International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:1, 2007

A System for Performance Evaluation of Embedded Software

Yong-Yoon Cho, Jong-Bae Moon, and Young-Chul Kim

***Abstract*—**Developers need to evaluate software's performance to make software efficient. This paper suggests a performance evaluation system for embedded software. The suggested system consists of code analyzer, testing agents, data analyzer, and report viewer. The code analyzer inserts additional code dependent on target system into source code and compiles the source code. The testing agents execute performance test. The data analyzer translates raw-level results data to class-level APIs for reporting viewer. The report viewer offers users graphical report views by using the APIs. We hope that the suggested tool will be useful for embedded-related software development, because developers can easily and intuitively analyze software's performance and resource utilization.

***Keywords*—**Embedded Software, Performance Evaluation System, Testing Agents, Report Generator

I. Introduction

According as embedded system has become increasingly sophisticated and user's requirement for embedded software has become complicated, developing efficient embedded software against the restricted resource has become much more difficult and important. Because embedded system generally offers less computing resource than general-purpose computer system does, embedded software that is so poor in quality or performance wastes the scarce resources [3][4]. Developers want to improve the quality of their embedded software and make it to have always a good performance in resource usage. To do this, developers occasionally use embedded software evaluation system to decrease time and increase efficiency in developing embedded. The evaluation system is useful for developers, because they know whether developed software is efficiently optimized in the embedded system's restricted resource. By using evaluation system, developers can execute embedded software on target system during the development process, test its performance in advance, and know what must be fixed to make it more efficient. But, because the testing results are text-based string, developers have to analyze the raw-level data to find software's portion where they has to revise. It is often very tiresome and time-consuming work. The evaluation system needs occasionally additional hardware that exists between host computer and target board and executes such functions as software's performance testing and result analyzing. But, that may impose heavy burden on developers, because they have to pay additional cost and study how to use the instrument. In this paper, we suggest graphic-based evaluation system to test and analyze embedded software's performance. Because the suggested evaluation system is based on pure software without any additional hardware, developers don't have to spend a lot of time studying about how to operate additional equipment. The suggested evaluation system involves a graphic report viewer, which reports various results and shows them graphically and according to such test items as memory usage, code coverage, and function call times. By using the viewer, developers can analyze software's performance at a glance and find easily what is fixed to make software more efficient. As a result, developers can improve development efficiency of embedded-related software, because they can have opportunity to analyze software's performance instantly and intuitively through the graphical report viewer. The paper is organized as follows. Section II reviews works related in embedded software evaluation system. Section III describes the proposed evaluation system and graphic report viewer. Section IV conducts a testing and presents the results. Finally, Section V states our conclusions and presents a summary of our research.

Manuscript received November 30, 2004.

Yong-Yoon Cho is with the Department of Computing, Soongsil University, Seoul, CO 156743 Korea (corresponding author to provide phone: +82-02-824-3862; fax: +82-02-824-3862; e-mail: sslabyycho@hotmail.com).

Jong-Bae Moon is with the Department of Computing, Soongsil University, Seoul, CO 156743 Korea (e-mail: comdoct@ss.ssu.ac.kr).

Young-Chul Kim is with the Department of Computing, Soongsil University, Seoul, CO 156743 Korea (e-mail: yckim@ss.ssu.ac.kr).

II. Related Works

Telelogic's Tau TTCN Suite is a system to test telecom and datacom equipment ranging from built-in communication chips to huge switches and intelligent network services. It includes various tools such as script editor, compiler and simulator. But, it is not suitable for testing embedded software because it is test system for telecommunication vendor. It also is very expensive because it is mostly additional hardware equipment to test telecom system.

AstonLinux's CodeMaker is IDE(Integrated Development Equipment) to develop embedded software based on linux in Windows. It supports remote debugging and source-level debugging. But, it doesn't offer function to test and analyze embedded software's performance, because it is only IDE for specific RTOS/chip vendor. It is also suitable for testing general-purpose target system, because it uses linux-based utilities.

TestQuest Pro is automated test solution for embedded systems with sophisticated human interfaces. It is automated test solution for virtually any application and device ranging from cell phones, PDAs and tablets to devices with embedded technologies, including wireless enterprise systems, smart appliances and automobiles. It offers functions related in debugging and simulating embedded software's source code but doesn't proffer information for software's performance in usage of resource.

Rational's TestRealTime is target-based evaluation system for embedded software's performance. It proffers various result views that users can easily analyze software's performance. It can also execute various performance testing ranging from memory usage, memory leak, cpu usage to code coverage. But, The result view is somewhat complicated to understand result's meaning at a glance.

III. Proposed Performance Evaluation System

In this paper, we suggest system for testing embedded software's performance that consists of pure software without additional hardware equipment and offers such various performance testing as memory, code coverage, code trace and function performance [1]-[7]. The evaluation system proffers users graphical report views that they can easily and intuitively analyze the test result. Fig. 1 is the proposed architecture for a performance evaluation system.

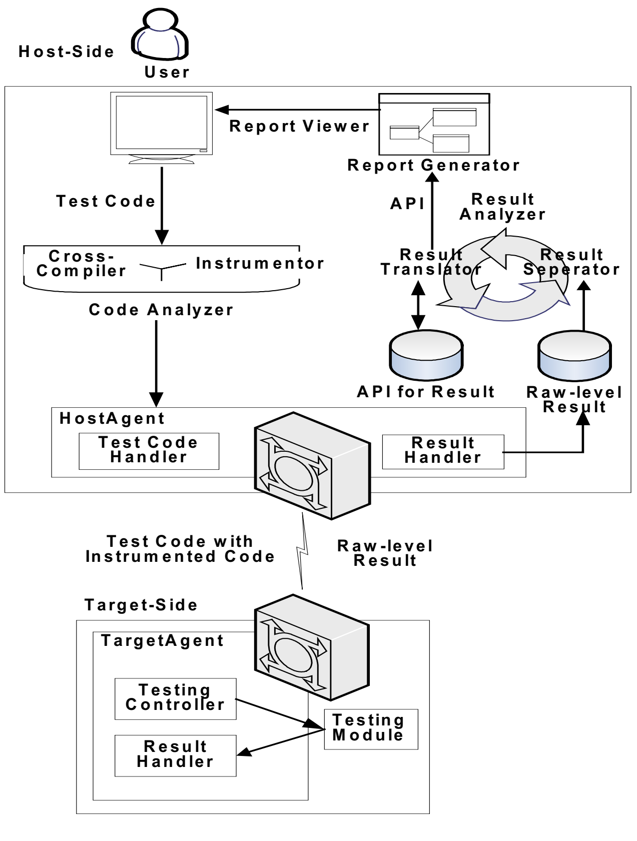


Fig. 1. A Proposed Architecture of Performance Evaluation System

In Fig. 1, the proposed evaluation system is composed with GUI, host/target-side agents, code analyzer, result analyzer, and report viewer. The code analyzer consists of instrumentor to insert additional code into source code and cross compiler to create target-execute file for the source code. The evaluation system is a client/server model based in host-target architecture. Because embedded system offers insufficient memory and inconvenient user interface, the suggested tool places agent not only on host-side to proffer users convenient GUI but also on target-side to execute software's performance testing in target board. The agents keep a communication connection to deliver source file and test result to each other. Firstly host-side agent transfers inputted source to target-side agent through serial cable or wireless network. Then, target-side agent executes testing process, gains results from the test events, and send the results to its host-side counterpart. Consequently, host-side agent stores the raw-level result received from target-side one into result DB. Generally, embedded software must use minimum process and memory resources [3][4]. To meet the requirement, the suggested tool tests software's performance for the 4 items described in Table I [5][6].

TABLE I
Units for Magnetic Properties

Testing Items	Function
Trace Profiling	Tracing software's runtime execution in UML sequence diagram
Memory Profiling	Profiling software's resource usage related memory.
Performance Profiling	Profiling function or method's execution performance.
Code Coverage Profiling	Separating and Profiling code block [8].

Through trace profiling, users can trace what functions are executed according to software's execution process and find what functions are unnecessarily called. Report viewer shows result for trace profiling as UML sequence diagram [9][10]. Through memory profiling, users can know information about memory allocation/de-allocation, memory leak, and code sections frequently to use memory. Users can use performance profiling to estimate how much time it takes to execute the whole or part of embedded software and confirms whether it becomes optimized in embedded system. Code coverage profiling offers users information about used or not used code section, and frequently or not frequently code section. Users can make embedded software more efficient by using information profiled according to the 4 items. Usually result created from profiling software exists in raw-level strings. It is difficult and tiresome for users to analyzing software performance with it.

Result analyzer classifies raw-level result according to the items referred in Table I and converts it into refined data type. It contains result separator and result translator. The result separator classifies raw-level result into different data type in accordance with the profiling items. The result translator converts the classified result into API classes or XML files that report generator can use to make a report view for user's requirement. The API or XML produced by the result analyzer is stored into API DB.

Table II describes a part of memory-related API class that raw-level result created through memory profiling is converted into.

TABLE II
A PART OF API CLASS FOR MEMORY-RELATED RAW-LEVEL RESULT

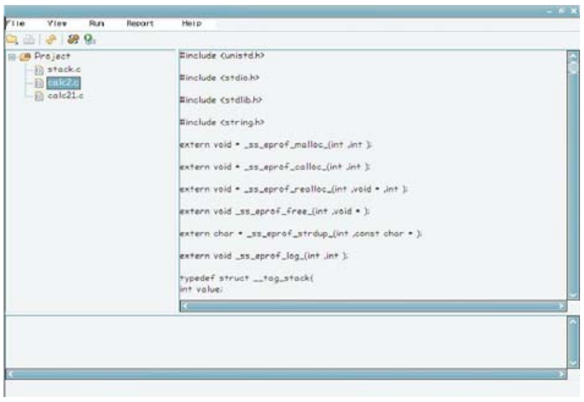
```
    public class DataConverter {
    private CInstrumentationTable instrumentations;
    public instrumentations(CInstrumentationTable instrumentations){
    this. instrumentations = instrumentations;
    }
    public static IMemoryElement convertingMemoryElement(IEvent
event){
    /* Read the symbol from raw-level result produced according to
memory profile test event and convert it into * /
            return new MemoryElement();
        }
```

Report generator makes graphical report view by using result APIs, when users select one of report views in evaluation system's menu. The suggested evaluation system includes report viewer, through that users can see selected report view.

IV. TESTING AND RESULTS

The suggested evaluation system is implemented in Java and its testing environment is embedded board launched strong ARM chip and embedded Linux. To evaluate performance for example C source program, we use a calculator program in C language as input source code. The source's code size is about 520 lines. It consists of three modules. We test them against the 4 items appeared in Table I and shows graphical report views for the results through report viewer.

Fig. 2 shows the evaluation system's initial screen to test calculator program.



```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
extern void * _ss_eprof_malloc_(int ,int );
extern void * _ss_eprof_calloc_(int ,int );
extern void * _ss_eprof_realloc_(int ,void * ,int );
extern void _ss_eprof_free_(int ,void * );
extern char * _ss_eprof_strdup_(int ,const char * );
extern void _ss_eprof_log_(int ,int );
typedef struct __tag_stack(
int value;
```

Fig. 2. The suggested evaluation system's initial screen for testing

After cross-compiling by code analyzer, the source code includes additional code that is dependent on target system. Then the evaluation system takes connection with target system to transfer source code to target system through host/target agents. After testing in target system, raw-level test result in string type is outputted. Table III shows a part of raw-level test result related in memory performance of the test source code appeared in Fig. 2.

TABLE III
A RAW-LEVEL TEST RESULT

```
index % time   self  children    called      name
                                  95000           func1 <cycle 1> [3]
[2]       0.0    0.00    0.00   95000         func2 <cycle 1> [2]
                                      900           func1 <cycle 1> [3]
-----------------------------------------------------------------
                                  900            func2 <cycle 1> [2]
                 0.00    0.00     1000/1000       main [13]
[3]       0.0    0.00    0.00    1900          func1 <cycle 1> [3]
                                      95000           func2 <cycle 1> [2]
-----------------------------------------------------------------
Index by function name
[3] func1                  [2] func2                [1] <cycle 1>
@ <Location>:(Called function + Called Location)[Instruction
Location] +/- Address Size= Start
@       ./ex-n:(mtrace+0x169)[0x80484e9]        +       0x804a378
0x12@ ./ex-n:[0x8048596] - 0x804a378
```

But, it is difficult for users to analyze the meanings. So, the evaluation system offers users graphical result views through data analyzer, report generator, and report viewer. Fig. 3 shows a graphical report view that report generator and report viewer make with API to that data analyzer translates raw-level trace result after testing the source in Fig. 2.

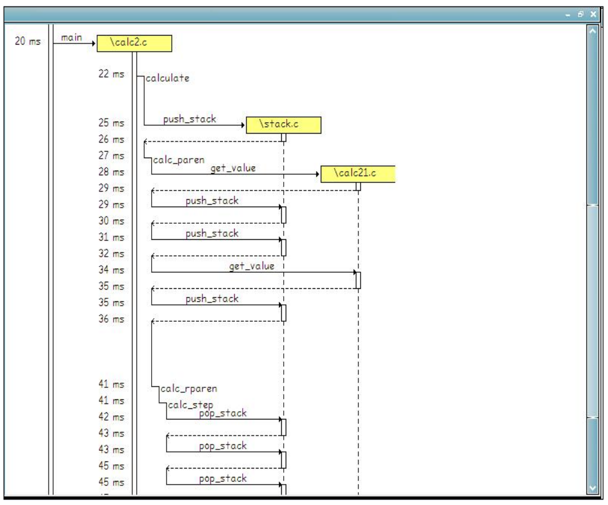
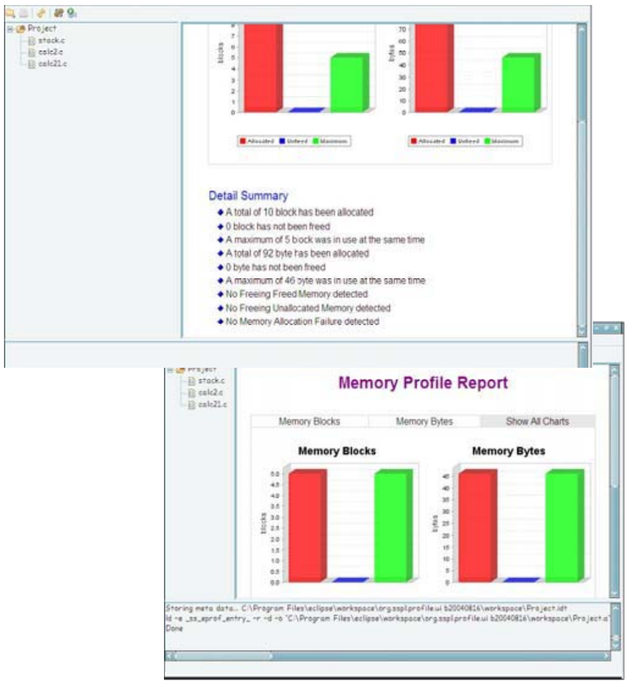


Fig. 3. Trace profiling view for raw-level test result

With the view showed in Fig. 2, we can know each function's execution time and its call order. So, developers can find a function that take a lot of time to process its processing, and can re-write source program to distribute a burden of works to others. Then, we can obtain other report views for test result of the test source in Fig. 2. Fig. 4 shows memory report view related in memory performance of the test result.

In Fig. 4, we can know information about software's memory usage such as memory allocation, freed memory, or a maximum of byte used at the same times through the memory report view.

So, developers can find a portion of source code that cause memory extravagance and dismiss it.

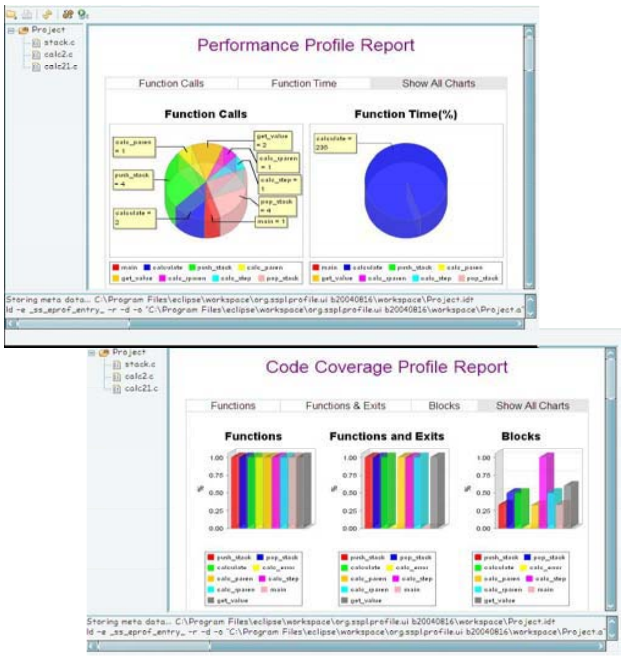


Detail Summary
- A total of 10 block has been allocated
- 0 block has not been freed
- A maximum of 5 bock was in use at the same time
- A total of 92 byte has been allocated
- 0 byte has not been freed
- A maximum of 46 byte was in use at the same time
- No Freeing Freed Memory detected
- No Freeing Unallocated Memory detected
- No Memory Allocation Failure detected

```
Storing meta data... C:\Program Files\eclipse\workspace\org.sspl.profile.ui b20040816\workspace\Project.idt
ld -e _ss_eprof_entry_ -r -d -o "C:\Program Files\eclipse\workspace\org.sspl.profile.ui b20040816\workspace\Project.o"
Done
```

Fig. 4. Memory report view

Fig. 5 shows code coverage and performance report views. In Fig. 5, through the code coverage report view, we can also know whether some function was executed or not and get information about execution rates of code blocks in function. And, with the performance report view, we can find information about each function's call times and execution time.



```
Storing meta data... C:\Program Files\eclipse\workspace\org.sspl.profile.ui b20040816\workspace\Project.idt
ld -e _ss_eprof_entry_ -r -d -o "C:\Program Files\eclipse\workspace\org.sspl.profile.ui b20040816\workspace\Project.o"
```

```
Storing meta data... C:\Program Files\eclipse\workspace\org.sspl.profile.ui b20040816\workspace\Project.idt
ld -e _ss_eprof_entry_ -r -d -o "C:\Program Files\eclipse\workspace\org.sspl.profile.ui b20040816\workspace\Project.o"
```

Fig. 5. Code coverage and Performance report views

So, we can get information about code blocks and make balance among code blocks. So, we can get information about each function's execution efficiency and tune it.

V. Conclusion

In this paper, we suggested a graphic-based system to easily evaluate embedded software's performance and intuitively to analyze results through graphic report views. The suggested evaluation system has adopted a client-server model based in agents. It is for the purpose of supporting development environment of embedded software based in cross-platform. The system adopts data analyzer to refine initial raw-level test result into API, which is class type. Developers can easily reuse API in various tools to evaluate embedded software's performance [12]. As a evaluation tool using API, the suggested system offers developers a report viewer. In test, we tested C source code and showed the results graphically through report viewer. Through the suggested tool, developers can clearly know what must be fixed in software's source code and can improve development efficiency of embedded software.

For the future, we will study method to automate a test case and to develop an internet-based testing system for embedded system by translating test data as XML document [13][14].

References

[1] Roper, Marc, *Software Testing*, London, McGraw-Hill Book Company, 1994.

[2] Boris Beizer, *Software Testing Techniques 2nd edition*, New York: Van Nostrand Reinhold, 1990.

[3] Bart Broekman and Edwin Notenboom, *Testing Embedded Software*, Addisson-wesley, Dec. 2002

[4] Dr. Neal Stollon, Rick Leatherman and Bruce Ableidinger, *Multi-Core Embedded Debug for Structured ASIC Systems*, *proceedings of DesignCon 2004*, Feb, 2004.

[5] David B. Stewart, Gaurav Arora, *A Tool for Analyzing and Fine Tuning the Real-Time Properties of an Embedded System*. IEEE Trans. Software Eng., Vol.TSE-29, No.4, April 2003, pp.311-326.

[6] Ichiro Satoh, *A Testing Framework for Mobile Computing Software*. IEEE Trans. Software Eng., Vol.TSE-29, No.12, December 2003, pp.1112-1121.

[7] Paul Anderson, Thomas W. Reps, Tim Teitelbaum, *Design and Implementation of a Fine-Grained Software Inspection Tool*. IEEE Trans. Software Eng., Vol.TSE-29, No.8, August 2003, pp.721-733.

[8] John Joseph Chilenski and Steven P. Miller, *Applicability of Modified Condition/Decision Coverage to Software Testing*, Software Engineering Journal, September 1994, Vol. 9, No. 5, pp. 193-200.

[9] Robert B. France, Dae-Kyoo Kim, Sudipto Ghosh, Eunjee Song, *A UML-Based Pattern Specification Technique*, IEEE Trans. Software Eng., Vol.TSE-30, No.4, April 2004, pp. 193-206.

[10] Ludovic Apvrille, Jean-Pierre Courtiat, Christophe Lohr, Pierre de Saqui-Sannes, *TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit*. IEEE Trans. Software Eng., Vol.TSE-30, No.7, July 2004, pp. 473-487.

[11] William E. Howden, *Weak Mutation Testing and Completeness of Test Sets*, IEEE Trans. Software Eng., Vol.SE-8, No.4, July 1982, pp.371-379.

[12] Brad Long, Daniel Hoffman, Paul A. Strooper, *Tool Support for Testing Concurrent Java Components*. IEEE Trans. Software Eng., Vol.TSE-29, No.6, June 2003, pp.555-566.

[13] Morell, Larry, *A Theory of Fault-Based Testing*, IEEE Trans. Software Eng., Vol.16, No.8, August 1990, pp.844-857.

[14] John P. Kearns, Carol J. Meier and Mary Lou Soffa, *The Performance Evaluation of Control Implementations*. IEEE Trans. Software Eng., Vol.SE-8, No.2, February 1982, pp.89-96.