A simple method for generating automatic documentation of computer programs

Erkki Lukkari

10.9.2007

Contents

1 Introduction

This document contains examples of diagrams (here: directed graphs)

Each of them is generated in following way:

1. an input file is converted into a Libgraph syntax file
2. Libgraph syntax file is filtered by string and/or merged with other similar files (this phase is an option)
3. Libgraph file is processed by Graphviz program package producing an image file

Input can be source code, log files, map files, list files, makefiles etc.

Most parsing an conversion tasks are done by Perl scripts.
More challenging parsing tasks are supported by command line programs such as C-Vision and Algoview
Filterings and merges are done by legacy programs: grep and copy

Graphviz output image files can be selected to be JPEG, GIF or some of several alternatives.

Graphviz (www.graphviz.org) offers rich variety of image element shapes, colors, fonts, sizes and layouts.
Such features or image quality are mainly neglected in these example diagrams .

2 Dynamic diagrams

Data Flow Diagrams

Source code file *CVIS2DOT.CPP* contains following macro calls (also line numbers listed) among other executable lines.

197 _____c("-open cvision file")
198 _____c("open result dot syntax file")
233 _____c("new line check start")
244 _____c("found line with valid start")
249 _____c("invalid start line found")
258 _____c("further checks passed OK")
263 _____c("further checks not passed")
275 _____c("all on same line")
286 _____c("read second tree line")
311 _____c("parse tree line data")
345 _____c("build send node")

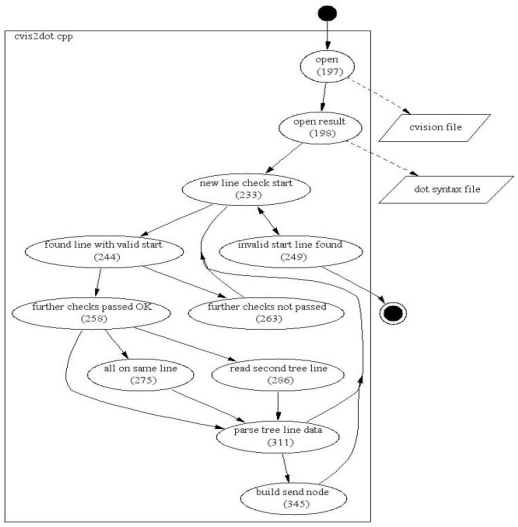
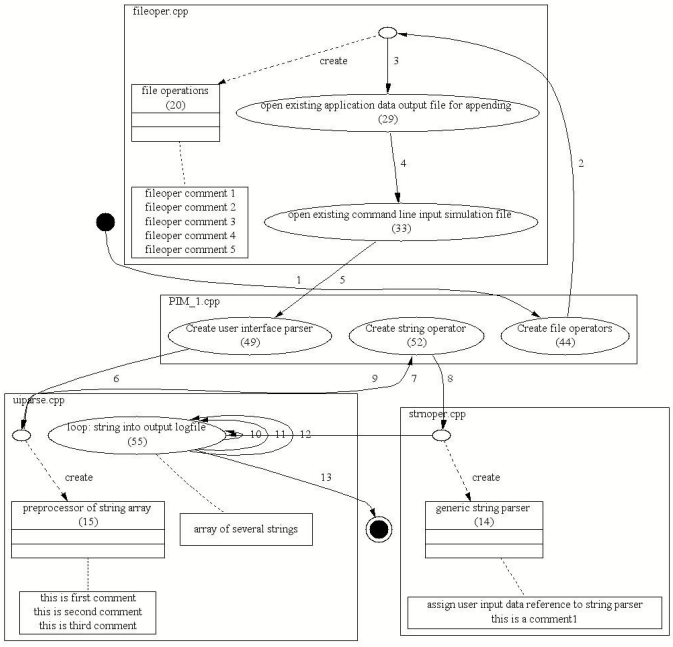


Fig 3-1 log file converted to Data Flow Diagram

Activity diagrams

More compicated macros can contain control syntax among commentary text.

```
File FILEOPER.CPP:
20      _____c("class: file operations");
21      _____c("// fileoper comment 1");
22      _____c("// fileoper comment 2");
23      _____c("// fileoper comment 3");
24      _____c("// fileoper comment 4");
25      _____c("// fileoper comment 5");
29      _____c("open existing application data output file for appending");
33      _____c("open existing command line input simulation file");
74      _____c("Read data from input simulation file");
File PIM_1.CPP:
44      _____c("Create file operators");
49      _____c("Create user interface parser");
52      _____c("Create string operator");
File STRNOPER.CPP:
14      _____c("class: generic string parser");
16      _____c("// assign user input data reference to string parser");
17      _____c("// this is a comment1");
File UIPARSE.CPP:
15      _____c("class: preprocessor of string array");
16      _____c("// this is first comment");
17      _____c("// this is second comment");
18      _____c("// this is third comment");
55      _____c("loop: string into output logfile");
56      _____c("// array of several strings");
```



Kuva 3-2 log file converted to Activity Diagram

Executed Functions

Every source code function contains a macro, which outputs current line number (using __LINE__) and file name, if it has has changed (using __FILE__)

Example of log file contents:

```
...
...
...
c:\project1\source\module2.cpp
153
1228
1501
7732
438
1501
1501
c:\project1\source\module1.cpp
338
2122
338
...
...
...
```

A script searches corresponding function names from source code.

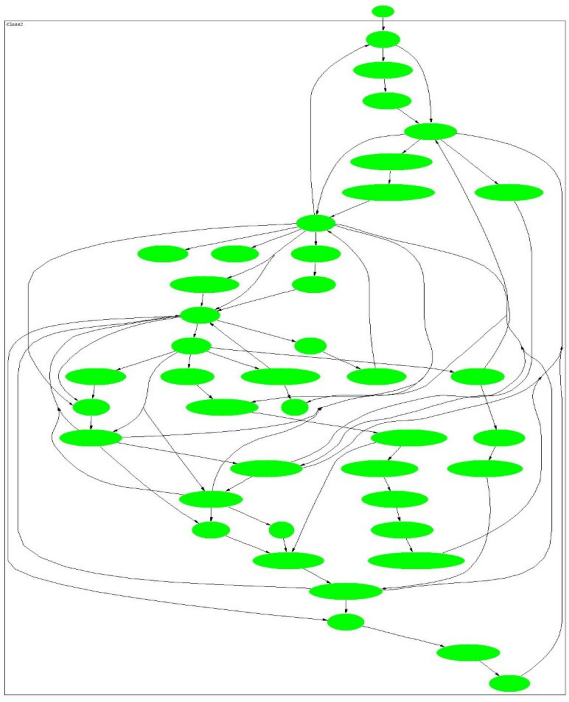


Fig 3-3 executed functions

Statechart Diagrams

If code contains state machines, writing state names to log file makes it possible to generate statechart diagrams.

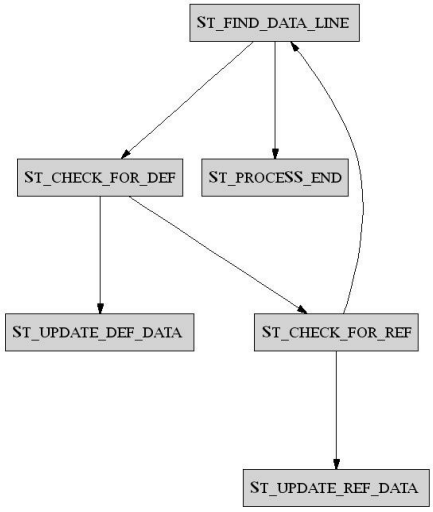


Fig 3-4

3 Static diagrams

Function calls

Fig 3-5 function calls (with parameters)

Data Structures

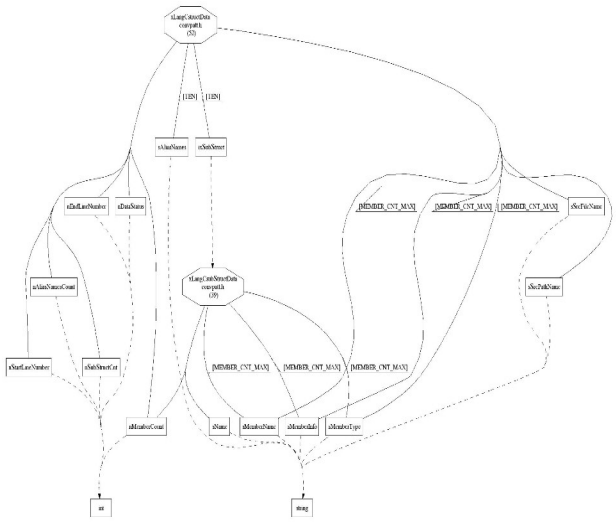
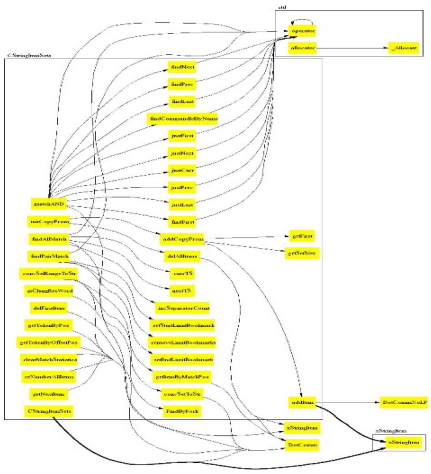


Fig 3-6 input: C++ header file

Method Calls



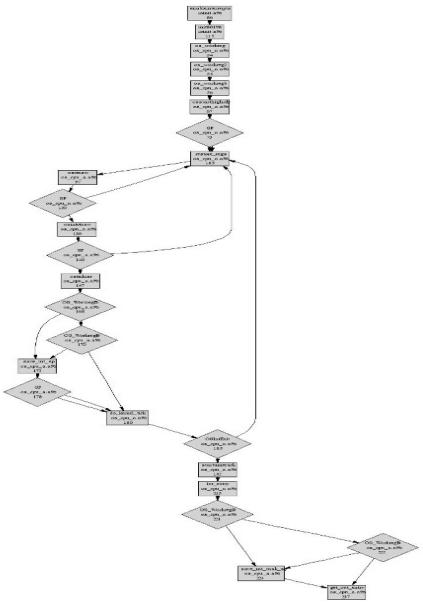
Kuva 3-7 input: Visual Studio compilation list file

Class Diagram

Kuva 3-8 input: C++ header file

Diagram contains classes and their associations (inheritance, aggregation and other types)
Methods and attributes are shown along their visibility (public, private, protected)

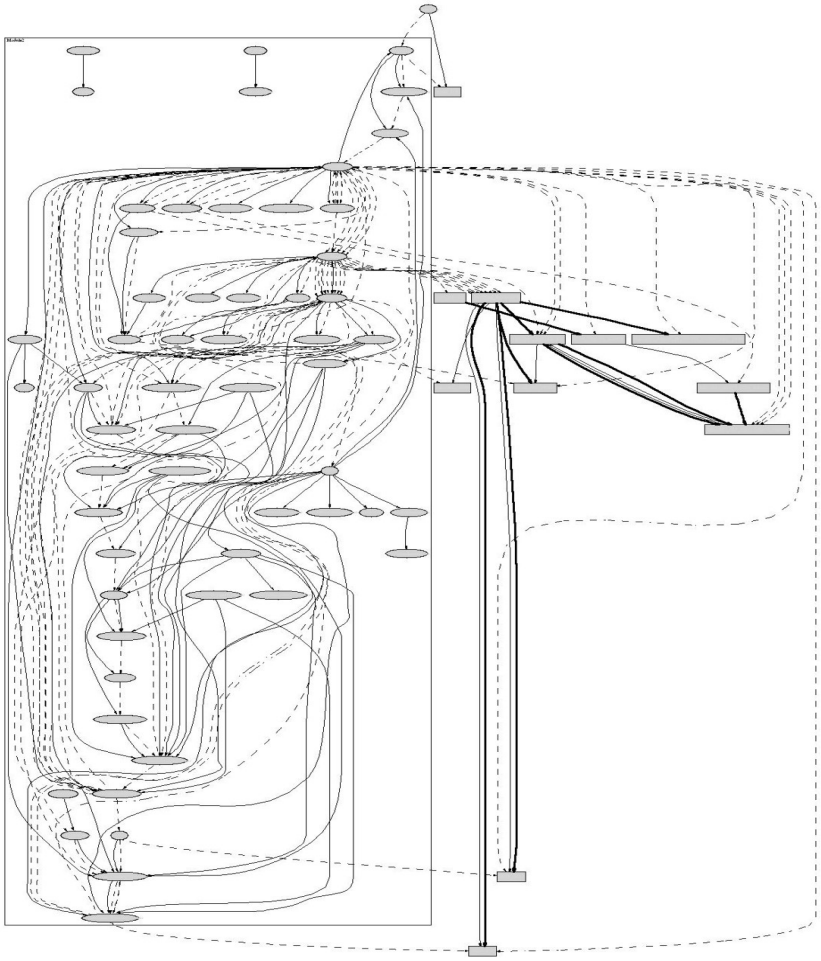
Flowcharts



Kuva 3-9 input: Assembler source code

4 Hybrid diagrams

Because same Libgraph syntax is used as an output of all these parsers, static and dynamic aspects can be combined into a single diagram.



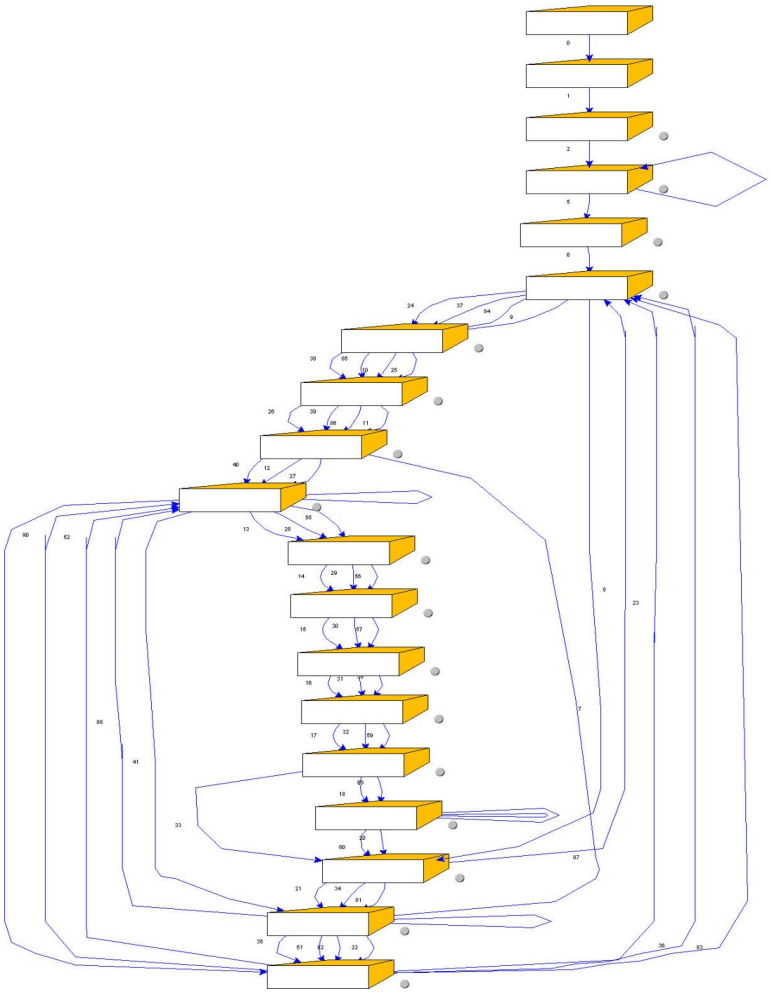
Kuva 3-10 two Libgraph syntax files merged

- ellipses are functions and boxes are states
- dashed lines are true sequence of executed functions
- thin solid lines are all possible function calls
- thick solid lines describe state transitions

This method can be used to describe static and dynamic combination diagrams, partial diagrams,partial diagrams of combination diagrams and combination diagrams of partial diagrams.

Interactive Diagrams

One of the file formats which can be produced by Graphviz, is that of Visual Thought, a Microsoft Visio -alike program



Kuva 3-11 State diagram generated from a log file

A grey spot can be seen close to every shaped box symbol. When clicking that with a mouse, corresponding file location is opeded in Notetab text editor