Development of Run-Time UML for JAVA Programming Language

Sulaiman Mohd Nor sulaiman@suria.fke.utm.my
Mohamed Khalil Hani khalil@suria.fke.utm.my
Mohsen Ashourian mohsena@ieee.org
Goh Hock Ann RunTimeUml@yahoo.com
MiCE Department,
Faculty of Electrical Engineering
Universiti Teknologi Malaysia
81310 UTM Skudai, Johor, Malaysia.

Abstract: Run-time modeling is a method of using the execution data for modeling the real process going on in a program. This paper introduces a run-time modeling scheme as a tool for software visualization and Unified Modeling Language (UML) is selected to accomplish this concept. To develop a Run-Time UML, it is necessary to devise user-friendly methods for showing static and dynamic modeling diagrams by mapping source codes information and real time execution data to UML diagrams. The proposed scheme is planned to be implemented into an Integrated Development Environment (IDE) using Java programming language.

Keywords: Software visualization, Unified Modeling Language (UML), Java.

I. INTRODUCTION

Software industry is experiencing a shift from procedural and linear paradigm into object-oriented. It could also say that it has move from subroutines into subsystems [3]. The shift is because the object-oriented approach promotes better modularity and reusability. Many other technologies also contribute to object-oriented software development. They are Component-Based Software Engineering (CBSE)[1], design pattern and the vast resource of Application Program Interfaces (API) available as a tool to assist the developers.

The fast growing of object-oriented software makes the task of understanding and debugging become a cumbersome if not impossible task. Furthermore, in order to improve software development, it is necessary to facilitating the process of comprehending existing programs. Software comprehension is the stumbling block that should be encountered in any effort to improve software industry. [6]

As the demand for software comprehension has escalated, many of the related fields such as software modeling, software visualization has also gain considerable attention from the developers. Software modeling is the field of modeling software before the development stage whereas software visualization is the discipline that makes use of various forms of imagery to provide insights and understanding and to reduce complexity of the existing software system under consideration [4].

However, these fields are far from perfect. This is because showing a thing that cannot be seen by the naked eye can be a difficult task [2]. Despite the effort of many developers, the problem still persists and lots of them remain unsolved. Thus, software comprehension proves to be a promising frontier to look into.

This paper centers on a run-time modeling scheme as a tool for software visualization. Run-time modeling is a method of using the execution data for modeling the real process going on in a program. The final goal is to integrate Run-time UML into an Integrated Design Environment (IDE) for developer to use. This project is implemented using UML and Java programming language.

II. ARCHITECTURE AND IMPLEMENTATION

The architecture of the proposed system was made to be as simple as possible to avoid too much development time. The project adopted the component based software architecture, swing architecture and also Java Platform Debugging Architecture (JPDA). JPDA provides a solution to reach into the Java Virtual Machine and get the run-time or execution information. A debugger implements this architecture. These collected data will be used later to show the dynamic execution of a program.

In order to jump-start the project the widely available open-source software from the Internet was used. Their implementation are studied and set as a guide to ensure better quality software. Besides, this will give more focus on the main objective of the project, which is to build a Run-Time UML. The components considered for development for are:

1. UML editor
2. Text editor
3. Project navigator
4. Compiler and parser
5. Debugger
6. Run-Time UML viewer

Most of the above components are self-explainable. However, further clarification is needed to differentiate between data from UML editor and Run-Time UML viewer. The UML editor contain UML diagram that will be built by the user. The user draws the class diagram and key in the necessary information. Another way to get the UML diagram is to extract it from the source code.

0-7803-6355-8/00/$10.00©2000 IEEE

On the other hand, the Run-Time UML fetches its information during the execution of the source code. As to the question of how it is rendered, static UML diagram will be used to provide hints on how to draw the Run-Time UML. This effort has been made so that the developer or user would not be alienated by his/her own work.

III. CONSTRAINTS AND LIMITATION

A number of constraints have to be put in order to reduce architecture complexity and to increase the feasibility of implementation. These constraints are for reducing the size of the Run-Time UML.

1. In this project, UML editor only have class diagram, whereas Run-Time UML have object diagram and collaboration diagram. However these diagrams may be modified so that it can be more intuitive and adapt better with the Java Programming Language.
2. Every source codes (*.java file) have a matching UML diagram (*.UML file). This is to reduce complexity. The architecture of the source code directly mapped to the UML file. Only the class that is implemented need to be drawn, while the ones that are associated will be set as a hyperlink to another UML where it is implemented.
3. The Run-Time UML may be queried at a certain level of depth only. May be ten levels deep from the `main()` function will be enough. This means that `x.y()` is one level deep while `x.y().z()` is two levels deep. It makes no sense to query so deep as it would not only overload the program but may also interfere the comprehension process of the developer that was working with the software.
4. Another limitation is that only classes, which are implemented, are queried for Run-Time UML. This means that Java API will not be queried. An example is `System.out.println()` in JAVA API examples.

Besides the above listed constraints and limitations, the user is also urged to apply good programming practice that facilitate comprehension. For example avoid long class or method implementation. Furthermore, intuitive and consistent names for fields are also encouraged. The developers are also advised to draw the diagram in a nice and intuitive way.

IV. BASIC WORKFLOW

This section discusses what would be the necessary actions taken by user in order to be able to see the Run-Time UML. The diagram (Fig 1) provides a summary of what will be discussed.

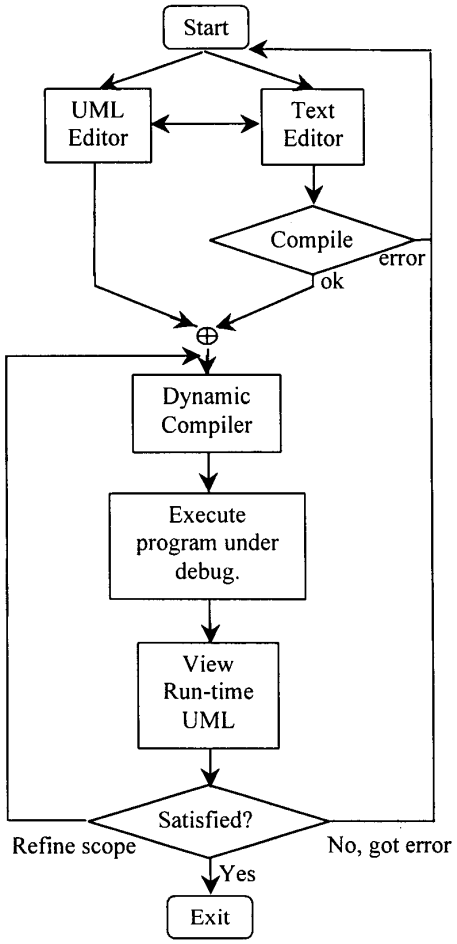

Fig.1 Basic Workflow

A. Before dynamic compilation

As shown in the diagram, there are two primary source of information that is needed before the user can move on to do a dynamic compilation. One is the source code and the other is UML diagram. The user can choose either one to start with. However, it is recommended that the user start with UML editor as it is a more rational choice, unless circumstances force otherwise.

Case 1

In case 1, the user chooses to work with UML editor. The user will need to draw all the class method, field and the relationship between one class and another. However, the user need not worry, as there will be drawing tools to help user with this.

After the user is satisfied with the model, the user can begin the programming stage. The programming stage will be facilitated, as the skeleton of the program has been generated. User only needs to insert the implementation and also necessary documentation.

After finish inserting the implementation, the user could compile the program. If error exists then the user is required to perform the semantic debugging. If the compilation is successful then it is possible to start dynamic compilation.

Case 2

The user could have also started with programming first. Usually this case happens when the source code has already exist or readily available. Then UML diagram can be generated from the source code. However, the user still needs to fill in all the relationship and place the class according to the user preference. Finally the user could compile it and if successful, it will be ready for dynamic compilation.

What needs to be noted here is that the text editor and UML editor will always keep track of each other changes. If the user changes one of them, this means the user will change the other. The program here ought to be robust and yet simple enough for the user to use.

Other features

Other interesting features that is in the plan, is that the UML should be able to collapse selectively. This means that it should be able to hide or collapse some of the fields and methods and replace it with ellipses. It can also be expanded if necessary.

Beside this, there should also be hyperlink to linkup between one UML diagram and the other that it is referencing. To get a bigger picture the user can also choose to expand the hyperlink within the current diagram.

For the text editor, a number of features will be added. Besides coloring and indentation, features like fast navigation [5] between previous and last edited section and also jumping between and to different declaration of classes, methods and fields will be included. Text editor should also have the capability of UML, which will enable it not only to collapse methods but also part of the implementation. This will not only save space but also reveal to the user the essence of the model.

B. Dynamic compilation

Dynamic compilation is a compilation for run time UML purposes. During the process, the user will be queried for necessary information for debugger to fetch from Java Virtual Machine during program execution. The information fetched will be used for Run-Time UML. By default, during dynamic compilation, it is based on the previous UML diagram for hint on what class, methods and fields that the user wishes to see during dynamic modeling. Those fields that are collapsible will be ignored by default. Moreover, handle name will also be ignored, as a run-time handle will be queried.

C. Run-Time UML viewer

When all the above steps have been carried out, the user could execute the program under the debugger. The debugger will query for the information from the Java Virtual Machine. The user could then end the program and get ready for the Run-Time UML navigation.

Next, the information there is to see and the way to navigate through the Run-Time UML will be described.

1. Information in Run-Time UML

In Run-Time UML the user not only see the classes that have drawn previously but also the process that involved them. Information that will be shown will be as follows.

1. Object creation and object deletion will be shown as soon as it happens.
2. Object will expand only when it is one of its methods or fields reached. If the object to be reach is in another diagram, then that diagram will be expanded. This means that at any one time only one class will be expand and the method that was called will be highlighted. At any other time the object will remain collapsed.
3. Every object in Run-Time UML ought to be reference by another object. If this is not the case the object that is not referenced will likely be garbage collected. (This is because an object that cannot be reach cannot be use.) At both end of the link, there can be a reference name or handle. These handles are use by the object at the other end of the link to reference it. A sequence number will appear in the middle if the link is use to invoke a function in another object. This will be similar to collaboration diagram, except that it is derive from run-time data.

2. Navigation in Run-Time UML

In Run-Time UML, the plan is to make it sort of tracing program. However, the difference is that it graphical representation is used in graphical form rather than text. The process ought to be like watching a video clip. The user will be able to move forward, backward, stop or pause. Like debugger, the user can choose to step into or skip an action or a function. This step will help user to navigate faster to the next interesting section.

However, while programming and modeling, the user ought to build up a mental model of what should be going on during the execution of the program the user has developed [7]. So during navigation, the user could check and see whether his hypothesis is correct and the program executes according to the requirement specification.

If the user detects any logic error or if it does not meet the requirement then the user could go back and do necessary amendment with the source code and diagrams, recompile and view the Run-Time UML again.

Anyway the user could choose to be passive and just sit back and leave the software to show the user what is going on.

The nice thing about this project is that the Run-Time UML is scalable to the user preference, and able to show only the essential information and leave out the unnecessary detail until queried. We hope this tool can become a media for changing ideas, opinion and to be use as a teaching aid. The help it can bring would be immense.

V. PROBLEMS AND DISCUSSION

Now to be realistic, problem exists in every field of studies. There are two main categories of problems that are to be solved. The first category is the problem of mapping the source code into an UML diagram and the second category would be to map the data collected during execution of the program into the dynamic diagram.

A. Mapping from source code to UML diagram

Since UML is build as a common modeling language and not for Java only, UML diagram doesn't show everything about the features of Java Programming Language. This means, some slight problem will be inevitable. The following list some of the problems.

a) It is difficult to show container and arrays. This problem also brings forward to dynamic visualization where size and content of a container may change across time.

b) Beside this, there is the problem of nested class such as inner class and anonymous inner class. These inner class and anonymous inner class could appear almost anywhere within a source code. Trying to draw this can be difficult.

c) In Java architecture it has exception handler and action handlers, which doesn't map very well with UML.

Because of the above problems the UML diagram should be modified to suit our purpose.

B. Mapping from run-time data to Run-Time UML

Besides having the problem of showing something that can't be seen, we have the problem of size and complexity. As we know the size and complexity of Run-Time UML can varies a lot during the execution of a program. Our aim is to reduce the size and decrease complexity. We wish to convince the user and not to confuse them. Bellow is some of the problem face.

a) It is hard to show the big picture in a small computer screen. The screen and also the limited processing power will constraints the performance of this project.

b) Showing more than one thread without confusing the user is difficult. We got to realize that thread programming itself is a complex task and showing it in a simple way is not going to be easy.

c) Showing iterative call and interleaving code is not easy either. This process could easily confuse the user.

d) There is also the problem of non-continuous method call such as event and exception that is hard to show. This is because it breaks the flow of comprehension.

e) It may also be hard for the user to imagine what is happening by merely viewing at the Run-Time UML without looking at the output pane. Action is going to be taken to tackle this problem.

VI. CONCLUSION

The basic working mechanism and issues in the development of Run-Time UML was discussed. The methodology used will hope to enrich the field of software Engineering. It is hoped that with this contribution, development of object oriented based programs such as java will be more efficient and reduce management and debugging time.

VII. REFERENCE

[1] M. Aoyama, "New Age Of Software Development: How Component-Based Software Engineering Changes The Way Of Software Development?" Proc. of International Workshop on Component Based Software Engineering, Kyoto, Japan, April 1998.

[2] Byung-do Yoon and Oscar N. Garcia, "Cognitive Activities And Support In Debugging", Proc. of the 4th Symposium On Human Interaction With Complex Systems, pp. 160-169 1998.

[3] P.C Clements, "From Subroutines To Subsystems: Component-Based Software Development", America Programmer, V8#1, Cutter Information Corp. Nov. 1995.

[4] C. Knight. "Visualization For Program Comprehension: Information And Issues". University of Durham, computer Science Technical Report 12/98 .

[5] J.Singer, T.lethbridge, N.Vingan and N. Anquetil. "An Examination Of Software Engineering Work Practices". In Proc. Of CASCON'97, pp 209-223. Toronto Canada 1997

[6] S R. Tilley, S Paul, D B. Smith, "Towards a Framework for Program Understanding". In 4th Workshop On Program Comprehension pp 19-28. IEEE. Comp. Soc. Press Mar 1996

[7] T. Systa, "On The Relationships Between Static And Dynamic Models In Reverse Engineering Java Software". Proc of the 6th Working Conference On Reverse Engineering. pp 304-313. 1998.