

Code Quality Analysis Toolset for Embedded Systems

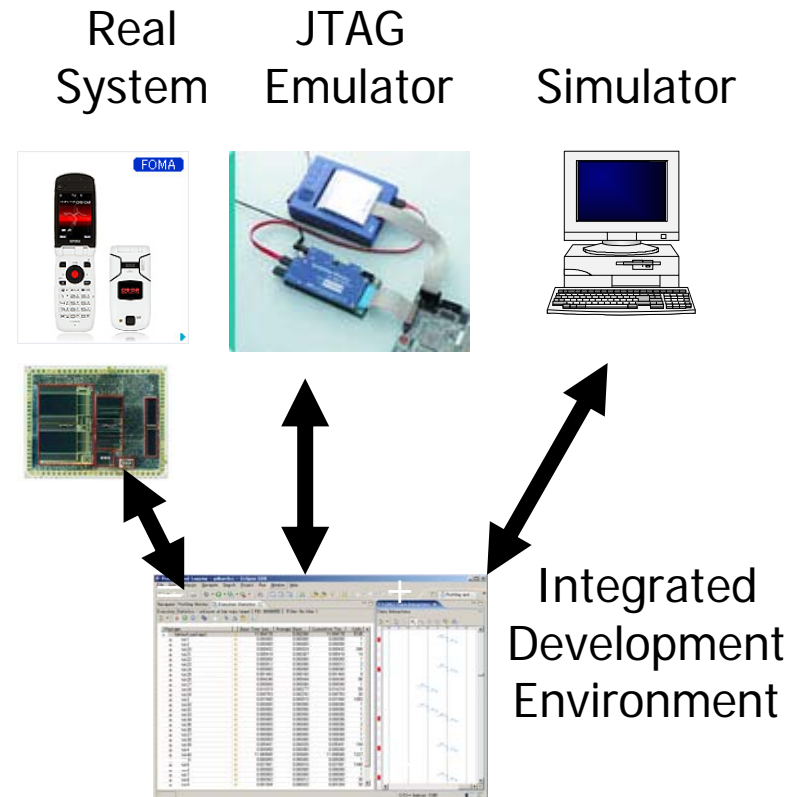
Kouhei NADEHARA

Yuichi NAKAMURA

System IP Core Res. Labs., NEC Corporation

Embedded System Development with Eclipse

- Deals with various “remote” target systems
 - ◆ Software simulator
 - ◆ Emulator via JTAG or USB
 - ◆ Real system via Ethernet connection
- Remote connection framework is required
 - ◆ Employ various existing tools
 - Open-source, GNU, or commercial tools
 - For performance and quality analysis



• How do we verify the quality of the target code?



Our Environment

Demonstration at EclipseCON 2007*

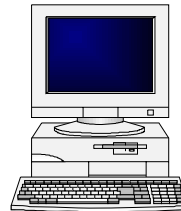
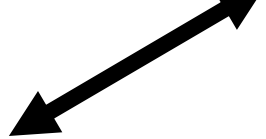
Small
Embedded
System



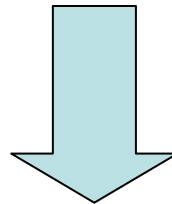
Remote
Connections



LAN



Linux



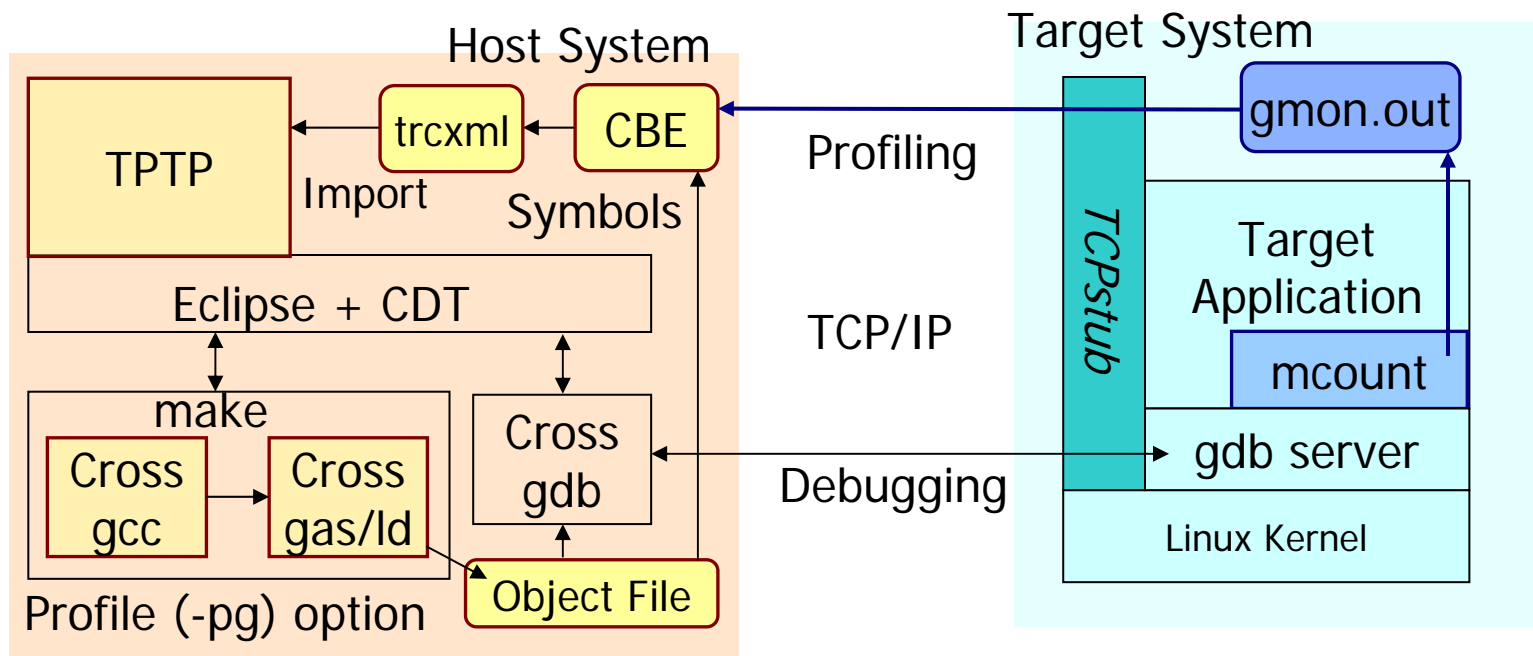
Thread names	Calls	Methods missed	Methods hit	% Methods hit
Unknown java applet	80	2	8	78.00%
main0	5	0	4	100.00%
org.eclipse.ui	25	0	4	80.00%
org.eclipse.ui	9	1	0	0.00%
org.eclipse.ui	5	0	1	100.00%
org.eclipse.ui	10	0	1	100.00%
org.eclipse.ui	10	0	1	100.00%
org.eclipse.ui	15	0	1	100.00%
org.eclipse.ui	20	0	1	100.00%
org.eclipse.ui	0	1	0	0.00%
org.eclipse.ui	0	0	0	0.00%

Eclipse front-end

- New in 2008: code quality analysis
 - ◆ Memory management with Valgrind (leak, cache statistics)
 - ◆ Code coverage testing with Gcov
 - ◆ TPTP-based and our original GUIs

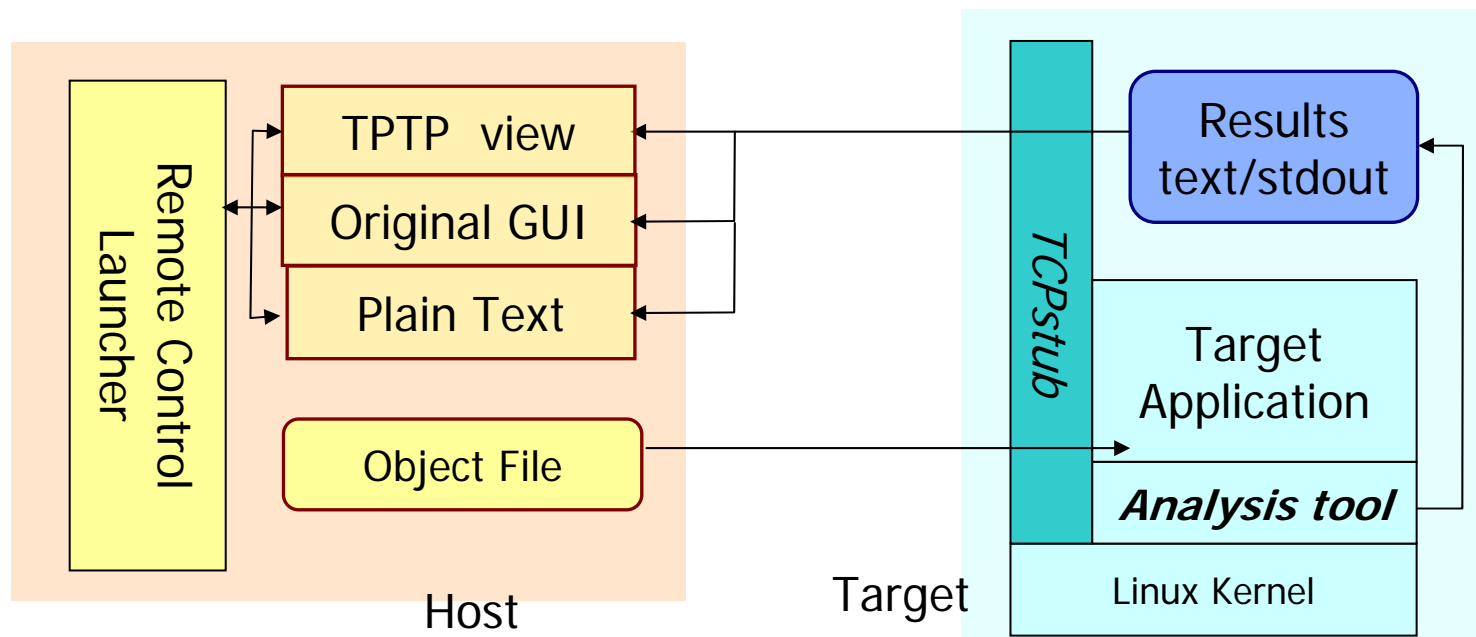
Base System: Remote Debugging/Profiling Architecture

- Communication managed by the “TCPstub”
- Remote debugging with gdb server and CDT
- Remote profiling with GNU profiler (gprof) and TPTP



Code Quality Analysis with "TCPstub"

- Network agent "TCPstub" manages communication
 - ◆ Invokes the application under an **Analysis tool** on the target
 - ◆ Retrieve its results to the host
- Results shown by TPTP view, or our original GUI



Remote Debugger Launcher

Select analysis tool

Select GUI

Get Profile data

Select Execution Mode.

Variable profile Get profile Check MEM error Check CACHE hit rate Check code coverage

Convert result to TPTP format

Target IP address(It's would put modules to target and get the result file from target)

192.168.14.119

Host PC working path name

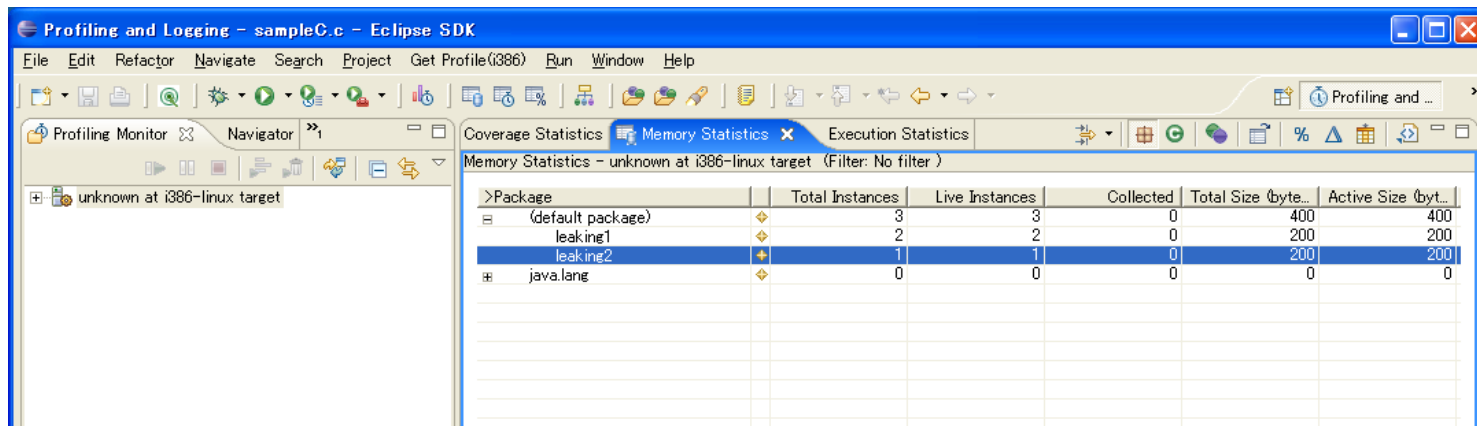
C:%pro_tool%cygwin%DEMO%i386%val-mem%

Target execute command

mem_leak

Memory Management with Valgrind

- Results shown by the TPTP memory statistics view



>Package	Total Instances	Live Instances	Collected	Total Size (byte...	Active Size (byt...
(default package)	3	3	0	400	400
leaking1	2	2	0	200	200
leaking2	1	1	0	200	200
java.lang	0	0	0	0	0

Because of the capability of the TPTP memory statistics view, results shown is memory leaks only.

Our GUI for Valgrind

Cache statistics and illegal memory accesses are shown by our original GUI

The screenshot shows a window titled "Cache Hit rate Information" with a table of cache statistics. The table has columns for index, function, file, lr, l1mr, mis hit rate, l2mr, mis hit rate, Dr, D1mr, mis hit rate, D2mr, mis hit rate, Dw, D1mw, mis hit rate, D2mw, and mis hit rate. The data is as follows:

index	function	file	lr	l1mr	mis hit rate	l2mr	mis hit rate	Dr	D1mr	mis hit rate	D2mr	mis hit rate	Dw	D1mw	mis hit rate	D2mw	mis hit rate
TOTALS	---	---	167904024	899	0.000535%	506	0.000301%	67153545	1301	0.001937%	685	0.001020%	16794303	262393	1.562393%	262393	1.562393%
1	main	./cache-hit.c	167809037	2	0.000001%	1	0.000001%	67121155	0	0.000000%	0	0.000000%	16781314	262144	1.562118%	262144	1.562118%

index	function	file	lr	l1mr	mis hit rate
TOTALS	---	---	167904024	899	0.000535%
1	main	./cache-hit.c	167809037	2	0.000001%

lr	l1mr	mis hit rate	l2mr	mis hit rate	Dr	D1mr	mis hit rate	D2mr	mis hit rate	Dw	D1mw	mis hit rate	D2mw	mis hit rate
167904024	899	0.000535%	506	0.000301%	67153545	1301	0.001937%	685	0.001020%	16794303	262393	1.562393%	262393	1.562393%
167809037	2	0.000001%	1	0.000001%	67121155	0	0.000000%	0	0.000000%	16781314	262144	1.562118%	262144	1.562118%

Cache statistics

```

==19298== Invalid write of size 1
==19298==    at 0x8048564: leaking1 (mem_leak.c:8)
==19298==    by 0x80485B9: main (mem_leak.c:22)
==19298== Address 0x402708D is 1 bytes after a block of size 100 alloc'd
==19298==    at 0x40046EA: malloc (vg_replace_malloc.c:149)
==19298==    by 0x804854B: leaking1 (mem_leak.c:6)
==19298==    by 0x80485B9: main (mem_leak.c:22)
--10700--
    
```

Illegal memory access

Gcov GUIs

Item names	Calls	Methods missed	Methods hit	% Methods Hit
<--Summary-->	80	2	6	75.00%
(default package)	80	2	6	75.00%
exp04	30	0	2	100.00%
main0	5		hit	
sub_proc10	25		hit	
exp04_sub1	0	1	0	0.00%
ext_sub_proc10	0	miss		
exp04_sub2	5	0	1	
ext_sub_proc20	5		hit	
exp04_sub3	10	0	1	
ext_sub_proc30	10		hit	
exp04_sub4	15	0	1	
ext_sub_proc40	15		hit	
exp04_sub5	20	0	1	

Function-level result shown by TPTP

Item names	Calls	Methods missed
<--Summary-->	80	
(default package)	80	
exp04	30	
main0	5	
sub_proc10	25	
exp04_sub1	0	miss
ext_sub_proc10	0	
exp04_sub2	5	
ext_sub_proc20	5	
exp04_sub3	10	
ext_sub_proc30	10	
exp04_sub4	15	

```

exp04.c.gcov
--: 0:Source:exp04.c
--: 0:Graph:exp04.gcn0
--: 0:Data:exp04.gcda
--: 0:Runs:32
--: 0:Programs:1
--: 1:#include <stdio.h>
--: 2:extern void ext_sub_proc1(int);
--: 3:extern void ext_sub_proc2(int);
--: 4:extern void ext_sub_proc3(int);
--: 5:extern void ext_sub_proc4(int);
--: 6:extern void ext_sub_proc5(int);
--: 7:extern void ext_sub_proc8(int);
--: 8:
function sub_proc1 called 160 returned 100% blocks executed 71%
160: 9:void sub_proc1(int a) {
160: 10:int i;
480: 11:   for(i=0;i<a;i++) {
320: 12:     switch(a) {
--: 13:       case 0:
#####: 14:         ext_sub_proc1(i);
#####: 15:         break;
--: 16:       case 1:
32: 17:         ext_sub_proc2(i);
32: 18:         break;
--: 19:       case 2:
64: 20:         ext_sub_proc3(i);
64: 21:         break;

```

Detailed line-level result shown by our GUI

Extension to general analysis tools

The results shown by plaintext

The 'Get Profile data' dialog box contains the following fields and options:

- Select Execution Mode:**
 - Variable profile
 - Get profile
 - Check MEM error
 - Check CACHE hit rate
 - Check code coverage
- Convert result to TPTP format
- Target IP address (It's would put modules to target and get the result file from target):** 192.168.0.250
- Host PC working path name:** C:\pro_tool\cygwin\DEMO\mipsel\%gcov%
- Target execute command:** exp04
- Result restore path. (If not specify this parameter then used "Host PC working path name" value):** C:\pro_tool\cygwin\DEMO\mipsel\%gcov%
- Local convert command:** mipsel-linux-gcov %r
- Target result module names:** exp04.gcda, exp04_sub1.gcda, exp04_sub2.gcda, exp04_sub3.gcda, exp04_sub4.gcda, exp04_sub5.gcda, exp04_sub6.gcda
- HOST PC result file names:** exp04.c.gcov, exp04_sub1.c.gcov, exp04_sub2.c.gcov, exp04_sub3.c.gcov, exp04_sub4.c.gcov, exp04_sub5.c.gcov
- All parameters load from text file.

The 'exp04.c.gcov' file content is as follows:

```

-: 0:Source:exp04.c
-: 0:Graph:exp04.gcno
-: 0:Data:exp04.gcda
-: 0:Runs:32
-: 0:Programs:1
-: 1:#include <stdio.h>
-: 2:extern void ext_sub_proc1(int);
-: 3:extern void ext_sub_proc2(int);
-: 4:extern void ext_sub_proc3(int);
-: 5:extern void ext_sub_proc4(int);
-: 6:extern void ext_sub_proc5(int);
-: 7:extern void ext_sub_proc8(int);
-: 8:
function sub_proc1 called 160 returned 100% blocks executed 71%
160: 9:void sub_proc1(int a) {
160: 10:int i;
490: 11:   for(i=0;i<a;i++) {
320: 12:     switch(a) {
-: 13:       case 0:
#####: 14:         ext_sub_proc1(i);
-: 15:         break;
-: 16:       case 1:
32: 17:         ext_sub_proc2(i);
32: 18:         break;
-: 19:       case 2:
64: 20:         ext_sub_proc3(i);
64: 21:         break;

```

Setting for various tools

Conclusion

- Code quality analysis for embedded systems
 - ◆ Memory management with Valgrind
 - ◆ Code coverage testing with Gcov
- Remote system analysis with network agent
 - ◆ By extending “TCPstub,” originally designed for debugging and profiling
- Seamlessly integrated to Eclipse
 - ◆ TPTP-based and our original GUIs
 - Because of the capability of the TPTP view