Dynamic Generation of Sequence Diagrams from Java™ Programs

Nuno Fortes

Abstract—This project contributes to the area of tools for generating sequence diagrams in an automatic way. To increase the flexibility and interaction with other applications, the developed application is able to export the generated diagrams to multiple formats allowing its visualization by diverse tools.

Index Terms—Diagram, Generator, Java, Sequence.

1 Introduction

We start by presenting the motivation and objectives for the development of this work.

1.1 Motivation

The *debugging* (monitoring and analyzing) of Java™ [18] applications can be very boring and annoying, specially when we do not have easy access to the code that is being implemented in the program. The problem starts to get more complicated when an application is in a remote environment and does not produce any *output* in the console or in a Log file (registration of transactions), mainly in the case of computer software of considerable size. Due to the increasing size and complexity of software applications the understanding of their structure and behaviour has become more and more important. Proper specification and design activities are known to be indispensable for producing understandable *software*. The *reverse engineering* of dynamic behaviour of a program is still a topic of on-going research. However, there are allready several types of diagrams which allow us to visualize the sequence of communication between objects of a program, also called **Message Sequence Charts** [5]. In **UML2** [13] specification there is a type of diagram for this purpose which is called **Sequence Diagram**. It allows to represent in a graphic way, the program behaviour and mainly the sequence of messages between objects (call and return method events). This type of diagrams can be very useful in analysis of object-oriented computer applications. Because allow us to know many informations about what the application does throughout its implementation (In the course of its execution). They therefore are called behaviour diagrams. One of the issues to be considered when trying to develop a tool for the generation of this type of diagram is, if there is some kind of support for the language through an API (**Application Program Interface**), to be able to automatically extract all the information required for the implementation of a program, in order to build the Sequence Diagram. Java™ [18], is an object-oriented programming language developed by *Sun Microsystems* [11] in 1991, with the main characteristic of creating applications which can run in any platform or Operating System, by a Java™ Virtual Machine. The Java™ applications usually are compiled to *bytecode*, which is a binary language the virtual machine will understand, thus to interpret or compile to native code, in order to run nativily by the operating system. So, for the purpose of creating Sequence Diagrams, the Java™ language technology is very useful, because it enables to obtain such information as the sequence of

• *Nuno M. Fortes is a Student of Computer Engineering at the Management and Technology School (ESTIG) of the Polytechnic Institute of Beja (PORTUGAL) (e-mail: nunofort@ gmail.com).*

Manuscript received October 31, 2007.

objects and methods found in real time, when the program is running on the *virtual machine*. The technology **Java Platform Debugger Architecture** or simply JPDA [16], developed by *Sun Microsystems*, facilitates this type of work, through various types of interfaces or APIs (**Application Program Interfaces**) to retrieve information about the status of many types of data at some point during the execution of a Java™ application. There is an interesting study [14] made on development of a reverse engineering tool for UML sequence diagrams using the Java™ platform technology, which explains how we can implement such a tool in order to analyse the behavior of any kind of Java™ program.

1.2 Objectives

The main goal of this project was to develop an application in the Java™ language which will function like a *reverse engineering* tool and allow us to generate automatically and in real time a Sequence Diagram, which graphically represents the execution sequence of any kind of Java™ program. The project focus on collecting data during Java™ program execution and creating the Sequence Diagram structure in memory. Then we can export the sequence diagram to several image formats for representation or visualization. This allow us to analyse (do some kind of *debugging*) effectivily, on the works and implementation of various parts of the code, help to find errors and *bugs* faster or, even code optimization, in a development process. The implementation of the code of this application relies heavily on several advanced features of the Java™ platform and uses the JDI [15] (**Java Debug Interface**) API which is part of the JPDA [16] (**Java™ Platform Debugger Architecture**), to analyse and retrieve the information needed to construct the behaviour of a running Java™ program into a Sequence Diagram. This API interface is easily accessible from developed Java™ code and the application does not need to use functions of a low-level layer, like the JVMTI [17] (**Java Virtual Machine Tool Interface**). So, the main purpose of such a tool is to provide a mapping from a Java™ code execution, mainly events of *call* and *return* methods, to a **UML2** [13] sequence diagram. The methods that are called when there is an event of input or output method are used to extract the information at the time the event occurs and create the respective arrows to insert into the sequence diagram.

2 About the Application

THE application developed in Java™ [18] within this project, can generate a graphical representation of the **UML2** (Unified Modeling Language) [13] Sequence Diagram, by executing the code of a Java™ program. That is, this tool can generate automatically and in real-time, a schematic presentation of all sequence execution. This allows accurate values to be shown at the moment of every event of call or return method of the Java™ program to be analyzed. This possibility is very valuable in any analysis and when debugging Java™ code. But nevertheless, the functionality of the application goes far beyond the simple generation of Sequence Diagrams. There is also the opportunity to interact with other similar applications, in order to increase its flexibility of use. The result diagram can be exported to various image formats, such as PIC [6], **JPEG** [4] or **EPS** [1], and visualize it on external tools. It can also save the Sequence Diagram description to a **XML** [2] format which can be loaded and visualized in the future. The value of some variables necessary for the operation of the application, can be modified using a configuration file of the Application. This application can be used by Java™ programmers to view and analyze (*debugging*) of various functional parts of the executed code, helping to quickly find errors, *bugs* or even make some optimization on code instructions.

2.1 Main Features

- Uses JDI (**Java Debug Interface**) [15] to get real time Sequence Diagram trace information.
- Can load/save the Sequence Diagram presentation in a based XML [2] format description.
- The information of sequence diagram construction operations are saved to a *LOG* file

(usefull for *debugging* of proper application functionality).

- The sequence diagram can be visualized by the application, through Java™ Graphics.
- Permits to export the diagram to the PIC [6] language image format by using the UMLGraph [12] macros for specially design sequence diagrams. Then we can use a graphics processing tool like *pic2plot* [7] to convert the PIC format to other Image format such as, PNG, GIF, SVG or PostScript (PS) [8].
- Can export the sequence diagram to the format of the *Sequence* tool [9].
- There is a simple configuration file to easily change some variable values defined in the application so, the program will operate differently from the pre-established or what comes by default.
- Option to generate Sequence Diagram of a small part of the code executed by application (This is very useful in diagrams of large application projects).
- Option to generate Sequence Diagrams from a simple *trace* file description without the need to write a line of Java™ code. With this feature, the application can be used to create Sequence Diagrams from *trace* files generated from any source program.

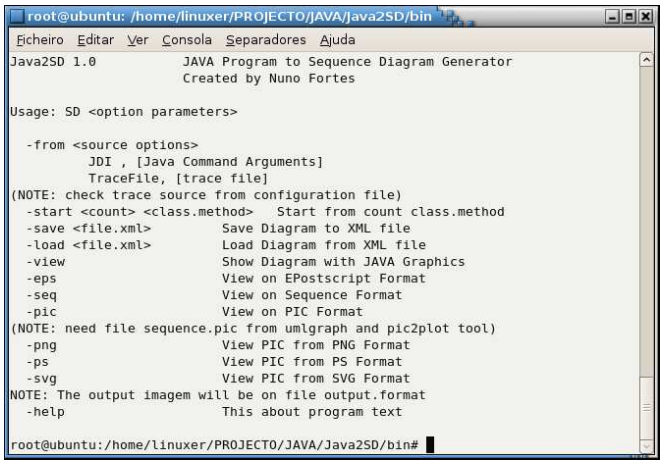

Fig. 1. Application Screenshot

3 Implementation

Besides the use of the Java™ Debug Interface to get the Sequence Diagram information, we needed to incorporate or establish a connection with other tools and formats or technologies for further development, sush as:

- A XML (**Extensible Markup Language**) based format [2] was used to store all sequence diagram information that can be easily loaded at any time in the future.
- A large part of the export flexibility of the developed application, was achieved through the language PIC [6]. This language can specify diagrams in terms of objects such as boxes with arrows between them and can be translated into concrete drawing commands by a compiler. This project uses the *pic2plot* [7] tool to convert these drawing commands to a binary image format. Uses a modified version of the *UMLGraph* [12] sequence diagram PIC macros file for the specific purpose of the application.
- Presentation of sequence diagram in the *Sequence* tool [9] format.
- Presentation of sequence diagram by the application, through Java™ Graphics.
- Built-in support for Sequence Diagram generation to Encapsulated Postscript (EPS) [1] image format.

4 Example Diagrams

Because of the limited space on article, we will present only a small example (Fig. 2) for clarification on what may look like a sequence diagram generated by the application. The respective code is on Listing 1. Note that, in the example diagram you can see the actual parameter values passed through the methods.

Listing 1. Java™ Code of the Diagram

```
1  package pt.ipbeja.estig.test;
2
3
4  /**
5   * Class for Testing the Application.
6   * uses some arithmetic type objects to perform the operations.
7   *
8   * Aplicacao de Teste para criar um diagrama de sequencia
9   * que faz chamadas a varios tipos de objecto
10  * serve de exemplo para facil verificacao do funcionamento
11  * das varias componentes da aplicacao em questao.
12  *
13  * @author Nuno Fortes
14  * @version 1.0
15  *
16  */
```

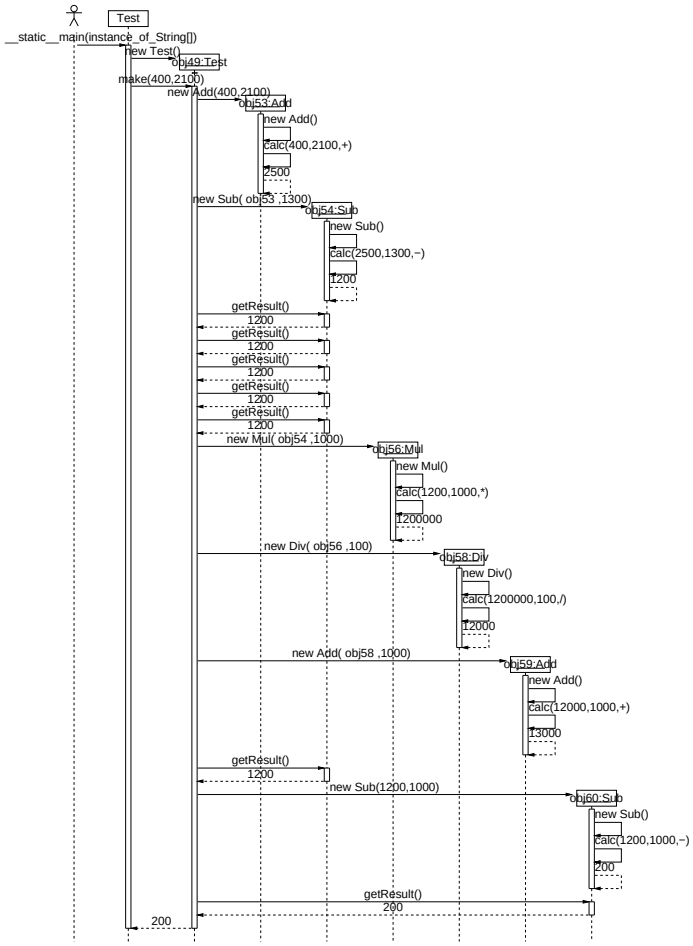

Fig. 2. Included Test Example

```
17 public class Test
18 {
19     public Test()
20     {
21     }
22
23
24     public String make(int n1, int n2)
25     {
26         //int res;
27       String s = "";
28       int[] subs = new int[5];
29         Sub sub = new Sub( new Add(400,2100), 1300 );
30       for (int i = 0; i < subs.length; i++)
31           subs[i] = sub.getResult();
32       for (int i = 0; i < subs.length; i++)
33           s = s + "  " + String.valueOf( subs[i] );
34       Mul mul = new Mul( sub, 1000 );
35       Div div = new Div( mul, 100);
36       Add add = new Add( div, 1000 );
37         return String.valueOf( new Sub( sub.getResult(), 1000 ).
               getResult() );
38       //return 1000;
39     }
40
41
42    public static void main(String args[])
43    {
44      System.out.println("Test Program");
45
46      Test test = new Test();
47      //int res = test.make(400,2100);
48      System.out.println("(400 + 2100)-1300 * 1000 = "+test.make(400,
            2100));
49    }
50 }
51
52 //debugging option
53 //javac -g
```

5 Conclusion

The Java™ development platform relies on a complex library of classes. Moreover, the JDI (**Java Debug Interface**) [15], makes a lot easier the development of *debuggers* or applications that need to monitor the various types of events generated by the applications. we achieved the desired objectives implemented on this project. It is possible to generate the sequence diagram of any Java™ application through the information available at the time of the input and output method events generated when running the program. That knowledge, enabled better understanding of the inner workings of the platform and technology Java™ . Personally, I hope to put in use the knowledge I have achieved about the Java™ language, by developping other programming projects of interest and use this technology in conjunction with other languages, mainly in the development of applications related to the Internet. About the distribution of this project, in order to serve as an example and usefulness to the general public, it was decided to put the package of the application available on the Internet for wide spread, on the world's largest Open Source software development web site which is *Sourceforge dot Net* [10] with a GPL [3] license (**GNU General Public License**).

Acknowledgments

This document is part of the final project of the Course of Computer Engineering. I want to thank Professor João Paulo Barros, the availability and the precise manner his guided me during the various stages of development of this Java™ project, without which it would not be possible to complete this work.

References

[1] Encapsulated Postscript. http://en.wikipedia.org/wiki/Encapsulated_PostScript, http://en.wikipedia.org/wiki/PostScript. Reviewed in 2007.

[2] eXtensible Markup Language. http://pt.wikipedia.org/wiki/XML. Reviewed in 2007.

[3] GNU General Public License. http://pt.wikipedia.org/wiki/GNU_General_Public_License. Reviewed in 2007.

[4] JPEG Image Format. http://en.wikipedia.org/wiki/JPEG. Reviewed in 2007.

[5] Message Sequence Charts. http://www.sdl-forum.org/MSC2000/index.htm. Reviewed in 2007.

[6] PIC Language. http://en.wikipedia.org/wiki/Pic_language. Reviewed in 2007.
[7] Pic2plot Tool. http://www.gnu.org/software/plotutils/, http://en.wikipedia.org/wiki/Pic_language. Reviewed in 2007.
[8] Postscript Image Format. http://en.wikipedia.org/wiki/PostScript. Reviewed 2007.
[9] Sequence Tool. http://www.zanthan.com/itymbi/archives/cat_sequence.html. Reviewed in 2007.
[10] Sourceforge dot Net. http://www.sourceforge.net. Reviewed in 2007.
[11] Sun Microsystems, inc. http://www.sun.com. Reviewed in 2007.
[12] UMLGraph Tool. http://www.spinellis.gr/sw/umlgraph/, http://www.spinellis.gr/sw/umlgraph/doc/indexw.html. Reviewed in 2007.
[13] Unified Modeling Language (UML). http://www.omg.org/cgi-bin/doc?formal/07-02-05, http://www.agilemodeling.com/style/, http://pt.wikipedia.org/wiki/UML . Reviewed in 2007.
[14] Matthias Merdes and Dirk Dorsch. Experiences with the development of a reverse engineering tool for uml sequence diagrams: a case study in modern java development. In *PPPJ '06: Proceedings of the 4th international symposium on Principles and practice of programming in Java*, pages 125–134, New York, NY, USA, 2006. ACM.
[15] Sun Microsystems, inc. Java Debug Interface (JDI). http://java.sun.com/javase/6/docs/jdk/api/jpda/jdi/index.html, http://java.sun.com/j2se/1.5.0/docs/guide/jpda/jdi/index.html, http://bbs.cjsdn.net/Doc/Mustang/guide/jpda/jdi/index.html. Reviewed in 2007.
[16] Sun Microsystems, inc. Java Platform Debugger Architecture (JPDA). http://java.sun.com/javase/6/docs/technotes/guides/jpda/jpda.html, http://java.sun.com/javase/6/docs/technotes/guides/jpda/architecture.html, http://bbs.cjsdn.net/Doc/Mustang/guide/jpda/architecture.html, http://builder.com.com/5100-6370_14-6139512.html. Reviewed in 2007.
[17] Sun Microsystems, inc. Java Virtual Machine Tool Interface (JVMTI). http://java.sun.com/javase/6/docs/platform/jvmti/jvmti.html, http://bbs.cjsdn.net/Doc/Mustang/guide/jvmti/jvmti.html. Reviewed in 2007.
[18] Sun Microsystems, inc. Java™ Language. http://java.sun.com/, http://en.wikipedia.org/wiki/Java_(programming_language), http://www.sun.com, 1991. Reviewed in 2007.