Open-Loop Testing of a MARS System [1]

Werner Schütz

Institut für Technische Informatik
Technische Universität Wien
Austria

schuetz@vmars.tuwien.ac.at

Research Report Nr. 13/90

September 1990

Institut für Technische Informatik
Technische Universität Wien
Treitlstraße 3/182/1
A-1040 Wien
Austria

Phone: (43) 222-58801-8181
Fax: (43) 222-569149

[1] This work was supported in part by the ESPRIT Basic Research Project 3092 "Predictably Dependable Computing Systems"

Abstract

This paper is a first step towards a practical evaluation of some previously proposed system architectures for conducting application software tests of time-driven real-time systems. Of special interest was an evaluation of the reproducibility of test runs. While familiarity with MARS is assumed, the application which was used to conduct these experiments is described in some detail. Then the experiments which have been performed are given, along with their necessary preparations. Finally, we present the results and their interpretation and conclude with the consequences of these results on the proposed test system architectures.

1 Introduction

Previously, this author has developed and proposed a test strategy for time-driven real-time systems (and especially for MARS [Kop85,Kop89]), as well as some test system architectures for actually conducting such tests [Sch89,Sch90]. Two different test modes are distinguished: With open-loop tests, prepared test data are input by the system and its outputs (test results) are recorded for later (off-line) evaluation. With closed-loop tests, test data are generated on-line by environment simulation software. It is considered especially important that the test system architectures are designed such that such tests are reproducible, i.e. they must give the same results if repeated. This is generally considered to be very difficult, if not impossible, to achieve with current state-of-the-art real-time systems, let alone with state-of-the-practice systems.

This work is a first step towards a practical evaluation of the proposed test system architectures. The focus is on open-loop tests; these are easier to implement because no environment simulation software needs to be written. Some aspects of the results, however, do also apply to the test system architecture for closed-loop tests.

The experiments and their evaluation are based on a MARS system executing the recently developed "Rolling Ball" application [Kop90]. In the following sections we first describe this application in more detail, then describe the experiments that were conducted and the necessary preparations. A summary of the results is given, followed by an interpretation of these results. The paper concludes by giving the consequences of the results for the proposed test system architectures.

2 The "Rolling Ball" Application

The application which was used to conduct the experiments described in section 3 is the so-called "Rolling Ball" [Kop90]. This is a real-time control problem whose aim it is to balance a ball on a flat surface (plane) in such a way that the ball follows a pre-defined course. The position of the ball is periodically (40 ms) sent to the MARS cluster which then computes commands for a set of servo motors to set the inclination of the plane. A two-dimensional orthogonal coordinate system on the plane is used.

In Figure 1 the configuration used for the experiments is shown. For brevity, the connection to the host computer for loading the MARS components has been omitted.

The individual parts of this system cooperate as follows: Since the video camera

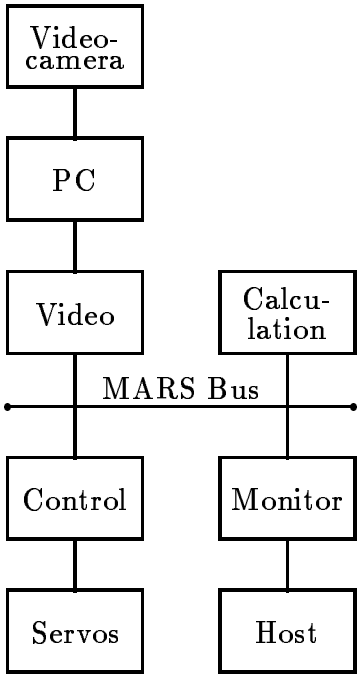

Figure 1: Experiment Configuration

could not be connected directly to a MARS component, a PC has to be used as an intermediary. The PC reads a picture from the camera every 40 ms, processes it, and sends data (i.e. one of coordinate system initialization, course initialization, or ball position) over a serial line to the Video component.

The Video component processes all the initialization data, then, as long as it continues to receive ball positions, it computes the current status of the ball (consisting of acceleration, velocity, and position, each for both the x- and the y-axis), and sends this status to the Calculation component.

The Calculation component uses this information and the description of the required course (also received from the Video component) to calculate the accelerations for the x- and the y-axis, respectively, which are required to keep the ball on the defined course (or bring it back to this course).

Finally, the Control component computes the angles that the plane must be set to in order to achieve these accelerations. These angles are then sent to the servo motors via another serial line, thus closing the control loop.

The Monitor component is not part of the original application; it has been written for observing and recording the experiment results. (Its function is described in Section 3).

Note that in this application there is only one application task per component; this simplifies things a bit. Also, for running the experiments, no redundant components were used, i.e. there was just a single instance of each application task. Makefiles and Application Description Files (`ball.adf`) can be found in `~schuetz/project.test/open_loop`.

3 The Experiments

This section describes the experiments that have been undertaken, their preparation, and how they were carried out. Some information about the effort that was necessary is also given.

3.1 Preparations

In order to run the intended experiments one had to decide first how inputs should be supplied to the system, and which outputs should be observed and recorded for later evaluation. Three different methods for supplying inputs have been used which will be described in more detail in the following subsections because these turned out to be important parameters of the experiments.

There were essentially two choices for the outputs that should be observed and recorded. One possibility was to choose the messages containing the required accelerations that the Calculation component sends to the Control component; the second possibility was to use the data (i.e. angles) sent by the Control component to the servo motors via a serial line. For all the experiments that were conducted the first possibility was selected since the time stamps of each MARS message allowed an easy check of completeness of the recorded experiment outputs. No such time stamps are available with the data sent by the Control component on its serial line.

Thus, a Monitor component executing a Monitor task was added to the system for recording experiment outputs. The Monitor task periodically (i.e. each 40 ms) tries to receive a message from the Calculation component. If it succeeds the receiving time stamp and the two acceleration values are output to the Monitor component's serial line interface. Otherwise, the string "`no message`" is output. The serial line is connected to a host computer which records the data received in a file. (On `idefix` use `tip m1` with the script option enabled.) During all the test runs it was generally the case that the output was recorded completely, as intended. Only occasionally did the Monitor (or the host – it is undecidable which part of the system is responsible) miss the output of one iteration. This was easily detected by checking the recorded time stamps.

Another question was which test data (i.e. input data) to use. Ideally these should be as similar as possible to data occurring during "real" executions of the system. Therefore, a "trace function" was added to the PC software. When this function is enabled, the PC software executes as before, but in addition each data packet sent to the Video component is also stored in a "trace file". The modified version of the PC software can be found in `C:\TC\SRCHBALL\OPENLOOP`. The trace function is enabled by invoking the program with `video -t <trace_file_name>`.

The effort required to prepare the experiments was approximately as follows: 2 days for understanding, modifying, and testing the Makefile and the Application Description File, less than half a day for implementing the Monitor task, and again about half a day for modifying the PC software, including adding a "replay function" (see below). There was no need to change the configuration of the MARS system and its peripheral devices, only a serial line between the Monitor component and a host computer had to be added.

3.2 Data Input, Type 1

The first version of providing input to the Video component was based on the fact that it is easy (with the modified version of the PC software) to get a "trace file". Now, all that needs to be done is to use the data in this trace file as inputs to the MARS cluster.

Therefore, a "replay function" was added to the PC software which does not read and process pictures from the video camera, but reads the trace file and sends the data packets found there to the Video component. In this version this is done line by line, i.e. one line of the trace file is read and sent to the video component, then the next line of the trace file is read, etc. The replay function is enabled by invoking the PC program with `video -r <trace_file_name>`.

We will call this type of input "Simple Input" subsequently.

3.3 Data Input, Type 2

Since using Simple Input did not give the expected results (see Section 4), we tried to improve the method of supplying input data. The goal was to reduce the (obvious) variation of the delivery time of input data supplied by Simple Input. It was assumed that reading a file from disk (i.e. the trace file) is one such major source of time variability.

Therefore, a more sophisticated version of the replay function was implemented. It first reads all the data packets in the trace file and stores them in an internal

data structure (subject to space available). Only after that does it begin to send the data packets to the Video component.

This type of input will be called "Cached Input" from now on.

3.4 Data Input, Type 3

Finally, while Cached Input generally turned out to be a slight improvement, it is still far from optimum (see Section 4). The last version of input delivery was therefore designed for removing all uncertainties concerning the time of input arrival. This means that input data could not be delivered via the PC and the serial line.

Instead, the Video component had to be modified such that it simulates input reception via the serial line. An internal data structure for holding the input data (i.e. the content of the trace file) was added to the source code of the Video task; this data structure is initialized at compile time. In addition, instead of receiving input from the serial line interface, input is copied from this internal data structure into the message buffer of the serial line and is time-stamped with the current time. For simplicity, this is done in such a way that a new input packet is ready with every iteration of the Video task, i.e. there are no gaps in the input sequence.

Of course, a new version of the Video task has to be edited and compiled for each new trace file. However, performing the above modifications to the Video task was accomplished in less than one hour, and should be a matter of minutes subsequently.

From now on, we will call this type of input "Builtin Input".

3.5 Execution of the Experiments

All the test runs have so far been conducted with one set of test data which is contained in the file `C:\TC\SRCHBALL\OPENLOOP\RUN1.DAT` on the PC. It consists of 3 coordinate system initialization packets, 185 course initialization packets, and 591 ball position packets, and is thus of moderate, but not trivial size.

In the course of the experiments, it also turned out that the time delay between sending data packets has significant influence on the results in the cases of Simple and Cached Input. This is implemented by a `delay` statement in the PC software. Therefore, experiments were run with this time delay set to 20, 35, 40, and 45 ms. (Remember that the basic execution period of this application is 40 ms.)

Thus there are two independent variables to characterize each test run: the type of input, and the time delay between individual data packets (for Simple and Cached Input only). These variables were systematically set to the above-mentioned values,

and for each combination of values at least three test runs were conducted. The most interesting results are given in the next section.

4 Results

When summarizing the results obtained by the experiments, it is simplest to start with Builtin Input: When using Builtin Input, the recorded test results of all test runs are identical.

This means that the sequence of acceleration values is identical for the whole test run. The associated time stamps differ approximately by 100 μs maximum (also for Simple and Cached Input) and are never taken into account for evaluating and comparing test runs.

As an aside, it is noteworthy that test runs with Builtin Input that were conducted without re-loading all the MARS application software gives identical results also, but these results differ from the results of test runs conducted immediately after re-loading. [1] As a consequence, only results from test runs executed immediately after a complete re-load are taken into account from now on. This is no loss of generality because all test runs with Simple or Cached Input exhibited some deviations and differences at some point in time during such runs; test runs conducted without re-load are then expected to differ even more widely since execution starts with some "arbitrary" (i.e. unknown and irreproducible) internal state.

The following tables show the amount of agreement between individual test results by indicating the number of the execution period where the recorded acceleration values differed for the first time since the start of the recording, i.e. a value of '1' means that the two test results differed in their first recorded message, a value of '2' means that the first message was identical, but the second was not, etc. Note that both acceleration values must be identical for the message to be considered identical, and that the time stamps are not taken into account, as mentioned above. For Simple and Cached Input individual test outputs are identified by a number identical to the delay time used and by a lower-case letter for distinguishing those with identical delays.

First, we show the "internal consistency" of test runs with the same input type and the same amount of delay, and in the same tables compare test runs with the same input type, but different delay times.

[1] A note to the "ball implementors": This means that there remains some (hidden?) state in the system, even after re-initializing the coordinate system and the course information. I do not know if this is good or bad, intended or unintended, or if it does not matter.

a) Simple Input

	20a	20b	20c	35a	35b	35c	40a	40b	40c	45a	45b	45c
20a	–	206	6	21	21	21	21	21	21	21	21	21
20b	–	–	6	21	21	21	21	21	21	21	21	21
20c	–	–	–	6	6	6	6	6	6	6	6	6
35a	–	–	–	–	111	111	111	111	111	111	111	111
35b	–	–	–	–	–	116	116	116	116	116	116	116
35c	–	–	–	–	–	–	306	306	306	306	306	306
40a	–	–	–	–	–	–	–	341	345	338	338	338
40b	–	–	–	–	–	–	–	–	341	338	338	338
40c	–	–	–	–	–	–	–	–	–	338	338	338
45a	–	–	–	–	–	–	–	–	–	–	371	371
45b	–	–	–	–	–	–	–	–	–	–	–	372
45c	–	–	–	–	–	–	–	–	–	–	–	–

b) Cached Input

	20a	20b	20c	35a	35b	35c	35d	40a	40b	40c	45a	45b	45c
20a	–	6	6	21	21	21	21	21	21	21	21	21	21
20b	–	–	206	6	6	6	6	6	6	6	6	6	6
20c	–	–	–	6	6	6	6	6	6	6	6	6	6
35a	–	–	–	–	302	303	302	302	302	302	302	302	302
35b	–	–	–	–	–	302	303	303	303	303	303	303	303
35c	–	–	–	–	–	–	302	302	302	302	302	302	302
35d	–	–	–	–	–	–	–	303	303	303	303	303	303
40a	–	–	–	–	–	–	–	–	337	335	336	336	336
40b	–	–	–	–	–	–	–	–	–	335	336	336	336
40c	–	–	–	–	–	–	–	–	–	–	335	335	335
45a	–	–	–	–	–	–	–	–	–	–	–	378	368
45b	–	–	–	–	–	–	–	–	–	–	–	–	368
45c	–	–	–	–	–	–	–	–	–	–	–	–	–

Next, we compare the amount of agreement when varying the type of data input, while keeping the delay time (where appropriate) constant.

c) Simple and Cached Input with delay 20 ms

Cached Input	Simple Input 20a	20b	20c
20a	203	203	6
20b	6	6	203
20c	6	6	203

d) Simple and Cached Input with delay 35 ms

Cached Input	Simple Input 35a	35b	35c
35a	111	116	302
35b	111	116	303
35c	111	116	302
35d	111	116	303

e) Simple and Cached Input with delay 40 ms

Cached Input	Simple Input 40a	40b	40c
40a	336	336	336
40b	336	336	336
40c	335	335	335

f) Simple and Cached Input with delay 45 ms

Cached Input	Simple Input 45a	45b	45c
45a	369	369	369
45b	369	369	369
45c	368	368	368

g) Simple and Builtin Input

Simple Input	Builtin Input
20a	21
20b	21
20c	6
35a	111
35b	116
35c	266
40a	266
40b	266
40c	266
45a	266
45b	266
45c	266

h) Cached and Builtin Input

Cached Input	Builtin Input
20a	21
20b	6
20c	6
35a	266
35b	266
35c	266
35d	266
40a	266
40b	266
40c	266
45a	266
45b	266
45c	266

5 Interpretation of Results

An obvious observation is that it is not easy to achieve reproducible behaviour of the MARS system if inputs are delivered via a serial line. However, if one does try to do so, the following observations can be made:

The chosen time delay has a big influence on the length of the reproducible output data sequence. If the time delay is about half the application's execution period, reproducibility is generally bad, except for lucky coincidences. The receiving component is "overwhelmed" with input packets, and it is unpredictable which ones are actually received.

Reproducibility improves when the time delay is raised to match the application's execution period. In the vicinity of this value, reproducibility seems to be directly proportional to the chosen time delay.

Using Cached Input instead of Simple Input has no visible effect when the time delay is too small, improves reproducibility when the time delay is slightly less than the application's execution period, but seems to impair more rather than improve reproducibility if the time delay is equal to or slightly larger than the application's execution period.

From tables a) and b) one can also see that changing the time delay not only changes the *length* of the reproducible test result sequences, but that the initial part of these sequences is the *same*, even for different delay times. That means that the

same behaviour is reproduced initially, independent of the selected delay time, and that a good choice of the delay time improves the time interval during which the reproduction of this behaviour can be sustained.

In general, choosing Cached Input over Simple Input does not improve reproducibility substantially. On the other hand, these two input methods do not contradict each other either. This can be seen with tables c) to f) which show that the two input methods produce identical initial sequences of test results. The lengths of these initial sequences depend on the chosen time delay and are (with only a few exceptions) very close to the minimum of the length of identical initial sequences from the appropriate table entries from a) and b).

If we assume that the results achieved with Builtin Input are the "correct" ones, then tables g) and h) show that correct reproducibility can be achieved for some time even when sending inputs over a serial line. The time delay has to be chosen "approximately right", but the exact input method (Simple or Cached) is not of significance. However, there seems to exist an upper limit for the maximum time interval during which test results are correctly reproduced. Moreover, it is not possible to reliably determine for which time interval an individual test result is actually identical to the "correct" one.

But anyhow, we are not interested in partial reproducibility, but in complete reproducibility. Using Builtin Input is so far the most promising way to achieve this, and it has been proven to be practical also.

Even so, the fact that complete reproducibility was so easy to achieve comes as a bit of a surprise. This is because with Builtin Input time stamping of the external messages was simulated by supplying the current time. Thus there was no explicit control over the time differences between message receptions, i.e. over the difference of receiving time stamps. However, such time differences are used later in this application, e.g. for calculating velocities and accelerations. The value and the time domain are therefore not independent of each other in this application. Thus differences in receiving time might manifest themselves in the computed values eventually.

Two explanations can be offered for the fact that no such effects were observed here: First, the receiving time differences are identical (i.e. reproducible) after all because the call to get the current time was issued a fixed number and type of instructions after each task invocation. Second, the actual variations are very small (i.e. of the order of μs) and are eliminated by the effects of numerical rounding.

6 Conclusions

With this work we have tried to evaluate possibilities for executing reproducible open-loop test runs on the MARS system. These experiments were conducted on top of the recently developed "Rolling Ball" application.

The timing of inputs sent via a serial line is essentially uncontrollable, so using Builtin Input (as described above) is the only practical possibility for achieving reproducibility, at least until alternate methods of external input are designed and implemented. This is despite its limitation of an upper limit on the size of the test data, due to the finite size of a component's main memory.

This methodology is totally compatible with the test system architecture proposed in [Sch89] and [Sch90]. The results presented in this paper prove that it is indeed possible to achieve reproducible test runs with this proposed architecture and thus reinforce the previously developed test system architecture and test strategy.

Simultaneously, however, the results indicate that the proposed test system architecture for closed-loop tests [Sch89,Sch90] is not yet feasible because inter-cluster communication can so far be only implemented using serial lines, i.e. in the current application there exists no time-controlled method for inter-cluster communication. Temporary work-arounds, e.g. integrating application and simulation software into one cluster, are possible, however.

References

[Kop85] H. Kopetz and W. Merker. The Architecture of MARS. In *Proc. 15th Int. Symposium on Fault-Tolerant Computing*, pages 274–279, Ann Arbor, Michigan, June 1985.

[Kop89] H. Kopetz, A. Damm, Ch. Koza, M. Mulazzani, W. Schwabl, Ch. Senft, and R. Zainlinger. Distributed Fault-Tolerant Real-Time Systems: The MARS Approach. *IEEE Micro*, 9(1):25–40, Feb. 1989.

[Kop90] H. Kopetz, R. Noisser, et al. The Rolling Ball: A Real-Time Control Problem. Research Report 2/90, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, 1990.

[Sch89] W. Schütz. Testing a Distributed Real-Time System — The MARS Approach. Research Report 11/89, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, July 1989.

[Sch90] W. Schütz. A Test Strategy for the Distributed Real-Time System MARS. In *IEEE CompEuro 90, Computer Systems and Software Engineering*, pages 20–27, Tel Aviv, Israel, May 1990.