From Software Sensitivity to Software Dynamics: Performance Metrics for Real-Time Software Architectures

Janusz Zalewski
Computer Science Department
Florida Gulf Coast University
Ft. Myers, FL 33965-6565, USA
zalewski@fgcu.edu

Abstract

This paper outlines an approach and presents preliminary results to describe behavior of real-time software architectures using concepts from dynamical systems. The basic idea is to change deadlines by decreasing them within certain range and take measurements on the number of missed deadlines and the total time by which the deadlines have been missed. A resulting curve, illustrating the dependence between the missed deadlines and their decrease, characterizes system dynamics. It can be approximated by a straight line or an exponential shape, from which certain dynamic parameters can be calculated.

1. Concept of Basic Metrics

This paper addresses the problem of evaluating behavioral properties of real-time software architecture. When building a real-time application, especially when designing its software architecture, the developers need to have some way of assessing its performance before deployment. Therefore it is essential to find good metrics and measures characterizing behavioral properties of software, that is, its dynamics. In this section, we briefly explain the basic concept of our approach, as well as present a benchmark, which is a starting point to deriving more involved metrics of real-time architecture performance.

1.1 Responsiveness and Timeliness

To deal with performance of real-time software, we need to realize first, how such software responds to events that it has to handle. Our model uses a representative real-time application, embedded avionics architecture, widely accepted by the aviation and aerospace industry (Fig. 1 [1]). It shows the layered approach to handling events by a real-time computer.

In general, an event, after passing through hardware, is first handled by a real-time kernel (part of an operating system layer in Fig. 1) and only then by an application task. A respective sequence of handling events, following the layered approach from Fig. 1, has been outlined in Fig. 2.

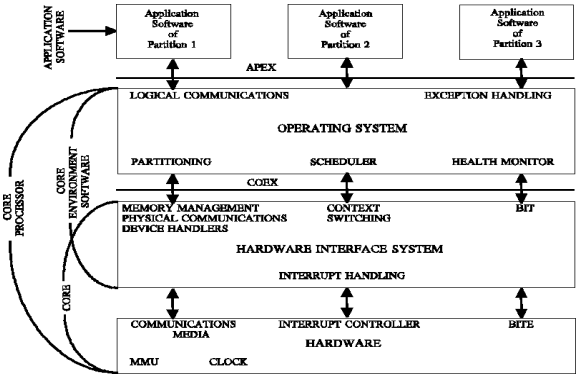

Fig. 1. Embedded avionics architecture (from ARINC 653 [1]).

The key point to remember is that before an event reaches the application task designed for it, it has to pass through several layers of hardware and software.

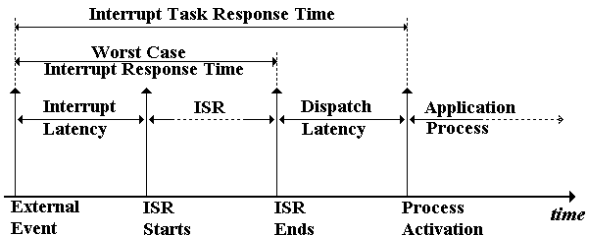

Fig. 2. Systematic view of event handling.

All latencies and response times illustrated in Fig. 2 contribute to the basic real-time system parameter we can call *responsiveness*. It can be measured as the worst-case time that elapses from the occurrence of a particular event to the start of its processing by an application task, and in absolute terms is equal to the interrupt task response time defined in Fig. 2. This time can be determined separately for every real-time kernel and underlying hardware platform independently of any application, therefore it is not of interest for this work.

What we concentrate on in this paper is how to evaluate behavior of the application processes (tasks). To do this, we have to look at another parameter, which can be termed *timeliness*, that is, the property characterized by the worst-case time that is needed for processing of an already perceived event by an application task. The question we are trying to address below is: How to measure this property?

1.2 Real-Time Software Benchmark

For a real-time architectural pattern described in [2], composed of all essential elements of a real-time application (such as measurement/control, human, communications, database, timing, and processing tasks and interfaces), we developed a simple benchmark to analyze its behavior. For simplicity, we include here only a brief description of a five-task data acquisition system, which performs the following functions, illustrated in Fig. 3 [3]: periodic and aperiodic sensor readout (T1 and T2), computational algorithm (T5), database access (T4), and user interface (T3). Each of these tasks may run on the same processor or on separate nodes communicating with other selected nodes.

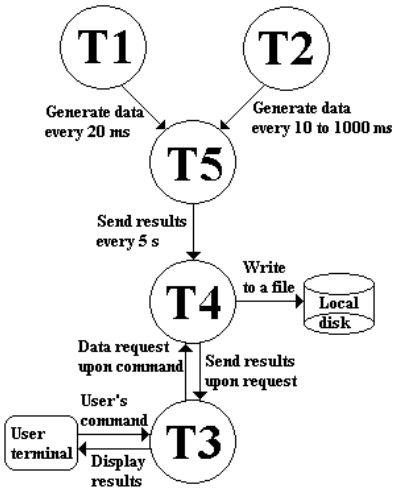

Fig. 3. Five-task benchmark architecture.

Under the assumption that a real-time application demonstrates satisfiable performance if it meets its timing constraints (deadlines), we proposed the following two metrics for evaluation of *timeliness*:

- the number of times the deadlines are missed (percentage of missed deadlines),
- the overall accumulated time by which the deadlines are missed.

They can be evaluated for a particular software module on a particular node.

To validate the metrics so conceived, we performed sample experiments, for various configurations of VxWorks real-time kernel and C/Java sockets, and for various CORBA implementations of the five-task benchmark. The results reported in [3] validated its concept confirming essential facts, such as superiority of shared memory communication over sockets or much better performance of real-time CORBA/TAO than CORBA's non-real-time versions.

At the same time, based on validation experiments, several other observations were made, such as:

- how much one implementation is better or worse than another or
- at what particular values of deadlines performance of the application begins to degrade.

The benchmark was then applied to two large scale applications implemented in CORBA: an air traffic control system simulator (ATCS) [3] and a satellite ground control station (SGCS) [4].

In the air traffic control problem, it was found to be effective for detecting unusual behavior of software in real time, when applied to the evaluation of performance. For example, experimental results show quantitatively how much degradation in performance of a certain component in ATCS (module performing conflict detection and resolution) can be expected, if the number of aircraft directly involved in computations increases. If the number of aircraft is increased five times, the performance of the whole system degrades from two times for light-load modules (such as a Collision Detection module, Fig. 4) to two orders of magnitude for heavy-load modules, in terms of the overall time the deadlines are missed.

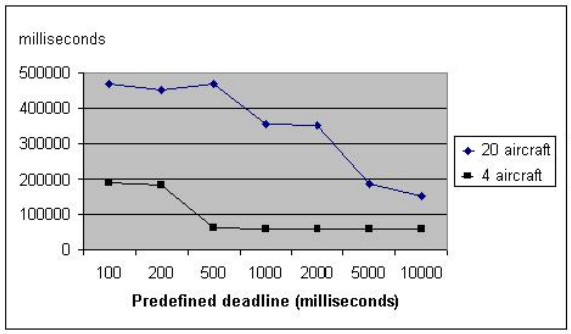

Fig. 4. Overall time the deadlines are missed (ms) in 100 experiments, for Collision Detection module.

In addition, applying the benchmark to the ATCS showed quantitatively where the computational bottleneck of the entire system was located (it turned out to be in the GUI module, where most functions were concentrated).

In the SGCS system, the response time was analyzed using the percentage of deadlines missed for the satellite tracking module in such implementation. The experiments measured the response time in the range of 3000 to 10000 ms for each client. On increasing the load, i.e., the number of clients sending requests, from one to three and five, the range of response time also increases linearly. Fig. 5 [4] shows typical deterioration of performance measured in terms of missed deadlines.

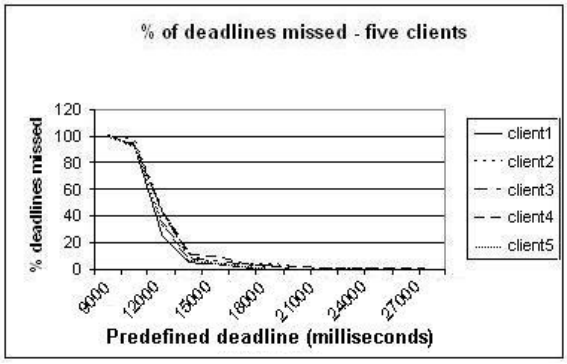

Fig. 5. Percentage of deadlines missed by the Tracking Module for 5 clients.

These basic results are used in the next section to derive two new measures of performance.

2. Derived Metrics

2.1 Software Sensitivity

Analyzing more closely the graphs in Fig. 4 and 5, one is tempted to ask a question: How significantly the degradation in meeting deadlines progresses, when the deadlines are shortened? For example, some curves ascend slowly, as in Fig. 4, which corresponds to slow performance degradation, while some other curves may ascend very sharply, as in Fig. 5, which would mean a relatively rapid degradation of performance.

Referring to Fig. 4, one can notice that, within the range of deadlines studied, the implementation is relatively indifferent to the shortening of predefined deadlines, since the performance degrades roughly twice over the range studied. This means that the system is not very sensitive to changes of deadlines within this range. However, the same statement is definitely not true for the implementation shown in Fig. 5, where the degradation of performance progresses significantly faster with the deadlines shortened. The performance of the Tracking Module under study deteriorates rapidly within 20-25% of deadline change in a certain range.

Based on these observations, we can define a new parameter as a metric of software performance, called *software sensitivity* [3]:

> *Software sensitivity is a measure of the magnitude of a software response to changes in the values of deadlines (when they are increased or decreased).*

The interpretation of software sensitivity in this sense is that the faster the curves in Fig. 4 and Fig. 5 ascend/descend the more sensitive respective software architecture is. When software is sensitive, a small change of a deadline length causes relatively larger changes in the number of deadlines missed. Practically, software sensitivity considered as metric shows whether performance degradation occurs sharply or gracefully.

It is important to note that quantitatively, it is not just the slope of the curve, what we mean by sensitivity, therefore it is different from traditional understanding of this concept. Software sensitivity can be represented quantitatively by the ratio of the change of response over the range of deadline lengths for the changed interval. Formally, sensitivity is a parameter that takes into account the slope of each curve, in relative terms, making curves and respective systems comparable.

Practically, the first step in calculating the value of sensitivity, S, is to linearize the curve in the interesting region. Then, to account for relative differences in absolute values of deadlines for different systems, the actual value of sensitivity is calculated from the straight line fits, according to the following formula, where (x_1,y_1) and (x_0,y_0) are coordinates of respective points on the straight line reflecting the range considered:

$$S = \frac{(y_1 - y_0)/[(y_1 + y_0)/2]}{(x_1 - x_0)/[(x_1 + x_0)/2]}$$

By including relative values of ranges, the formula allows for comparison of different systems, for which deadline lengths may significantly differ in absolute values, but the system's speed of response, that is, sensitivity, may be equivalent.

Using this method, we can assess sensitivity of various implementations. For example, as reported in [3], for a sophisticated air-traffic control system simulator, sensitivity measurements showed that the least sensitive implementation is the one with the interaction between the Communication component and the GUI component, which provided a significant insight into the way aircraft hand-offs were handled in software. Discovering such dependencies has a positive impact on software redesign at the development level, before deployment. Overall, the sensitivity parameter tells designers, how fast the system degrades, if the deadlines are shortened, that is, how fast it gets saturated.

2.2 Software Dynamics

Software sensitivity, although used for assessing the timing behavior of software, is essentially a steady-state parameter, because it does not depend on dynamic properties. One would, however, be tempted to characterize software behavior using true dynamic terms.

Looking closely at the respective performance curves presented in Fig. 4 and 5, one may ask a question: How linear is the curve in its descending region and would another type of approximation describe it better? Trying exponential approximation, under the assumption that the descending part of the curve is not a straight line but an exponential curve, we can characterize it by a new notion called *software dynamics*, capturing the dynamic response to changes in software behavior under varying load. The initial physical interpretation of the exponential curve is that the software performance is gradually descending when the load increases.

Quantitatively, for an exponential curve of the form $y=K*\exp(-x/\tau)$, representing the behavior of software, to characterize it one can use the following transfer function representing it as a first-order dynamic system:

$$G(s) = y(s)/x(s) = K / (\tau*s + 1)$$

where its gain $K=1$, τ is a time constant, and s is a complex variable. Time constant τ characterizes system dynamics and is a measure of the speed of system's response to changes in load. Following the practice of control engineering it can be calculated as 25% of a settling time. Settling time, in turn, is the point where the descending curve reaches 2%-4% of its maximum.

With this formula, the dynamics of software architecture can be represented by a dynamic equation using a Laplace transform. This approach to software is completely unique in a sense that there are no available records in scientific literature of attempts to study the concept of software dynamics as a continuous property of software, which is by its nature discrete. A review of computing literature for the last two decades indicated that there were only two papers dealing with software dynamics at all, but in a different sense than ours.

First, Motus, in 1985 [5], used the term software dynamics but in a different meaning, at a specification not the execution level, to study "the set of time constraints that a software system has to satisfy". Bernstein, in 1996 [6] used this term in a meaning similar to ours, postulating the use dynamic measures to evaluate software quality. However, Bernstein's idea is consistent with the concept of software rejuvenation, which refers to software's long-term execution. In contrast to that, we are postulating to analyze software behavior in short term, by studying software in real time, under changing load.

3. Experiments

To verify the concepts of software sensitivity and software dynamics we conducted a series of experiments for a satellite ground control station being built at Florida Space Institute, in Orlando and Kennedy Space Center [4]. In a testbed performing the basic functions of the satellite ground control station, the following software modules were implemented: telemetry, GPS and database components connected to CORBA with GUI, running on a variety of heterogeneous platforms, as illustrated in Fig. 6.

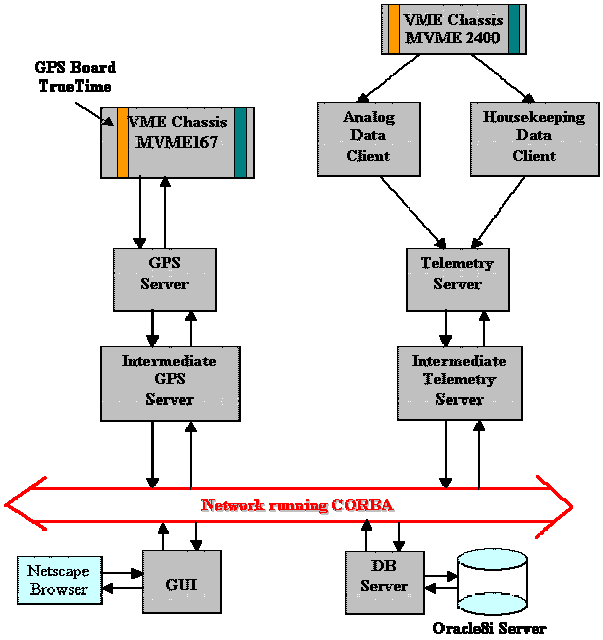

Fig. 6. Satellite ground control station – partial implementation of a testbed.

To measure the response time to GUI commands directed to telemetry, GPS and database components, experiments were conducted in the above configuration. Figures 7 and 8 present the results for a telemetry component, to evaluate its sensitivity and dynamics, respectively. Results for the GPS component have been published elsewhere [7]. The combined results for all components are presented in Table 1. The software dynamics is measured as a time constant of the first order system, as explained in Section 2.2. The results show consistency with the intuitively perceived relative speed of respective modules.

Table 1. Results of evaluating sensitivity & dynamics.

Component	Sensitivity	Dynamics [ms]
Database	--	165.0
Telemetry	1.00	87.5
GPS	1.64	15.0

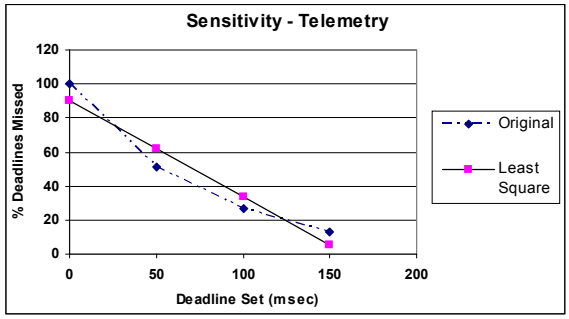

Fig. 7. Illustration of the sensitivity measurement for the telemetry component of the satellite ground station.

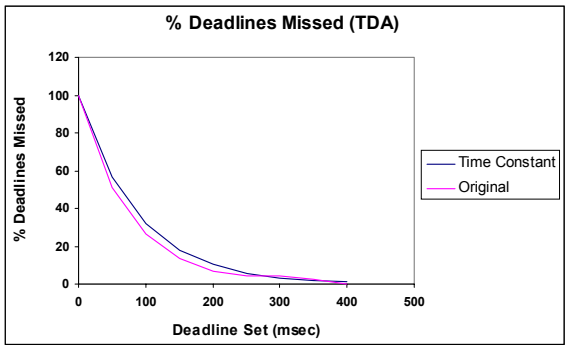

Fig. 8. Illustration of the software dynamics evaluation for the telemetry component of the ground station.

4. Conclusion

The objective of this work was to introduce a new metric to characterize dynamic properties of real-time software architectures. Using a benchmark built on the concept of changing deadlines to see impact on the number of times (percentage) the deadlines are missed, and the overall time the deadlines are missed, we introduced two composite measures:

- software sensitivity, that characterizes the magnitude of software response to changes in deadlines, and
- software dynamics, which leads to using a time constant as a characteristic of dynamic properties of software.

Software sensitivity is similar in technical meaning to the concept of *elasticity* used in economics. Software dynamics goes further and involves a concept of a *first-order dynamic system* by associating dynamic properties with software. Thus, software can be described by notions from the theory of dynamical systems and its characteristics can be included in computational models involving Laplace transforms and time constants.

Experiments conducted for the telemetry module in a satellite ground control station testbed have initially confirmed the usefulness of both concepts, software sensitivity and software dynamics. They characterize steady-state and dynamic properties of software, respectively, and can be used to evaluate software architectures both for implementation properties and at the design phase.

References

[1] ARINC Inc., *Avionics Application Software Standard Interface*, ARINC Specification 653, Baltimore, MD, 1997

[2] Sanz R., J. Zalewski, Pattern-Based Control Systems Engineering, *IEEE Control Systems*, Vol. 23, No. 3, pp. 43-60, July 2003

[3] Guo D., J. van Katwijk, J. Zalewski, A New Benchmark for Distributed Real-Time Systems: Some Experimental Results, *Proc. 27th IFAC/IFIP/IEEE Workshop on Real-Time Programming*, Łagów, Poland, May 14-17, 2003, pp. 141-146

[4] Bhatia M., R. Johnson, J. Zalewski, Evaluating Performance of Real-Time Software Components: Satellite Ground Control Station Case Study. *Proc. SEA2003, 7th IASTED Int'l Conf. on Software Engineering and Applications*, Marina del Rey, Calif., November 3-5, 2003, pp. 781-786.

[5] Motus L., A. Lomp, Distributed Computer Control System's Software Dynamics Specification, *Proc. 9th IFAC Congress*, Pergamon, Oxford, UK, 1985, Vol. 5, pp. 2657-2661

[6] Bernstein L., Better Software Through Operational Dynamics, *IEEE Software*, Vol. 13, No. 2, pp. 107-109, 1996

[7] Zalewski J., Software Dynamics: A New Measure of Performance for Real-Time Software, *Proc. SEW-28 28th NASA Goddard Software Engineering Workshop*, Greenbelt, MD, December 3-4, 2003, pp. 120-126