Analysis (OOA)

Main objective for analysis is to present complete graphical view, non-obscured and consistent for system requirement and operation in order to fulfilled users' requirement and need. This will be achieve by developing model for the system focusing on the explanation about what the system will do rather than how to do it [26]. Bahrami add that analysis is the process of transferring the problems definition from obscure facts into system requirement statements.

In this phase will be analyze how users use the system and what are needed for complete requirement of system operation. The methods use in analysis study to gets all information about CONCEIVER++ system is from literature reviews that has been done. OOA phase in UA are using *actor* and *use case* for describing about the system in user perspective. Actor is an external factor which will interact with the system mean while use case is the scenario that explaining how the actor use the system. Use case that been identified here is involved in the overall development process.

The OOA process consists of several steps that can referred to Fig. 1 follows. The steps involves are:

1. Identify actor
   - For CONCEIVER++, actor is the users that use the system.
2. Create process model by using UML activity diagram.
3. Create use case
   - Use case gives views on what user can do with CONCEIVER++ system.
4. Develop interaction diagram
   - Interaction diagram is used to assigned execution sequence of the system.
   - Interaction inside the system can be show in the collaboration diagram
5. Classification, creating a static UML class diagram
   - Identify class
   - Identify relationship
   - Identify attribute
   - Identify method

Iterate and refine. If needed, repeat the previous steps.

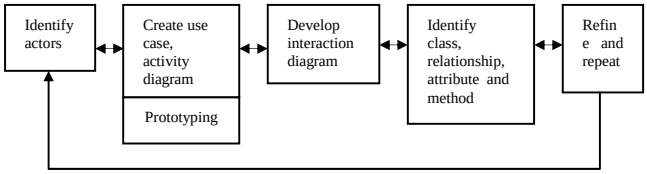

Fig. 1 Object-oriented Analysis (OOA) Process

3.2 Object-oriented Design (OOD)

In the study of UML model for a prototype also involve the object-oriented design phase. Rumbaugh et al. [27] state that the main focus in analysis phase of system development is into what we need to do. Objects that identify during analysis can be used as the framework for design. Attribute methods and relationship for class that has been identified during analysis must be design for implementation such as data type must be describe in the implementation language. According to Bahrami [26], in design phase is the starting of thinking how the problem be really implemented in the system. But, in this phase, the centre is on the class view and access such as for fixed the information or the better way for interact with user or presenting the information. In this level, it is useful for getting the good understanding of the classes in the development environment.

During the design phase, the identified class from the OOA for CONCEIVER++ must be called-back with up shift focus to its implementation. New classes and attributes including methods is been added for the purpose of implementation and user interface. The OOD process consists of such activities as shown in Fig. 2:

1. Design class, attribute, method, association, structure
   1.1 Refine and completing the static UML class diagram with additional details to the diagram. The steps are with below activities:
   1.1.1 Refine attribute

1.1.2 Design methods using the activity diagram to illustrate the algorithm of the method
1.1.3 Refine the relationship between classes (if needed)
1.2 Repeat and refine more

2. Design access level
2.1 Simplify class and the relationship. Main goal is take out the redundant class and structure.
2.1.1 Redundant class: Do not need to keep two classes that executing the changes of the same request and result. Choose either one and drop the other one.
2.1.2 Method class: Explore all classes with one or two methods whether it can drop it or combine it with the other class.
2.2 Repeat and refine again
3. Design the form of display level
3.1 Develop prototype for display level program
3.2 User satisfaction testing
3.3 Repeat and refine

Repeat and refine overall design. If needed, repeat all steps.

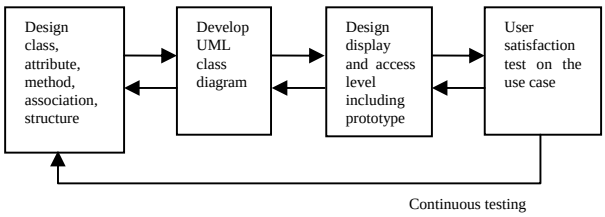

Fig. 2 Object-oriented Design (OOD) Process

The result of the object-oriented analysis and design of the UA method is the model for the CONCEIVER++, object-oriented program understanding system. CONCEIVER++ has two main modules, understanding and editor modules. The focus of this paper is to presents the details of execution with its classes and object involves for each module. In terms of that, in order to give better understanding of the system, UML sequence diagram is use for that purpose. In sequence diagram, the flow of implementation for every module can be show it clearly, what are classes has been used and what are the objects has been created to execute for each module.

4. Understanding Module

As mentioned earlier, CONCEIVER++ consists of two main modules: understanding and editor. In this section will describe on the understanding module. An overall process in this module is that, programming code written by students will be parse and transform using the parser and transformer components. Output from it is in form of control flow graph (CFG) and being kept in one file. The CFG file will read by the code/CFG processor and then all the CFG information will be put in binary search tree structure. In the other side, plan which has been kept in plan base will be access by the plan processor. Plans is read and put into linked list structure. These tree and linked list will be as input to the recognition engine. With these two type of data structure, the recognition engine will match the data from both structure type will come out with the result of understanding to the user of this system.

This module has been divided into three parts which has been mentioned just now, that are code/CFG processor, plan processor and recognition engine. In this research, the user in this module is students or lecturers. Users will write a Java program codes and then insert to the understanding module to be inferred by the system. At the same time, plans will be accessed from plan base for the purpose of inferring the codes. Below discuss on each part of the understanding module.

4.1 Plan Processor

The process involve in this plan processor part is to read the plan in the plan base. Linked list structure is generated and each node in the linked list will contain plan, including the information of the plan. This generated linked list with plan inside is the output from this plan processor part and will be as input for the recognition engine. The objects created for executing this processor, there are *plan*, *list*, *next* and *element*. The UML sequence diagram in Fig. 3 below shows flow of execution of this part and all its objects.

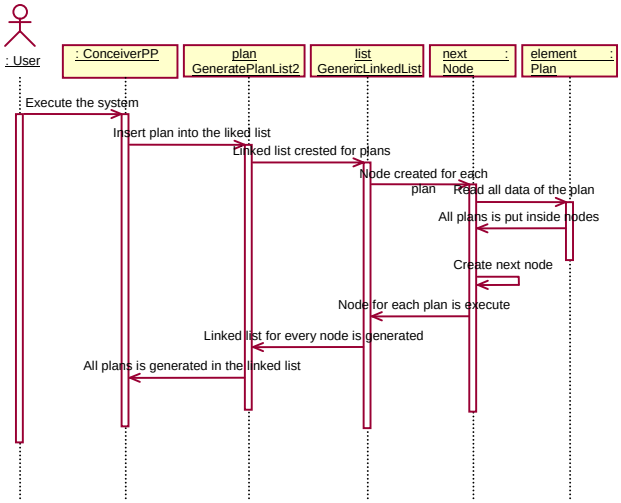

Fig. 3 Plan Processor

Plan object will hold linked list that contain all plans from the plan base. *Plan* object is the object reference for *GeneratePlanList2* that has been used in main class which is *ConceiverPP*. This object is use next in the recognition engine part.

List is an object with role as object reference to *GenericLinkedList* class. In this object will activate the linked list that holds all plans. Each node of linked list represents one plan.

Object that responsible for generating each node inside the linked list is *next*, object reference for *Node*. For each node of plan will keep all information for that particular plan. Therefore, object for retrieving this information is *element*. This object is the reference object of class *Plan*, that pointing to the information of plan. All the information is the data about knowledge of language based on the plan formalism that had mentioned in above section. Because of the design of the plan formalism is not the focus of this paper, researched has been done and based on discussion has agreed on specifying the information needed for representing the knowledge.

4.2 Code/CFG Processor

Second part on implementing the Understanding module is code/CFG processor. Operations that involve in here are information about code which is from the control flow graph (CFG) will generate into binary search tree structure (BST). Each node in three is represents each line of program code. The BST will hold information for programming codes which then will use in recognition process. Simultaneously, number of nodes is collected to be use during the process.

There are several object created for the execution of this code/CFG processor: *startTree*, *tree*, *root*, *data* and *countTree*. The UML sequence diagram in Fig. 4 below shows flow of execution of this part and all its objects.

Object *startTree* is an object reference for *TreeTest* class. The program code that has been through parsing and transforming processes will be change into control flow graph form. The external component named KONSIS [28](Najib and Abdullah, 2005) will parse and transform the code. The result from KONSIS, which is an abstract syntac tree (AST) will be used and formed the CFG with its' data flow information. This CFG will be read by *startTree* object and hereinafter use in the recognition engine part.

Object *tree* is an object that will generate the binary search tree structure. It is an object reference for class named *Tree*. In order to generate the BST, each node created is referred to node in the CFG. These nodes of BST is created using *root* object, an object reference of *TreeNode*.

Each node will saved and contains data or information for the CFG node, with one CFG node represents one line of code. The data or information for the CFG node will be hold by object *data*. *Data* is an object reference of *ParseTree* class. The number of CFG node created will be counted by object *countTree*, an object reference for class *TypeNodeCount*.

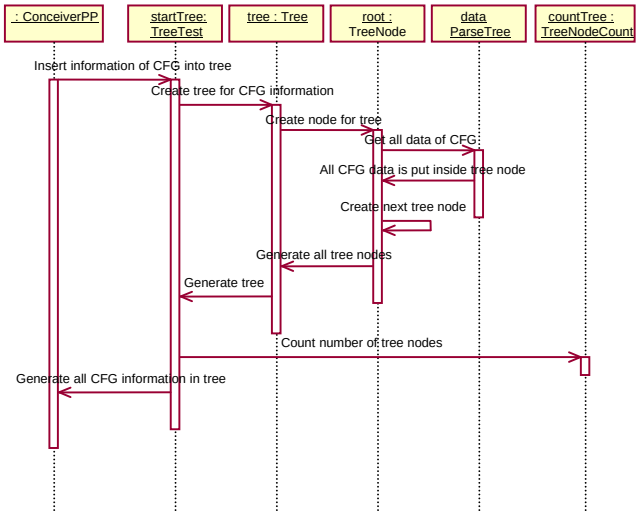

Fig. 4 Code/CFG Processor

4.3 Recognition Engine

This is the third division of understanding module called recognition engine. Operation involve in this part is to match results from plan processor and code/CFG processor part. Result of the recognition produce the information of the meaning or understanding of the program code. Objects created in this recognition engine are *MM* (for matcing), *ttree* and *methodname*. The UML sequence diagram in Fig. 5 below shows flow of execution of this part and all its objects involved.

Object *MM* is an object reference of *Match* class is build to use the linked list of plans and the binary search tree of code/CFG. In object *MM*, these two data structures will be matched and then produce the understanding information for program code.

During the matching execution, *ttree*, an object reference of *TreeTest* is created. This object is use to check the

binary search tree contents which are methods. Linked list name of methods is created by object *methodname*, instantiate by using *CollectMethodName* class. This object is needed during matching process.

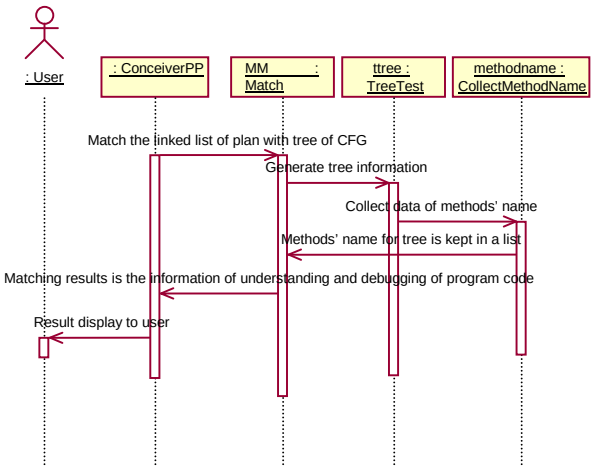

Fig. 5 Recognition Engine

5. Editor Module

The second main module of CONCEIVER++ is an editor module. This module specially created for use by knowledge engineer or lecturers. The two types of users are assumed to understand on the standard that has been set for the plan that is plan formalism, so that they can use this module correctly. This module is created to ease the lecturers to add new plan into the plan base. This editor also allowed lecturer to check each plan that exist.

The importance of this module is when lecturer given a different sets of programming exercises. Even though the lecturers sometimes are using the same programming codes, but they may need statement of code which still did not have the plan for recognition process. So, editor module is the solution to help them to improve and add new plan. This module has been divided into two parts that are: add plan and view plan.

5.1 Add Plan

Operation involve in this add plan part is, new data will inserted by lecturer through add panel available. It will be kept in plan base and be use for understanding purpose. Objects created in this part are *TE*, *planPanel* and *pplan*. The UML sequence diagram in Fig. 6 below shows flow of execution of this part and all its objects involved.

Object *TE* is referenced to *TestEditor* class is an interface for editor module. From this interface, knowledge engineers that are lecturers have to select tab 'Add' in order to add new plan. Hence, *AddPlan* class is executed then an object for panel view is executed.

Object that display panel for lecturer to key-in new plan is *planPanel*, referenced object of *PlanPanel* class. Hereinafter, *PPlan* class is use to create an object reference *pplan* which will hold all data or information input through the panel. Each data inserted will be checked the size by using the *FixedLengthStringIO* class.

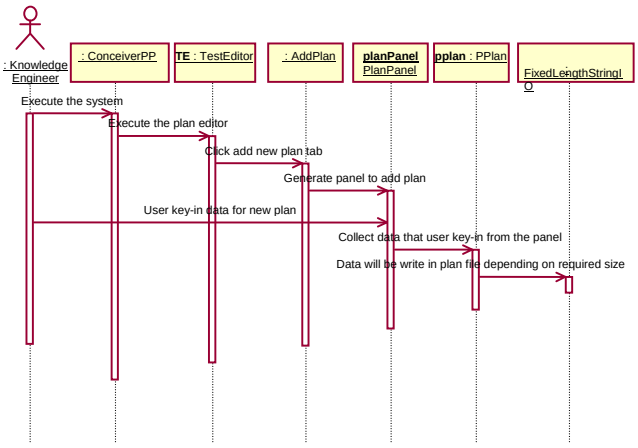

Fig. 6 Add Plan

5.2 View Plan

Operation involve in this view plan part is for accessing information of available plans from the plan base. Objects created in this part are *pplan* and *planPanel*. The UML sequence diagram in Fig. 7 below shows flow of execution of this part and all its objects involved.

Object *TE* is referenced to *TestEditor* class is an interface for editor module. From this interface, knowledge engineer that are lecturers have to select tab 'View' in order to display all plans. Hence, the contents of plan base can be view with execution of *ViewPlan* class.

Object reference of *Pplan* class, which is *pplan*, will hold data or information of plan. The other object, *planPanel* is referring to *PlanPanel* class is use to execute panel for displaying the information for each plan. Knowledge engineer can use button on the view panel to look on the content of the plan.

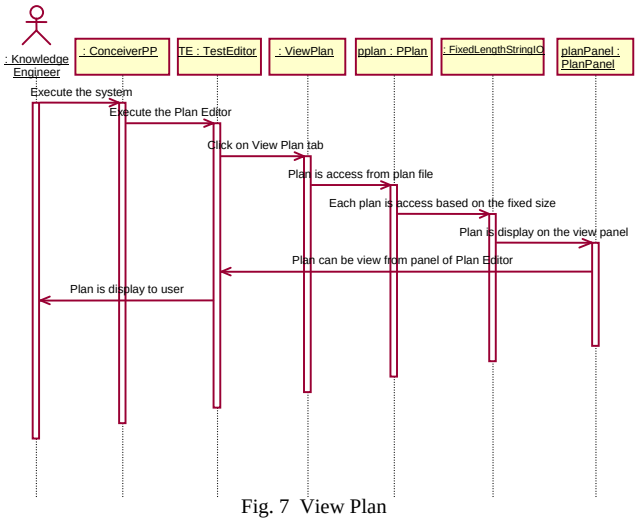

Fig. 7 View Plan

6. Result

The output of the model is the user interface that has been developed by using the Java programming language asmentioned earlier, the name of this system is CONCEIVER++. The main user interface of CONCEIVER++ is as in Fig. 8, which has four menus at the menu bar: File, Edit, Program and Help menus. In File menu contain instructions such as open file, open new file and save file. For Edit menu, the instructions are cut, copy and paste. In Program menu contain the main instruction of the system: Parser, Transformer, Understanding and Editor. The last menu, Help give brief explanation of this system.

Toolbar is also included in this main interface, with the purpose is to give user friendliness environment, easy and fast touch for accessing any command to users. Buttons that has been provided are New File, Open File, Save File, Parser, Transformer, Understanding, Editor and Help buttons.

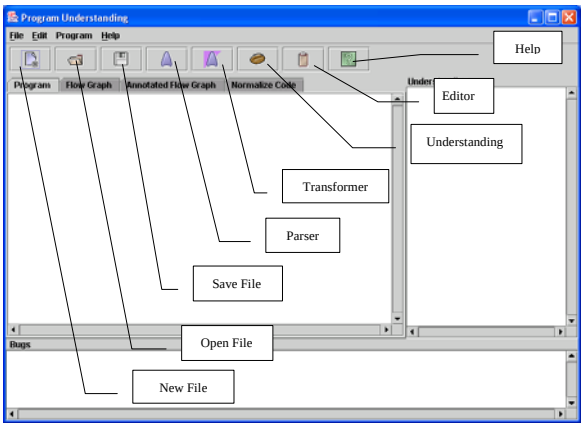

Fig. 8 User interface of CONCEIVER++

6.1 Understanding Module Interface

The result of developing the understanding module, based on the model discussed earlier will be presented in this section. In order to execute this module, user has to click on button Understanding or by using the Program menu bar. After clicking, another interface will be display as shown in Fig. 9.

Based on the UML sequence diagram for Understanding module, there are three processes involved which are code/CFG processor, plan processor and recognition engine. From figure 9 has showed the three panel that labeled as A, B and C. A is named as Plan Base panel which is the result of the plan processor process. B is named as Node of Flow Graph is the result of code/CFG processor process. C named as Understanding & Debugging is the result of the recognition engine process.

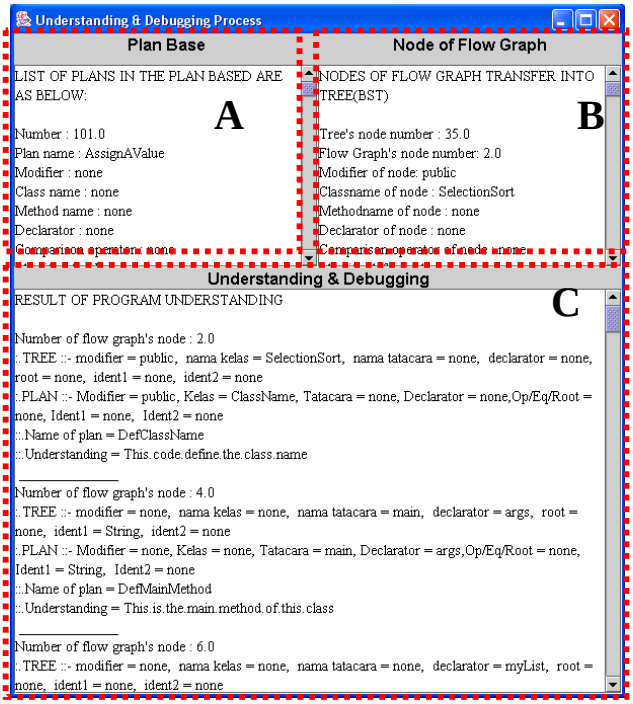

Fig. 9 Understanding Module Interface

6.2 Editor Module Interface

The result of developing the editor module, based on the model discussed earlier will be presented in this section. This module is restricted the knowledge engineer, which is lecturer only. In terms of that, this module is executed once the users key-in the password. If correct, the interface of the Plan Editor will be display as in Fig. 10

shown. There are two tabs in this interface, Add Plan and View Plan. If the user wants to add new plan into plan base, they have to select Add Plan tab. The added plan must follow the specification or the formalism that has been agreed from the plan design phase, which do not discuss in detail in this paper. User can click Add button at the bottom, and the clear field will be display for user to key-in.

Otherwise, if the user only wan t to view all plans available in the plan base, they have to click tab View Plan. Interface of view plan as in Fig. 11. From the figure shows that this interface has four buttons at the bottom labeled as First, Next, Previous and Last. These buttons is use for searching plan with its data and to update any plan.

Fig. 10 Add Plan Interface

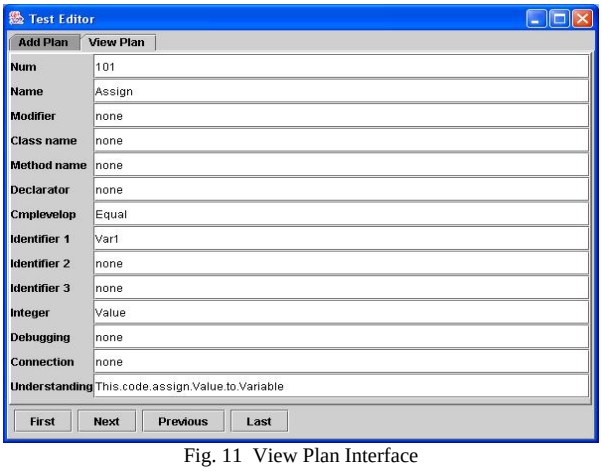

Fig. 11 View Plan Interface

7. Conclusion

This paper has discussed on the model of an object-oriented program understanding system, called CONCEIVER++. Based on the model, we have developed the system and the system has run successfully. The model has been created and design according to the object-oriented analysis and design phases of Unified Approach methodology. The model has been produced and the detail for two main modules of the system, that are understanding and editor modules, has been discussed in detail by using the UML sequence diagram. This diagram has showed and presented a clear discussion on the flow of execution that involved on each module, the classes and object created also has been presented. Then the result of the presented model has been discussed for every module created. The developed module understanding and editor is by using the Java programming language. Finally, future work should be made for evaluating the system to investigate the usability of the system.

References

[1] Romero, P., Cox, R., du Boulay, B. & Lutz, R. 2003. A survey of external representation employed in object-oriented programming environments. *Journal of Visual Languages and Computing* 14(15): 387-419.

[2] Barr, M., Holden, S., Philips, D. & Greening, T. 1999. An Exploration of Novice Programming Errors in an Object-Oriented Environment. *SIGCSE Bulletin* 31(4): 42-46.

[3] Jun Shen, G.V. Cormack, D. Duggan. 1995. On Abstraction and Sharing in Generic Modules. Object-oriented Technology for Database and Software Systems, World Scientific

[4] Kozaczynski, W. & Ning, J.Q. 1989. SRE: A knowledge-Based Environment for Large-Scale Software Re-engineering Activities. *Proc. of 11th International Conference on Software Engineering*, pp. 113-122.

[5] Quicili, A., Yang, Q. & Woods, S. 1998. Applying Plan Recognition Algorithms to Program Understanding. *Journal of Automated Software Engineering* 5(3): 347-372.

[6] Woods, S. & Yang, Q. 1995. Program Understanding as Constraint Satisfaction. *Proc. Computer-Aided Software Engineering* 7: 318-327.

[7] Kozaczynski, W. & Ning, J.Q. 1994. Automated Program Understanding by Concept Recognition. *Automated Software Engineering* 1(1): 61-78.

[8] Rich, C. & Wills, L.M. 1990. Recognizing a Program's Design: A Graph-Parsing Approach. *IEEE Software* 7(1): 82-89.

[9] Johnson, W.L. & Soloway, E. 1985. PROUST: Knowledge-based Program Understanding. *IEEE Transaction on Software Engineering* SE-11(3): 267-275.

[10] Murray, W.R. 1986. Talus: Automatic Program Debugging for Intelligent Tutoring Systems. Technical Report AI TR86-32. AI Laboratory, University of Texas.

[11] Ning, J.Q. 1989. A Knowledge-Based Approach to Automatic Program Analysis. Ph.D. Thesis. University of Illinois.

[12] Al-Omari, H.M.A. 1999. CONCEIVER: A Program Understanding System. Ph.D. Thesis. Universiti Kebangsaan Malaysia.

[13] Burnstein, I. & Saner, F. 2000. Using Fuzzy Reasoning to Support Automated Program Understanding. *International*

Journal of Software Engineering and Knowledge Engineering 10(1): 115-137.
[14] Bruhn, R.E., and Burton, P.J. 2003. An Approach to Teaching Java Using Computers. *SIGCSE Bulletin* 35(4): 94-99.
[15] Gearailt, A. 2002. Using Java to increase Active Learning in Programming Courses. *Proceedings of the Inaugural Conference on the Principles and Practice of Programming*, 107-112.
[16] Madden, M., and Chambers, D. 2002. Evaluation of Student Attitudes to Learning the Java Language. *Proceedings of the Inaugural Conference on the Principles and Practice of Programming*, 125-130.
[17] Rich, C. & Wills, L.M. 1990. Recognizing a Program's Design: A Graph-Parsing Approach. *IEEE Software* 7(1): 82-89.
[18] Kozaczynski, W., Ning, J. & Engberts, A. 1992. Program Concept Recognition and Transformation. *IEEE Transactions on Software Engineering* 18(12): 1065-1075.
[19] Limpiyakorn, Y. 2002. The Signature Approach to Program Plan Retrieval. Ph.D. Thesis. Illinois Institute of Technology.
[20] Quicili, A. 1993. A Hybrid Approach to Recognizing Programming Plans. *Proc. Working Conference on Reverse Engineering*, pp. 126-133.
[21] Quicili, A. 1994. A Memory-Based Approach to Recognizing Program Plans. *Communications of the ACM* 37(5): 84-93.
[22] Mei, H., Yuan, W., Wu, Q. & Yang, F. 1997. BDCom-C++: A C++ Program Understanding System. *Chinese Journal of Electronics* 6(2): 64-69.
[23] Rich, C. 1981. A Formal Representation for Plans in the Programmer's Apprentice. *Proc. 7th International Joint Conference on AI*, pp. 1044-1052.
[24] Letovsky, S. 1988. Plan Analysis of Programs. Ph.D. Thesis. Yale University.
[25] Müller, H.A. 1996. Understanding Software Systems Using Reverse Engineering Technologies Research and Practice. *Tutorial presented at 18th International Conference on Software Engineering*, Berlin, Germany.
[26] Bahrami, A. 1999. *Object Oriented System Development using the Unified Modeling Language*. Boston: McGraw-Hill.
[27] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorenson, W. 1991. *Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice-Hall.
[28] Mohd Najib, M.K. & Abdullah, M.Z. 2005. KONSIS: Set Komponen Bagi Membangunakan Sistem Pemahaman Aturcara. *Prosiding Persidangan Kebangsaan Sains Pengaturcaraan 2005*, pp. 202-210.

Nor Fazlida is a lecturer in Department of Computer Science, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. She obtained Ph.D. focus on Program Analysis and Program Understanding System from Universiti Kebangsaan Malaysia in 2007. Her research interests include reverse engineering, software maintenance and program debugging.