Getting started with the `gen_trace` software

Thomas Arts
Ericsson
Computer Science Laboratory

February 11, 2002

1 Installation

We assume Erlang release 8 to be installed `http://www.erlang.org/`. We expect the software also to work with release 7 and 6, but haven't tested this.

For the generation of traces you need to install the gen_trace.erl program (just compile the file and put it in your code path).

For the visualization of traces, as described in Chapter **??**, you need a graph drawing program. The following possibilities are supported:

- **DaVinci** a serious graph drawing tool for which the license only permits to use it for research purposes. Available for Solaris and Linux.
- **Scaled Vector Graphics** (svg) a standardized format to draw vector graphics. The format can be produced from the Erlang graph by the open source package GraphViz.

  Available from: `http://www.research.att.com/sw/tools/graphviz/`

  (We used version 1.7.15 for Sun Sparc Solaris 5.6+, but Linux, Windows and several unix systems are supported as well.)

  Visualizing the Scaled Vector Graphics format is possible with any SVG Viewer. We recommend Adobe's plug-in for web browsers, available from: `http://www.adobe.com/svg` for most platforms. Note that you might have to up-grade your browser to make it work.
- **Postscript** can be generated directly, but only works nicely for small graphs (that fit on a page, say). The software you need is again the open source package GraphViz.

  Available from: `http://www.research.att.com/sw/tools/graphviz/`.
- **Cæsar/Aldébaran Development Package** is a toolset for formal verification of software. If you have this package installed for other purposes, you might as well use it for visualizing graphs. We do not recommend installing the package for visualization only. Available from: `http://www.inrialpes.fr/vasy/cadp/`

2 Using the software

In order to give a feeling on how the software can be used, some callback modules for `gen_trace` are supplied. We present a little example in which we use `abs_code_server.erl` as callback module.

Code server example

Standard in the Erlang kernel is a generic server process running, which is called `code_server`. We take this code server as an example for the use of the `gen_trace` software.

2.0.1 Dumping

Make sure you have installed the generic trace module and that you are in a directory in which you can compile `abs_code_server.erl`. We now describe the process of making a trace and visualizing it in different ways. It would be interesting to see if this visualization increases your understanding of the `code_server` process.

- start recording a trace of the code server by typing: `gen_trace:dump([code_server],"cs.dump",{timeout,60})`. The trace of the process named `code_server` is stored in the file `cs.dump` and the trace is taken during 60 seconds. This might be far too much for you to perform the following operations, but then just wait a little longer in the end. In case the time is too short. Please remove the file `cs.dump` before you start again.
- In the Erlang shell, start triggering the code server process by typing

```
1> c(abs_code_server).
{ok,abs_code_server}
2> l(lists).

=ERROR REPORT==== 01-Feb-2002::15:49:43 ===
Can't load module that resides in sticky dir
{error,sticky_directory}
3> l(help).
{error,nofile}
4>
```

- After a while (about 60 seconds after starting the tracer), the message `Tracing has finished` should appear in the shell. Now the file `cs.dump` is created and the analysis can begin.

2.0.2 Abstracting

In order to view whether the file indeed contains information, one may use the functions `proc_info` and `show` in the `gen_trace` module. The aim of the tracing software is to visualize the trace in a better way than just textual. First of all we have the possibility of abstracting away from certain details. The trace as shown in Figure 1 is obtained by evaluating:

```
gen_trace:abstract(abs_code_server,"cs.dump",
                   [{tool,davinci,codeserver}]).
```

which created a file `codeserver.daVinci` that is saved to postscript. You could as well use

```
gen_trace:abstract(abs_code_server,"cs.dump",
                   [{tool,svg,codeserver}]).
```

to obtain a file `codeserver.svg` that can be viewed in your webbrowser (provided that you have installed the **dot** software and **svg** plug-in.

If you do not want to be bothered with installing graph drawing software before you have seen the result, try

```
gen_trace:abstract(abs_code_server,"cs.dump",
                   [{tool,dot,codeserver}]).
```

and look with your text editor in the `codeserver.dot` file.

What you see is still a trace of the system. However, The yellow boxes contain the abstracted state information of the generic server, the white boxes contain the sequence of events between a state transition. As you can see, the state does not change much (this is because we have not evaluated an `add_path` function in the `code` module). Thus, we could identify these states as one and the same abstract state, which has a number of possible transitions. This would give the graph as shown in Figure 2. The graph can be obtained using the same abstraction and an extra option to the `abstract` function:

```
gen_trace:abstract(abs_code_server,"cs.dump",
                   [{tool,dot,codeserver},{connect,graph}]).
```

Feel free to play around with the abstraction module `abs_code_server` to make different abstractions or use the same abstraction on different traces that you make with the `dump` function. Note that if you use the same name for the file to dump the trace in, you will append it to the already existing file.

3 Acknowledgements

Lars-Åke Fredlund from the Swedish Institute of Computer Science contributed to the generic tracing software by some practical tips and many good discussions.

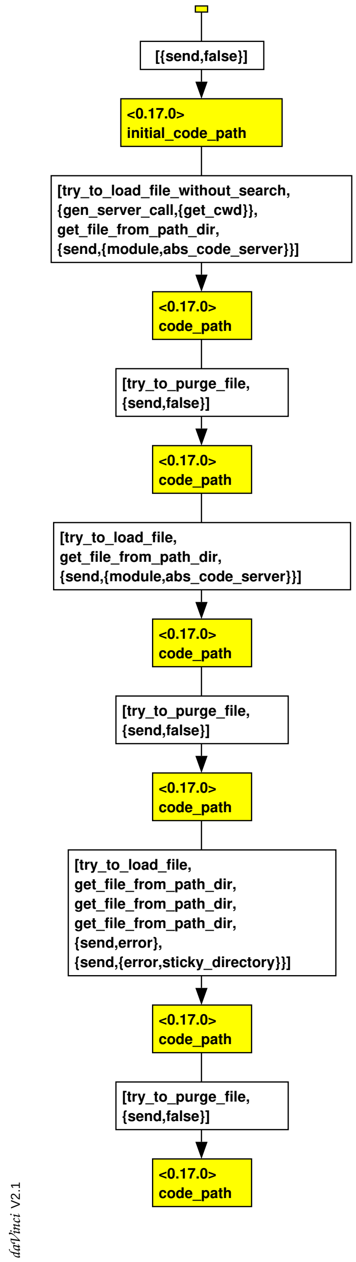

Figure 1: code server trace

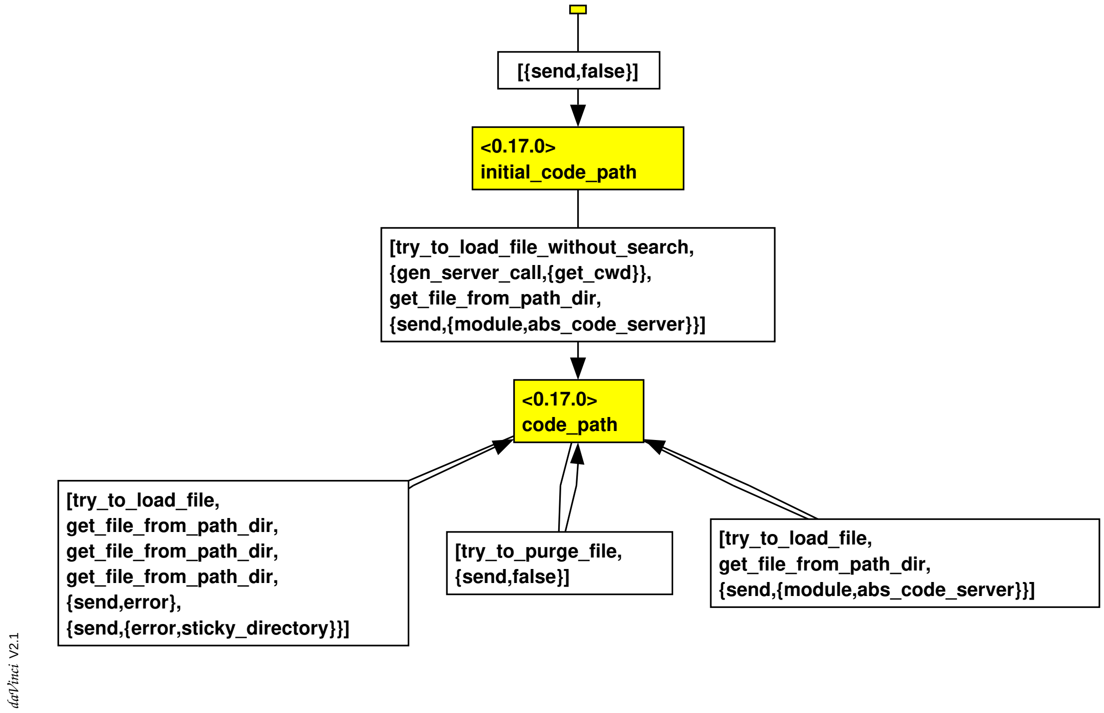

Figure 2: code server graph