CCVisu 3.0 Introduction and Reference Manual — Draft —

Dirk Beyer
University of Passau, Germany

2010-03-03

Abstract

CCVisu is a lightweight tool for visual graph clustering and general force-directed graph layout. Although the tool was originally developed for computing clustering layouts of software systems, based on dependency and co-change graphs, CCVisu is applicable to many graph layout problems.

Contents

1 Getting Started

CCVisu is a lightweight tool that reads and writes data using simple text formats (cf. pipes-and-filters architecture). It is written in Java, and should be usable on all platforms. The tool is designed as a framework, easy to use, and easy to integrate into existing re-engineering environments; several formats for data interchange are already implemented. CCVisu provides a command-line interface as well as a GUI. The graph layout is currently provided in VRML format, in SVG format, in a standard text format, or directly drawn on the screen.

1.1 Download and Installation

Download the current release from the CCVisu web page at `http://ccvisu.sosy-lab.org`. Start the program from the command line using `java -jar CCVisu-3.0.jar` or via double-click on the jar file in a file browser. If no parameters are given, a GUI is started that asks for the parameter. Option `-h` prints a list of available options. Alternatively, pull the sources from the repository or unpack the zip archive, and compile the Java sources; the package contains a `Makefile`. It is a good idea to set the `CLASSPATH` variable of your shell to the directory `bin`, which contains the Java byte-code files after successful compilation. If you do not want to set your `CLASSPATH` variable globally, you can use the file `setup.sh` (generated by `make setup`) to set the variables in the current shell: `source setup.sh`. This also sets the `PATH` variable to the CCVisu directory, to enable the use of the shell wrappers `ccvisu.sh` and `disp.sh` to abbreviate the command line a bit (invocation of the Java virtual machine).

1.2 Usage and Command-Line Options

```
Usage: java ccvisu.CCVisu [OPTION]...
Compute a layout for a given (co-change) graph (or convert).

Options:
General options:
   -h  --help        display this help message and exit (current value: true).
   -v  --version     print version information and exit (current value: false).
   -q  --quiet       quiet mode (current value: false).
   -w  --warnings    enable warnings (current value: false).
   -verbose          verbose mode (current value: false).
   -a  --assertCheck check if assertions are enabled (current value: false).
   -g  --guiMode     GUI mode (provides a window to set options) (current value: false).
   -inputName <str>  read input data from given file <str> (current value: 'stdin').
   -outputName <str> write output data to given file <str> (current value: 'stdout').
   -inFormat FORMAT  read input data in format FORMAT (default: RSF, see below).
   -outFormat FORMAT write output data in format FORMAT (default: DISP, see below).

Layouting options:
   -dim <int>        number of dimensions of the layout, up to 3 (current value: 2).
   -iter <int>       number of iterations of the minimizer;
                     choose appropriate values by observing the convergence of energy (current value: 100).
   -initLayout <str> use layout (LAY format) from file <str> as initial layout (current value: '').
   -fixedInitPos     fix positions for vertices from initial layout given by -initLayout (current value: false).

Energy model options:
   -attrExp <int>    exponent for the distance in the attraction term
                     (default: 1).
   -repuExp <int>    specifies that <int> is applied as exponent to the distance in the repulsion term
                     (if <int> != 0) or that log is applied to the distance (if <int> = 0) (default: 0).
   -vertRepu         use vertex repulsion instead of edge repulsion
                     (default: edge repulsion).
   -noWeight         use unweighted model (default: weighted).
   -grav <float>     gravitation factor for the Barnes-Hut-procedure
                     (default: 0.001).

DOX reader option:
   -relSelect <rel>  selects a relation for visualization
```

```
                     (default: REFFILE).

CVS reader options:
   -timeWindow <int> time window for change transaction recovery, in milli-seconds (current value: 180000).
   -slidingTW        use sliding time windows instead of fixed time window, i.e., the time window 'slides':
                     a new commit node is created if the time difference between two commited files is bigger
                     than the time window (current value: false).

Layout writer options:
   -hideSource       draw only vertices that are not source of an edge.
                     In co-change graphs, all change-transaction vertices
                     are source vertices (current value: false).
   -minVert <float>  size of the smallest vertex disc; diameter (current value: 2.0).
   -fontSize <int>   font size of vertex annotations (current value: 14).
   -backColor COLOR  background color (default: WHITE).
                     Colors: BLACK, GRAY, LIGHTGRAY, WHITE.
   -noBlackCircle    no black circle around each vertex (default: with).
   -ringColor <str>  Color of the ring around the vertex discs (current value: 'GRAY').
   -depDegreeColor   Color of the vertex disc determined by dep-degree (current value: false).
   -showEdges        show the edges of the graph; available only for CVS and RFS inFomat (current value: false).
   -scalePos <float> scaling factor for the layout to adjust; VRML and SVG only (current value: 1.0).
   -noAnim           layout not shown while minimizer is still improving it
                     (default: show).
   -annotAll         annotate each vertex with its name (current value: false).
   -annotNone        annotate no vertex (current value: false).
   -shortNames       shorten vertex labels (current value: false).
   -dispFilter       show extra controls for display filter (current value: false).
   -initGroups <str> assign vertices to groups (and colors) according to file <str>
                     (cf. file marker_script.example.txt as example) (current value: '').
   -openURL          the vertex names can be considered as URL and opened in a web broswer.
                     This option used with DISP output requires to hold CTRL KEY while clicking (current value: false).

DISP specific option
   -browser <str>    browser <str> will be invoked; if empty, CCVisu will try to guess (current value: '').

Input Formats:
   RSF               Graph (relation) in Relational Standard Format (RSF).
   LAY               Graph layout in textual format.
   CVS               CVS log format (produce with 'cvs log -Nb').
   SVN               SVN log format (produce with 'svn log -v --xml').
   DOX               Doxygen XML dump format (produce with 'doxygen').
   ODS               ODS Spreadsheet.
   AUX               Graph is passed as data structure from a third-party client.
Output Formats:
   RSF               Graph (relation) in Relational Standard Format (RSF).
   LAY               Graph layout in textual format.
   SVG               Graph layout in SVG format.
   VRML              Graph layout in VRML format.
   GRAPHML           Graph in GraphML format.
   DISP              Display graph layout on screen.
```

2 CCVisu Tutorial

In this tutorial, we write `ccvisu.sh` for `java -jar CCVisu-3.0.jar`. If no command-line option is given, CCVisu opens a GUI to enter necessary parameters, e.g., an input file to operate on. A double-click on the jar file in a file browser should do the same.

2.1 Generating Graphs in RSF Format / CCVisu as Fact Extractor

Relational Standard Format (RSF) is a common exchange format for relations. RSF is used by many relation-based tools, for example, the relational programming tool CrocoPat and the re-engineering tool Rigi (Rigi Standard Format). RSF is based on plain text with whitespace (most common: tabs) as delimiter, which makes it easy to process data on the command line with standard Unix tools like `grep`, `cut`, and `sed`.

When CCVisu is used to produce layouts (visualizations), it expects input graphs in one of the following file formats: Relational Standard Format (RSF) (cf. Sect. 3.3), Doxygen XML format (DOX) (cf. Sect. 3.2), Subversion log format in XML (SVN) (cf. Sect. 3.1), and CVS log format (CVS) (cf. Sect. 3.1). Since CCVisu uses RSF as file format for data exchange, it can export (to RSF) relations that it extracted from one of the above mentioned input formats.

Open Document Spreadsheet (ODS) to RSF Converter. For example, given a file `crocopat-2.1.ods`, which is an ODS file that contains one table sheet named 'CO-CHANGE' with several columns representing a relation. Then the command
`ccvisu.sh -informat ODS -i crocopat-2.1.ods -outformat RSF -o crocopat-2.1.rsf`
produces a file `crocopat-2.1.4.rsf`, which contains the co-change graph that CCVisu extracted from table sheet 'CO-CHANGE'.

Doxygen (DOX) to RSF Converter. For example, given a directory `crocopat-2.1.4-dox` of XML files that was produced using the command `doxygen Doxyfile`, where `Doxyfile` is an example configuration file for Doxygen. (Doxygen usually generates a directory `xml` which contains a file `index.xml`.) Then the command
`ccvisu.sh -inFormat DOX -i crocopat-2.1.4-dox/index.xml -outFormat RSF -o crocopat-2.1.4.rsf`
produces a file `crocopat-2.1.4.rsf`, which contains the relations that CCVisu extracted from the Doxygen output.

CVS to RSF Converter. For example, given a file `crocopat-2.1.log`, which is a CVS log file of CrocoPat and was produced using the command `cvs log -Nb`. Then the command
`ccvisu.sh -informat CVS -i crocopat-2.1.log -outformat RSF -o crocopat-2.1.rsf`
produces a file `crocopat-2.1.4.rsf`, which contains the co-change graph that CCVisu extracted from the CVS log file. As example for how an RSF file looks like we show an extract from the generated co-change graph of CrocoPat:

```
CCG    125    src/bddSymTab.h
CCG    126    src/Makefile
CCG    126    src/relLex.l
CCG    127    src/crocopat.cpp
CCG    127    src/relExpression.h
CCG    127    src/relLex.l
CCG    127    src/relYacc.y
```

Subversion (SVN) to RSF Converter. TODO

Other Fact Extractors. There are many tools available for graph extraction from software systems. Examples are: cawk$_{cg}$, GCT, Imagix, Rigiparse, Field, cflow, CIA, LSME$_{cg}$, Mawk$_{cg}$, Portable Bookshelf (grok), Doxygen, DepFinder, java2rsf.

Comparison of Fact Extractors. A good (but somewhat outdated) comparison of fact extractors was done by Murphy et al. [MNGL98].

2.2 Generating Layouts / CCVisu as Graph-Drawing Tool

Generate a Layout. Once we have the input graph in RSF or in one of the above mentioned formats, the next step is to run CCVisu on the input graph in order to produce a (clustering) layout for the software graph. For example, consider the dependency graph of a (small) compiler system (extracted using Bauhaus) in file `compiler.rsf`. If we run CCVisu using the command line
`ccvisu.sh -informat RSF -i compiler.rsf -outformat DISP -iter 100`,
a screen display will open and show the resulting layout, running 100 iterations of the minimizer. Since RSF is the default input format, and DISP (screen DISPlay) is the default output option, and by default it runs 100 iterations, the following command does the same: `ccvisu.sh -i compiler.rsf`

Save a Layout. CCVisu supports several different output formats: LAY (plain text format), SVG (2D Scalable Vector Graphics), VRML (3D format). Any layout that CCVisu shows on the screen can be saved to a file in one of these formats, by opening the corresponding SAVE dialog and giving a filename to store the layout in the file. The filename extension .lay, .svg, or .wrl determines the file format.

2.3 Introduction to Force-Directed Graph Layout

Force-directed graph layout consists of two parts: an energy model that assigns an energy value to each layout for evaluation —the smaller the number, the better the layout—, and a minimizing algorithm that computes a layout with minimal energy.

A tool for force-directed graph layout takes as input a graph (undirected, connected, irreflexive). Another parameter of the tool is the energy model, which encodes the user's requirements for 'good' layout, e.g., uniform edge-length or interpretable distances. The output of the tool is a layout that has minimal energy according to the energy model.

CCVisu is such a tool for force-directed layout. It uses the well-known algorithm of Barnes and Hut as minimization algorithm (cf. Sect. 6.1), and it provides several standard energy models, and lets the user define (optional, per command line) certain parameters of the energy model (cf. Sect. 6.2), in order to adapt it to the user's application area.

2.4 Examples

Chain. Consider first a graph that forms a chain, i.e., a connected graph where the first and the last vertex has one edge and all other vertices have two edges. Figure 1 shows the graph (on the left) and the resulting layout (on the right). To produce this result, the tool CCVisu was invoked with the following command line (start java interpreter with class `CCVisu` of package `ccvisu`, read the input graph in RSF format from file `simple-chain.rsf`, and use the attraction exponent 3 in the energy model, cf. Section 6.2, Generic Model):

```
ccvisu.sh -i simple-chain.rsf -attrExp 3
```

An energy model usually consists of two terms, one is interpreted as the energy resulting from an attraction force, and the other is interpreted as the energy resulting from a repulsion force. In the example, the attraction force ensures that neighboring vertices in the chain have closed positions in the layout (e.g., vertices 1 and 2). The repulsion force between each two vertices ensure that vertices that are connected by a long path have distant positions (e.g., vertices 1 and 20).

V1V2. todo

3times two. todo

CrocoPat's Co-Change Graph. Now we consider the co-change visualization of the tool CrocoPat[1]. The input is the CVS log file of the version history, obtained using the command `cvs log` [2]. The co-change visualization can be either computed and displayed in one direct step:

```
ccvisu.sh -inFormat CVS -i crocopat-2.1.log -hideSource
```

or obtained by several steps (to store intermediate results for later reuse):

```
ccvisu.sh -inFormat CVS -i crocopat-2.1.log -outFormat RSF -o crocopat-2.1.rsf
```

computes the co-change graph and stores it into a text file in RSF format.

```
ccvisu.sh -inFormat RSF -i crocopat-2.1.rsf -outFormat LAY -o crocopat-2.1.lay -hideSource
```

computes the layout and writes it to a text file in LAY format. The option `-hideSource` hides the change transaction vertices in the visualization.

```
ccvisu.sh -inFormat LAY -i crocopat-2.1.lay -outFormat SVG -o crocopat-2.1.svg
```

converts the layout in LAY format to a vector graphics in SVG format.

[1]CrocoPat is a tool for relational calculation, available at ../../CrocoPat.

[2]Example files are available at ../ccvisu/examples.

```
R 1 2
R 2 3
R 3 4
R 4 5
R 5 6
R 6 7
R 7 8
R 8 9
R 9 10
R 10 11
R 11 12
R 12 13
R 13 14
R 14 15
R 15 16
R 16 17
R 17 18
R 18 19
R 19 20
```

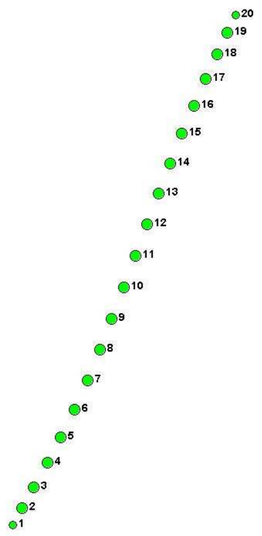

Figure 1: Chain example

3 Input Formats

There are five different input formats for the different purposes of CCVisu: the CVS/SVN log-file formats for extracting the co-change graph from a CVS/SVN repository, the Doxygen XML output format (DOX) for extracting the software relations from a source code directory after Doxygen had processed the directory, graphs in Relational Standard Format (RSF) to enable the use of CCVisu for any graphs or previously stored relations, and layouts in text format (LAY) (previously computed with CCVisu or a compatible tool).

3.1 CVS/SVN Log File (CVS/SVN)

CCVisu can be used to extract a co-change graph [BN05a], which is an abstraction of the history in a software repository, from either CVS log files in text format (produced with 'cvs log -Nb') or SVN log files in XML format (produced with 'svn log -v –xml'). CCVisu uses the log files to directly compute a co-change visualization, or to dump the co-change information into an RSF file, which can serve as input for another tool (if CCVisu is used as pure fact-extractor) or as input for CCVisu in a later processing phase. The mode of operation depends on the parameter `-outFormat`.

The version control system CVS does not directly keep the information about which files were checked-in together in the repository. The transactions need to be recovered from the logged information about time, user, and log message. The current CVS reader implements the heuristic used in cvs2cl (available at http://www.red-bean.com/cvs2cl) : it considers a sequence of changes of files as one change transaction if the changes have the same user login, the same log message, and time stamps that differ by at most 180 s (the constant can be adjusted by parameter -timeWindow). The co-change graph is extracted on file level. However, if a more fine-grained visualization is necessary (e.g., on method level), the techniques used in Rose [ZDZ03] can be integrated as additional reader. On the other hand, co-change graphs on higher levels (e.g., on package level) can be obtained by applying a technique called 'lifting'.

3.2 Doxygen File (DOX)

CCVisu can be used as fact-extractor for Doxygen XML files. For a given software system, Doxygen can be applied to the source-code directory in order to produce a directory of XML files that describe the structure of the software system (the option to produce XML output can be enabled in the Doxygen configuration file `Doxyfile`).

Given such an XML output directory, CCVisu can be applied to extract an RSF file that contains the most important software graphs, such as the inheritance graph, containment graph, and call graph:

```
ccvisu.sh -inFormat DOX -i xml/index.xml -outFormat RSF -o DOXgraphs.rsf
```

The main XML file in this case is `xml/index.xml`, and the following relations are written to the file `DOXgraphs.rsf`.
Basic Doxygen relations:

```
COMPOUND     <compound-kind>    <compound-id>       <compound-name>
MEMBER       <member-kind>      <member-id>         <member-name>
CONTAINEDIN  <member-id>        <compound-id>
BASEDON      <compound-id>      <basecompound-id>   <protection-kind>  <virtual-kind>
REFERSTO     <member-id>        <referredmember-id>
LOCATEDAT    <compound-id>      <file-path>         <line-no>
LOCATEDAT    <member-id>        <file-path>         <line-no>
```

Derived relations:

```
REF<kind>    <referrer-name>         <referred-name>
refFile      <referrer-file-name>    <referred-file-name>
refClass     <referrer-class-name>   <referred-class-name>
```

The relation `REF<kind>` is derived from relation `REFERSTO`, where the ids are replaces by their names and the second element is of the category `<kind>`, e.g., `REFvariable`, `REFfunction`, ... The category `<kind>` can be any of the following: `define, property, event, variable, typedef, enum, enumvalue, function, signal, prototype, friend, dcop, slot`.

The relation `refFile` (`refClass`) is derived from relation `REFERSTO`, where the ids are replaced by the names of the containing files (classes), i.e., this relation is the 'lifting' of the `REF<kind>` relation to the file (class) level.

3.3 Graph (RSF)

Graphs are provided in RSF format. This format is used to provide co-change graphs, which were previously computed by CCVisu or other extraction tools, or graphs in general (e.g., graphs representing the static structure of a software system, gene expression networks, etc.).

Each line in an RSF file represents an edge in the following format:

```
<graph name> <source> <target> <weight>
```

For example, the line

```
CALL A B 0.5
```

represents an edge between vertices `A` and `B` of weight `0.5`.

Graph requirements. Input graphs for force-directed graph layout must be irreflexive (no self-edges) and connected (no isolated subgraphs). The graph must be connected because for most energy models, the distance of two unconnected vertices is infinite in a layout with minimal energy. A software system consisting of several unconnected components must be visualized using several layouts, one for each component (small unconnected components are usually skipped because of its unimportance).

3.4 Layout (LAY)

For the purpose of transforming a given layout to the VRML or SVG format, or to display the layout on the screen, CCVisu accepts layouts in the layout text format that is described in Section 4.2 (output formats).

4 Output Formats

CCVisu produces two kinds of results: co-change graphs, when used as a pure extractor of co-change relations from the CVS repository, or graph layouts in different formats (text format LAY, VRML, SVG, and directly drawn on the screen).

4.1 Graph (RSF)

When used as a co-change graph extractor only, CCVisu stores the graph in RSF format for further use (the RSF format is described in Section 3.3).

4.2 Layout (LAY)

CCVisu can save layouts in its own text format, either by specifying LAY as output format or by pressing the save-button while viewing a screen layout using the DISP output format. The tool can produce all other provided layout formats from the LAY format.

Each line contains the data needed to represent a single vertex. For example, the line

```
LAY  -126.0  -72.0  0.0  32  src/Makefile  255  false
```

represents a vertex with name `src/Makefile` at coordinates `x=-126.0`, `y=-72.0`, `z=0.0`, edge degree `32`, drawn in color `255` (which is blue, in usual 3 byte RGB code), and the name is by default not annotated (`false`). ('LAY' is the name of the relation if the file is interpreted in RSF format.)

4.3 Displaying the Layout

This section describes common features of the different possibilities for actually viewing the computed layout. The vertices are drawn as filled circles in the visualizations. Edges are always omitted. The area of the filled circles is proportional to the degree of the corresponding vertex.

All three output options support the following mouse feature to show the names of vertices:

- If the mouse pointer is moved on a vertex, the name annotation is shown. After the mouse pointer leave the vertex, the annotation is removed.
- If the mouse pointer points to a vertex and the mouse button is clicked, the name annotation is permanently added to the vertex. Clicking again on a vertex removes the annotation.

4.3.1 Layout (VRML)

CCVisu supports the VRML graphics format as output format
(cf. http://tecfa.unige.ch/guides/vrml/vrml97/spec/
or http://tecfa.unige.ch/guides/vrml/pointers.html).

For VRML viewers we refer to the VRML Viewers, Browsers and Plug-ins web page of the *Web 3D Consortium* (http://www.web3d.org/x3d/vrml/tools/viewers_and_browsers/), or, e.g., directly to the *Cortona VRML Client* (http://www.parallelgraphics.com/products/cortona/), which is a plug-in for some standard web browsers.

However, VRML viewers are not appropriate for viewing large layouts (thousands of vertices).

4.3.2 Layout (SVG)

CCVisu supports the SVG graphics format as output format
(cf. http://www.w3.org/TR/SVG/ or http://www.w3.org/Graphics/SVG/).

For SVG viewers we refer to the SVG Implementations web page of the *World Wide Web Consortium* (http://www.w3.org/Graphics/SVG/SVG-Implementations), or, e.g., directly to the *Adobe SVG Viewer* (http://www.adobe.com/svg/), which is a plug-in for some standard web browsers.

4.3.3 Layout on Screen (DISP)

The output-format option DISP provides the display of layouts without any additional viewer software, and at the same time this is the only possibility for viewing layouts up to a million vertices (where even an SVG viewer breaks down).

In difference to the above mentioned viewing techniques, the direct display annotates the vertex name (if mouse pointer moves on a vertex) in a separate frame to avoid repainting the layout. The separate frame also contains a SAVE button to save the layout using the LAY output format.

5 Tool Overview and Architecture

Input/Processing/Output (black-box view. Figure 2 shows the more general usage of the tool. The input is either (1) a CVS log file —extracted from the CVS version repository with the command `cvs log`—, or (2) a textual representation of the co-change graph in Rigi Standard Format (RSF) to compute layouts for co-change graphs extracted from other version control systems. To display a previously computed layout, the input can also be (3) a text file containing the layout (LAY).

After the input graph is read, the tool computes the layout for the vertices of the graph. This part is done by an implementation of an energy model and an algorithm that computes a layout that has minimal energy according to the energy model.

At the end, the graph needs to be displayed on the screen, or written to a file. The layout of the artifacts can be produced in three forms. (1) The text file (LAY) can later be read by CCVisu or other tools, such that the tool can be embedded in different environments. (2) The VRML format allows the use of an external VRML viewer (or a web browser with VRML plug-in) to view the layout, and is enabled for 2D as well as 3D layouts. Artifacts are drawn as spheres, and when the mouse pointer moves on the artifacts the name is annotated. (3) The layout can be directly displayed on the screen. This form is the preferred output method for huge graphs, when a VRML viewer is not able to reproduce the layout on the screen. Only two-dimensional layouts are supported. Artifacts are drawn as filled circles, and the names of the artifacts are shown in a separate frame when the mouse pointer moves onto an artifact, or permanently annotated to the artifact via mouse click. (4) Besides the layouts, the tool can also output the extracted co-change graph in RSF format.

Framework (white-box view). CCVisu is designed as a framework to make improvements and extensions easy, and to enable integration into other reengineering tools. Figure 3 shows the components of the tool. Basically, the input graph is read by a reader component, passed to the layout algorithm, and the output is written by a writer component. The reader interface has currently three implementations: for reading CVS log files, co-change graphs in RSF, and layouts in text format. The writer interface has five implementations so far, for writing co-change graphs in RSF format, layouts in text format, VRML format, SVG format, and for writing the layout directly to the screen.

Using these flexible input/output formats, the tool can be used as a general tool for force-directed graph layout, not only for co-change graphs. To provide more control over the concrete layout computation, the minimizer algorithm and the energy model themselves are also abstract components. Currently, CCVisu includes the Barnes-Hut algorithm [BH86] as minimizer, and the following energy models for the evaluation of layouts: the Fruchterman-Reingold model [FR91], the LinLog model [Noa04a], the edge-repulsion LinLog model [BN05a, Noa04b], and the weighted edge-repulsion LinLog model.

API. The integration of CCVisu into other tools is possible either by invoking it as a command line tool from a shell, or by invoking the methods of the Java classes using CCVisu's API.

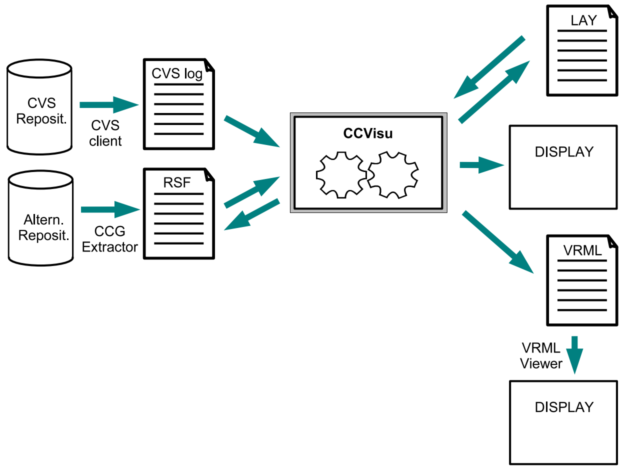

Figure 2: CCVisu's input/output interface

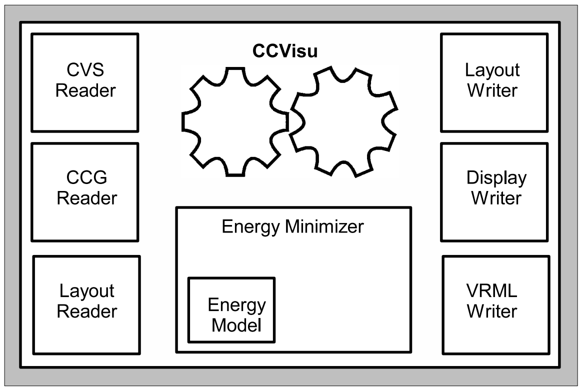

Figure 3: CCVisu's framework architecture

6 Force-Directed Graph Layout

6.1 Minimizing Algorithm

An algorithm for minimizing the energy of the model usually works in the following way: It starts with an initial layout, where the positions of the vertices are randomly assigned. Then, in every iteration, the algorithm tries to improve the layout according to the energy model (by using the first derivation of the energy function to compute a direction and a distance for the movement of each vertex). Since the graphs we have to deal with are usually large (especially software graphs), we cannot afford to use algorithms with complexity in $O(|V|^2)$ per iteration. The algorithm of Barnes and Hut [BH86] is in $O(|E| + |V| \log |V|)$ per iteration, and is therefore sufficient for our purposes.

6.2 Energy Models

The energy model encodes decisions about what is considered to be a good layout. In the following, we briefly introduce some of the energy models that are supported by CCVisu. First give the concrete models, and at the end we explain our generic model that can be instantiated for each of the concrete models.

We use the following notation: A graph $G = (V, E)$ consists of a set of vertices V and a set of edges $E \subseteq V^{(2)}$, where $V^{(2)} = \{\{u, v\} \mid u, v \in V\}$ contains all sets with two vertices (i.e., undirected edges). A layout $p : V \rightarrow \Re^d$ is a function that assigns to each vertex a position in the d-dimensional space ($d \in \{2, 3\}$ is the number of dimensions, $\Re$ is the set of real numbers), P is the set of all layouts. An energy model $U : P \rightarrow \Re$ is a function that assigns to each layout a real number (the smaller the value the better the layout). For a given layout p and two vertices u and v, the term $||p(u) - p(v)||$ denotes the Euclidean distance of the two vertices.

6.2.1 Fruchterman Reingold

The energy model of Fruchterman and Reingold was designed to enforce layouts with uniform edge length [FR91]. It requires graph with a large diameter (or graph-theoretic distance), for which it produces esthetic graph layouts. The energy for a layout p is defined as:

$$U(p) \quad = \quad \sum\nolimits_{\{u,v\} \in E} \frac{1}{3}\, ||p(u) - p(v)||^3 \quad + \quad \sum\nolimits_{\{u,v\} \in V^{(2)}} - \ln ||p(u) - p(v)||$$

The first term of the sum is interpreted as attraction between connected vertices, because its value decreases when the distance of such vertices decreases. The second term is interpreted as repulsion between all pairs of (different) vertices, because its value decreases when the distance between any two vertices increases.

6.2.2 Vertex-Repulsion LinLog

The vertex-repulsion LinLog energy model was designed for visual graph clustering, i.e., to produce layouts that fulfill certain clustering criteria [Noa04a]. It was the first energy model that was explicitly designed for computing clustering layouts (of software graphs). This model requires graphs with uniform edge degree to produce good layouts. The energy for a layout p is defined as:

$$U(p) \quad = \quad \sum\nolimits_{\{u,v\} \in E} ||p(u) - p(v)|| \quad + \quad \sum\nolimits_{\{u,v\} \in V^{(2)}} - \ln ||p(u) - p(v)||$$

6.2.3 Edge-Repulsion LinLog

The edge-repulsion LinLog energy model is an extension of the vertex-repulsion LinLog model to overcome the limitation to graphs of uniform edge degree. It was successfully used in the initial study of co-change visualization [BN05a]. Noack provides a detailed introduction and comparison with the vertex-repulsion LinLog model in the technical report [Noa04b]. For a comparison with the Fruchterman-Reingold model we refer to Section 7 (or to the technical report [BN05b]). The energy for a layout p is defined as (the edge degree of a vertex v is denoted by $\deg(v)$):

$$U(p) \quad = \quad \sum\nolimits_{\{u,v\} \in E} ||p(u) - p(v)|| \quad + \quad \sum\nolimits_{\{u,v\} \in V^{(2)}} - \deg(u)\ \deg(v)\ \ \ln ||p(u) - p(v)||$$

In this energy model, the second term can be interpreted as repulsion between all pairs of edges (more precisely, between the end vertices of the edges).

6.2.4 Weighted Edge-Repulsion LinLog

The weighted edge-repulsion LinLog model is defined for graphs with weighted edges, as a straight forward extension of the edge-repulsion LinLog model. A weighted graph $G = (V, E, w)$ consists of a set of vertices V, a set of edges $E \subseteq V^{(2)}$, and a function $w : E \rightarrow \Re$, which assigns a real number (edge weight) to each edge. (A graph is a weighted graph with $w(e) = 1$ for all edges $e \in E$.) The weighted edge degree of a vertex v is $\deg_w(v) = \sum_{\{u,v\} \in E} w(\{u, v\})$. The energy for a layout p is defined as:

$$U(p) \quad = \quad \sum\nolimits_{\{u,v\} \in E}\ w(\{u, v\})\ ||p(u) - p(v)|| \quad + \quad \sum\nolimits_{\{u,v\} \in V^{(2)}} - \deg_w(u)\deg_w(v) \ln ||p(u) - p(v)||$$

6.2.5 Generic Model

The tool CCVisu implements a generalization of the above mentioned energy models by using three parameters, which can be used to adjust the actual energy model by command-line options. (1) Option `-attrExp a` applies value a as exponent to the distance in the attraction term. (2) Option `-repuExp r` applies value r as exponent to the distance in the repulsion term if $r \neq 0$, and applies the logarithm to the distance in the repulsion term if $r = 0$. (3) Option `-vertRepu` eliminates the edge-weight factor in the repulsion term by setting e to value 0. The default values are $a = 1$, $r = 0$, and $e = 1$, i.e., the default energy model is the weighted edge-repulsion energy model. Table 1 describes how to set the parameters for each above-mentioned energy model.

For a weighted graph $G = (V, E, w)$, the energy for a layout p is defined as:

$$\begin{aligned}
if r = 0: \quad U(p) &= \sum\nolimits_{\{u,v\} \in E} \; w(\{u,v\}) \; \frac{1}{a} \; ||p(u) - p(v)||^a \\
&+ \sum\nolimits_{\{u,v\} \in V^{(2)}} - (\; \deg_w(u) \deg_w(v) \;)^e \; \ln ||p(u) - p(v)|| \\
if r \neq 0: \quad U(p) &= \sum\nolimits_{\{u,v\} \in E} \; w(\{u,v\}) \; \frac{1}{a} \; ||p(u) - p(v)||^a \\
&+ \sum\nolimits_{\{u,v\} \in V^{(2)}} - (\; \deg_w(u) \deg_w(v) \;)^e \; ||p(u) - p(v)||^r
\end{aligned}$$

with the attraction exponent $a \in \Re$, the repulsion exponent $r \in \Re$, and the edge-repulsion factor $e \in \{0, 1\}$.

Energy model	attrExp a	repuExp r	vertRepu e
Fruchterman Reingold	3	0	0
Vertex-repulsion LinLog	1	0	0
Edge-repulsion LinLog	1	0	1
Weighted edge-repulsion LinLog	1	0	1

Table 1: Parameters for the generic energy model

The energy models for clustering software graphs —e.g., co-change graphs and call graphs— have to fulfill several clustering criteria, in particular, it should separate clusters and lead to interpretable distances. In difference to other graph-drawing applications, the energy models for software graphs must not enforce uniform edge length, must not be biased to the size of the clusters, and must be normalized to non-uniform degrees of the vertices. That is why we use the weighted edge-repulsion LinLog energy model (or its unweighted version) as the standard energy model. However, the generic energy model allows to express the many important energy models for software graphs in the current implementation, and the tool CCVisu is easy to extend to further variants and completely different energy models.

7 Co-Change Graphs

Co-change visualization is a method to compute clustering layouts based on the change history of the system. Intuitively, we want to compute layouts where two artifacts have close positions if they were often changed together, and they have distant positions if they were rarely commonly changed. We model the system's change history as the so called co-change graph, which is described in Section 3.1. Then, the usual graph-layout algorithm can be applied to compute a layout (cf. [BN05a] for details). The precondition for achieving good layouts is to use an energy model that fulfills certain clustering properties (cf. the discussion in Section 6.2).

The motivation for using the co-change graph is threefold: First, frequently co-changed artifacts are likely to be logically coupled, and grouping them together in one subsystem restricts the scope of changes to the local context. Second, the co-change graph is not limited to program source code, unlike call graphs and other syntax-based models; the co-change graph includes also artifacts for test data, shell scripts, SQL scripts, examples, documentation, and subsystems in different programming languages. Third, the co-change graph can be efficiently and inexpensively extracted from version control repositories.

The (weighted) co-change graph for a given version-control repository is an undirected graph $G = (V, E, w)$. The set V of vertices represents the artifacts of the system (e.g., files, classes, methods, packages) and change transactions (e.g., commits in CVS). An edge $\{c, a\}$ is contained in the set E of edges if artifact a was changed by change transaction c (also called 'commit'). The weight $w(\{c, a\})$ of an edge is interpreted as the importance of the edge. For an unweighted graph, the weight is 1 for all edges. A detailed discussion on edge weights for co-change graphs is given in the technical report [BN05b].

Information provided by the visualization. The layouts produced by the tool CCVisu provide information on two levels:

- If the co-change graph contains clusters, then —due to the clustering quality of the energy model— the clusters are separated. Therefore, on the higher level, it reveals the subsystem structure of the system if the repository information allows so, and provides an overview over the relationships between the subsystems on the coarse level.

- Artifacts that were often changed together are placed closed together in the layout. Therefore, on the lower level, the engineer can use the visualization to find out in which context the artifact is used, which other artifacts need to be understood to understand the artifact, and if the artifact needs to be changed, it provides the artifacts that are most likely to change as well in the close neighborhood of the artifact.

The visualization can, for example, provide some guidance for answering concrete questions like the following:

High level: What are the subsystems of the system, according to common changes? If there is a decomposition into subsystems available, does it match the subsystems suggested by the co-change visualization? (If not, what are the reasons?) If we want to restructure the system, what do the clusters in the co-change layout suggest? Are there files that need to be assigned to other subsystems, which they are closed to in the layout?

Low level: Which SQL query files correspond to which module of the system? Which test input file is related to which code file? Which configuration file corresponds to which module files? If we change a certain file, which files should we understand because of potential impact? If we are interested to unterstand a certain code file, which documentation file shall we read? If we want to test a certain part of the program, which example files and test cases are closely related to the source file of that part?

Example Visualization. We have applied the CCVisu method to the well-known software project Mozilla, in particular to the *mailnews* component without the base package. The co-change graph was extracted from a CVS log file with 270 000 lines (13 MB). In this example, the artifacts of the co-change graph are files. The graph consists of 1 804 artifact vertices, 9 950 vertices for change transactions, and 30 938 edges (changes). Figure 4 shows a screen-shot of the layout, which was computed within 5 min on a 1.7 MHz Pentium machine, using only 100 iterations of the minimizer.

The vertices for the change transactions and the edges are omitted for readability. The artifact vertices were drawn in different colors, in order to compare the grouping suggested by the layout with the authoritative decomposition, according to the documentation. We considered 8 major subsystems of the *mailnews* component and assigned colors to them: AddrBook (blue), Compose (magenta), IMAP (pink), MAPI (yellow), MIME (red), Import (cyan), DB (orange), and Extensions (gray). The rest (minor components, build utils, etc.) is labeled as Misc (green) in the figure. (The subsystem labels are also annotated in gray boxes, to improve readability for gray-scale printouts.) Now we can compare whether CCVisu has positioned the 1 804 files in groups in agreement with the authoritative decomposition: Some of the subsystems are clearly separated from the rest (Extensions, IMAP, DB, MAPI, AddrBook), some are not separate clusters but almost all files of the same subsystem are closed together (Import, MIME, Compose), and Misc is not grouped at all (as expected).

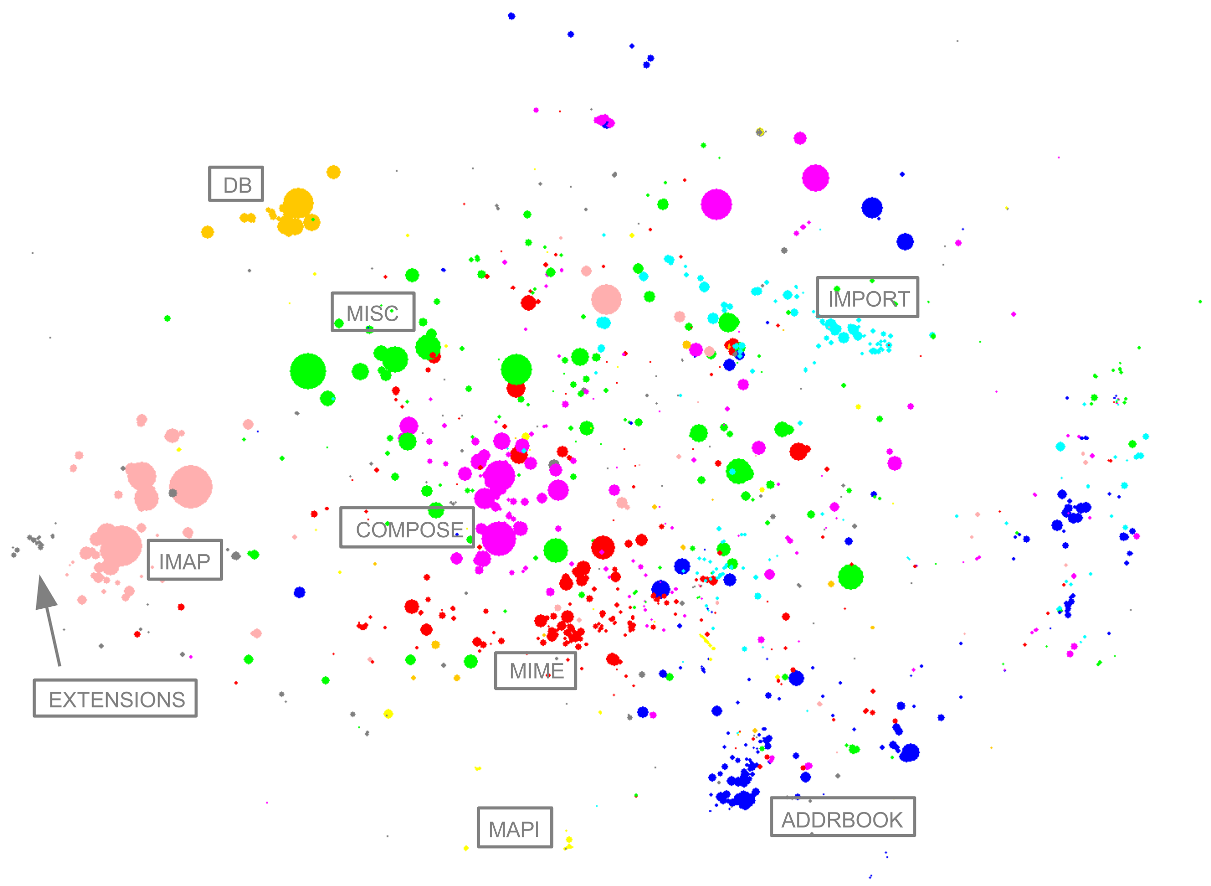

Figure 4: Co-change visualization of MOZILLA's mailnews component

Comparison of two Energy Models. Table 2 compares layouts of co-change graphs created with the edge-repulsion LinLog energy model (described in Section 6.2.3) and with the standard force model of Fruchterman and Reingold (cf. [1] or Section 6.2.1). The figures show that the Fruchterman-Reingold model separates clusters less clearly (because it enforces uniform edge lengths) and has a strong bias towards placing vertices with high degree (i.e., vertices that participated in many change transactions and are drawn large in the figures) in the center (because it models vertex repulsion instead of edge repulsion). This is typical for state-of-the-art force and energy models, because they are not primarily designed for clustering.

SW system	CC graph	Edge-Repulsion LinLog			Fruchterman-Reingold		
CrocoPat 2.1	crocopat.rsf	crocopat.png	crocopat.svg	crocopat.wrl	crocopat-FR.png	crocopat-FR.svg	crocopa
Rabbit 2.1	rabbit.rsf	rabbit.png	rabbit.svg	rabbit.wrl	rabbit-FR.png	rabbit-FR.svg	rabbit-F
Blast 1.1	blast.rsf	blast.png	blast.svg	blast.wrl	blast-FR.png	blast-FR.svg	blast-F

Table 2: Comparison of energy models

For each software system, the table provides the co-change graph (RSF), the layouts created with the edge-repulsion LinLog model, and the layouts created with the Fruchterman-Reingold model. For a detailed explanation of the formats and how to get viewers, we refer to Section 4. The PNG files are static pictures that can be viewed with standard web browsers. The WRL (VRML) files can be viewed with a VRML viewer (cf. Section 4.3.1). The advantage of the VRML files is that the names of graph vertices can be selectively shown and that one can navigate through the layout.

The artifact vertices are drawn as circles in the figures. The vertices for change transactions and the edges are omitted. The area of the circles is proportional to the degree of the corresponding vertice. The circles are colored according to the authoritative decomposition. Different subsystems in the authoritative decomposition correspond to different colors. (In the Blast visualizations, some colors are difficult to distinguish because of the large number of different colors.)

A discussion and interpretation of the layouts are given in the technical report [BN05b].

8 Related Work

A comprehensive discussion of the related work is given in our technical report [BN05b]. The two most related approaches are Bunch [MMCG99] and the work of Eick and Wills [EW93]. The novelty of CCVisu is twofold: First, it is based on the co-change graph, not on syntax-based models. (There are other approaches using change history, but not for clustering.) Second, it is based on an energy model that is designed for clustering layout.

References

[BH86] Josh Barnes and Piet Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324:446–449, 1986.

[BN05a] Dirk Beyer and Andreas Noack. Clustering software artifacts based on frequent common changes. In *Proc. IWPC*, pages 259–268. IEEE, 2005.

[BN05b] Dirk Beyer and Andreas Noack. Mining co-change clusters from version repositories. Technical Report IC/2005/003, EPFL Lausanne, 2005.

[EW93] Stephen G. Eick and Graham J. Wills. Navigating large networks with hierarchies. In *Proc. Visualization*, pages 204–210, 1993.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991.

[MMCG99] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proc. ICSM*, pages 50–59. IEEE, 1999.

[MNGL98] G. C. Murphy, D. Notkin, W. G. Griswold, and E. S. Lan. An empirical study of static call-graph extractors. *ACM Trans. Softw. Eng. Methodol.*, 7(2):158–191, 1998.

[Noa04a] Andreas Noack. An energy model for visual graph clustering. In *Proc. GD'03*, LNCS 2912, pages 425–436. Springer, 2004.

[Noa04b] Andreas Noack. Visual clustering of graphs with nonuniform degrees. Technical Report 02/04, BTU Cottbus, 2004.

[ZDZ03] Thomas Zimmermann, Stephan Diehl, and Andreas Zeller. How history justifies system architecture (or not). In *Proc. IWPSE*, pages 73–83. IEEE, 2003.