ZGRViewer: A GraphViz DOT Visualizer

Contributing to an Open Source Project

Nikos Korfiatis
Department of Management Science and Technology
Athens University of Economics and Business
nkorf@eltrun.gr

Contents

Introduction

1.1. Overview

Scope of this report is to document our contribution to the open source project **ZGRViewer**[1] which was done as part of the course "*Software Comprehension and Maintenance*", taught this semester by professor Diomidis Spinellis. Following the course requirements we collaborated with Open Source Developers in order to understand the process of open source software development and to this end to contribute/implement some of our perceptions to an interesting open source project. The project that we finally choose to contribute was the ZGRViewer which is a GraphViz dot source visualizer.

Following a formal way of communication we contacted the chief developer of the application and addressed our thoughts about addressing some functionality and usability issues through our contribution. The original mail message that was send to Emanuell Pietriga (PhD, Visiting Research Scientist, MIT/W3C) can be seen in Figure 1

```
Received:  from gateway.nuxeo.com ([213.56.215.224]) by cosmos.eltrun.gr with Microsoft
SMTPSVC(5.0.2195.6713); Tue, 16 Mar 2004 09:47:22 +0200
MIME-Version: 1.0
Content-Type: multipart/alternative;
          boundary="----_=_NextPart_001_01C40B2A.EA42E100"
Received:  from nuxeo.com (unknown [192.168.2.139]) by gateway.nuxeo.com (Postfix) with
ESMTP id 2E6551DC94 for <nkorf@eltrun.gr>; Tue, 16 Mar 2004 08:47:21 +0100 (CET)
X-MimeOLE: Produced By Microsoft Exchange V6.0.6375.0
content-class: urn:content-classes:message
Subject: Re: RFC - ZVGR Viewer
Date: Tue, 16 Mar 2004 10:47:21 +0300
Message-ID: <CA22BB1CDF3EF74EB522054264D167470A6E02@cosmos.eltrun.gr>
X-MS-Has-Attach:
X-MS-TNEF-Correlator:
Thread-Topic: RFC - ZVGR Viewer
Thread-Index: AcQLKurDn+MgTxxqQN+1KBJKoIgOeg==
From: "Emmanuel Pietriga" <epietriga@nuxeo.com>
To: "Nikos Korfiatis" <nkorf@eltrun.gr>

Hello,

I've added you as a developer to the ZVTM project. But before you commit
anything, could you give me more information about what you intend to do
exactly?

I'm not sure I understand what is the extra editor window you mention.
Is it a textual representation of the DOT file that you would be able to
edit from ZGRViewer so that you could then refresh the graph?

Thanks,
Emmanuel

--
Emmanuel Pietriga (epietriga@nuxeo.com)
```

```
http://claribole.net

Nikos Korfiatis wrote:
> Dear Emmanuel
> My name is Nikos Korfiatis and i'm a software engineering student at
> Athens Greece. The reason that i'm sending you this email is to ask your
> permission to do some comits to the ZVGR Viewer. I like GraphViz because
> it generates clear graphs and is a very handfull tool. Although i find
> graphviz a very nice tool, i can't convince my colleagues to use it
> because they have been stack into the WYSIYG hell... I think i can add
> an extra editor window to allow user's edit their dot files and then
> parse them to the viewer. I think ZVGR Viewer could be an excellent
> frontent editor for the graphviz.
> If you agree you can add me to the developers of your project (sf acount
> name: nkorf)
>
> Cheers,
> Nikos
```

Figure 1: The mail message that was send to the developer including the reply

In terms of communication we had a good contact since the ideological foundations of open source community are openness to everyone that has a clear perception of the project and wants to contribute.

Next we give some background information about the nature of the application, the context of use and the issues that are behind our contribution and finally we present in technical terms the contribution that was done to the existing source code of ZGRViewer.

2. Background Information

ZGRViewer is a wrapper application between the GraphViz graphics suite (citation) and the *"Z Visual Transformation Machine"* developed at Xerox-Inria Research Center Europe (?). By referring to ZGRViewer as a wrapper application we mention that ZGRViewer cannot function properly without these two programs/libraries installed. Following this specification we provide some background information about the components/libraries that ZGRViewer uses to complement it's functioning.

Development Status:	4 - Beta
Intended Audience:	Developers, Science/Research
License:	GNU Library or Lesser General Public License (LGPL)
Natural Language:	English, French
Operating System:	OS Independent
Programming Language:	Java
Topic:	Vector-Based, Viewers, Visualization, Build Tools

Figure 2: Sourceforge Project Info

2.1. The ZGRViewer Component Model

Component based software engineering is a modern trend to every aspects of the Software Projects, from architectural design to testing and maintenance issues, software components are being used widely in order to disseminate the programming effort, give a clarity into the resulted application and finally foster the time required to develop the application. ZGRViewer follows a component model as can be inferred from the Figure 1.

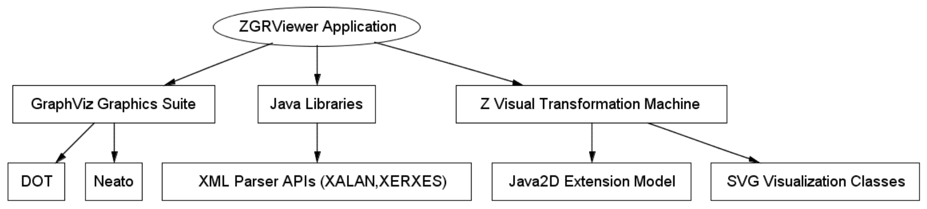


Figure 3: The ZGRViewer Component Model

The main components of the ZGRViewer are:

- ***The GraphViz Graphics Suite***

  Graphviz [2] is a collection of Graph Drawing Tools containing batch layout programs (such us dot, neato, fdp, twopi); a platform for incremental layout (Dynagraph); customizable graph editors (dotty, Grappa); a server for including graphs in Web pages (WebDot); support for graphs as COM objects (Montage); utility programs useful in graph visualization; and libraries for attributed graphs. The features and the functionality of GraphViz programs began to expand at the beginning of 2000 following a decision of ATTs Bell Labs to release the software as an open source. Many and different contributions have been made to Graphviz with most of them being modifications to the contextual use of the GraphViz such us using it to create different types of graphs to cover different needs. We provide some examples at the end of this section.

  - ***DOT***

    DOT is a part of the GraphViz suite that makes directed graphs using advanced layout algorithms described in (citation). DOT Visualizes a graph in four different faces: First phase is to break any cycles which occur in the input graph by reversing the internal direction of certain cyclic edges. Secondly it assigns nodes to discrete ranks or levels thus schematizing a non or less crossing virtual graph to be computed in the third face. The fourth step sets coordinates of nodes to keep edges short, and the final step routes edge splines. DOT accepts input in the DOT language which we will present following. This language describes three kinds of objects: graphs, nodes, and edges.

The main (outermost) graph can be directed (digraph) or undirected graph. DOT only draws directed graphs.

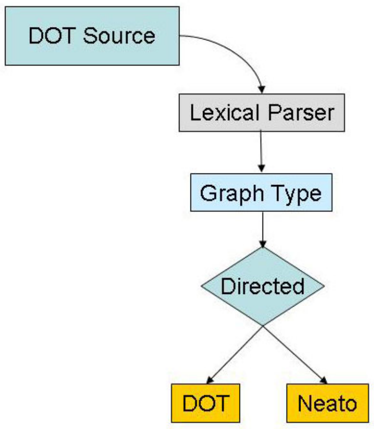


Figure 4: The DOT Language Processing Model

- ***Neato***

Neato works using a different algorithmic model in contrast with DOT but it is compatible with it as it takes the same input format (DOT Source) and global parameters. The main principle which is behind Neato is that is uses an iterative problem solver to find a "low energy" configuration in the graph. Low energy is mentioned as placed between every pair of nodes such that its length is set to the shortest path distance between the endpoints. A comparative figure of the result of the same source parsing by DOT and Neato can be seen bellow:

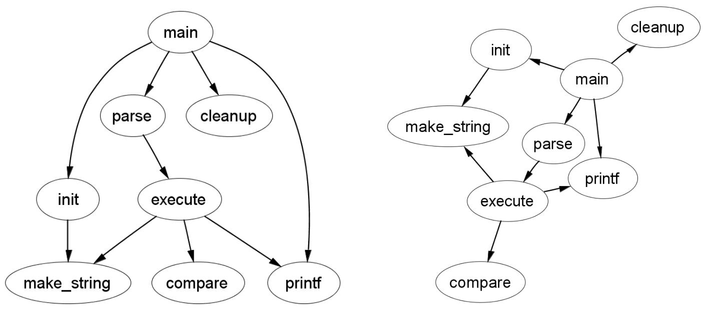


Figure 5: The same declarative specification projected by DOT and Neato

DOT has a wide acceptance as a comfortable way of visualizing

Contextual Uses of GraphViz	
Doxygen	DOT is used to create documentation diagrams for Software
UMLGraph	DOT is used in the same context to create UML Class Diagrams declared with the javadoc
Sqldot	DOT is used to visualize the Database Schema from the SQL Source
Bibliometrics	Neato can be used to measure the co-authoring impact in research publications
ISAViz	Layout algorithms are being used to visualize RDF nodes

- ***Java Libraries***
  - ***Xerces XML APIs***

    ZGRViewer is a Java application that makes a light use of XML in order to complement some functions that deal with Input/Output or read from user parameters. Xerces is used to provide the ability to process and parse xml data files that are being used to store specific information related to the application such us the user preferences. In our case ZGRViewer needs to know where is the appropriate path to the DOT and Neato executables, some user preferences such us font antialiazing etc. Through the application when a parameter has to be extracted from user preferences a Xerces Method is being invoked to retrieve the value of the specific parameter.

- ***Zoomable Visual Transformation Machine (ZVTM)***

ZVTM is also an open source project that has its roots in the work of Emannuel Pietriga at the Xerox Research Center / INRIA France. The basic feature of ZVTM is the ability to provide a coherent application programming interface that can be used easily to provide the functionality of a graph drawing application. Furthermore the Java2D extensions that have been implemented within the engine provide a convenient way to navigate through large settlements of graphs using a 2,5 D space diameter.

- ***Java2D Extension Model***

  As already mentioned ZVTM provides an extension to the excisting Java2D drawing API making it easier to use by the programmers. ZVTM has the word “Zoomable” in its word as it provides an accelerated zoom function giving to the user the perception that he is navigating through a 3D environment. The actual fact is that the user when making a zoom to the graph takes an accelerated transformation of the graph to convenient his request giving him the illusion that he is behind a 3D representation.

- ***SVG Visualization Classes***

  The second major component of the ZVTM is the SVG Visualization classes that are used to provide the graphical representation. SVG is a graphics specification that is written as an application of the XML and provides information on how to display geometrical shapes and graphics in a viewer or a web page. Practically most of the applications' functionality relies to this feature as the common point of the ZVTM and Graphviz is the SVG input and output.

Beyond the application components a major issue is the input format that ZGRViewer accepts as an input. Practically any file containing validated DOT source is accepted and visualized immediately. To this end the DOT Language specification and syntax is important to furthermore understand the functionality of ZGRViewer and in the next section we provide some information regarding the syntax of DOT Declarative Specifications.

2.2. The DOT Language

As already mentioned DOT and Neato use its own language to visualize the results of an appropriate declarative specification:

```
digraph G
        {
//      rankdir=LR ; // comment if you don't want horizontal align
        node [shape=box];
                zgrviewer [label="ZGRViewer Application",shape=ellipse];
                graphviz  [label="GraphViz Graphics Suite"];
                dot       [label="DOT"];
                neato     [label="Neato"];
                javalibraries [label="Java Libraries"];
                zvtm   [label="Z Visual Transformation Machine"];
                java2d [label="Java2D Extension Model"];
                svg [label="SVG Visualization Classes"];
                xml [label="XML Parser APIs (XALAN,XERXES)"] ;
                zgrviewer ->graphviz ;
                graphviz ->dot ;
                graphviz ->neato ;
                zgrviewer ->zvtm;
                zvtm ->svg ;
                zvtm ->java2d ;
                zgrviewer ->javalibraries ;
                javalibraries -> xml ;
        }
```

Figure 6: A declarative Specification of the Figure 1

The Dot language specifies the following semantic entities:

- **graph:** Can be directed (digraph) or undirected
  - each graph can have an identifier, in our example in fugure3 G is an id for the graph
- Each graph has an atribute list which is contained between two brackets. An atribute list contains the main part of the specification

such us nodes, subgraphs and properties

- **Nodes** represent the main entities of the graph and are interconected using the edge operator "->". Each node can have a specific atribute list or the atributes can be inhereted by global statements in the beginning of the graph. Furthermore DOT and Neato accept global parameters in their call method invocation and format their vizualization acordint to the values invoked.

In our example a global parameter regarding the shape of the node is defined with the atribute box. This atribute can be ovewritten when it is defined though an atribute list. In our example the shape of the node labeled "ZGRViewer Application" is ellipse instead of box as defined in the beginning of the file.

```
        graph : [ strict ] (graph | digraph) [ ID ] '{' stmt_list '}'
    stmt_list : [ stmt [ ';' ] [ stmt_list ] ]
         stmt : node_stmt
              | edge_stmt
              | attr_stmt
              | ID '=' ID
              | Subgraph
    attr_stmt : (graph | node | edge) attr_list
    attr_list : '[' [ a_list ] ']' [ attr_list ]
       a_list : ID [ '=' ID ] [ ',' ] [ a_list ]
    edge_stmt : (node_id | subgraph) edgeRHS [ attr_list ]
      edgeRHS : edgeop (node_id | subgraph) [ edgeRHS ]
    node_stmt : node_id [ attr_list ]
      node_id : ID [ port ]
         port : port_location [ port_angle ]
              | port_angle [ port_location ]
port_location : ':' ID
              | ':' '(' ID ',' ID ')'
   port_angle : '@' ID
     subgraph : [ subgraph [ ID ] ] '{' stmt_list '}'
              | subgraph ID
```

Figure 7: The DOT Grammar

DOT and Neato read the declarative specification and depending on the input parameters it produces the output in a format depending on the argument passing when the call of the visualization engine begins as can be seen in the Figure 7

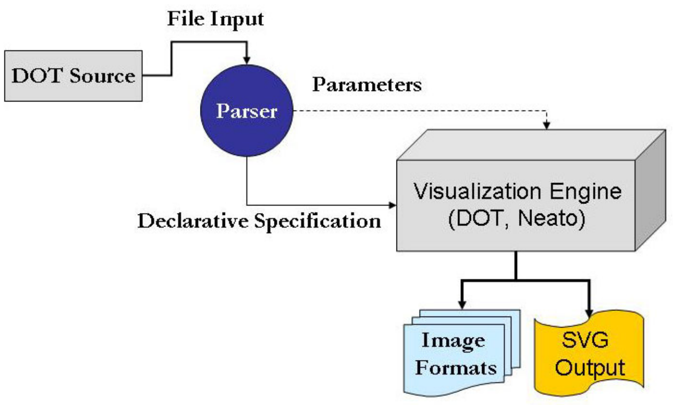


Figure 8: DOT language excecution model

3. Architectural Specification of ZGRViewer

ZGRViewer is curently consisted of 14 classes and aproximately 119 methods. The functionality handled from each class is sumarized in the table bellow:

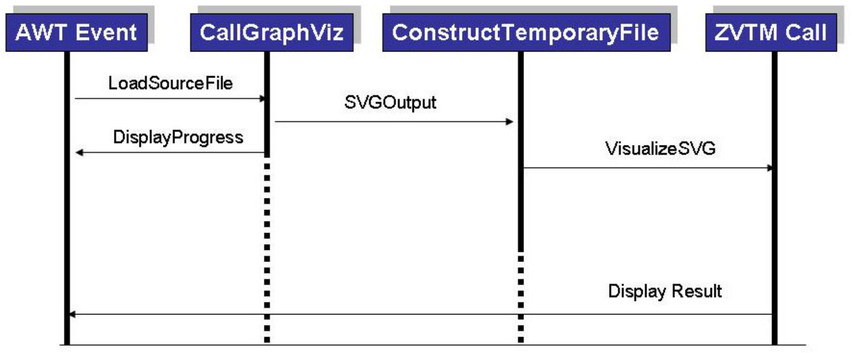
Filename	**Functionality**
ConfigManager.java	Handles configuration parameters such us the path where the DOT/Neato executables relie.It uses xerces to access the user preferences stored in an XML file.
DOTManager.java	Handles Calls to the Graphviz DOT/Neato
Messages.java	Handles the Messages tha are presented to the user each time an operation demands them to show.
PrefWindow.java	Gui class to construct the interface for the preferences window
PrintUtilities.java	Handles the printing of the current vizualization
ProgPanel.java	Handles the progress bar that is displayed while graphviz computes the SVG output.
TextViewer.java	A simple class that act as a container for the messages that are displayed when
Utils.java	Contains various utility methods such us checking for the version of JVM, the operating Sytem etc.
WebBrowser.java	When defined in the DOT Source this class handles the calls to the apropriate web browser in order to open the url
ZGRApplet.java	A wrapper class to use ZGRViewer through a web browser
ZgrAppletEvtHdlr.java	Handles user interactions (button clicks etc)

	when ZGRViewer is deployed as Java Applet
ZgrvEvtHdlr.java	Handles user interactions (button clicks etc)
ZGRViewer.java	Contains the main method as along as with the user interface specification (menu bar etc)

Beyond this classes our application use some other classes that are not part of the Java Foundation Classes and require the apropriate jars in order to be used during the program compilation. Some of this classes/packages are summarized bellow:

Import Statement	Functionality
com.xerox.VTM.engine.*;	The main ZVTM engine classes
com.xerox.VTM.glyphs.*;	Classes to handle interaction with the visualized drawing
org.apache.xml.serialize.XMLSerializer;	Several Classes from the Xerces library to serialize

The call sequence of the ZGRViewer is documented in the following figure:



3.1. Expressing our Perceptions for Contributing to the Project

4. Contributing to the Project

Having clarified the goals of our contribution by using the application we then had to configure our development environment in terms of soure code control, compilation and deployment. ZGRViewer uses the Jar utility in order to pack its classes into one file making the deployment process much more easier as having to replase only one file each time we compiled the programme.
Obtaining the source code was very easy by using the cvs access method

that sourceforge provides to its' registered users.

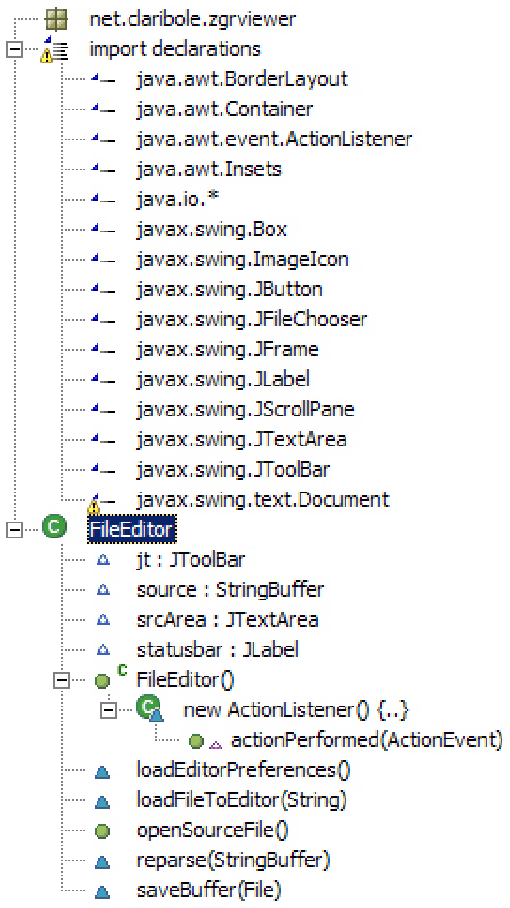
```
:pserver:nkorf@cvs.sf.net/cvsroot/zvtm/zgrviewer
```

The major issue that we had to address was the automation of the development processes. Following the triple ***write->compile->deploy*** we had to create a process specification in order to adress the issue of software process automation. Since our development was caried out using the Java Programming Language we considered reasonable to use a wide accepted build tool by the java developers. ANT was the choise to automate the process and a specification of the software process was compiled as can be seen bellow:

```
<!-- $Id: build.xml,v 1.3 2004/04/05 17:47:34 nkorf Exp $ -->

<project default="dist" basedir=".">
        <!—Defining Basic Properties -->
       <property name="src"  location="net/claribole/zgrviewer"/>
        <property name="dist" location="dist"/>
        <property name="lib"  location="lib"/>

        <target name="init">
                <mkdir dir="${dist}"/>
        </target>

        <target name="compile" depends="init">

        <javac srcdir="${src}" destdir="${dist}"/>
        </target>

        <target name="dist" depends="compile">
        <jar jarfile="${lib}/zgrviewer.jar" basedir="${dist}"/>
        <delete dir="${dist}"/>
        </target>
</project>
```

Figure 9: A simple ant Build file that we compiled for the project purposes

The second step was to integrate well our source code with the excisting one in the application but not having it



```
net.claribole.zgrviewer
import declarations
    java.awt.BorderLayout
    java.awt.Container
    java.awt.event.ActionListener
    java.awt.Insets
    java.io.*
    javax.swing.Box
    javax.swing.ImageIcon
    javax.swing.JButton
    javax.swing.JFileChooser
    javax.swing.JFrame
    javax.swing.JLabel
    javax.swing.JScrollPane
    javax.swing.JTextArea
    javax.swing.JToolBar
    javax.swing.text.Document
FileEditor
    jt : JToolBar
    source : StringBuffer
    srcArea : JTextArea
    statusbar : JLabel
    FileEditor()
        new ActionListener() {..}
            actionPerformed(ActionEvent)
    loadEditorPreferences()
    loadFileToEditor(String)
    openSourceFile()
    reparse(StringBuffer)
    saveBuffer(File)
```

4.1. Testing and Functionality

In order to assure that nothing was harmed through our contribution we decided to provide a test case using the JUnit API. This test cases assures that when a file is loaded to the ZGRViewer and a request to edit it is being made by the user then the file will load normally instead of making an exception to the application. This is needed because the dot source file can be simultaneusly accessed by the ZVTM methods or from the FileEditor class that we have presented previously

Bibliography

[1] Zgrviewer home page.
http://zvtm.sourceforge.net/zgrviewer.html.
Last Access Date: 2004/06/08.

[2] Stephen C. North Emden R. Gasner, Eleftherios Koutsofios and Kiem-Phong Vo. A technique for drawing dricted graphs. *IEEE Transactions on Software Engineering*, 19(3):124–230, May 1993.