Scheduler 1-2-3: An Interactive Schedulability Analyzer for Real-Time Systems

Hideyuki Tokuda and Makoto Kotera

June 15, 1988

CMU-CS-88-178

Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Advances in Software Engineering provided us a set of modern programming tools for building large, complex software systems. Various toolsets can cover the wide range of the software development life cycle from the requirement analysis phase to the debugging and the maintenance phases. However, many of these modern tools are not quite effective for building and analyzing complex real-time systems. An additional toolset such as a timing tool, a schedulability analyzer, and a real-time monitor/debugger should be developed to reduce the complexity of real-time software. In this paper, we describe an interactive analysis tool, called *Scheduler1-2-3*, which can perform the schedulability analysis for development of real-time computing systems. The schedulability analysis can verify whether all hard real-time tasks in a target system will complete by their deadlines or not at the system design phase. *Scheduler1-2-3* is a window-based interactive standalone tool, and it can also be used as a synthetic workload generator as a part of an integrated toolset that consists of a timing tool, a schedulability analyzer, and a real-time monitor/debugger.

This research was supported in part by the U.S. Naval Ocean Systems Center under contract number N66001-87-C-0155, by the Office of Naval Research under contract number N00014-84-K-0734, and by the Federal Systems Division of IBM Corporation under University Agreement YA-278067. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of NOSC, ONR, IBM, or the U.S. Government.

Table of Contents

List of Figures

1. Introduction

Advances in Software Engineering provided us with a set of modern programming tools for building large, complex software systems. Various tool sets can cover the wide range of the software development life cycle from a requirement analysis phase to the debugging and the maintenance phases [1, 5]. However, we cannot simply reuse the existing tool set for designing and building complex real-time computing systems without considering the time management capabilities. For instance, an interactive debugging tool in Smalltalk-80 [6] allows us to track down a "logical" bug in a program very well. However, it is almost impossible to detect or fix a timing bug (error) for real-time programs.

What we are missing in real-time systems engineering is good real-time programming methodologies, real-time specification methods, high-level real-time languages, and a real-time toolset which allows us to manage real-time programs in a reliable and predictable fashion. For instance, Ada[1] [22] is a good high-level language for managing a large, complex programs, however, there are many practical problems in its tasking semantics for embedded hard real-time applications [4]. Classical linear time temporal logic is also rather insufficient for expressing real-time systems safety or liveness properties. It cannot specify its temporal property within a certain time bound. An extended model of the temporal logic such as a Real-Time Temporal Logic (RTL) [16] is necessary to deal with the real-time programs. Similarly, many existing software tools cannot provide support for detecting and eliminating a timing error in the target program.

Recently, several attempts have been made to provide better real-time software tools for developing complex real-time systems. At Carnegie-Mellon University, development of a real-time kernel, called ARTS, is currently underway. The ARTS kernel provides a distributed real-time computing environment based on an object-oriented computation model. This kernel cooperates with an integrated toolset which consists of a timing tool, a schedulability analyzer and a real-time monitor/debugger. In this paper, we will focus on our interactive schedulability analyzer, *Scheduler1-2-3* [2]. First, we present the overview of the real-time computational model we use. Then, we present the objectives of *Scheduler 1-2-3* and its internal structure. We also describe both techniques adopted for the schedulability analysis and its interactive operations, and we compare our approach with other related approaches. We then summarize our works and mention future extensions. First of all, let us review software tools which focus on the prediction, detection, and elimination of the timing errors in real-time systems, before stepping into the detail discussion of *Scheduler1-2-3*.

1.1. Timing tools

Unfortunately, most high-level programming languages do not treat "time" as a first class object, then any timing constraint on a real-time activity must be imposed implicitly. This type of "implicit binding" between a code segment and time causes many common timing errors. One way of avoiding the error is to use "time" in the real-time program explicitly. Then, a timing tool which can estimate or measure the average, the worst, and the best case of the timing characteristics of a given program module would be very handy. The timing tool should be available at the design phase as well as the debugging phase.

---

[1] Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

[2] Of course, the name of *Scheduler1-2-3* came from *Lotus1-2-3* which is a registered trade mark of Lotus Development Corporation.

An example of such timing tools has been developed for the SARTOR Environment [15], and it can estimate the timing information (i.e., cpu execution time) for a given code segment using a microprocessor simulator. There is also a real-time programming language, called Real-Time Euclid [7] which was designed with a set of schedulability analysis provisions built-in for the sake of schedulability analysis.

1.2. Schedulability Analysis Tools

A schedulability analysis tool can verify whether a given real-time task set can meet their timing constraints or not. The benefits of this tool is that it is possible to foresee the schedulability of the target system in prior to the implementation phase. In general, the predictability of the analyzer heavily depends on the scheduling policies the target system use. It is desirable that the analyzer can be applied to various scheduling policies. Moreover, the analyzer should be able to take into account various types of practical system overheads.

There are well-known schedulability analysis algorithms [10, 11, 14], however, practical interactive analysis tools have not been demonstrated yet.

1.3. Real-Time Monitoring/Debugging Tools

By virtue of systematic programming paradigms like data encapsulation, modern debuggers fully support to detect and eliminate logical errors in non-real-time programs; on the other hand, testing and verifying timing correctness has been done only in *ad hoc* manner. It is very hard to track down the timing bugs, since the timing bug sometimes affect the system beyond the task boundaries. This makes it really difficult to find out the origins of the timing errors.

Time encapsulation, the notion to capture timing error at the point where the problem originates, is needed [21]. A powerful real-time monitoring/debugging tool should be able to capture a timing bug in cooperation with the previously mentioned tools.

Furthermore, a monitor/debugger should be able to show the runtime system behaviors and is hopefully noninvasive. Like non real-time debugging tool, a real-time debugger may also require ordinary debugging features such as traces, breackpoints, and stack examiners. However, the difficult problem in real-time debugging is that the system overhead due to the debugger might cause a new timing error in the integrated testing stage. For instance, debugging statements seen in non-real-time programs are likely to affect timing correctness, even though logical correctness can be verified with such features. In this sense, the real-time monitor/debugger should minimize the system overhead.

A real-time monitoring tool has been built in the DCT (Distributed Computer Testbed) system [2]. In DCT, a large amount of hardware is used for system instrumentation and it could achieve a high degree of noninvasive monitoring. A global synchronized clock provided the system with the system-wide monitoring activities. The ART Real-Time Monitor [21] employs a software approach to the runtime monitoring. Many probes are embedded in a real-time kernel to capture interesting events, and collected events are reported to a remote monitoring subsystem. A Distributed Program Monitor by Miller [13] is also a software approach for monitoring system behavior in a distributed environment. Software oriented approaches cannot be completely noninvasive; nonetheless, maximum interference caused by monitoring activities should be predictable and minimized.

2. Real-Time Computational Model

The biggest difference between real-time systems and non real-time systems is time-critical nature. Real-time systems must respond on external activities. Response must be done in time. Among various types of real-time systems, we first classify the real-time computing tasks as *hard real-time* or *soft real-time*. By the hard real-time task we mean that the task must complete its activities by its "hard" deadline time, otherwise it will cause undesirable damage or a fatal error to the system. The soft real-time task, on the other hand, does not have such "hard" deadline and it still make sense to the system to complete the task even if it passed its "critical" (i.e. soft deadline) time. Suppose we use a "value function" [12, 20] which represents the task's contribution (i.e., semantic value) to the system as a function of time, we can illustrate the difference. The hard real-time task indicates a step function where the discontinuity occurs at its deadline while the soft real-time has a continuous (linear or non-linear) decreasing function after its critical time.

In both types, a task can be either *periodic* or *aperiodic*. A periodic task P_i is defined by the total computation time C_i, period T_i, while an aperiodic task AP_i is defined by the total computation time C_i, its distribution D_i, mean arrival time M_i, and standard deviation S_i. In particular, when a hard real-time task is an aperiodic, we call it a *sporadic* task where consecutive requests of the task initiation are kept at least Q unit of time apart [14]. It should be noted that it is not our intention to limit the task execution time as a constant. We can always incorporate a stochastic execution model, once we have a better analytical result. The classification of the hard and soft real-time tasks is depicted in Figure 2-1.

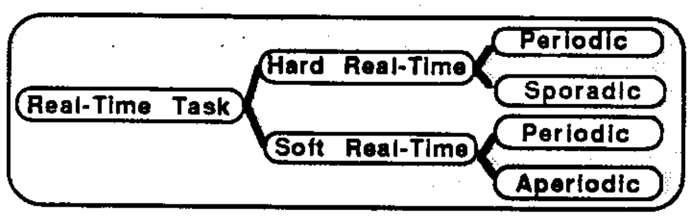

Figure 2-1: The Real-Time Computational Model

In general, real-time systems do not consist of a specific type of real-time tasks. It often consists of a mixture of hard and soft, and periodic or aperiodic (sporadic) activities. For instance, in the Inertial Navigation System (INS) experiment [3] , there are seven hard and periodic tasks: Attitude Updater (0.9/2.5), Velocity Updater (4/40), Attitude Sender (10/62.5), Navigation Sender (20/1000), Status Display (100/1000), Runtime BIT (5/1000), Position Updater (25/1250) where a pair of numbers indicates its execution time over the period. There are also a few soft and aperiodic tasks such as a Keyboard Command Processor, Console Keyboard Interrupt Service Routine, and Console Screen Interrupt Service Routine. In some of embedded real-time systems, the majority of the task are hard and periodic real-time tasks, while some of large, distributed real-time systems may have more soft and aperiodic activities due to the use of distributed database systems or expert systems.

3. The Schedulability Analyzer

3.1. Objectives

The main objective is to provide an interactive design tool that makes it easy to design a complex real-time system. For the existing systems, the *Scheduler1-2-3* can be used to predict the timing effects due to software and hardware modification. Another objective is to utilize *Scheduler1-2-3* as a synthetic workload generator, which can be integrated with the other test tools, the timing tool and the real-time monitor/debugger. The objectives are summarized as following.

- **Schedulability Analysis**

  The schedulability is verified for the given hard deadline task sets under the given scheduling algorithm. Currently, we are interested in analysis on the the rate monotonic [11], the first-in first-out (FIFO), the closest deadline first and the round robin scheduling. If a given task set is not schedulable, the following information is reported, namely, when a deadline will be missed first time and which task misses its deadline. Furthermore, *Scheduler1-2-3* gives some intelligent suggestions based on the period transformation method [18].

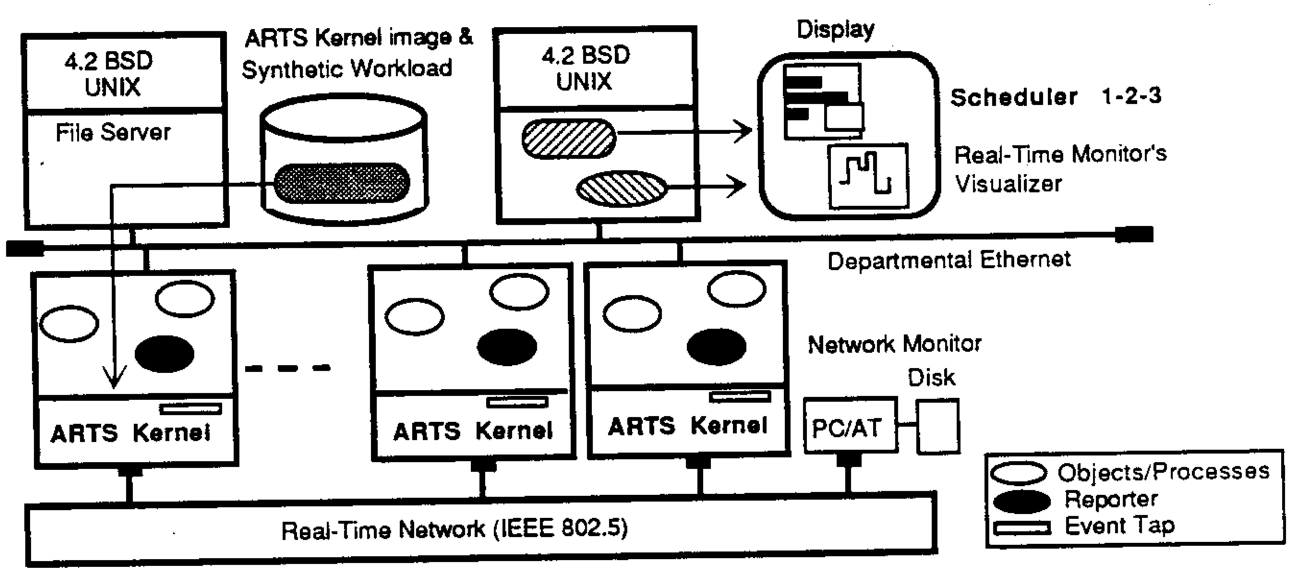

Figure 3-1: System Configuration of the ART Real-Time Testbed

- **Response Time Analysis for Aperiodic Tasks**

  Performance of the given sets of aperiodic tasks with soft deadlines is given under arbitrary scheduling algorithms for aperiodic tasks. By means of a simulation, *Scheduler1-2-3* estimates the average response times of those aperiodic tasks based on the given statistical parameters, like mean arrival time, standard deviation and distribution.

- **Monitorability Analysis**

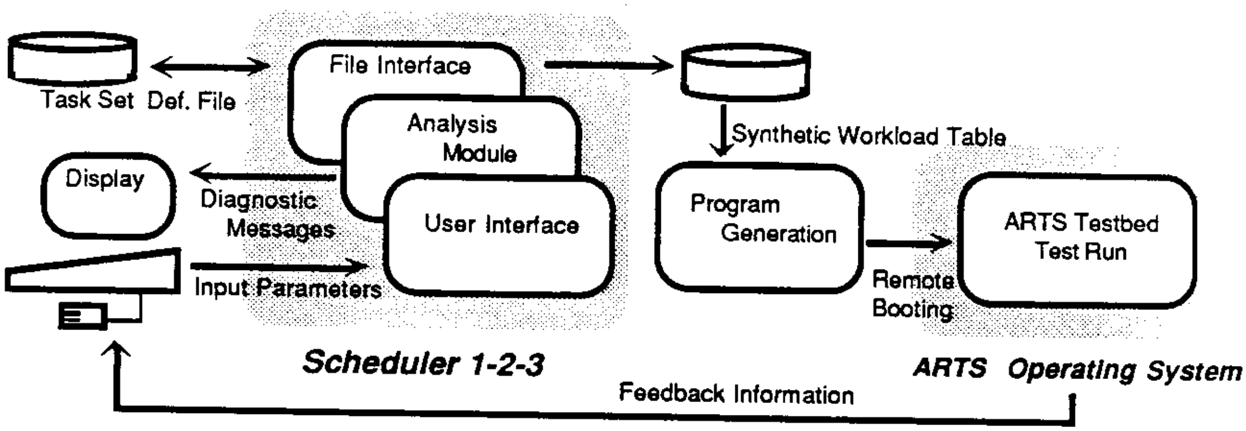

Figure 3-2: System Flow in the ART Real-Time Testbed

The interference of the monitoring and debugging activities in runtime should be not only minimized but also predicted because real-time systems are time-critical. Our runtime monitoring approach [21] adds a monotoring task in advance to the target task set so that monitoring overhead can be analyzed by *Scheduler1-2-3* before hand. In addition, *Scheduler 1-2-3* predicts the maximum capacity of the monitoring task under a given scheduling policy so as not to have the monitoring capacity overwhelmed by too many event occurrence. For the sake of simplicity, we call this analysis *monitorability analysis* in remider of this paper.

- **Easy-to-use Interface**

An interactive user interface is provided on a window system of bitmap display for ease of use. This interface makes it possible for users to enjoy schedulability analysis without precise knowledge of internal analysis procedures.

- **Synthetic Workload Generator**

To confirm the schedulability of the given task set on a practical environment, *Scheduler1-2-3* outputs a workload table on which a synthetic real-time task set can be generated. Synthetic tasks with workload specified would be tested on the ART Real-Time Testbed as illustrated in Figure 3-1.

3.2. Structure

Scheduler1-2-3 consists of three major modules. They are the Analysis module, the File interface module and the User interface module, as depicted in Figure 3-2. It is easy to port *Scheduler1-2-3* to other hosts or to apply the analysis module to another purpose. Those major modules are structured independently of each other.

- **Analysis Module**

The analysis module performs schedulability analysis, response time analysis for aperiodic tasks and monitorability analysis. The rate monotonic algorithm [11] is currently available for schedulability analysis of hard deadline tasks. The scheduling algorithm for aperiodic tasks is currently based on the deferrable server algorithm [9]. Analysis steps are explained in the following section.

- **File Interface Module**

  Generating a synthetic workload table and reading/writing a task definition file are done by this module. As depicted in Figure 3-2, a test program is created from a synthetic workload table to be executed on the ART Real-Time Testbed. A task set definition table is a collection of binary data of the parameters of the given task set. This table is used to save the time to re-input the task set definition.

- **User Interface Module**

  This module provides an easy-to-use interactive user interface based on a window system. The user interface is implemented independently of the analysis part. The current implementation has been done on the window system on Sun3 workstation[3]. It is possible both to use *Scheduler1-2-3* on windlowless hosts and to port it on other window systems [4]owing to the independent structure of major modules.

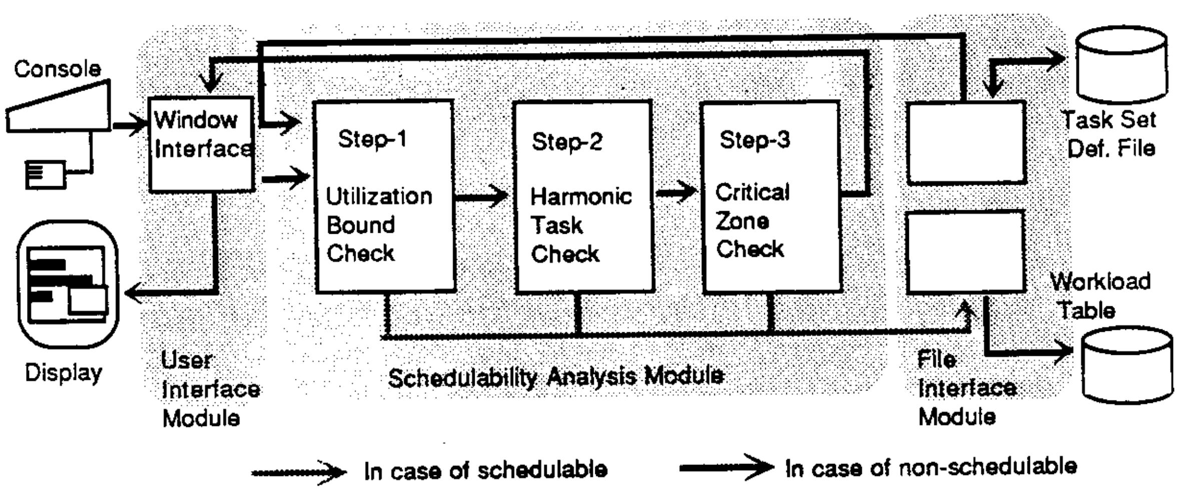

Figure 3-3: Analysis Steps in *Scheduler1-2-3*

[3]Sun3 is the trademark of Sun Microsystems, Inc.

[4]Porting it onto X.11-window is now underway.

3.3. Task Analysis

We will describe the task analysis techniques based on the rate monotonic scheduling algorithm in this section. Although schedulability of real-time tasks is very difficult to predict, the rate monotonic scheduling makes it possible to analyze the schedulability with the closed formula described in this section. Even if the activities of the task set include interprocess communication or mutual exclusion among the task set, the closed form analysis is still possible by using the priority inheritance algorithm [18]. Simulation part will be added to deal with the other scheduling policies.

3.3.1. Schedulability Analysis

The rate monotonic scheduling gives higher priority in execution to more frequently executed periodic tasks. The schedulability analysis is carried out along with the following three steps as shown in Figure 3-3. Suppose there are n periodic tasks whose periods and execution times are represented as T_i and C_i respectively, where $1 \leq i \leq n$ and $T_1 \leq T_2 \leq \cdots \leq T_n$.

Step1 - Utilization Bound Check:

Under the rate monotonic scheduling, if processor (CPU) utilization is less than $n \cdot (2^{1/n} - 1)$ with n periodic tasks in the given task set, which is approximately equal to $ln\ 2 \approx 0.693$[5], it has been proved to be schedulable [11]. Therefore, a task set with CPU utilization which is not higher than this limitation is reported to be schedulable, and the subsequent steps are omitted.

Step2 - Harmonic Task Check:

Further research on the rate monotonic algorithm leads to a more sophisticated verification technique, called the Task-Lumping Technique [17]. This technique is used to predict the schedulability if CPU utilization by a given task set exceeds $ln\ 2$. This Lumping technique tries to lump tasks together so that schedulability test becomes easier and simpler. The Lumping technique is applied iteratively from the most frequency task up to the least frequency task. If all tasks are successfully lumped together, the given task set is proved to be schedulable. One lumping is described as below.

1. The first step tried to lump task$_i$ and task$_{i+1}$, where $1 \leq i \leq n-1$. If T_{i+1} is a multiple of T_i, those two tasks are called harmonic and lumped together into one task, τ_i whose period is T_{i+1}.
2. CPU utilization of task τ_i is verified, if task$_i$ and task$_{i+1}$ are harmonic. CPU utilization of τ_i, $C_i / T_i + C_{i+1} / T_{i+1}$, is below 100 percent, schedulability is guaranteed up to task$_{i+1}$.
3. If those two tasks are not harmonic, namely T_{i+1} is not a multiple of T_i, schedulability has to be checked explicitly. In period of T_{i+1}, if both tasks get allocated CPU time that are longer than or equal to their execution times, those tasks turned out to be schedulable and those two are lumped into τ_i whose period is set to the least common multiple of T_i and T_{i+1}.
4. If task$_i$ and task$_{i+1}$ are schedulable, τ_i would be regarded as task$_i$ in the next lumping.

Step3 - Critical Zone Check:

Because the Lumping technique is slightly pessimistic, all schedulable task set cannot be found by the lumping technique. In this case, the decision is made by the third stage test, which is a kind of numerical

[5]In average case, the rate monotonic algorithm can schedule task sets with 0.88 or more CPU utilization [8].

simulation of the critical zone as illustrated below.

1. Schedulability of task$_1$, the highest frequent periodic task, is verified by confirming its execution time is less than or equal to its period ($C_1 \leq T_1$).

2. For every periodic task$_i$($2 \leq i \leq n$), check points are set by calculating arrival points of all periodic tasks that are higher frequent than task$_i$

   Sets of check points are represented as $S_i = \{k \cdot T_l\}$ for $1 \leq k \leq \lfloor T_i / T_l \rfloor$ and $1 \leq l \leq i$.

3. At every check point, CPU utilization is calculated and checked whether utilization is less than or equal to 100 percent. If CPU utilization exceeds 100 percent at any check point, it is reported that task$_i$'s $\lfloor T_i / T_l \rfloor + 1^{st}$ arrival causes missed deadline, and the simulation failed; otherwise, there is at least 1 way to schedule i tasks so that all of them will meet their deadlines.

4. If this simulation finishes successfully for all n periodic tasks, the task set is schedulable.

An amendment - Context switching overhead:

Overhead caused by the context switching must be taken into account, because task creation and elimination consume considerable processor time, and this overhead is not avoidable when periodic tasks which repeat creation and elimination are handled. *Scheduler1-2-3* can take context switching overhead into its schedulability analysis by adding those overhead twice into the execution time of every periodic tasks before an analysis begins.

3.3.2. Integrated Schedulability Analysis with Aperiodic Tasks

It is important to minimize the response time of aperiodic soft deadline tasks as well, because real-time systems are often mixture of hard real-time tasks and soft real-time tasks. The deferrable server algorithm [9] is a newly proposed algorithm which performs better than other algorithm for handling aperiodic tasks, (i.e. the back ground or the polling) under the rate monotonic scheduling. Suppose a set of n periodic tasks, this algorithm adds an extra periodic task, called the deferrable server, with C_s execution time and T_s period with priority defined by T_s in the rate monotonic scheduling. If T_s is the shortest, the deferrable server has the highest priority. The deferrable server uses preserved CPU time C_s to service to aperiodic tasks which arrive by the end of period T_s. Because the highest priority is given to the deferrable server and the maximum cpu time is reserved for the execution time for the server, namely, aperiodic tasks, this algorithm improves response time of aperiodic tasks in hard real-time environment in comparison with the back ground or the polling algorithms.

Scheduler1-2-3 sets the deferrable server's period to the highest period in the task set, when an analysis includes aperiodic tasks. Then maximum slack time that may be allocated to the deferrable server is found by means of a binary search. Nonetheless, it is possible to set the period arbitrarily by hand and to see the schedulability.

3.3.3. Monitorability analysis

The analysis that verifies whether or not the runtime system behavior can be monitored is done by the worst case analysis. First, if the monitoring task (Reporter) is not included in the task set, it reports that the task set cannot be monitored. The Reporter task is in charge of sending event information over the network to the "Visualizer", where event information is analyzed and visualized. If there is the Reporter, the maximum number of events that can occur in one Reporter's period is estimated. If the number of events taking place in one Reporter's period overwhelms the Reporter's capability of sending event information over the network, event information will be lost eventually. In that case, this task set is not "monitorable". The precise analysis steps are described in [21].

3.4. User Interface

Scheduler1-2-3 is characterized by its interactive user interface based on a window system of bitmap display. As illustrated in Figure 3-4, the window of *Scheduler1-2-3* is conceptually divided into 3 subwindows. To begin with, a user gives a set of parameters to *Scheduler1-2-3* through *the editing window*, and all input parameters is displayed on *the display window* where the task set under analysis are displayed. Operations are triggered by clicking the buttons on *the operation window* with a mouse. For instance, clicking the **Cyclic** button tells *Scheduler1-2-3* to start schedulability analysis for periodic tasks. The results would be printed on *the message window*, which appears and disappears as necessary. The schedulability analysis, handling context overhead and integrated analysis including aperiodic tasks, are operated in the following interactive manner.

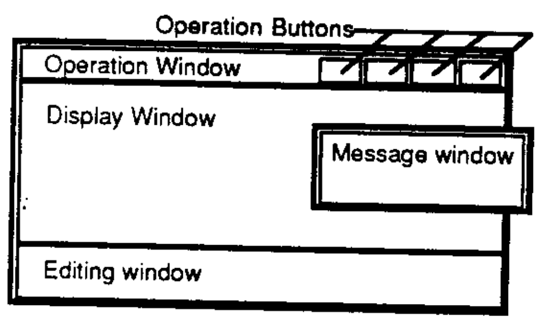

Figure 3-4: The Window Layout of *Scheduler1-2-3*

3.4.1. Schedulability analysis

An analysis is performed for the INS task set [3], (see Section 2), in Figure 3-5. All necessary timing parameters were already set through *the editing window*. By clicking the **Cyclic** button on *the operation window* the analysis is started, and then as displayed in *the message window*, the result of analysis shows this task set is schedulable with 0.86 CPU utilization.

3.4.2. Context switching overhead

Context overhead was taken through *the editing window*. The effect context overhead has on schedulability is typically illustrated in Figure 3-6, which first shows the task set that used to be schedulable in Figure 3-5 becomes non-schedulable when context overhead is added. Then, as a result of the analysis, *Scheduler1-2-3* reports that the *Velocity Updater* task will miss its deadline at its 4th arrival.

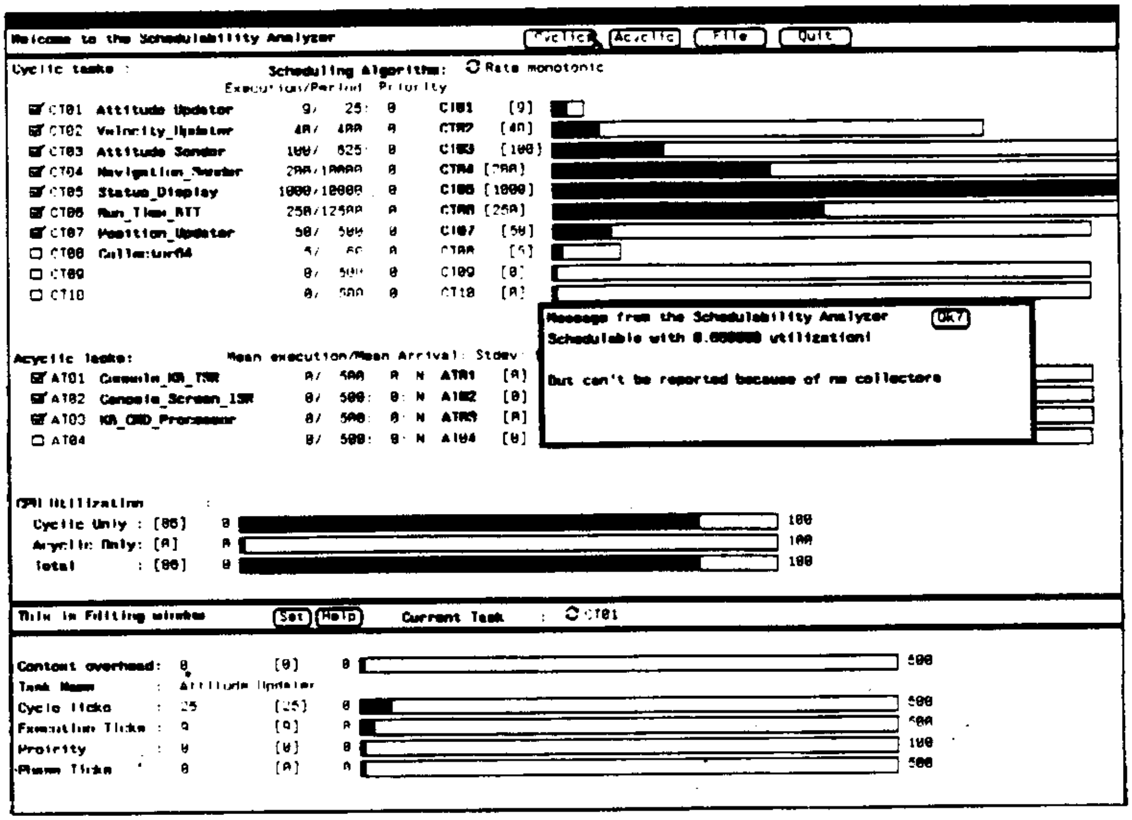

Figure 3-5: The INS Task Set

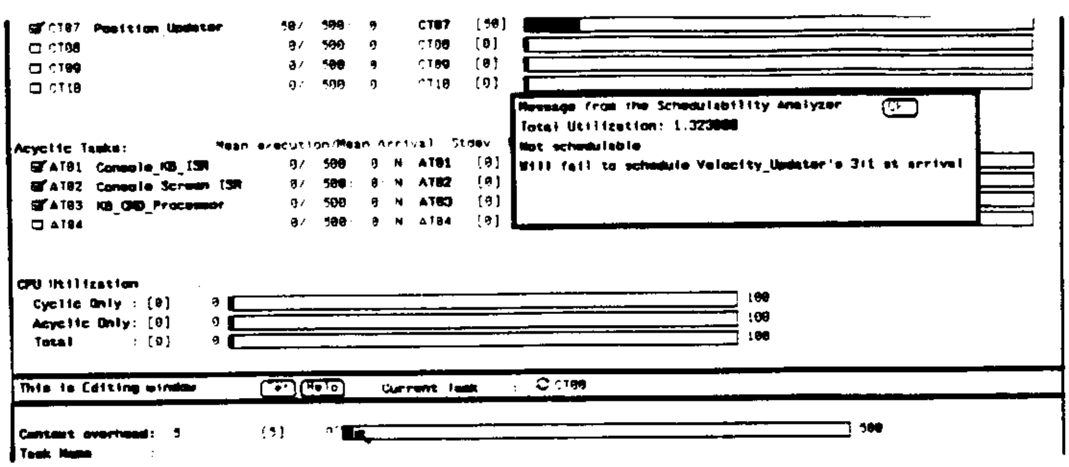

Figure 3-6: The Effect of Context Switching Overhead

3.4.3. Integrated analysis

Creation of the deferrable server and the result of analysis with the deferrable server task is depicted by Figure 3-7 in case of the INS task set. Clicking the **Acyclic** button creates the deferrable server task automatically and goes into the analysis steps. This example indicates that the schedulability is maintained with 0.96 CPU utilization.

3.4.4. Monitorability analysis

Monitorability is analyzed simultaneously along with other analyses as shown in Figure 3-8, where the INS task set can be monitored with maximum number of events coming up in a period of the monitoring task being about 51.6.

3.5. Synthetic Workload Generator

This facility makes *Scheduler1-2-3* an efficient integrated tool, which can work in the practical world. For the purpose of testing on the ART Real-Time Testbed, *Scheduler1-2-3* generates a workload table after the schedulability analysis. The workload table would be included into a testing program that generates synthetic workload, then the schedulability is practically tested on the ART Real-Time Testbed. A workload table is shown in Figure 3-9, which would be an include file of a C program.

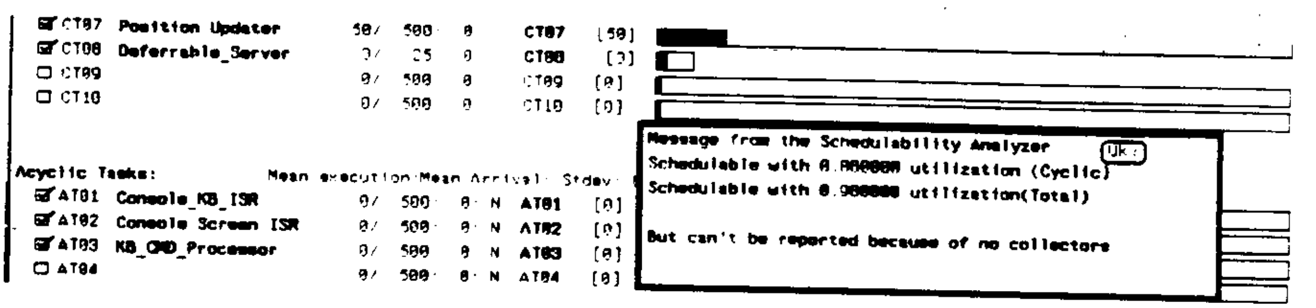

Figure 3-7: An Analysis with a Deferrable Server

4. Related Work

We introduced our approach to schedulability analysis based on an interactive analysis tool. Our approach can be classified as the "tool oriented" approach introducing an integrated toolset. Similar approaches has scarcely attempted. On the other hand, some theoretical approaches and language oriented approaches has been proposed. An strong point of the theoretical approach is its generality. Theoretical approach is not limited to a specific scheduling algorithm, the host machine's architecture or the language used to describe the target system. Meanwhile fewer constraints sometimes leads to a pessimistic or oversimplified analysis. The language oriented approach uses real-time programming languages. This approach takes advantage of precise analysis based on the language specification. By a real-time programming language, we mean a high-level language which explicitly binds the timing

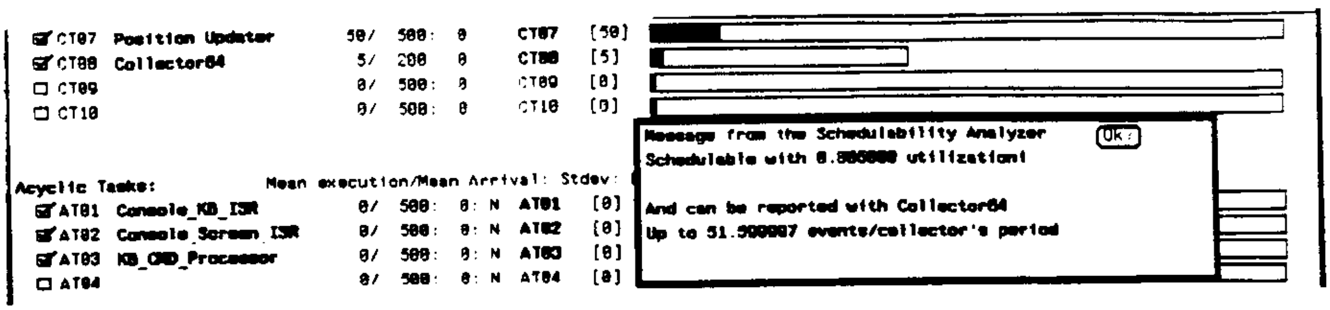

Figure 3-8: Monitorability Analysis

```
typedef         struct{
      char      t_avail;
      char      t_name[MAXTASKNAME];
      char      t_type;
      unsigned int      t_period;
      unsigned int      t_exec;
      unsigned int      t_prio;
      unsigned int      t_phase;
} XL_SCHED_ELM;
```

Data structure

```
'\001', "Attitude_Updater", '\000', 25, 9, 0, 0,
'\001', "Velocity_Updater", '\000', 400, 40, 0, 0,
'\001', "Attitude_Sender", '\000', 625, 100, 0, 0,
'\001', "Navigation_Sender", '\000', 10000, 200, 0, 0,
'\001', "Status_Display", '\000', 10000, 1000, 0, 0,
'\001', "Run_Time_BIT", '\000', 12500, 250, 0, 0,
'\001', "Position_Updater", '\000', 500, 50, 0, 0,
'\000', "NA", '\000', 500, 0, 0, 0,
'\000', "NA", '\000', 500, 0, 0, 0,
'\000', "NA", '\000', 500, 0, 0, 0,
'\001', "Console_KB_ISR", '\001', 500, 20, 0, 0,
'\001', "Console_Screen_ISR", '\001', 500, 54, 0, 0,
'\001', "KB_CMD_Processor", '\001', 500, 0, 0, 0,
'\000', "NA", '\001', 500, 0, 0, 0,
```

File format

Figure 3-9: The Workload Table for the INS Task Set

constraints with the code segment. However, a big drawback is that it is very hard to analyze systems written in other languages.

In this section, we review two theoretical approaches: Leinbaugh's well known formula [10] and Real-Time Temporal Logic. A real-time programming language (Real-Time Euclid) oriented is also discussed.

4.1. Leinbaugh's Algorithm

A formula well known for analyzing the schedulability of periodic tasks with hard deadline has been given by Leinbaugh. The schedulability analysis technique introduced in this approach consists of the following items.

$$rate_i = TOTN_i / (GT_i - TOTR_i - TOTD_i - B_i)$$

- $rate_i$: The ratio of processor time allocated to perform non-critical sections of $task_i$. The following constraint must be kept. $\sum_{i=1}^{n} rate_i \leq 1$
- $TOTN_i$: Total processor time for performing non-critical sections, which implies $task_i$ can be interrupted and taken over by $task_j$ that begins non-interruptable execution.
- GT_i: The Guaranteed response time of $task_i$, representing the period of $task_i$.
- $TOTR_i$: Total execution time for critical sections included $task_i$'s execution. Critical sections are non-preemptable. $TOTR_i$ may be called non-interruptable execution time.
- $TOTD_i$: Total time spent by $task_i$ on handling devices; the worst case should be known precisely.
- B_i: Blockage delay by other tasks due to preemption, queuing delay in handling device, critical section and delay by time sharing policy. In the worst case, B_i is calculated with the following formula.

  $$B_i = \sum_{j=1}^{n} TOTR_j \cdot (\lceil GT_i / GT_j \rceil + 1) + BD_i,$$

  where $i \neq j$ and if $task_j$ has possibility to interfere with the activity of $task_i$. BD_i represents queuing delay in handling devices.

In other words, if $TOTR_i$, $TOTD_i$, $TOTN_i$, B_i and BD_i are known for all $task_i$, where $1 \leq i \leq n$, and schedulability can be verified by check whether or not the condition $\sum_{i=1}^{n} rate_i \leq 1$ is kept. This approach takes into critical sections in schedulability analysis. And it is scheduling algorithm independent. However, the items above strongly depend on the scheduling policy, so the analysis results in scheduling policy dependent; otherwise, the analysis is too pessimistic. For example, B_i can be reduced to $TOTR_j \cdot (\lceil \frac{T_i}{T_j} \rceil + 1)$. The priority inheritance protocol provides us with another approach for handling critical sections.

4.2. Real-Time Euclid and *Schedalyzer*

Stoyenko introduced Real-Time Euclid [19], which is designed with a set of schedulability analysis provisions built-in for the sake of schedulability analysis. The worst case analysis is done by the closest deadline first algorithm based on the Leinbaugh's approach. The Real-Time Euclid based Schedulability Analyzer, called *Schedalyzer* consists of the front end and the back end. The front end is a sort of timing tool which collects timing and task dependency information from target programs written in Real-Time Euclid and informs them to the back end where schedulability is actually analyzed. With *Schedalyzer*, the execution time of each task can be measured without relaying on a timing tool. Implication among tasks can be also analyzed at the compilation time. However, the available scheduling policy is limited. Besides, it is very hard to apply this approach to existing systems implemented in various languages other than Real-Time Euclid.

4.3. Real-Time Temporal Logic

Real-Time Temporal Logic (RTL) is proposed by Ostroff [16] extending Extended State Machine (ESM) with time bounded events. This approach models a Real-Time system such as a chemical plant with ESMs, and specifies timing specifications of those systems in RTL. Then, system behaviors, i.e. including schedulability, can be verified. Therefore, this approach might give us a scheme to analyze system behavior more precisely, not only in the task level but the procedure or function level, if all timing specifications of target systems can be represented completely in RTL.

5. Future Work

One of our primary objectives is to provide an interactive tool for a real-time system designer to modify or upgrade the existing real-time systems. However, the current implementation of *Scheduler1-2-3* can analyze the rate monotonic algorithm for a hard and periodic task set. To analyze other scheduling policies, *Scheduler1-2-3* should be extended to be able to perform an efficient simulation of the given policy. Another immediate extension is related to the response time analysis for aperiodic tasks. Not only the deferrable server algorithm, but a background, polling, priority exchange algorithms should be included to compare the effectiveness of the algorithms.

Two important extensions are also planned in more sophisticated capabilities for the analysis. One is an extension for intelligent diagnostics which include intelligent suggestions (e.g. a result from a period transformation method) to improve the schedulability. Another extension is related with the response time analysis for aperiodic tasks. The current analysis for the aperiodic task set is to estimate their response time under a given scheduling policy. However, a reversed analysis is also important. That is, for a given response time requirement of the aperiodic task set, *Scheduler1-2-3* should be able to estimate the necessary specifications of the deferrable server or anything that can produce the given response time. The goal of this reversed analysis would be to exploit a scheduling scheme for a mixture of periodic and sporadic tasks. Finally, we are also planning to extend the domain of the real-time computational model to a distributed environment, a stochastic execution model, as well as a cooperating task set using shared variables or message passing by using the priority inheritance algorithm.

6. Summary

Despite advances in Software Engineering, a good programming tools for real-time systems are missing. In particular, we need to develop a better tool set which can easily detect or eliminate nasty "timing error" from a real-time program. In this paper, we described an interactive schedulability analyzer, *Scheduler1-2-3*, which can analyze whether a given task set can meet their timing constraints or not at the system design phase. *Scheduler1-2-3* can also be used as a synthetic workload generator for the ART testbed and can cooperate with a real-time monitor. It then allows us to verify our scheduling policy on the testbed. We also demonstrated the practicability and ease of use of *Scheduler1-2-3* illustrated by a few examples of the INS task set. Our priority based scheduling analysis scheme is much effective in the communication domain. Finally, various extensions of the analysis capabilities of *Scheduler1-2-3* are under way and soon it will be able to perform more sophisticated integrated analysis for the mixture of the hard periodic and soft aperiodic tasks.

Acknowledgment

The authors would like to thank John Lehoczky, Lui Sha and Ragunathan Rajkumar for their contributions to the analysis algorithms for *Scheduler 1-2-3*. We also are indebted to Cliff Mercer for valuable comments on this paper.

References

[1] Barstow, D. R., Shrobe, H. E. and Sandwell, E. (editors).
Interactive Programming Environment.
McGrow-Hill, 1984.

[2] Bhatt, D., Ghonami, A. and Ramanujan, R.
An Instrumented Testbed for Real-Time Distributed System Development.
In *Proceedings of 8th IEEE Real-Time Systems Symposium.* December, 1987.

[3] Borger, M. W.
VAXLEN Experimentation: Programming a Real-Time Periodic Task Dispatcher using VAXLEN Ada 1.1.
Technical Report CMU/SEI-87-TR-32(ESD-TR-87-195), Carnegie Mellon University, September, 1987.

[4] Cornhill, D., Sha, L., Lehoczky, J. P., Rajkumar, R. and Tokuda, H.
Limitations of Ada for Real-Time Scheduling.
In *International Workshop on Real-Time Ada Issues.* 1987.

[5] Dart, S. A., Ellison, R. J., Feiler, P. H. and Habermann, A. N.
Software Development Environment.
COMPUTER 20(11), November, 1987.

[6] Goldberg, A.
Smalltalk-80.
Addison-Wesley, 1984.

[7] Kligerman, E. and Stoyenko, A. D.
Real-Time Euclid: A Language for Reliable Real-Time Systems.
IEEE Transaction on Software Engineering SE-12(9), September, 1986.

[8] Lehoczky, J. P. and Sha, L.
The Average Case Behavior of The Rate Monotonic Scheduling Algorithm.
Technical Report, Computer Science Department, Carnegie Mellon University.
1986

[9] Lehoczky, J. P., Sha, L. and Strosnider, J. K.
Enhanced Aperiodic Responsiveness in A Hard Real-Time Environments.
In *Proceedings of 8th IEEE Real-Time Systems Symposium.* December, 1987.

[10] Leinbaugh, D. W.
Guaranteed Response Times in a Hard-Real-Time Environment.
IEEE Transaction on Software Engineering SE-6(1), January, 1980.

[11] Liu, C. L. and Layland, J. W.
Scheduling Algorithm for Multiprogramming in a Hard Real-Time Environment.
JACM 20(1), 1973.

[12] Locke, C. D., Jensen, E. D. and Tokuda, H.
A Time-Driven Scheduling Model for Real-Time Operating Systems.
Proc. IEEE Real-Time Systems Symposium . December, 1985.

[13] Miller, B. P., Secherest, S. and Macrander, C.
A Distributed Program Monitor for Berkley Unix.
Technical Report UCB/CSD 84/201, University of California, Berkley, 1984.

[14] Mok, A. K.
Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment.
PhD thesis, Massachusetts Institute of Technology, May, 1983.

[15] Mok, A. K., Amerasinghe, P. and Chen, M.
Synthesis of a Real-Time Processing System with Data-driven Timing Constraint.
In *Proceedings of 8th IEEE Real-Time Systems Symposium*. December, 1987.

[16] Ostroff, J. S. and Wonham, W. M.
Modelling, Specifying and Verifying Real-Time Embedded Computing Systems.
In *Proceedings of 8th IEEE Real-Time Systems Symposium*. December, 1987.

[17] Sha, L., Lehoczky, J. P. and Rajkumar, R.
A Schedulability Test For Rate-Monotonic Priority Assignment.
Computer Science Department ART project, Carnegie Mellon University.
July, 1987

[18] Sha, L., Rajkumar, R. and Lehoczky, J. P.
Priority Inheritance Protocols: An Approach to Real-Time Synchronization.
Computer Science Department ART project, Carnegie Mellon University.
April, 1988

[19] Stoyenko, A. D.
A Schedulability Analyzer for Real-Time Euclid.
In *Proceedings of 8th IEEE Real-Time Systems Symposium*. December, 1987.

[20] Tokuda, H., Wendorf, J. W. and Wang, H.-Y.
Implementation of a Time-Driven Scheduler for Real-Time Operating Systems.
In *Proceedings of 8th IEEE Real-Time Systems Symposium*. December, 1987.

[21] Tokuda, H., Kotera, M. and Mercer, C. W.
A Real-Time Monitor for a Distributed Real-Time Operating System.
In *Proceedings of ACM SIGOPS and SIGPLAN Workshop on Parallel and Distributed Debugging*. May, 1988.

[22] *Reference manual for Ada programming language*
United States Department of Defense, 1983.
ANSI/MIL-STD-1815 A.