Real-Time Simulation in the Distributed Real-Time System MARS[1]

Werner Schütz

Institut für Technische Informatik
Technische Universität Wien
Austria

schuetz@vmars.uucp

Research Report Nr. 7/90

March 1990

A slightly shorter version was accepted for the
European Simulation Multiconference 1990,
Erlangen, BRD, June 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 (PDCS)".

Abstract

This paper explores the problems of using a real-time environment simulator for testing real-time systems. An overview of real-time systems and of real-time simulation is given first. Then it is shown how real-time simulation has to be defined in order to be consistent with our view of real-time systems. Within the context of our distributed real-time system MARS (*MA*intainable *R*eal-Time *S*ystem) these concepts are finally applied. A technique to construct real-time environment simulators is developed, and a test bed architecture is described that permits the use of this simulator for driving test runs of a MARS system. This technique is specifically tailored to allow easy integration with the existing MARS Design Methodology and Design System.

1 Introduction

Real-time systems are more and more applied in increasingly complex and critical applications. Application areas include control of automobile engines, nuclear power plants, industrial process control systems, aircraft avionics, and robotics [Sta88]. The research community agrees to a large extent [Sta88] that *predictability* is one of the most important properties of next-generation real-time systems in order to achieve the level of dependability [Lap89] which the public legitimately expects and demands in such application contexts.

In our work on MARS (*MA*intainable *R*eal-time *S*ystem) we have concentrated on a specific class of real-time systems, the so-called *hard real-time systems* (see below). In such systems the required timing constraints (deadlines) absolutely have to be met under all anticipated fault and load conditions. Design and implementation of a hard real-time system have to *guarantee* that all deadlines will always be met. In MARS, predictability is achieved by *deterministic system execution* and pre-runtime verification of the timing behaviour.

One possibility to evaluate the correct and timely performance of a system and to gain confidence in the achieved level of dependability is *testing* the system. This paper deals both with the problems and with techniques of using real-time simulation for test purposes. Before going into any details, we give a general introduction to real-time systems and the most important background information on our MARS system. Then we summarise our work on testing real-time systems. One specific test tool, the environment simulator, is introduced and its use for testing real-time systems is motivated. A general survey on the state-of-the-art of real-time simulation follows. Finally, we describe the development and implementation of an environment simulator and its use for executing tests within the context of our distributed real-time system MARS.

2 On Real-Time Systems

We define a *real-time system* as a system that is required to react to stimuli from the environment (including the passage of physical time) within time intervals dictated by the environment.

Real-time systems can be classified according to a number of different viewpoints. Here it is sufficient to consider their *criticality*. Under this criterion we classify with reference to the costs (consequences) of a system failure. We distinguish between *benign* and *catastrophic* failures [Lap89]. The cost of a benign system failure is in the same order of magnitude as the benefits of the system during its

normal operation, i.e. the consequence is basically a loss of the benefits of system service. In contrast, if the consequences of a system failure can exceed the benefits of the system by several orders of magnitude, then the failure is catastrophic.

Thus, a *critical real-time system* is defined as a real-time system where a failure can lead to a catastrophe.

There are several areas of criticality, e.g. safety criticality, financial criticality, security criticality, environmental criticality, etc. An area of criticality which is of particular interest and importance is the time behaviour of the system. Therefore, a special term has been coined for systems which are critical in this respect:

If in a critical real-time system catastrophic consequences can arise because of failures in the time domain, the system is called a *hard real-time system.*

3 An Overview of MARS

MARS [Kop89] is a system designed to support hard real-time applications and to guarantee that all deadlines are met under all anticipated load and fault conditions. To achieve this, MARS is a *periodic* (*time-driven*) system where all activities are triggered by the passage of (real) time. External events (stimuli) are detected only at the end of predefined (finite-length) time intervals. This contrasts to *event-driven* systems where each external event is detected upon arrival (interrupt) and handled immediately.

A MARS system consists of a set of *clusters* that may be interconnected in an arbitrary topology depending on application requirements. What subset of the application tasks is assigned to a given cluster is informally determined by requiring that these tasks perform (highly) related functions; it is the designers' responsibility to make reasonable choices. Each cluster consists of a set of *components* interconnected by an Ethernet-type real-time local area network (*MARS bus*). A component is a self-contained computer that executes an identical copy of the MARS Operating System and a set of system and application *tasks.* Tasks are activated periodically by the operating system and may interact with each other solely by the exchange of standardised MARS messages. Fault tolerance is incorporated into MARS by duplicating components.

The so-called "Interface Components" are also connected to a second bus, the *Interface Bus.* This enables a cluster to communicate with its environment, which is either another MARS cluster or the controlled object (the physical process), via appropriate peripheral devices. Standardised MARS messages are used on the Interface Bus as well; a MARS task needs then not be aware of the nature of its communication partner.

In order to guarantee that all deadlines are met, several techniques are used: The system is designed for peak load, i.e. for the case of all stimuli occurring with their maximum frequency. The resource requirements for this case are determined, and resource allocation (CPU, bus access) is done according to these worst-case (or maximum-demand) situations. A CPU schedule for each component is calculated off-line [Koz87], taking into account any task precedence constraints and the maximum execution time of each task [Pus89]. Furthermore, access to the MARS bus must be deterministic; a Time-Division Multiple Access protocol is used in order to avoid queueing delays and message collisions. This guarantees that at any point in time at most one component of a cluster is granted send-access to the bus. In order to optimise response times, CPU and bus scheduling can be carried out concurrently. Finally, a prerequisite for the above is that all components have access to a common global time-base. Therefore, a synchronisation protocol has been implemented that uses special hardware to keep components' clocks synchronised [Kop87].

Besides the physical MARS system (i.e. the target system), there is also a MARS Design System [Sen89] that supports system (and software) development and evaluation. Its most important characteristic is that it allows the specification and refinement of time information throughout the life cycle, from requirements definition to task implementation. It is configured as a set of independent design, evaluation, and management tools that communicate by "contracts" exchanged via e-mail.

The following development phases are supported by the Design System: Basic system requirements are collected and described in the *Requirements Definition* phase. During *System Design and Cluster Definition* the system is decomposed into a set of MARS clusters and their interconnections. Each cluster is then refined into a set of components in the *Cluster Design and Component Definition* phase. The purpose of *Component Design and Task Definition* is to describe the functionality of each component by a collection of tasks. Lastly, all tasks are implemented in the *Task Implementation* phase using the programming language C.

Within the Design System, system functions are specified as a set of *transactions* each of which consists of a set of attributes. Attributes are the "stimulus" describing when or how a transaction is activated, the "response" describing the required reaction of the transaction, an informal description of the transaction's functionality, and various time attributes (maximum time between stimulus and response, maximum stimulus arrival rate, etc.). Transactions are used both to specify system behaviour and to describe the functions of subsystems (e.g. clusters). Thus, during design, transactions are iteratively refined into subtransactions until they become simple enough to be implemented as MARS tasks.

4 Testing Real-Time Systems

Since no design methodology (even if computer-aided) can prevent the commitment of errors during the design process, we have also addressed the question of how to evaluate the results of the design process and to gain confidence in the correct and timely performance of a system. *Testing* is one possibility to do that; it is applied to the final outcome of a design effort, the code. During testing a system is executed and provided with input values with the intention of finding and subsequently correcting errors [Lap89].

The main differences between real-time software testing and "traditional" (non-real-time) testing are:

1. Not only functional properties (the correctness) must be tested, but also timing properties (the time behaviour).
2. Real-time systems are usually developed on a *host system*, but eventually executed on a *target system*, sometimes a custom-designed piece of hardware, specifically tailored to efficiently execute the real-time software.
3. A real-time system interacts closely and is tightly coupled with its environment. In most applications, it is not possible to exercise explicit flow control over the environment of the real-time system. The mechanisms by which the system is able to influence its environment are limited. Thus the interdependence between system and environment is much greater than for non-real-time systems and it is more difficult to consider the behaviour of the environment (both in the value and the time domain) during testing.

In previous studies the state-of-the-art of testing hard real-time systems has been assessed, a test strategy for MARS has been developed, and techniques for testing real-time systems without altering or disturbing their time behaviour have been proposed [Sch90]. One major conclusion of these studies is that an *environment simulator* is a well-accepted and useful test tool, especially for target system tests [Gla80], and that our proposed test strategy should enable the use of such a tool.

An environment simulator is a device or a system that mimics the behaviour of the real-time system's environment, thus providing inputs to and accepting outputs from the system during test runs. These outputs are then transformed into a new set of inputs. There are several reasons for using a simulated environment for testing purposes:

- In most cases it is not permissible to use the real environment because of safety and/or cost considerations, especially since confidence in the correctness of the real-time system is still low.

- Simulation provides the only possibility to test system behaviour in "rare event situations" which occur only rarely in the real world. Often it is either difficult or unsafe to obtain these situations from the real environment. Frequently, system dependability is of utmost importance if such a "rare event" occurs (e.g. emergency handling of a chemical process).

- Generally, the tester has more control over the simulated environment than over the real one. Test execution is thus easier, and flexibility is enhanced. Evaluating system behaviour under invalid environmental behaviour (either in the value, or the time domain, or both) becomes feasible.

- It is often much easier to obtain system responses and other intermediate test results from a simulated environment than from the real one because the simulated environment is easier to instrument.

- The need to generate large amounts of input data is significantly reduced. Not all the test data have to be generated before test execution because new inputs are calculated on-line by the environment simulator, thus enabling longer and more extensive test runs.

- It might also be easier to run tests with *realistic* inputs and sequences of inputs if test data need not be generated in advance.

In order to achieve these goals the behaviour of the environment simulator must match that of the real environment as closely as possible, so that test runs become as realistic as possible. This holds not only for the input *values* delivered to the system under test, but also for the behaviour of the simulator in the *time* domain. The reaction of the environment has to be available within the same time interval as if the real environment was used. The software for simulating the environment is therefore real-time software as well and must be executed under real-time constraints by a real-time system.

5 Real-Time Simulation: A Survey

5.1 Modelling and Simulation

Simulation can be very generally defined as a method to investigate the properties or the behaviour of a real (physical) system by building a second system which is a *model* of the original system [Sch85]. Experimentation is then performed with the model, and results can be applied to the original system.

In the context of Computer Science, (computer) simulation is also more narrowly defined as "experiments run in a computer" [Dud85] or "experimentation with computer models" [Van83]. This is a special case of the general definition, implying that the model system is implemented as a computer program and results are obtained by executing this program.

From the above definitions it is apparent that modelling and simulation are closely related. A model is a representation or a mapping of one system by (to) another [Ame85]. Constructing a model that can be used to perform the desired experiments usually proceeds in two steps [Van83]:

First, the real (physical) system is analysed, relevant system properties are identified, and unimportant system properties are abstracted from. The decision which system properties are important and which are not is guided by the intentions of the modeller, i.e. the purpose of the model, or the problems one wants to investigate with the model. The result of this step is a conceptual or abstract model of the system which is formulated in some adequate and convenient notation, either formally (e.g. graphs, petri nets, queueing theory models, etc.) or informally.

The second step is to "implement" the abstract model in order to get a model with which the desired experiments or measurements can be performed. Again depending on the purpose of the model and on practicality, several types of models are possible: analytical (mathematical) models (e.g. a set of equations), physical models (e.g. flight simulators), or computer models. Manipulating an analytical model by analytical methods, either exact or approximated, is not considered simulation, instead "the model is solved". In all other cases, experimenting with the model is termed "simulation".

From now on, we will restrict our attention to computer models.

5.2 What does "Real-Time Simulation" mean?

The term "real-time simulation" appeared in the literature at least as early as 1966 [Sag66]. From the beginning, this term has been meant to denote a computer simulation where simulation time has to be kept synchronous to real (external) time or, more generally, to the system time of the original system being simulated [Cro82,Hau84]. This is always necessary when a simulation does not model a closed system, but when there is instead some interaction of the simulation with some external system, e.g. parts of the original system not being modelled.

This is also consistent with our view of model construction above and with our view of real-time systems: When the computer model is intended to be used as part of a larger system, it is not permissible to abstract model time from real or system

time; instead, these have to be synchronous. In traditional simulation, it is often the case that time is one of the first properties to be abstracted from. There may either not even be a notion of simulation time (static models), or simulation time may proceed at an arbitrary rate, faster than real time when the model is used for projections and forecasts, slower than real time when the underlying model is very complex (e.g. simulating elementary particle behaviour), or it may even proceed at different rates at different points in (simulation) time (e.g. variable step integration techniques, discrete event simulation).

While the meaning of "real-time simulation" has always been (at least informally) well understood, the main problem during the early years, however, was that computers were not powerful enough to allow the real-time execution of simulation programs of any realistic complexity. Thus the primary necessity was to reduce program run time. To this end, fast numeric integration techniques have been investigated (e.g. [Sag66]), analogue and (special-purpose) hybrid computers have been developed (e.g. for simulating electric power generation equipment [Bry76], car and jet engines [Led76], or flight simulators [Tom76]), and (digital) multi-processor systems which enable parallel processing were used to execute simulation software (e.g. general-purpose simulation systems [Fey83] or flight simulators [Kod85]).

Unfortunately, due to this emphasis on the need to speed up execution time, "real-time simulation" often seems to be interpreted that "execution needs to be as fast as possible in order to have a chance to keep up with real time". This is sometimes explicitly acknowledged [Tro83], but often the term "real-time simulation" is simply left undefined [Fey83,Cho84]. It also seems that the goal is sometimes just to speed up execution to such an extent that interactive simulation becomes possible – a gross misuse of the term "real-time". Note that "real-time computing is equivalent to fast computing" is one of the more common misconceptions about real-time computing that has been identified and refuted in a recent article [Sta88].

5.3 Applications of Real-Time Simulation

A first application area for real-time simulation are *test purposes*. This has been suggested before: Ledger describes a real-time simulation of a diesel engine which is used to facilitate testing of controllers designed for this engine [Led76]. Presumably, these controllers consist of a combination of hardware and software. Thus, this is an example where a physical system is tested by a (software) simulation of its environment. Hauser proposes a scheme very similar to the MARS scheme that uses a model of a process to test the control system for this process [Hau84]. This model must of course be simulated in real time. Here, a software system is tested by a software simulation of its environment.

However, none of these two examples describe how to achieve real-time execution of their models, or how to guarantee that all timing constraints are met. It is another objective of the MARS approach to provide an integrated, structured method to derive and specify the real-time requirements for the simulated environment (model construction) and to implement and execute the developed model such that these requirements are guaranteed to be met (model implementation and test execution). No such consistent methods or techniques have been described in the literature yet.

A second purpose for using real-time simulation is *user or operator training*. Flight simulators are probably the most well-known example for this type of systems; they are built exclusively for training purposes. In MARS we envision the approach that the same simulation software and the same (hardware) architecture as for testing can be used for training purposes.

Run-time error detection is yet another application of real-time simulation. Sturm proposes a scheme where a physical process (or parts thereof) is simulated by a software-implemented model and error detection is performed by a comparison of the simulation results with the outputs of the real process [Stu82]. Again, however, there is no information of how the real-time execution of the simulation software is enforced or guaranteed, if at all. In MARS, a similar approach seems possible, by simply using the environment simulator not only for testing, but also during the operational phase of the system.

Finally, it is often the case that knowledge of some internal or intermediate process variables is required for process control, especially for optimising process control. These process variables may or may not be measured readily. In the case that they are difficult to measure, software simulation of the process is often used to obtain the necessary values [Kel88]. This means that simulation is used as a substitute for measurement; required input parameters are computed instead of being measured. From the MARS standpoint, this type of simulation is part of the application software and must be designed and implemented in conjunction with application development. It is not discussed further in this paper, although software reuse might be possible.

6 Real-Time Simulation in MARS

In this section we describe the proposed simulator life cycle, consisting of:

- Deriving the requirements for the simulation software (modelling),
- Developing and implementing the simulator,
- Executing the simulator for conducting test runs.

The focus is on suggesting a way to integrate the simulation software life cycle into the existing MARS Design System, so that simulator construction can proceed in parallel with overall system development. The general approach is to reuse existing (design and management) tools as much as appropriate.

6.1 Deriving the Specification for the Simulation Software

Before one can actually begin to write the simulation software, one must somehow develop a model of the object to be simulated (the environment, in this case). While there is probably no single universally applicable way of deriving such a model, it is not unreasonable to demand that the *results* of this modelling process be specified in terms of the usual *transactions*. As mentioned above, the simulation software is also real-time software because it is intended to be used for test purposes; therefore, the same criteria as for application software development apply: One needs a Design System which supports the specification and refinement of time attributes during design, and which guarantees that all timing requirements will be met during execution. The MARS Design System meets these requirements and is, of course, readily available. Thus the MARS design tools can be used also for developing the necessary test support tools, in this case consisting of environment simulation software.

The specification of an environment simulator consists then of a set of transactions. There are two sources of information which must be considered during modelling: First, a description of the environment that must be simulated is needed (*environment specification*). The environment is either (a part of) the physical machine (the process) that is going to be controlled by the MARS application, or some other cluster of the MARS application, or a mixture of both. This depends on whether the object whose environment is simulated is a single MARS cluster or the complete MARS system. Second, a description of the (sub-)system that will be tested by the environment simulator is also needed (*application specification*). This second description will in many cases implicitly include a lot of information about its environment, e.g. about the assumed behaviour of its environment; therefore, these two sources of information are partly redundant.

One might suspect that an environment specification alone is sufficient to derive a simulation software specification. Such an approach would be more appealing since development of the environment simulator and of the application software could proceed independently from each other. Furthermore, the strict separation between the viewpoints of the application designer and the simulator designer introduces an element of "diversity" into the system development process and thus provides the potential for early removal of errors and inconsistencies by applying mutual checks. Using a similar methodology, the time required to install and field-test a new control system in an Italian automobile assembly plant was reduced from

projected 6 months to 1 week [Gra90].

Because the simulation software has to mimic the functionality of the real environment and also has to adhere to its timing properties, problems can arise when a function of the environment is implemented in hardware (e.g. in the physical machine or process), and thus is very fast. In some cases, the time intervals involved may be so short that the function is performed essentially "instantly" and that no possible software implementation is able to satisfy the associated time requirements.

Luckily, there is a way out: It is not really necessary to complete such a function within its "real" time, but one needs only to make sure that the function is completed before the application software requires its results. Thus it is possible to see at what period the application software needs the results of each such function and to derive the time requirements for the simulator from this information. This information is contained in the application specification, and that is the reason why both descriptions mentioned above are necessary. The application software will still not perceive any difference between its real and its simulated environment, despite this relaxation of time properties.

The drawbacks of this approach, however, are that the application and the simulation software become mutually dependent on each other, which means that changes in the application software (especially in its time requirements) may have to be reflected in the simulation software, and that errors in the application specification easily propagate into the simulator. An additional benefit exists, too: This approach indirectly "tests" (validates) the application specification. What we mean by this statement is the following: If (after simulator implementation) the behaviour of the simulated environment agrees with the behaviour of the real environment, confidence increases that the simulator specification — and thus the application specification, too — is correct.

Integration of simulator development into the MARS Design System is achieved by using contracts. As soon as an object which will be tested with the help of an environment simulation is specified, the project management directs a "simulator team" to design and implement the simulation software for this object and creates a suitable contract which contains all the pertinent information, as outlined above. Thus application software and simulation software development can proceed essentially in parallel, but management must assure that any changes in the requirements of the application software will be propagated to simulation development, perhaps by "update contracts".

The first step of simulator development is now to model the simulator, i.e. to derive a set of transactions specifying the simulator. The simplest approach is to derive the simulator's functionality by "inverting" the functionality of the tested application (sub-)system and to copy the latter's timing requirements (for

an example see [Sch90]). However, it is yet to be determined if this approach is general enough for all conceivable situations.

A tool that extracts the relevant information from the contract and generates a first version of the model specification is conceivable. This set of transactions probably has then to be fine-tuned and manipulated (manually), until finally arriving at a set of transactions specifying a MARS cluster (or a MARS system, for very complex simulations).

6.2 Developing the Simulation Software

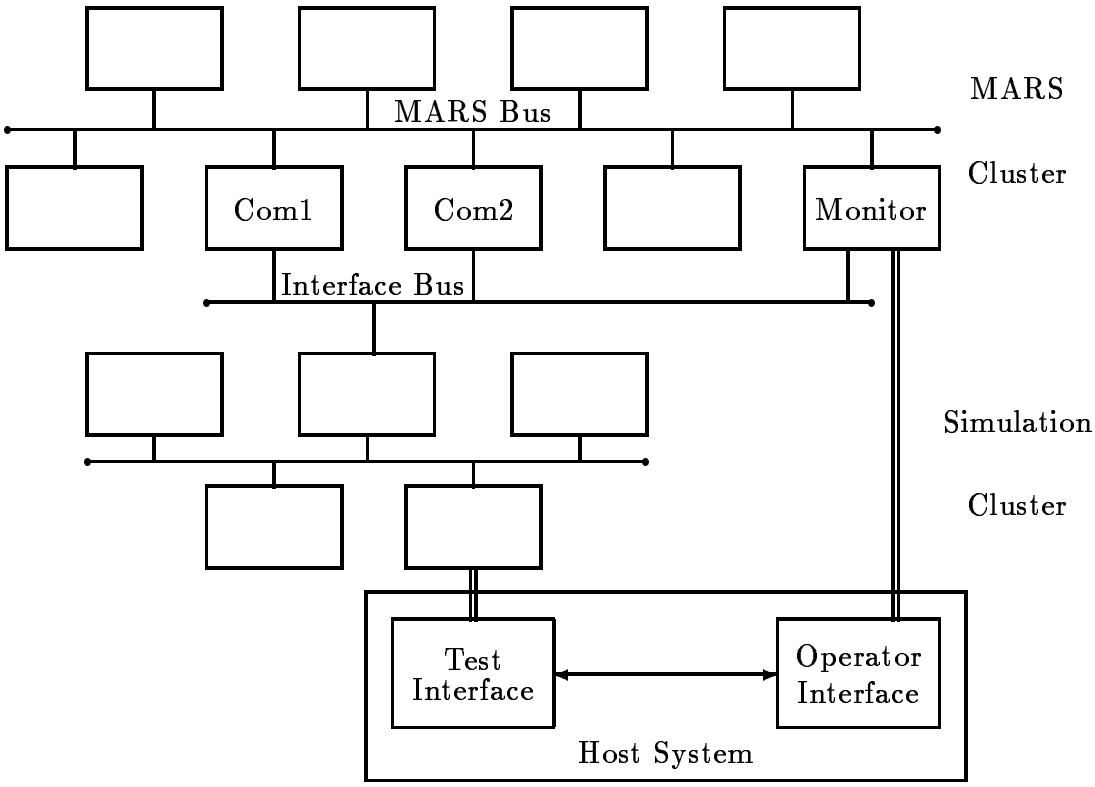
Once the appropriate set of transactions has been derived and serves as the requirements of the simulation software, simulator development is in the same state as any other application software development. Using the appropriate tools of the MARS Design System, the simulator is implemented as a MARS cluster. A suitable CPU and bus schedule is also calculated by the appropriate tools.

In principle, simulation software development can proceed in parallel to application software development. However, due to the (possible) interdependence between application requirements and simulator requirements described in the previous section, there is an increased likelyhood of changing simulator requirements during the first phase of application development. Therefore, it is a good idea to delay simulation software development until requirements become more stable, in order to avoid duplication of effort.

The problem of testing the simulation software itself remains to be addressed. Clearly, it is not practical to implement yet another simulator which simulates the environment of the first simulator; it would be identical to the application system itself. The suggested approach is to use those test phases of the MARS Test Strategy [Sch90] that do not make use of the environment simulation technique. If at all possible, these tests should be designed such that the behaviour of the simulated environment can be compared to that of the real one. Afterwards, the simulator is declared ready for its original purpose: aid in running tests. However, one has to keep in mind that any failure of a test run is due to *either* an error in the application cluster under test *or* an error in the simulation software. Genuine data obtained from the real environment might then be helpful for speeding up detection of the error location.

6.3 Executing the Simulation Software

Within the MARS Test Strategy a test bed architecture was proposed for conducting tests driven by an environment simulation. Figure 1 shows this proposed architecture.



Com1, Com2 ... MARS interface components
Monitor ... MARS component monitoring bus traffic

Figure 1: Test System for the Cluster Test with Environment Simulation

The environment simulator is executed on a separate "simulation cluster", and communicates with the application cluster via the Interface Bus. The application cluster remains totally unchanged (apart from adding the monitor component). The simulation cluster has its own MARS bus to exchange any messages needed by the simulation algorithm. Thus neither the application cluster nor the simulation cluster is aware of the other's messages, and neither cluster's functional or timing behaviour is influenced by the other's presence.

A *monitor component* is a passive component that does not send any messages. (Clearly, such a component does not alter the cluster's time behaviour.) It listens to all messages transmitted over the MARS bus and thereby monitors the responses of the cluster in reaction to the stimuli the simulation cluster provides. Via the monitor component, outputs can be transmitted to the host system and be stored for later evaluation.

The monitor component can not only hear the final (response) message of a

transaction but also all intermediate messages; therefore, the component (and presumably the task) where an error first occurred can be identified. As far as testing the time behaviour is concerned, every message is automatically time-stamped by the operating system; therefore it is possible to determine response times from the collected results.

Using this scheme, the behaviour of the cluster under test can be observed very closely.

7 Conclusion and Future Work

In this paper a general overview of real-time systems and real-time simulation was given, and we have explained how the notion of real-time simulation fits into our model of real-time systems. Specific attention was paid to the problem of applying real-time simulation for testing purposes. Within the context of our distributed real-time system MARS we have introduced general methods and techniques for constructing a real-time environment simulator and for executing it in a particular test bed, thus providing an opportunity for testing a real-time application without altering its timing behaviour. A method for integrating simulator development into the MARS Design System has also been considered.

Currently, the construction of an example real-time application [Kop90] is under way. The aim of this application is to balance a ball on a flat plane such that it follows a predefined course. With the help of a video camera the coordinates of the ball are input periodically, and the required inclination of the plane is computed and output to a set of servo motors. Within the scope of this experiment it is planned to use the techniques described in this paper for testing the software, in order to gain practical experience with and to evaluate the practical usefulness of the proposed techniques.

Acknowledgements
The author would like to thank the members of the MARS research group, in particular H. Kopetz, G. Fohler, G. Grünsteidl, H. Kantz, P. Puschner, J. Reisinger, and R. Zainlinger, for stimulating discussions and helpful comments on an earlier version of this paper.

References

[Ame85] W. Ameling. Methoden und Aspekte zur Planung und Analyse technischer Systeme. In D. Möller, Editor, *Proc. 3. Symposium Simulationstechnik*, pages 87–103, Bad Münster, BRD, 1985. Springer Verlag (IFB 109).

[Bry76] G. W. Bryce, T. R. Foord, D. J. Murray-Smith, and P. W. Agnew. Hybrid Simulation of Water-Turbine Governors. In R. Crosbie and J. Hay, Editors, *Toward Real-Time Simulation: Languages, Models, and Systems (Part 1)*, pages 35–44, La Jolla, CA, June 1976. Simulation Councils Proceedings Series, Vol. 6, No. 1.

[Cho84] C. K. Cho, E. K. Lin, and C. L. Jen. On Performance Evaluation of Multiprocessor Systems for Real-Time Simulation. In E. Comer, Editor, *Proc. 17th Annual Simulation Symposium*, pages 209–225, Tampa, Florida, 1984. IEEE Computer Society Press.

[Cro82] R. E. Crosbie. Interactive and Real-Time Simulation. In F. E. Cellier, Editor, *Progress in Modelling and Simulation*, pages 393–406. Academic Press, London, 1982.

[Dud85] E. J. Dudewicz and Z. A. Karian. Modern Design of Simulation Experiments. In E. J. Dudewicz and Z. A. Karian, Editors, *IEEE Tutorial: Modern Design and Analysis of Discrete-Event Computer Simulations*, pages 186–190. IEEE Computer Society Press, Washington, D.C., 1985.

[Fey83] S. Feyock and W. R. Collins. ADA and Multi-Microprocessor Real-Time Simulation. In L. Holbrook, Editor, *Proc. 16th Annual Simulation Symposium*, pages 211–228, Tampa, Florida, 1983. IEEE Computer Society Press.

[Gla80] R. L. Glass. Real-Time: The "Lost World" of Software Debugging and Testing. *Communications of the ACM*, 23(5):264–271, May 1980.

[Gra90] F. Grandoni. personal communication. 2nd European Workshop on Dependable Computing, Florence, Italy, Feb. 1990.

[Hau84] G. Hauser. Test von Steuerungssoftware mit Hilfe der Realzeitsimulation. In F. Breitenecker and W. Kleinert, Editors, *Proc. 2. Symposium Simulationstechnik*, pages 495–499, Wien, 1984. Springer Verlag (IFB 85).

[Kel88] H. B. Keller. Verteilte/modulare Echtzeitsimulation komplexer Systeme. In W. Ameling, Editor, *Proc. 5. Symposium Simulationstechnik*, pages 73–83, Aachen, BRD, 1988. Springer Verlag (IFB 179).

[Kod85] R. Kodweiß. Software Konzept für Echtzeit-Simulation. In D. Möller, Editor, *Proc. 3. Symposium Simulationstechnik*, pages 251–253, Bad Münster, BRD, 1985. Springer Verlag (IFB 109).

[Kop87] H. Kopetz and W. Ochsenreiter. Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, 36(8):933–940, Aug. 1987.

[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 and 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.

[Koz87] Ch. Koza. Scheduling of Hard Real-Time Tasks in the Fault-Tolerant Distributed Real-Time System MARS. In *Proc. 4th IEEE Workshop Real-Time Operating Systems*, pages 31–36, July 1987.

[Lap89] J. C. Laprie. Dependability: A Unifying Concept for Reliable Computing and Fault Tolerance. In T. Anderson, Editor, *Dependability of Resilient Computers*, pages 1–28. BSP Professional Books, Oxford, 1989.

[Led76] J. D. Ledger, A. J. Fisher, and G. C. Barney. Real-Time Hybrid Simulation of a Turbocharged Diesel Engine. In R. Crosbie and J. Hay, Editors, *Toward Real-Time Simulation: Languages, Models, and Systems (Part 1)*, pages 45–50, La Jolla, CA, June 1976. Simulation Councils Proceedings Series, Vol. 6, No. 1.

[Pus89] P. Puschner and Ch. Koza. Calculating the Maximum Execution Time of Real-Time Programs. *Real-Time Systems*, 1(2):159–176, Sep. 1989.

[Sag66] A. P. Sage and S. L. Smith. Real-Time Digital Simulation for Systems Control. *Proceedings of the IEEE*, 54(12):1802–1812, Dec. 1966.

[Sch85] B. Schmidt. "Was tut man, wenn man simuliert?" Versuch einer Begriffsbestimmung. In D. Möller, Editor, *Proc. 3. Symposium Simulationstechnik*, pages 104–111, Bad Münster, BRD, 1985. Springer Verlag (IFB 109).

[Sch90] W. Schütz. A Test Strategy for the Distributed Real-Time System MARS. In *IEEE Compeuro 90, Computer Systems and Software Engineering*, Tel Aviv, Israel, May 1990. Also available as Research Report 1/90.

[Sen89] Ch. Senft and R. Zainlinger. A Graphical Design Environment for Distributed Real-Time Systems. In *Proc. 22nd Annual Hawaii International Conference on System Sciences, Vol. II*, pages 871–880, Hawaii, Jan. 1989.

[Sta88] J. A. Stankovic. Misconceptions about Real-Time Computing. *IEEE Computer*, 21(10):10–19, Oct. 1988.

[Stu82] K. H. Sturm. Simulation von Systemkomponenten in Realzeitsystemen. In M. Goller, Editor, *Proc. 1. Symposium Simulationstechnik*, pages 494–504, Erlangen, BRD, 1982. Springer Verlag (IFB 56).

[Tom76] B. N. Tomlinson. Hybrid Computing for Real-Time Flight Simulation. In R. Crosbie and J. Hay, Editors, *Toward Real-Time Simulation: Languages, Models, and Systems (Part 2)*, pages 105–114, La Jolla, CA, Dec. 1976. Simulation Councils Proceedings Series, Vol. 6, No. 2.

[Tro83] K. C. Trott and F. K. Frantz. A Detailed Interactive Simulation System for Developing Command and Control Systems. In L. Holbrook, Editor, *Proc. 16th Annual Simulation Symposium*, pages 11–31, Tampa, Florida, 1983. IEEE Computer Society Press.

[Van83] G. C. Vansteenkiste. Simulation Methodology for Improved Process Interaction. In W. Ameling, Editor, *Proc. First European Simulation Congress ESC 83*, pages 32–43, Aachen, BRD, 1983. Springer Verlag (IFB 71).