ADVANCES ON TESTING SAFETY-CRITICAL SOFTWARE
Goal-driven Approach, Prototype-tool and Comparative Evaluation

Guido Pennella, Christian Di Biagio
MBDA-Italy SpA, Via Tiburtina, Roma, Italy

Gianfranco Pesce
Centro di Calcolo e Documentazione, Università degli Studi di Roma "Tor Vergata", Via O. Raimondo, Roma, Italy

Giovanni Cantone
Dip. di Informatica, Sistemi e Produzione, Università degli Studi di Roma "Tor Vergata", Via O. Raimondo, Roma, Italy

Keywords: Software engineering, Distributed and parallel systems, Hard Real-time Systems, Performance-measurement Tools.

Abstract: The reference company for this paper – a multination organization, Italian branch, that works in the domain of safety-critical systems – evaluated the major tools, which the market provides for testing safety-critical software, as not sufficiently featured for her quality improvement goals. Consequently, in order to investigate the space of possible solutions, if any, the company's Research Lab. started an academic cooperation, which leaded to share knowledge and eventually to establish a common research team. Once we had transformed those goals in detailed technical requirements, and evaluated that it was possible to realize them conveniently in a tool, we passed to analyze, construct, and eventually utilize in field the prototype "Software Test Framework". This tool allows non-intrusive measurements on different hard-soft targets of a distributed system running under one or more Unix standard OS, e.g. LynxOS, AIX, Solaris, and Linux. The tool acquires and graphically displays the real-time flow of data, so enabling users to verify and validate software products, diagnose and resolve emerging performance problems quickly, and enact regression testing. This paper reports on the characteristics of Software Test Framework, its architecture, and results from a case study. Based on comparison of results with previous tools, we can say that Software Test Framework is leading to a new concept of tool for the domain of safety-critical software.

1 INTRODUCTION

This paper expands on a previous work (Di Biagio, 2006b), which investigated the major available technologies for testing hard real-time software. The main result of that study was the characterization of those technologies from the point of view of a certain company – a multination organization, Italian branch, which works in the domain of safety-critical systems.

Based on the results from that study, the company's management evaluated the major tools that the market provides for testing safety-critical software, as not sufficiently featured for their quality improvement goals. Consequently, in order to investigate the space of possible solutions, if any, the company's Research Lab. was allowed to start an academic cooperation, in the aim of sharing knowledge and eventually establish a common project and research team. This paper reports on some results and a product that derived from such an experience.

Let us briefly present the context of real-time performance testing, remanding to a technical report for further details (Di Biagio, 2006a). The usage of a monitor is strongly recommended for the test of performance of hard real-time systems (Tsai, 1995) and quality assurance of new digitalized safety-critical systems (EPRI, 1994). A monitor is a system able to observing and analyzing behaviors shown by another, in case remote, system (a.k.a.: the "target"),

Pennella G., Di Biagio C., Pesce G. and Cantone G. (2006).
ADVANCES ON TESTING SAFETY-CRITICAL SOFTWARE - Goal-driven Approach, Prototype-tool and Comparative Evaluation.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 218-223
DOI: 10.5220/0001311002180223

comparing the actual states of the target with expected ones – as produced by the same monitor performing in the role of "oracle" (Weyuker, 1982) – or reporting on system failures – as detected by the same monitor performing in the role of "supervisor" (Simser, 1996) – respectively. In safety-critical applications, the system should be monitored by another safety system to ensure continued correct behavior. To achieve these goals, observed behaviors must be quickly accepted or rejected; this task is quite difficult to enact when complex real-time systems are involved, and the requested response time is not in the range of human capabilities. Additionally, software practitioners cannot diagnose, troubleshoot, and resolve every component affecting a critical software performance by using just manual methods.

The goal (Basili, 1994) of the present paper is concerned with the purpose of measuring system test performances. The focus is on measurement of CPU and memory loads, performance monitoring of distributed heterogeneous processes and their threads, intrusiveness, and other key attributes. The point of view consists in the reference organization practitioners. The context is the development of critical software. In particular, we want to proceed by: (i) expressing the reference company need of testing safety-critical software in terms of conveniently feasible features and capabilities; (ii) developing a new software tool that meet those needs; (iii) Characterizing that tool, comparing it with other testing tools, accepting it by a case study, and eventually (iv) accrediting the tool in field and continually improving it, based on feedback from practitioners (Cantone, 2000).

In the remaining of the present paper, Section 2 transforms the reference organization's needs and goals in required testing features. Section 3 presents the philosophy, architecture, and functionalities of Software Test Framework (STFW), a new prototype tool, which is based on those features. Section 4 shows results from a case study, which involved the STFW. Section 5 briefly compares STFW with major professional tools that the market provides. Section 6 presents some conclusions and points to future research.

2 TESTING FEATURES

There is not enough room here to report on the interview-based requirement elicitation process that we enacted with the customer stakeholders (rhe reference company's software practitioners and project managers). Anyway, based on the expected use cases and the resulting requirements, a list of testing features (F) follows, which, in our view, characterizes a software test framework and is able to satisfy the needs that the reference organization expressed. Each of the shown features is augmented with the F's: (i) function or capability, (ii) measurement model applied (in round brackets), (iii) relative importance or weight, as expressed by the involved stakeholders [in square brackets] (values are not shown; see Section 5).

F1 Heterogeneous targets monitoring (N|(Y, heterogeneous target types) [w1].
F2 Average CPU percentage used during data acquisition on a target system. CPU and memory (see F3) occupancies are calculated under their maximum load, i.e. when all possible data are required for acquisition, and the acquisition interval is the one suggested by the tool producer, respectively (%) [w2].
F3 Memory occupancy on a target system (MB) [w3].
F4 Persistent data repository and management (N|Y) [w4].
F5 Tailor the test system to suit special user needs or purposes (N|Y) [w5].
F6 Un-intrusiveness (Intrusiveness: time for data acquisition in seconds) [w6].
F7 Distributed targets monitoring. TCP/IP over Ethernet (N|Y) [w7].
F8 Plug-in architecture (N|Y) [w8].
F9 System CPU (idle and used) percentage measurement (N|(Y, %)) [w9].
F10 System memory load (free and occupied) measurement (N|(Y, MB)) [w10].
F11 Process CPU (idle and used) percentage measurement (N|(Y, %)) [w11].
F12 Process memory load (free and occupied) measurement (N|(Y, MB)) [w12].
F13 Thread CPU (idle and used) percentage measurement (N|(Y, %)) [w13].
F14 Thread memory load (free and occupied) measurement (N|(Y, MB) [w14].
F15 Support multi platform for all the major operative systems (N | (Y, Checkbox for LynxOS, Solaris, AIX, Linux, POSIX etc., respectively)) [w15].
F16 Allow regression testing (N|Y) [w16]
F17 Utilize software sensors (N|Y) [w17].
Cost (0|*$) [w18].

3 SOFTWARE TEST FRAMEWORK

Software Test Framework is a complex analysis tool that deals with capturing resource occupation data of one or more target systems.

3.1 Architecture

In order to introduce minimal perturbation in the target system, STFW is developed for performing flexibly non-intrusive as-accurate-as-possible measurements. These results are achieved by employing a distributed architecture, which works on different computers in such way that only the measurements operations are performed on the target system, leaving the most complex elaborations and activities, such as the graphical plot, to other computers. Figure 1 shows the architecture of STFW. STFW is build-up by three macro-units:

- **Target**: it resides on each target machine and is responsible of the execution of the measurements and the optimization of the sensor. Target is build-up by two sub-units:
  - **Test Manager** (TM): its task is to opportunely tailor the Sensor.
  - **Sensor**: its task is to acquire information.
- **Analysis System**: it does not reside on a target computer but on a different machine. The Analysis System is responsible of the analysis, interpretation and visualization, both in real and in deferred time of data, which the instances of Sensor send. The Analysis System is build-up by three sub-units:
  - **Data Manager**: it is responsible for the interpretation of information sent by Sensor. The Data Manager also forwards the Data Plotter.
  - **Data Plotter**: is able to graphically plot data that Data Manager sends.
  - **GUI** (Graphical User Interface): sends Test Manager the information to acquire, as specified by the user.
- **Repository**: it historicizes test related data. The Repository does not reside on a target computer but on a different machine.

The most interesting features and capabilities of STFW are:

- STFW supports regression test
- STFW supports data repository
- STFW supports threads monitoring
- Sensor is a tailor-made software
- Sensor is not intrusive
- Acquisitions form different targets are synchronous in the same conversation (Anderson, 1983).

3.2 Usage

STFW is very easy to use. After the installation

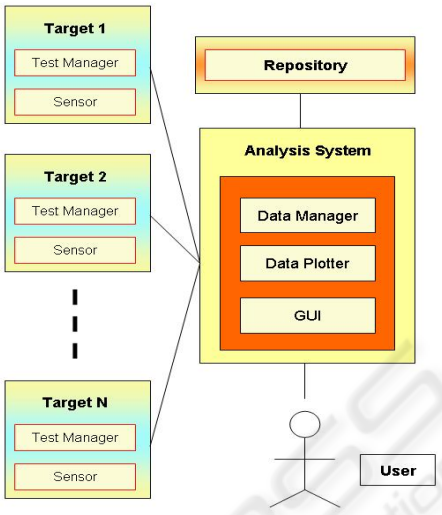

Figure 1: Architecture of STFW.

of the required software on Target, Analysis System, and Repository, a user is able to start with tests of any kind and proceed step by step. In the first step, the user chooses the information needed (concerning CPU, memory, and so on), the duration of the whole test, and the sampling interval by means of the STFW graphical user interface. In the second step, the user sets the IP addresses of the Target and Repository sub-systems. Now, the user is allowed to start the test. After a small time (1 – 20 sec), in which the Test Manager (TM) configures Sensor to acquire only the specified information (Sensor loads only the needed modules), data plotting is started on the user screen and, in parallel, the repository is populated.

The user, during the first step, can load and launch a historicized test: as result, the user is allowed to compare two different tests in the same plot, the historicized one, and the other one in running. Moreover, once a test is finished, the user can choose graphical or numerical presentation of results; plots are presented for each acquisition time.

3.3 Regression Test

STFW provides EXnee, which is an integrated and enhanced version of Xnee. This is a free software tool, which is able to record and playback all events used by the X Server. So, each time a user moves the mouse or digits a button on the keyboard, Xnee records these events and is then able to reproduce all the related actions. In this way, Xnee is able to replicate in the system the effects of all the activities performed by the user in the same temporal sequence.

After a session of events is recorded, an STFW user can reproduce that session every time it is needed. Let us consider, for instance, a user, who starts the execution of a (critical) software, and then begins to interact with it. Of course, if the user makes decision to change that software, Xnee allows that user (and all the authorized colleagues) to start replication of all those interactions. Once that such a replication has been started, Xnee is able to proceed autonomously (the physical presence of user is no more requested) by replicating events of user-system interactions and identifying differences in behaviors, if any, due to the injection of software changes since the last build (regression test).

3.4 Tailoring

Concerning the consumer side, STFW is configurable to the different operational environments. In order to allow the (static) specialization of STFW to the particular operational environment, some parameters are specified for the framework (i.e. operating system, process monitoring, thread monitoring etc.); parameters are easily handled, due the STFW modular structure.

3.5 Intrusiveness

Intrusiveness represents for a software application the OS load. It is complementary to, and can be quantified in terms of, CPU percentage and amount of memory used by the application software itself in situation of maximum performance.

STFW is able to guaranty CPU occupancy under 1%, while acquire data with a minimal interval of 1 second. Let us note that major tools suggest acquiring data on the target system with sampling period not less than 3 or 10 seconds, respectively. Such a STFW advantage derives from its tailoring features (see Section 3.4) and the system architecture of the Target module.

3.6 Parallelism, Synchronization and Heterogeneity

Based on the architecture of our tool (see Section 3.1), STFW supports data acquisition in parallel from different heterogeneous targets. On a target machine, a test is build-up by a configuration phase and a subsequent conversation phase for data acquisition. When all the Sensors have been configured, they synchronize on the reception of a start message. Following the reception of this message, all Sensors start to acquire their data and finally sending those data to the consumer.

Let us note that, in order to compare consistent data, starting and completing synchronously acquisitions from different targets is an essential requirement. Because the end of a communication time-window is in the control of the consumer, it is enough to start (multi-point to point) communications at the "same" time, as STFW actually does (notice that latencies - as introduced both by the TCP/IP over Ethernet, and the OS scheduler – are negligible in common test environments, compared to sampling interval).

3.7 Data Repository

The whole information, as each Sensor acquires, is stored in a relational data base (DB). In order to keep intrusiveness in control, the DB is installed on the computer that hosts the Analysis System, or any other machine but different from the ones where Sensors are installed.

Storing data in a repository is useful because it allows reusing previous test cases, analyzing previous results, and comparing such previous results with those generated by running test cases.

3.8 Process and Thread Monitoring

STFW is able to acquire information about processes and threads, as in the followings:

- PID: Process Identifier
- TID: Thread Identifier
- PPID: Parent PID
- S: Status; can be Ready, Running or Waiting
- MO: Memory occupancy; is the sum of the amount of memory allocated for the stack, the executable file, and data.
- CPUO: CPU occupancy; is the percentage of CPU used.

TID does not apply to processes. In case of threads, MO evaluates the stack size (a thread shares text and data with its parent process).

4 CASE STUDY

Let us present results from a case study, where we compared in real-time the behaviors of two applications running on two Single Board Computer (SBC). Monitored attributes were the system's target CPU occupancy, and the full information associated to the execution of two processes, Ubench 2.0 and Sensor, respectively. The Ubench job consists in computing senseless mathematical operations for 3 minutes, and then, in the successive 3 minutes, performing senseless memory allocation and de-

allocations (Ubench, 2006). The job of Sensor consists in auto-monitoring activities.

We conducted the case study in the reference company's industrial environment, built-up by three calculus nodes, as in the followings: (1) Thales – Vmpc6a Single Board Computer (SBC) with Lynx OS, (2) Concurrent - Intel SBC with Linux Red Hat Enterprise, and (3) x86 PC with Windows XP. Those nodes are one to each other connected through an Ethernet LAN.

Each SBC was arranged to perform in the role of target system, and had its own Test Manager and Sensor installed. The Windows PC was arranged to perform in the role of consumer, and hosted the graphical console. Hence, we proceeded with the case study by starting Test Managers (i.e. writing "./testman" on the bash consoles) and the GUI (i.e. double clicking the exe file in the PC window). Following the start of the GUI, we passed to configure the targets by entering "CPU", "Ubench" and "Sensor" and then pressing the OK button. When the Sensors were compiled, installed and ready to send data, we pressed the START button and then two plotting windows appeared on the PC screen, which showed the required information only.

Figure 2 shows an instance of process-monitoring windows in STFW.

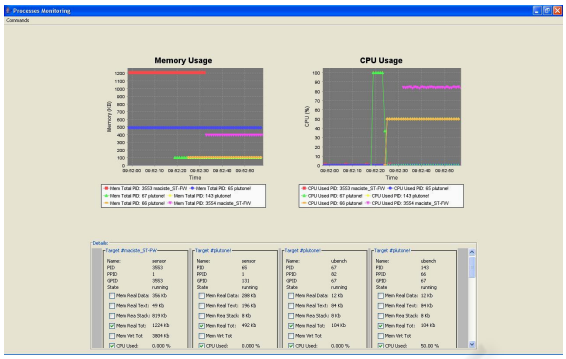

Figure 2: Process-monitoring windows in STFW.

5 COMPARATIVE ANALYSIS

In Table 1 we compare STFW with three major professional tools (Di Biagio, 2006a), (Di Biagio, 2006a).

Table 1 shows the limits of commercial measuring tool with respect to STFW.

In fact, for all the attributes of the evaluation model less the memory occupancy on a target (F3), STFW shows the same or better values than the other tools.

Consequentlu, in order to compare those technologies, we do not need to weight those attributes and develop a synthetic indicator: the advantage of STFW would persist to any practical set of weights chosen.

Anyway, the reader should notice that STFW is just a prototype (but in its second internal release).

While Table 1 is auto-explicative in terms of comparative analysis, let us use this opportunity to present some further considerations.

In our view, the measuring tools available are "heavy" both for data-producers and data-consumers. They admit the worst configuration only, so that they acquire all possible data.

Table 1: Characterization of T12, T2 and T3 monitoring tools (N≡0|Y≡1; Li ≡ Linux 2.6; Ly ≡ Lynx; S ≡ Solaris).

F	m	T1	T2	T3	STFW
F1	0..1	0	0	0	1
F2	%	3	60	3	1
F3	MB	1	0	0,5	<2
F4	0..1	0	0	0	1
F5	0..1	0	0	0	1
F6	(sec.)	3	10	1	1
F7	0..1	0	1	0	1
F8	0..1	0	0	0	1
F9	0..1	1	1	1	1
F10	0..1	1	1	1	1
F11	0..1	1	0	0	1
F12	0..1	1	0	0	1
F13	0..1	0	0	0	1
F14	0..1	0	0	0	1
F15	OS list	Li, Ly, S, AIX	Li	S	Li, Ly, S, AIX
F16	0..1	0	0	0	1
F17	0..1	0	0	0	1
Cost	0..*$	0	$$$	0	0

Consequently, the installation of all their data-acquisition modules is permanently requested. As a result, consumers receive data that they never requested. As a further result, the intrusiveness is unnecessary high; in fact, it is proportional to the amount of data acquired. Instead, STFW is a framework, fully tailor-made: tailoring introduces improvements both on the producer side (unnecessary modules are not loaded), and the consumer side (only explicitly requested data is processed and represented to the consumer).

With respect to other monitoring technology, two turning points make STFW a new concept tool. Concerning the target machine, STFW reduces the occupancy of the system resources in term of memory and CPU percentage occupied, because only user-required data is acquired (no overload of the system resources), and memory allocation is

minimal (only the requested modules are loaded, which correspond to the requested data). Concerning the consumer side, this is allowed to choose a-priori the data to acquire, so not having to discriminate a posteriori among all the received information for the interesting data.

6 CONCLUSION AND FUTURE WORK

We have presented the philosophy, architecture and features of a new tool, STFW, for testing time-behavior of safety-critical systems, and briefly compared that tool with major system performance measurement tools, as available from the market, to the best of our knowledge. STFW resulted to be much more supportive than other tools for our reference professional engineers. The most important features, which make STFW really a competitive tool, are: (i) Tailor-made non-intrusive data sensing; (ii) Synchronous conversations for acquiring state information from distributed targets; (iii) Repository of test cases for reuse, and their results for comparative analysis; (iv) Thread monitoring, (v) Ability to perform regression test.

Thanks to STFW, each product can be validate and verified in real-time by monitoring and comparing results from different tests, and reproducing complete scenarios build-up by different machines. Next step will be to extend STFW to VxWorks™ (VxWorks, 2006), the worldwide known OS for real-time system, and the most utilized for the control of automata.

REFERENCES

Anderson T. and Knight J.C., *A Framework for Software Fault Tolerance in Real-Time Systems*, IEEE Trans. Soft. Eng., Vol. SE-9, no.3, pp.355- 364, 1983.

Basili, V. R., Caldiera, G., and Rombach, H. D., *The Goal Question Metric Approach*, Encyclopedia of Software Engineering, Wiley&Sons Inc., 1994.

Di Biagio C., Pennella G., and Cantone G., *Comparing Tools for Testing Critical Software. The Case Study of "Software Framework 2.0"*, TR 20060426.1, MBDA Italy, 2006.

Di Biagio C., Pennella G., Lomartire A., and Cantone G., *An Introduction to Characterization of Monitors for Testing Safety-Critical Software*, Procs. of ICSOFT 06 (these Proceedings), Setubal, 2006.

Cantone, G., and Donzelli P., *Production and Maintenance of Goal-oriented Measurement Models*, International Journal of Software Engineering & Knowledge Engineering, World Scientific Publishing Company, Vol. 10, No. 5, pp. 605-626, 2000.

EPRI, Handbook for verification and validation of digital systems, Vol.1: Summary, EPRI TR103291, Vol.1, 1994.

IEEE, IEEE/EIA 12207.0-1996 Industry Implementation of International Standard ISO/IEC 12207: 1995 (ISO/IEC 12207) Standard for Information Technology Software Life Cycle Processes, in *IEEE/EIA* 12207.0-1996, 1998, pp. i-75.

Leveson. N. G., Software safety: Why, what, and how. *Computing Surveys*,18(2):125-163, June 1986.

Isaksen U., Bowen J. P., and Nissanke N., *System and Software Safety in Critical Systems*, December 1996.

Lilja D. J., *Measuring Computer Performance*, Ed. Cambridge University Press, 2000.

QUEST SPOTLIGHT™ http://wm.quest.com/library/docs/spotlightwindows/SpotlightWindows.pdf (last access, March 2006).

Simser D. and R.E. Seviora, Supervision of Real-Time Systems Using Optimistic Path Prediction and Rollbacks, *Procs. Int'l Symp. Software Reliability Eng.* (ISSRE), pp. 340–349, Oct. 1996.

SOLARIS PERFORMANCE METER™ 2.0.0 http://docsun.cites.uiuc.edu/sun_docs/C/solaris_9/SUNWabe/CDEUG/p125.html (last access, March 2006).

TOP™ - William LeFebvre's http://www.uwsg.iu.edu/UAU/system/top.html (last access, March 2006).

Tsai J.J., Yang S.J., Monitoring and Debugging of Distributed Real-Time Systems, *J.J. Tsai and S.J. Yang, eds*., IEEE CS Press, 1995.

Ubench 2.0™ , http://www.phystec.com/download/ubench.html (last access, March 2006).

Weyuker E.J., On Testing Non-Testable Programs, *The Computer J*., vol. 25, no. 4, pp. 465–470, 1982.

VxWorks, http://www.windriver.com (last access, April 2006).