Scheduling Strategies and Estimations for Concept-Oriented Rapid Prototyping

A. Burst, M. Wolff, M. Kühl, K. Müller-Glaser
Institute for Information Processing Technology, ITIV
University of Karlsruhe
76128 Karlsruhe, GERMANY
http://www-itiv.etec.uni-karlsruhe.de
email: {burst,wolff,kuehl,kmg}@itiv.etec.uni-karlsruhe.de

Abstract

In this paper scheduling strategies for a rapid prototyping system are discussed. Our rapid prototyping system is able to use several CASE-tools and generate code for models of heterogenous domains. By using the emerging CASE data interchange format CDIF the model data of CASE-tools is represented tool-independent. This tool-independent layer serves as a basis for simulation and code generation. The generated code is partitioned in tasks which must be scheduled as fast as possible with a real-time operating system to support high performance applications. We classify scheduling requirements for the constraints of rapid prototyping and present a new scheduling strategy called pseudo-rate scheduling which significantly improve the execution speed of rapid prototyping applications. Additionally, we provide a set of equations to estimate schedulability. Experimental results demonstrate the main advantages of our scheduling strategy.

1 Introduction

Increasing complexity of electronic real-time systems and time-to-market pressure require a methodical development approach. Beginning with the specification several development phases have to be performed in a top-down process to get the final release. If errors are detected in lower development phases or in the final release all development phases have to be performed once again. Therefore, it is important to start from a correct system specification that exactly reproduces the expected system behavior. To ensure a correct specification, rapid prototyping is used to clarify system goals and the behavior in the real environment. Rapid prototyping is a technique to develop prototypes early in the development process to permit early feedback and analysis.

Typical electronic systems for example an electric window regulator of a car consists of time-discrete components (drive up, stop, etc.) and time-continuous components (speed regulation, pinch protection). For the development of these systems several specification languages (*multilanguage specification*) and several CASE-tools are used. Today, no environment exists which completely fulfills the requirements of a methodical development approach using several CASE-tools. These requirements include tool-independence of simulation and rapid prototyping, support of several design domains (i.e. time-discrete and time-continuous techniques), support of an overall simulation and code generation for all used domains. In our work we focus on concept-oriented rapid prototyping which is a fast conversion of an executable specification into a functional software prototype. Our approach is based on CASE-tools like STATEMATE[1] and MATRIX$_X$[2], automatic code generation and powerful extensible hardware to obtain functional prototypes [3]. Using the standardized CASE data interchange format CDIF[3] [5] a tool independent rapid prototyping system can be built [4]. The generated code for a rapid prototyping system [2] consists of three components which are responsible for the logical behavior of the model, the integration of the input/output hardware and the scheduling. These components form several tasks which are executed on a specialized hardware platform equipped with I/O hardware. To obtain a high performance, these tasks have to be scheduled as fast as possible considering timing and resource constraints. Therefore scheduling strategies suitable for concept-oriented rapid prototyping have to be examined.

In the following we will introduce some terms used in scheduling theory. First, we have to distinguish *static* and *dynamic* scheduling. While static scheduling determines the scheduling strategy at compilation time, the dynamic scheduling determines the scheduling strategy at run time. *Preemptive* scheduling allows a task to be interrupted at execution time and go on working later. Every preemption is time-consuming because of the context switch of a task (*latency*). A task is called *data independent* if it shares no data with other tasks and if there is no need for intertask communication. If a task uses resources like functional resources (e.g. a processor), communication resources (e.g. a bus) or memory resources (e.g. RAM / ROM) it is called *limited on resources*. If a task is repeated repetitively with a constant interval the scheduling is called *periodic*. At last, we have to distinguish between *uniprocessor* and *multiprocessor* scheduling [14].

The generated code for a rapid prototyping system is partitioned into several tasks which have to be executed sharing a functional resource (the processor). This limitation means that more important tasks (like a task responsible for the timing) must be executed with a higher priority than less important tasks (*priority scheme*). A scheduling strategy which uses a priority scheme has to decide whether a task must be started,

1. STATEMATE is a registered trademark of I-Logix. Inc.
2. MATRIX$_X$ is a registered trademark of Integrated Systems, Inc.
3. CDIF is a registered trademark of the Electronic Industry Assoc.

suspended, interrupted or resumed. Scheduling theory typically uses *metrics* like minimizing the sum of completion times, minimizing the weighted sum of completion times, minimizing scheduling length or minimizing the maximum lateness. We have to decide metrics that meet the requirements of concept-oriented rapid prototyping. Furtheron, we have to determine a suitable scheduling strategy and a criterion for the schedulability estimation of an application.

The paper first summarizes the related work on scheduling in rapid prototyping systems. In section 3 we will classifiy the scheduling of a concept-oriented rapid prototyping system. Section 4 contains scheduling estimations for rate-monotonic scheduling and pseudo-rate scheduling. The second scheduling strategy takes advantage of rapid prototyping constraints and is presented for the first time. The results of both scheduling algorithms are presented in section 5. Finally, section 6 offers a conclusion.

2 Related Work

Most of the code generators of CASE-tools provide the user only with components which reproduce the logical behavior of the model. Support for executing this code on a real-time target platform or a scheduling concept for the execution is rare. If a scheduling strategy is offered, it is often written for simulation purpose only. In general, these scheduling strategies do not take deadlines of a real-time execution into account or even jump back in time like the Time-Warp algorithm [8].

STATEMATE, a CASE-tool for time-discrete modeling (statecharts), is an example for that. It only supports code generation and scheduling for simulation purpose [6]. If a user wants to build a prototype he has to implement scheduling and I/O connection on his own. MATRIX$_X$ [7] and Matlab[4] [13] which are mainly used for modeling time-continuous systems, provide better real-time support. MATRIX$_X$ for example offers three different kinds of tasks:

- *free-running periodic tasks* are executed repetitively at a fixed frequency
- *enabled periodic tasks* are executed repetitively but only when enabled.
- *triggered tasks* are executed when a trigger is detected.

The priority for execution is based on the kind of task. The free-running periodic task has the highest priority and the triggered task the lowest. Triggered tasks are of low priority because a static scheduling strategy cannot react to the time spent for an additional task and the real-time constraints of the periodic scheduled tasks must be met under all circumstances. The real-time code is mainly generated for a specialized hardware platform (RealSim series). Other real-time platforms are basically supported by MATRIX$_X$ but this costs additional user effort and the performance is not optimal.

So we can state that for rapid prototyping purpose scheduling strategies for simulation cannot be used. If a CASE-tool offers support for real-time scheduling it is optimized for a specific hardware platform. Scheduling strategies for several domains and optimization techniques for these are hardly examined.

4. Matlab is a registered trademark of The MathWorks Inc.

3 Scheduling Classification for Concept-Oriented Rapid Prototyping

In the following we will classify scheduling strategies considering constraints and requirements of concept-oriented rapid prototyping. According to the terms introduced in section 1 we have to distinguish scheduling for uniprocessor or multiprocessors. Multiprocessor scheduling causes a lot of communication overhead and the scheduling problem is np-complete [14]. Therefore we will concentrate on uniprocessor scheduling. Dynamic scheduling will be used if new tasks have to be generated at runtime. It is not deterministic and the computation of scheduling at run time is time-consuming. Especially in high performance environments, time used for the computation of scheduling can be missed to keep the deadlines of real-time tasks. In the field of rapid prototyping a deterministic behavior is necessary which requires a precise prediction of the execution scenario. Static scheduling guarantees determinism if deadlines, constraints and execution times are known at compilation time and is therefore preferred.

The interruption of a task during its execution (*preemption*) can be permitted in general. Particulary in case of unbalanced tasks (a mixture of tasks with short deadlines and tasks with long deadlines) this property is necessary to obtain a working scheduling. The scheduling strategy has to assign higher priorities for tasks with faster sample rates and lower priorities to the slower ones in order to guarantee schedulability. For periodic tasks this is the optimum static priority scheme [11]. However, if execution times are small compared to the period or the deadline it would not be useful to interrupt them because every preemption requires some additional time for changing the context (*latency*). Because the user can model tasks for a rapid prototyping system with any sample rates we have to consider preemption for the scheduling of concept-oriented rapid prototyping.

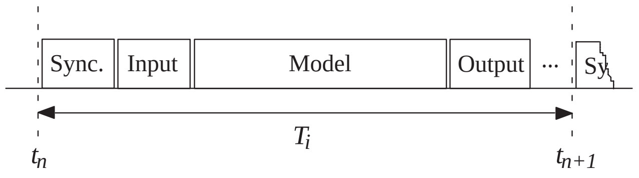

Figure 1: Structure of a task

To guarantee a deterministic behavior in a real-time environment the tasks are scheduled as encapsulated objects which accept inputs and return outputs strictly under the control of the scheduler. Such an encapsulated object has a limited completion time C_i and consists of four components (Fig. 1) which are executed repetitively. The synchronization component is responsible for the communication with the scheduler. Then, input variables are updated and the model component is executed. At last, output variables are written. Changes of output variables within a period T_i lead to a non-deterministic behavior during the execution of a model, because other tasks would rely on output variables which can be updated at any time. Therefore output variables of every task have to be updated at the end of a period. *Double-buffering* is a method to give every task its own local variables. Using double-buffering, the input variables are updated before the execution of a task. These input variables are valid for the entire period. The output variables are written in local memory at any time but the global output variables are only updated at the end of the task's period. This leads to a data independence of all executed tasks. Almost all rapid prototyping applications have resource limitations. Mainly functional resources are affected while memory and communication resources are not critical. The usage of a multiprocessor capable real-time operation system (e.g. VxWorks[5]) can improve the situation for functional resources but this will lead to a large communication overhead.

To solve algorithms of time-discrete and time-continuous models an accurate timing is needed. For time-discrete models the time to process one transition must be known. Time-continuous models which are processed in a time-discrete manner need a time base to solve differential equations or generate signals. A periodic scheduling of tasks offers the best solution, because it uses a time base which can also be used for the execution of these algorithms. A scheduling strategy with a fixed rate is called rate-monotonic scheduling.

For rapid prototyping it is important to maximize the number of tasks that will be completed in a given period. Accordingly we can use the minimization of the maximum lateness L_{max} as a metric. Lateness of a task is defined as the interval between deadline and completion time. So, exceeding these deadlines should be minimized (strictly spoken this should never happen because we are dealing with hard real-time tasks, but the given metric is able to schedule as fast as possible). According to Lawler's notation [9] in which the problem definition is given with $\alpha \mid \beta \mid \gamma$, where α indicates the machine environment (i.e. $\alpha = 1$ indicates a uniprocessor machine), β indicates the constraints (*prec* stands for precedence, *pmtn* for preemption, *resrc* for limited resources and *indpt* for data independence), and γ indicates the metric, concept-oriented rapid prototyping can be characterized as:

$$1 \mid static,\ prec,\ pmtn,\ resrc,\ indpt \mid L_{max}$$

In the following section we will discuss the schedulability of periodic tasks. Additionally, a new scheduling strategy considering the constraints of real-time execution is discussed leading to shortest possible execution times.

4 Scheduling Estimation for Concept-Oriented Rapid Prototyping

4.1 Rate-Monotonic Scheduling

The rate-monotonic scheduling is a special case of deadline monotonic scheduling. Deadline monotonic scheduling can be identified by the inequality

$$computation\ time\ C_i \le deadline \le period\ T_i$$

whereas rate-monotonic scheduling is characterized by

$$computation\ time\ C_i \le deadline \overset{!}{=} period\ T_i$$

If this inequality can be kept it is guaranteed that the periodic execution of a system can keep the deadline. The most obvious condition for a successful scheduling which is a necessary and sufficient condition [12] is

$$\sum_{i=1}^{n} \frac{C_i}{T_i} = \sum_{i=1}^{n} U_i \le 1.$$

U_i is called utilization. This condition can be enhanced with the introduction of an upper bound for rate-monotonic scheduling. This bound is characterized by the relation

$$\sum_{i=1}^{n} U_i \le n\left(2^{\frac{1}{n}} - 1\right),$$

where n is the number of independent periodic tasks [12]. This *first theorem* states that if the inequality above is fulfilled, all tasks can be executed in their period. The value of the upper bound converges for rising n to $\ln 2 \approx 0,693$. However, the condition is only sufficient, not necessary. Therefore the condition can indicate by mistake that a system is not schedulable. This leads to a necessary and sufficient set of inequalities [10]:

$$\forall i : 1 \le i \le n \quad \min_{(k,l) \in W_i} \left(\sum_{j=1}^{i} \frac{C_j}{l\,T_k} \left\lceil \frac{l\,T_k}{T_j} \right\rceil \right) \le 1$$

$$\text{and } W_i = \left\{ (k,l) \mid 1 \le k \le i\ ,\ l = 1, \ldots,\ \left\lfloor \frac{T_i}{T_k} \right\rfloor \right\}$$

This *second theorem* constructs ineqalities for all schedulability points. The schedulability points are determined by computing all successive periods for all tasks up to the end of the first period of the lowest frequency task. For one schedulability point, the inequality consists of a sum of possible execution times of all tasks that can be activated before the schedulability point, divided by the value of time corresponding to the schedulability point.

Unfortunately the computation of both of these theorems has a different complexity. While the first theorem is increasing with $o(n)$ the second is increasing considerably stronger because of it's data dependence [1]. The number of calculations required is dependent on the values of task periods. For the worst case the calculation can consist of a schedule for each task up to the period of the lowest frequency task.

4.2 Pseudo-Rate Scheduling

An additional constraint for real-time systems must be met if an onboard hardware timer is used to generate a periodic interrupt as a time base for tasks. Real-time operating systems usually support one onboard hardware timer which is called auxiliary timer. The execution of real-time systems which are not using this auxiliary timer as a time base are slower because they have to use a time-consuming watchdog timer (software-implemented). The advantage of watchdog timers is that they can produce any period they want. Tasks which are using inter-

5. VxWorks is a registered trademark of WindRiver Systems.

rupts of an auxiliary timer are limited because they have to use the greatest common divisor (gcd) of all task periods as a common time base. The required resolution δ for a set of tasks with different periods is given with the following equation:

$$\forall i : \frac{T_i}{\delta} \in \mathbb{N}, \qquad \underset{j,k=1\ldots n,\ j \neq k}{\forall} \delta = gcd\,|T_j - T_k|$$

In the equation above the time differences between all task periods are calculated. The required resolution δ is equivalent to the greatest common divisor of all differences. Sometimes it is not possible to generate resolutions which are small enough. For example if a task needs a sample rate of 1 kHz and another needs 800 Hz we will need a resolution δ of 0,25 ms although both sample rates are equal or larger than 1 ms (Fig. 2). If the resolution gets smaller additional interrupts must be executed by the real-time operating system. This shortens the time that remains for the execution of the model tasks. Therefore it is necessary to find the smallest possible resolution that keeps the real-time constraints and does not produce interrupts so frequent that the processor spends too much time servicing the interrupts. If the processor is overloaded with interrupts and cannot execute the model tasks this is called *trashing*. If the sample rate of a task must be adapted to get the same or a better precision like rate-monotonic scheduling the adapted sample rate must be higher than the previous one. In our example, a sample rate of 1 kHz for both tasks will result in a lower resolution ($\delta = 1ms$). The effects on the task with a higher resolution will be discussed in section 4.3. Scheduling strategies for systems using a single fixed rate as their time base are called pseudo-rate scheduling.

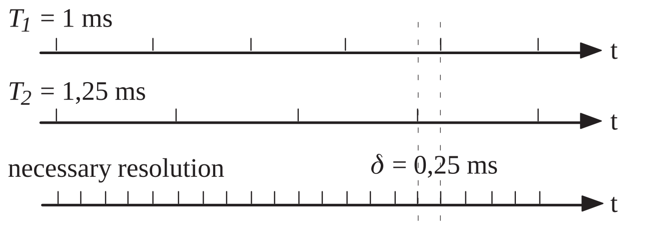

Figure 2: Necessary resolution δ

The schedulability of those systems cannot be determined by the two theorems obtained in section 4.1. Strictly spoken the deadline of the tasks which have to be adjusted will remain the same. Therefore we will obtain the case that the deadline is longer or equal compared to the period. Literature shows that there is no general solution to this problem [15]. In this special case however we can set the deadline equal to the period. Instead of T_i we obtain a shortened period T_i' which can be expressed as:

$$T_i' = \left\lfloor \frac{T_i}{T_{aux}} \right\rfloor T_{aux}$$

T_{aux} is the period of the auxiliary timer. With this, both of the theorems above can be adapted. The first theorem can be extended to:

$$\sum_{i=1}^{n} \frac{C_i}{\left\lfloor \frac{T_j}{T_{aux}} \right\rfloor T_{aux}} \leq n\left(2^{\frac{1}{n}} - 1\right), \text{ and } T_{aux} \leq T_j .$$

In this way we only get values for the period which are multiples of T_{aux}. The second theorem can be extended to:

$$\min_{(k,l) \in W_i} \left\{ \sum_{j=1}^{i} \frac{C_j}{l \left\lfloor \frac{T_k}{T_{aux}} \right\rfloor T_{aux}} \left\lceil \frac{l \left\lfloor \frac{T_k}{T_{aux}} \right\rfloor T_{aux}}{\left\lfloor \frac{T_j}{T_{aux}} \right\rfloor T_{aux}} \right\rceil \right\} \leq 1 ,$$

$$W_i = \left\{ (k,l) \mid 1 \leq k \leq i ,\ l = 1, \ldots , \left\lfloor \frac{\left\lfloor \frac{T_i}{T_{aux}} \right\rfloor T_{aux}}{\left\lfloor \frac{T_k}{T_{aux}} \right\rfloor T_{aux}} \right\rfloor \right\}$$

Like the second theorem for rate-monotonic scheduling ineqalities for all schedulability points are constructed but this time for the revised schedulability points. As an example for this complex theorem we will vary the example which is given in [15] (Tab. 1). We assume an auxiliary timer interrupt every 20 time ticks. Tab. 1 shows three tasks, their execution time C_i, the rate-monotonic period T_i, the pseudo-rate period T_i' as well as ratio, utilization and the upper bound of the first theorem. In [15] it was shown that a rate-monotonic scheduling is possible. In the following we will show that a pseudo-rate scheduling is not possible.

Task i	Execution time C_i	rate-monotonic T_i	pseudo rate T_i'	Ratio C_i/T_i'	Utilization 1 .. N	upper bound
1	45	135	120	0,375	0,375	1,000
2	50	150	140	0,357	0,732	0,828
3	80	360	360	0,222	0,954	0,779

Table 1: Example of pseudo-rate scheduling

To examine the schedulability of the three tasks given in Table 1 we only have to consider the case $i = 3$ of the second theorem because the first two tasks are schedulable according to the first theorem ($utilization \leq upper\ bound$). The index j and k must vary between 1 and 3. For $k = 1$ l must vary between 1 and 3, for $k = 2$ between 1 and 2, and for $k = 3$ $l = 1$. We will not show all of these terms because the calculation is simple. However, none of the inequalities is fulfilled indicating that the system cannot be pseudo-rate scheduled with the given values.

$k = 1, l = 1$:

$$\frac{C_1}{1 \cdot T_1'} \left\lceil \frac{1 \cdot T_1'}{T_1'} \right\rceil + \frac{C_2}{1 \cdot T_1'} \left\lceil \frac{1 \cdot T_1'}{T_2'} \right\rceil + \frac{C_3}{1 \cdot T_1'} \left\lceil \frac{1 \cdot T_1'}{T_3'} \right\rceil =$$

$$\frac{45 + 50 + 80}{120} = 1,458 \overset{!}{>} 1$$

$k = 1, l = 2$:

$$\frac{C_1}{2 \cdot T_1'} \left\lceil \frac{2 \cdot T_1'}{T_1'} \right\rceil + \frac{C_2}{2 \cdot T_1'} \left\lceil \frac{2 \cdot T_1'}{T_2'} \right\rceil + \frac{C_3}{2 \cdot T_1'} \left\lceil \frac{2 \cdot T_1'}{T_3'} \right\rceil =$$

$$\frac{2 \cdot 45 + 2 \cdot 50 + 80}{2 \cdot 120} = 1,125 \overset{!}{>} 1$$

. . .

$k = 3, l = 1$:

$$\frac{C_1}{1 \cdot T_3'} \left\lceil \frac{1 \cdot T_3'}{T_1'} \right\rceil + \frac{C_2}{1 \cdot T_3'} \left\lceil \frac{1 \cdot T_3'}{T_2'} \right\rceil + \frac{C_3}{1 \cdot T_3'} \left\lceil \frac{1 \cdot T_3'}{T_3'} \right\rceil =$$

$$\frac{3 \cdot 45 + 3 \cdot 50 + 80}{360} = 1,014 \overset{!}{>} 1$$

To enable scheduling, the periods of the tasks have to be increased, the execution times have to be shorter or a smaller resolution has to be chosen. If we assume a reduction of the execution time of task 3 from 80 to 75 time units (C_3'=75), according to theorem 1 the utilization is still above the upper bound (0,940 > 0,779). But this time according to the second theorem the tasks are schedulable. The critical case is given with

$k = 3, l = 1$:

$$\frac{C_1}{1 \cdot T_3'} \left\lceil \frac{1 \cdot T_3'}{T_1'} \right\rceil + \frac{C_2}{1 \cdot T_3'} \left\lceil \frac{1 \cdot T_3'}{T_2'} \right\rceil + \frac{C_3'}{1 \cdot T_3'} \left\lceil \frac{1 \cdot T_3'}{T_3'} \right\rceil =$$

$$\frac{3 \cdot 45 + 3 \cdot 50 + 75}{360} \overset{!}{=} 1$$

From this case we can obtain the scheduling for this example with pseudo-rate scheduling. For the critical case we have to execute three times C_1, three times C_2 and one time C_3' within 360 time units. The result is shown in Figure 3.

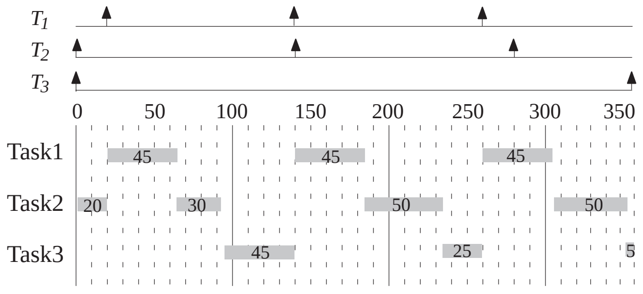

Figure 3: Pseudo-rate scheduling of Tab. 1

4.3 Influence of Pseudo-Rate Scheduling

The reduction of periods of a task leads to consequences for the execution of the time-discrete and time-continuous models. Tasks with time-continuous models will improve their precision because they use an integration algorithm with fixed step size like Euler or Runge–Kutta and the precision of these algorithms rises with shorter periods. The generation of signals like pulse width modulation (PWM) however might lead to problems. If a signal has to keep a sequence which cannot be met by the time base of the auxiliary timer the generated signal will differ. According to the sampling theorem the maximum difference τ is

$$\tau = \frac{\delta}{2}$$

A similar problem exists for tasks with time-discrete models. If two input changes occur within the time of a period it is not guaranteed that both events are processed in the same period. However, the simultaneous processing of both events can also not be guaranteed for every case if the period is not reduced (frequence shift). The dependence of simultaneous events of a model must be avoided because this will lead to a non–deterministic behavior. Modeling of absolute times can also cause problems because if a timeout event or a scheduled action cannot be executed in time the behavior of the system can be influenced.

These are undesired effects which can only be avoided if the user already choose in the model a sample rate which is a multiple of the desired auxiliary timer resolution. If an adaptation of a period have to be done in order to match the time base of an auxiliary timer the user of a rapid prototyping system must be informed about the consequences.

5 Results

Our results show the execution speed superiority of the pseudo-rate scheduling compared to the rate-monotonic scheduling. This speed superiority results mainly from the use of a hardware implemented auxiliary timer which produces an interrupt every period and does not cause the overhead of a software implemented timer. Rate-monotonic scheduling requires a timer implementation for every task with a watchdog timer but can be adapted to any period. Figure 4 shows the scheduling results of pseudo-rate scheduling compared to rate-monotonic scheduling for different tasks. For the tests we used a Motorola VMEbus based system with a 200 MHz PowerPC Processor 604, the real-time operating system VxWorks and self-developed general purpose I/O hardware. The advantage of the pseudo-rate scheduling is at least about 20 percent and is strongly rising with the number of tasks.

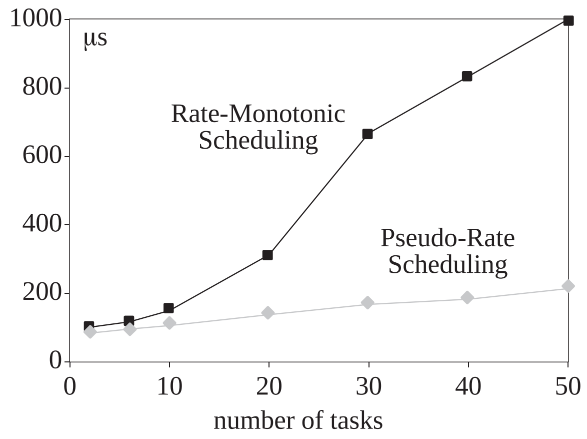

Figure 4: Comparison of scheduling strategies

To show the effects of the different scheduling strategies for an industrial application we have selected an electrical window regulator for demonstration purpose. The window regulator was built according to industrial specifications including position counting, pinch protection and theft protection. The application consists of a MATRIX$_X$ and a STATEMATE model and uses our code generation modules based on the application independent CASE data interchange format CDIF. The STATEMATE model, responsible for the operational control, consists of 22 basic states, 3 hierarchical and 3 orthogonal states while the MATRIX$_X$ model, responsible for filtering signals, consists of 5 nested FIR filters. Pseudo-rate scheduling was executed in 90 µs, whereas rate-monotoic scheduling was executed in 120 µs.

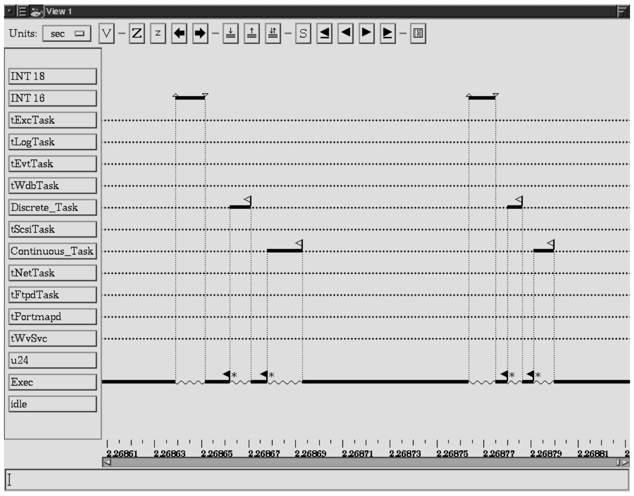

Figure 5: Screenshot of pseudo-rate scheduling

Figure 5 shows a screenshot done with the real-time visualization tool WindView of the real-time behavior with pseudo-rate scheduling. The auxiliary timer which generates an interrupt each period is called ’Int16’, the task named ’Exec’ corresponds with the scheduler and the time-discrete and time-continuous tasks are called ’Discrete_Task’ and ’Continuous_Task’. A filled flag indicates that the scheduler gives a semaphore for every model task. The two model tasks take them, execute their models and allow the scheduler to proceed. If a model task was not finished before the next auxiliary timer interrupt, an error occured and the execution will enter a fail safe mode.

6 Conclusion

In this paper we have classified different scheduling strategies for concept-oriented rapid prototyping systems. Rapid prototyping systems with their constraints of fast execution times and deterministic behavior require a periodic scheduling. To optimize the execution times we can use pseudo-rate scheduling which uses a hardware implemented auxiliary timer to generate periodic interrupts as a time base for tasks. For the first time schedulability of those systems were analyzed methodically. The advantages of pseudo-rate scheduling compared to rate-monotonic scheduling are shown in experimental results. An industrial application shows the potential of concept-oriented rapid prototyping which now can be used in areas with highest performance requirements like engine management systems.

Future work will include a worst-case estimation of the execution time C_i of tasks to obtain an optimal processor utilization. Furtheron a test of our rapid prototyping system with highest performance systems have to be realized. The optimization of the generated code will complete our work.

Acknowledgements

We thank WindRiver Systems, Inc. and DaimlerChrysler AG for their generous support of our work.

References

[1] Audsley, N.: *Deadline Monotonic Scheduling.* Internal paper, Department of Computer Science, University of York, YCS-90-146, 1990.

[2] Burst, A.; Spitzer, B.; Wolff, M.; Müller-Glaser, K.: *On Code Generation for Rapid Prototyping Using CDIF.* OOPSLA, Vancouver, Canada, 1998.

[3] Burst, A.; Wolff, M.; Kühl, M.; Müller-Glaser, K.: *A Rapid Prototyping Environment for the Concurrent Development of Mechatronic Systems.* ECEC, Erlangen, Germany, 1998.

[4] Burst, A.; Wolff, M.; Kühl, M.; Müller-Glaser, K.: *Using CDIF for Concept-Oriented Rapid Prototyping of Electronic Systems.* RSP, Leuven, Belgium, 1998.

[5] EIA / CDIF Technical Committee: *CDIF / CASE Data Interchange Format.* EIA Interim Std. EIA / IS- 106-112, 1994.

[6] i-Logix, Inc.: *Software Code Generator Reference Manual.* 1998.

[7] Integrated Systems, Inc. (ISI): *Autocode User’s Guide.* 1997.

[8] Jefferson, D.; Sowizral, H.: *Fast Concurrent Simulation Using the Time Warp Mechanism.* Distributed Simulation, SCS, LaJolla, 1985.

[9] Lawler, E.: *Recent Results in the Theory of Machine Scheduling.* In: Mathematical Programming: The State of the Art, A. Bachen, Springer-Verlag, New York, 1983.

[10] Lehoczky, J; Sha, L.; Ding, Y.: *The Rate-Monotonic Scheduling Algorithm: Exact Classification and Average Case Behavior.* Proc. Real-Time Systems Symp., IEEE CS Press, Los Alamitos, Calif., 1989.

[11] Leung, J.; Whitehead, J.: *On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks.* Performance Evaluation, 2, 1982.

[12] Liu, C.; Layland, J.: *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment.* JACM, No. 1, 1973.

[13] The Mathworks Inc.: *Real-Time Workshop User’s Guide.* 1998.

[14] Stankovic, J.; Spuri, M.; Di Natale, M.; Buttazzo, G.: *Implications of Classical Scheduling Results for Real-Time Systems.* IEEE Computer, 6, 1995.

[15] Zalewski, J.: *What every Engineer Needs to Know about Rate-Monotonic Scheduling: A Tutorial.* Advanced Multiprocessor Bus Architectures, IEEE Computer Society Press, Los Alamitos, 1995.