A FRAMEWORK FOR MODELING AND EVALUATING TIMING BEHAVIOUR FOR REAL-TIME SYSTEMS

Zmaranda Doina, Rusu Claudia, Gligor Marius

University of Oradea
5 Armatei Romane street,410087 Oradea, Romania
Email: zdoina@uoradea.ro; clarule79@yahoo.com; glimar78@yahoo.com;

Abstract: In this paper we present a tool for modelling and analyzing real-time systems. The description of the system being modelled is based on system's timing characteristics. Using the implemented tool, several quantitative timing information about the system such as schedulability, response time and loading factor could be estimated. Based on the developed simulated model of a real-time system, the timing behavior could be assessed prior to implementation.

Keywords: Real-time system, deadline, schedulability, Worst Case Execution Time (WCET).

1. INTRODUCTION

A real-time system is generally defined as a system in which time is part of the system's logic. Consequently, in a real-time system, the resource are managed explicitly to satisfy the completion of real-time constrains of the application. These constrains, such as deadlines, result from natural laws (physical, chemical biological, and other) which governs application's behavior and establish acceptable execution completion times for associated real-time computations. In many cases, real-time applications are confused with fast applications; instead of this, real-time means predictable and, if possible, deterministic computation, and these characteristics are independent of the order of magnitude of the time constrains imposed to a specific application.

Deterministic computation in real-time context means that all computation timing is known in advance, and there is no uncertainly about any parameters of the computation. These means that all elements like: arrival time of tasks, duration of execution, resource dependencies that affect system's timing are known in advance. Unfortunately, there are very few real-time applications that meet this determinism criterion.

The most likely situation is the one when we can consider that computation timeliness is not deterministic but predictable in the sense that it can be estimated with a high level of accuracy (acceptably). This implies that several parameters of the computation: arrival time, execution duration, resource dependencies and interactions with other computations are known sufficiently well. This knowledge is acquired generally by formal analysis, simulations, empirical measurements, etc. The resulting values may be expressed in different ways, but, for real time systems, the more suitable approach is as an upper bound or worst case execution time (WCET).

Given the predictability and determinism requirements described above, it is of major importance to be able to evaluate these characteristics for a real-time application. Besides the estimation of time, the degree of predictability is affected also by the scheduler, whose execution should be also controlled. Generally, is a general assumption that, real-time scheduling algorithms are deterministic, even if the parameters of these algorithms are not.

Moreover, it is known that both determinism and predictability are independent of timeliness magnitudes.

2. THE MODEL OF REAL-TIME SCHEDULING

In most cases, responsibility of satisfying application time constrains is granted to the application software. Several approaches are used; the simple one implies a static mapping of the application time constrains to priorities. But, because traditional real-time terminology is not very precise, we can consider that many computer systems are real-time to some different degree: thus, traditional real-time systems are classified into two categories: hard and soft, according to the degree for which they satisfy the predictability and determinism criteria.

Hard real-time systems are defined as deterministic systems, in the sense that it is mandatory that all system's computations must meet their deadlines, otherwise the system will fail. On the other side, soft real-time systems are not necessary deterministic, in the sense that, if in some cases a deadline will be missed, that is acceptable.

If, during the designing phase, an analytical verification could be done for a real-time system in order to prove that the system will function correctly in every possible situation (meet all deadlines), then we have a fully deterministic real-time system. Unfortunately, for many real-time applications such analytical verification is not feasible in practice: most of hard real-time applications were developed using the "best-effort" approach. This approach cannot guarantee that all deadlines should be met, but, at least, could minimize the probability to which this could happen.

In order to be able to estimate the timing properties of a real-time system, comprehensive methods for expressing time constrains and scheduling criteria should be used. In our model, we consider that the unit of computation (or computational entity) is the task.

Generally, most real-time systems are implemented to execute periodically the following: read data from sensors process the inputs and respond to the controlled system through the actuators. Because some (sometimes all) responses to external events must be obtained in a pre-determined time (usually this time is determined based on the specific requirements of the controlled system and its interaction with external environment), the tasks that implement these jobs are subjected to deadlines.

Consequently, the task is subject to a completion time-constraint or deadline: the time period during which completion of the real-time execution is acceptable. The execution of each real-time task is not necessary done to maximize its individual temporal acceptability: because a real-time system is composed normally by several tasks, each with its particular deadline, a collective temporal acceptability criterion instead of individual ones could be a better approach.

The main goal of the chosen scheduling algorithm is to assure that the set of task will meet their deadlines globally. A particular set of task is schedulable, if there is at least one algorithm which can schedule the tasks. That means that, the collective temporal accessibility can be sufficiently satisfied.

3. FRAMEWORK DESCRIPTION

3.1. Task specification

The main concept on which the constructed framework is based is the task. In this approach, a task is an executable program characterized by several parameters and constrains. For implementation of the above described model, a framework was developed in order to simulate the execution of several tasks, with specific characteristics and using a specific scheduling algorithm. The task could be divided based on task arrival pattern into two task models: periodic and a-periodic.

Usually, classical scheduling policies have as a main goal only to cope with satisfaction of deadline constrains. Our tool allows creating a model of the real-time system based on concurrent tasks and computes quantitative information about system's model. The resulting output allows the user to check the real-time system from the temporal point of view: issues like schedulability of tasks can be determined based on a-priori information given for each task.

Consequently, for each task i the timing characteristics can be modelled using the following parameters:
- task identifier ID_i and name (unique) N_i – defines a task uniquely
- task i execution or computing time C_i– the execution time of a task can vary within an interval $[C_{Bi}, C_{wi}]$ where and are the best respectively the worst case execution times for the task i (in most of the cases Worst Case Execution Time is denoted by $WCET_i$); generally, only the best and the worst execution time of each task are known. When modeling the timing behavior of tasks, this is a natural approach, because exact computation time of a task cannot be establish. Consequently, both times have to be considered when creating a task behavior model, and results must be compared.
- task deadline D_i – deadline is a typical task constraint in real-time systems: the time before which the task must complete its execution. Usually, the

deadline of the task is relative, meaning that, from the moment when a task i arrives, it should finish within D_i time units.
- task period T_i – for periodic tasks, task period T_i is considered to be equal to task deadline D_i
- task ready (arrival) time R_i – represents the moment of time when the task i is ready for execution. From the time being, all tasks are assumed to arrive at the moment 0; no arrival pattern is modeled into the actual implementation of our framework. This should be a development direction for future improvment of the framework
- other tasks constrains, for example precedence constrains - generally, in a real-time system, tasks are not running independently: several relations exist between the implied tasks. One important relation between tasks is the precedence relation: task j precedes task i, if task i could not be executed until task j finishes its execution. Such precedence relations are usually represented using precedence graphs; in our framework, precedence is implemented using a so called precedence list L. For a task i, precedence list contains all task that have to precede task i.

Consequently, we can characterize a task as a tuple of numbers, such as Ti = (N_i, R_i, D_i, C_i). Although these initial parameters are given based on system knowledge and approximations, the constructed model provides a valuable insight into the behaviour of the system and system's dynamics, helping, in many cases, to detect inefficiencies. Also, based on the results obtained from the model several optimisations could be done, and the way on which these optimisations affect the design could be then tested before the actual implementation.

3.2 Scheduling algorithms and schedulability analysis

Schedulability analysis can be done by computing the response time of every process in the system and comparing it to the process deadline. Several scheduling algorithms were implemented, and global system's performance could be compared also depending on the chosen algorithm. The framework permits to select the appropriate scheduling algorithm form several implemented possibilities, as presented in Table 1.

Table 1: Aperiodic/periodic scheduling algoritms

Aperiodic	**Periodic**
Round Robin	Rate Monotonic
Shortest Job First	Earliest Deadline First
First Come First Served	Least Slack Time First
Highest Response Ratio Next	First In First Out
Shortest Remaining Time Next	

For non-periodic tasks, First-come-first-served is the simples scheduling strategy, that does not invoke any task constrains. Tasks are executed in the order of their arrival. It is a non-preemptive scheduling policy, each task being allowed to run to completion.

Round-robin scheduling algorithm represents the preemptive version of First-come-first-served algorithm. The difference between the two ones is that the tasks are set o execute within a fixed time slice, after that, it will be forcibly removed from the processor and put at the end of the ready queue. The task will continue its execution, next time when it will be in the top of the queue.

Shortest-job-first is a priority based alternative to the First-come-first-served scheduling strategy. The criteria in defining task priority is task execution time: task with the shortest computation time C_i becomes highest priority one. It is a non-preemptive scheduling policy. Another algorithm, Shortest-Remaining -Time-Next defines the priority of task in a different way: based on remaining time for task execution.

Highest-response-ratio-next tries to overcome some of the weaknesses of the Shortest-job-first algorithm, by taking into consideration for establishing the prioriy of tasks, two criteria: task execution time and task waiting time. Consequently, priority P_i for a task i being calculated based on the formula presented in (1) where W_i representes task i waiting time (the amount of time task has been ready but waiting to be executed), C_i represents the computation time for task I and P_i is the resulting priority for task i):

$$P_i = \frac{W_i + C_i}{C_i} \qquad (1)$$

For periodic tasks, First-In-First-Out represents the periodic alternative to First Come-First-Served algorithm. Rate monotonic scheduling is a preemptive scheduling policy, based on fixed priorities given by the task period. Rate monotonic scheduling is the preferred scheduling algorithm when analysis of periodic tasks when deterministic behavior is considered. Rate monotonic algorithm is based on several initial assumtions, in order to simplify analytical schedulability analyzis, such as:
- tasks are equally important, and periodic
- task deadline is define by the task period

For non-periodic tasks, which are often triggered by events coming from the external environment and have a non-deterministic behavior, schedulability analysis implies more complicated procedures.

Erliest-Deadline-First algorithm use a single criterion for asigning task priority: task time to deadline. A task will be asigned the highest priority if it is the nearest to deadline.

Another important issue that can be obtained from the model is system's performance, based on the so called limit factor. The average limit factor of the system is computed based on information regarding how close to deadline the tasks are (this factor is computed only when the set of tasks are schedulable) when they finish their execution. If many tasks are close to their deadlines, then the system is characterized by a low limit factor, because adding a new task will lead probably, in many cases, to a non-schedulable system. So, the optimum solution will be chosen from all the schedulable solutions, based on the highest average limit factor.

Some scheduling algorithms are constructed in the idea of maximizing system's limit factor, some are not. For example, Earliest Deadline First or Shortest Remaining Time Next are based on this idea. On the other hand, if we choose another scheduling algorithm with fixed priorities (for example, rate monotonic), then the limit factor (average) for a schedulable set of n tasks could be calculated given the following formula:

$$limit\_factor = \sum_{i=1}^{n} \frac{D_i - C_i}{n} \qquad (2)$$

Where C_i represents the actual execution time and D_i is the deadline for the task I and n represents the total number of tasks in the task set. The above formula gives an average limit factor, and does not maximize the individual ones. The problem of choosing the right configuration of tasks in order to maximize individual limit factors is not simple. In our framework, this is carried out as an trial and error process, based on simulation results.

3.3 Framework implementation

The developed framework permits modeling, simulation and schedulability analysis for real-time systems, being a modeling and analyzing tool for real-time systems that can be described as a set of preemptive or non-preemptive tasks, either periodic or non-periodic. The framework was implemented in Visual C++. It is divided into two parts: a specification part and an analysis part (Figure 1).

In the specification part, user models the analyzed real-time system, by given all initial data about the set of tasks. The framework provides a graphical user interface for specifying system's (tasks) initial characteristics and for simulation of tasks execution.

In the analysis part, the given real-time system is analyzed from the schedulability point of view. Also, system's dynamic behavior could be observed and several optimizations could be done based on these observations.

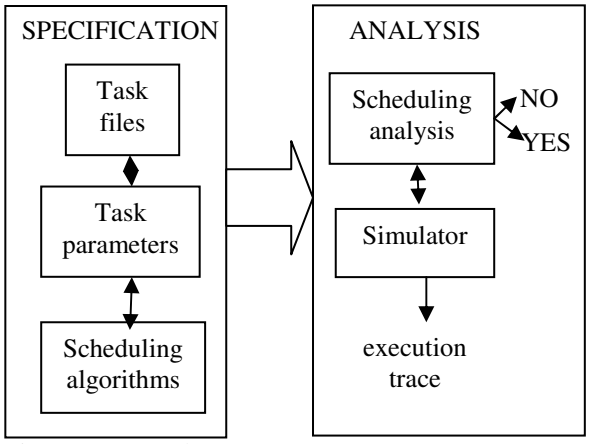

Fig. 1.Framework structure

The main features of our framework are the following:
- a graphical user interface for introducing tasks parameters, such as: deadline, execution time, priority, etc; these parameters are saved in a (text) file for each given task, in a specific format
- several scheduling algorithms are implemented, and the user could choose from a list of implemented algorithms. These algorithms are grouped into two categories: for periodic and non-periodic tasks
- a simulator, that shows a graphical representation of the generated trace of execution for the set of tasks, according to the chosen scheduling algorithm. Using this simulation, the points when tasks arrive, when they are suspended or completed could be easily observed. Also, when a tasks miss its deadline (for a non-schedulable set of tasks), this situation is marked in the simulation view using a red cross. By simulation user can validate the dynamic behavior of the system and see how tasks are executing according to their given parameters and scheduling policy
- schedulability analysis implies checking if all tasks meet their deadlines; a message to the user is displayed, that indicates is system is schedulable or not. For the time being, it is assumed that all the tasks released times are given through the interface, together with other tasks parameters. In the future, we plan to extend the framework by introducing the possibility of creating several arrival patterns for the set of tasks, and the framework will analyze schedulability for all possible resulting states. Also, the evaluation of limit factor as it was stated in (2) is not implemented yet.

Screenshots of the developed framework are presented in Figure 2, 3 and 4 respectively.

3.4 An example

The framework has been experimented with several scenarios. For a given number of tasks, with specific characteristics, simulation were done using several scheduling policies. For each simulation, schedulability analysis indicates if the set of task is

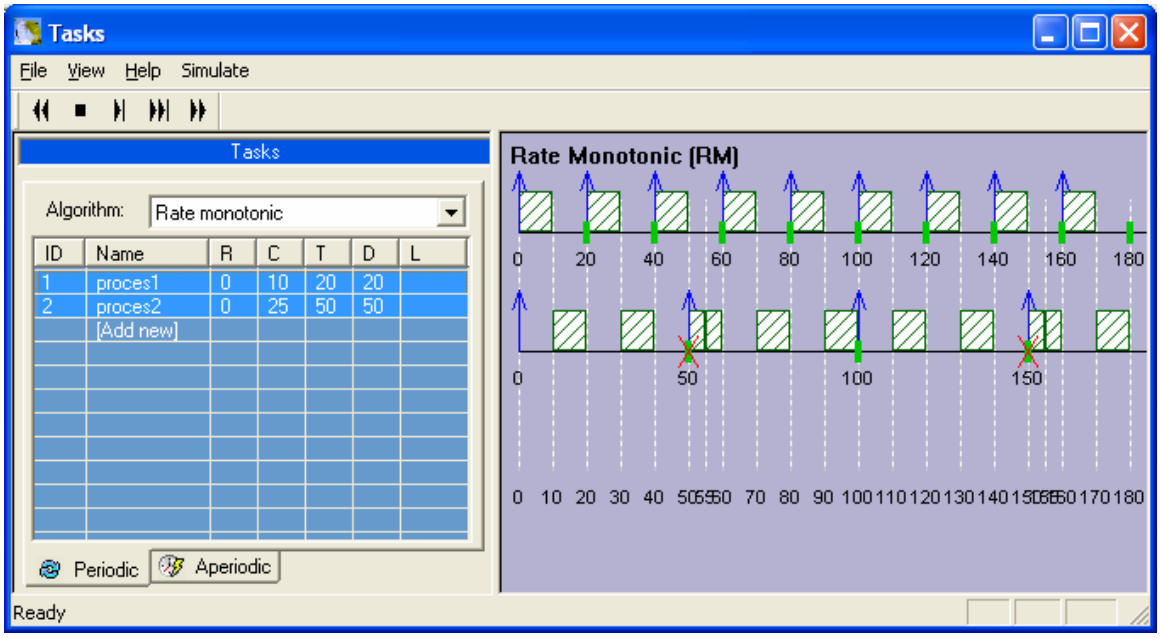

Fig. 2. Periodic task set scheduled with rate monotonic algorithm

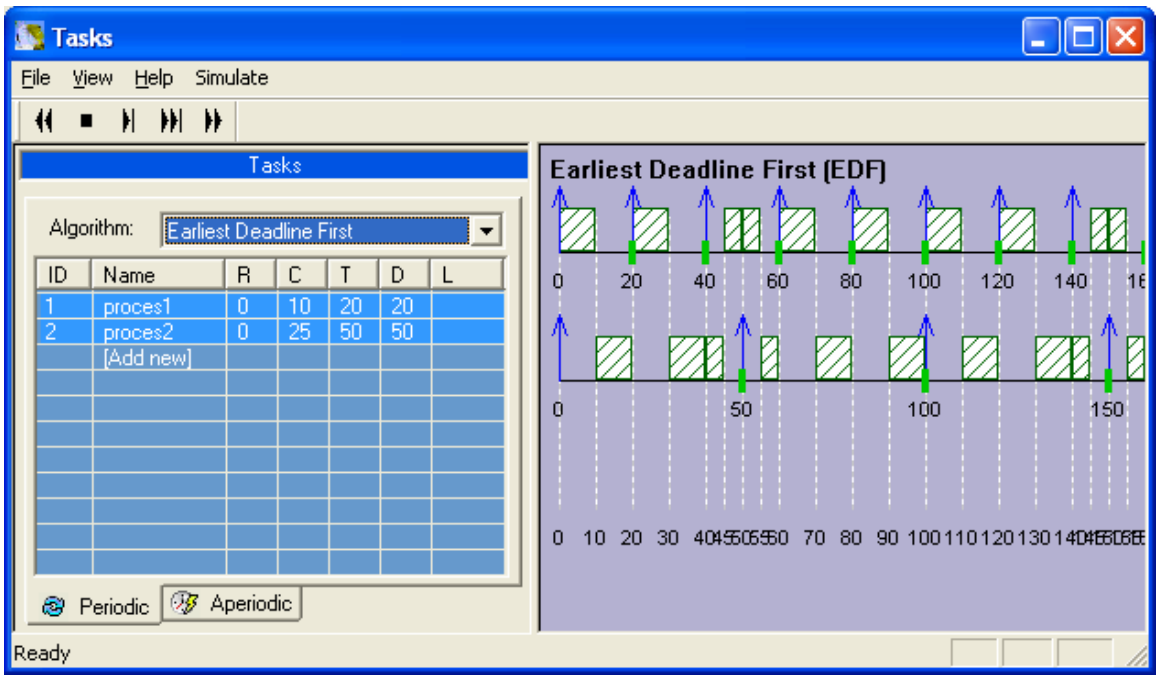

Fig. 3. Periodic task set scheduled with Earliest Deadline First algorithm

schedulable or not; if yes, the average limit factor is calculated based on individual ones.

The system analyzed is composed by several tasks, as it is presented in Table 2 and Table 3. Screenshots of the resulting scheduling for the tasks are presented in Figure 2 and 3, for periodic algorithms and Figure 4, for aperiodic algorithms. The system has been analyzed with two algorithms implemented within the framework: rate monotonic and shortest remaining time next. In addition to the analyzing process, it is possible to select other algorithms; also, to create different system's models for that are using worst case and best case execution times respectively.

The task characteristics (requirements) are given by the framework user and saved into specific files (text). Each task is presented as a sequence of components, each with a different execution time and deadline. Examples of task sets are given in Table 2 and Table 3, for periodic and aperiodic tasks respectively.

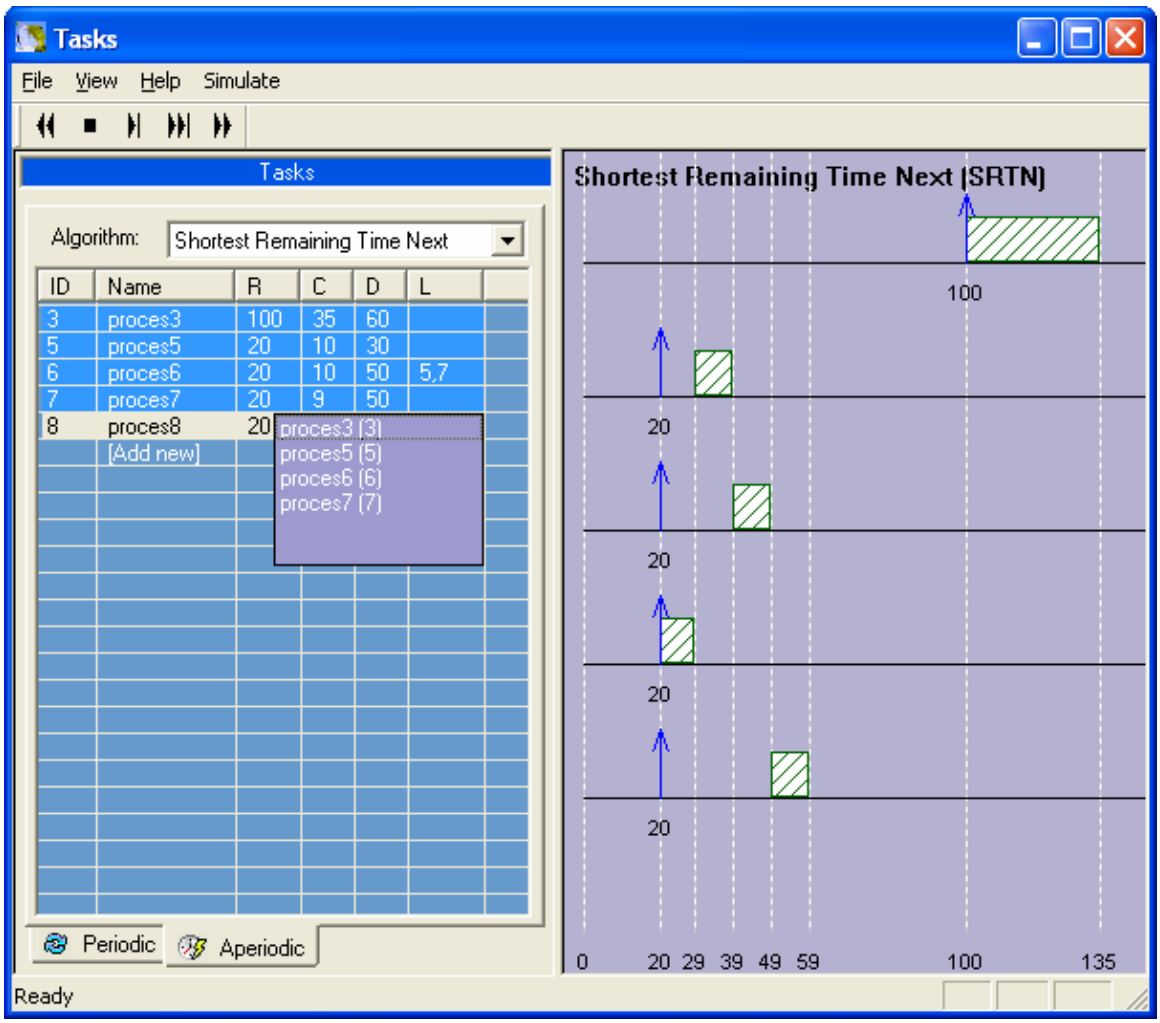

Fig. 4. Aperiodic tasks scheduled with Shortest Remaning Time next

Table 2: Periodic task requirements

ID	Name	R	C	T	D	L
1	Proces1	0	10	20	20	
2	Proces2	0	25	50	50	

Table 3: Aperiodic task requirements

ID	Name	R	C	D	L
3	Proces3	100	35	60	
5	Proces5	20	10	30	
6	Proces6	20	10	50	5,7
7	Proces7	20	9	50	
8	Proces8	20	10	60	

If, for example, the Rate Monotonic scheduling algorithm is choosen for the periodic set, by computing the response times for all tasks presented in Table 2 the results presented in Figure 2 are generated. From this, we conclude that the task set is not schedulable, and points when deadline is missed for task proces2 is marked in the simulation using a red cross. Information obtained from simulation could be used by system's designer further to optimize the system.

For example, using another scheduling algorithm, such as Earliest Deadline First, better results are obtained, the task set being schedulable, as it is presented in Figure 3. For aperiodic tasks, the task set from Table 2 is schedulable, as it is presented in Figure 4.

CONCLUSIONS

The tool that is presented in this paper could be used for modeling an analyzing several real-time systems of industrial complexity from the temporal properties point of view. By simulating the behavior of the tasks, the corectness of the system could be evaluated. We proposed a method of specifying and verifying timing properties such as presented using the above framework. The results obtained can be further used in the process of implementation, because it gives valuable information about system's behaviour under different circumstances. Consequently, the results can assist in designing better and more efficient real-time systems. The framework should be further developed and improved.

REFERENCES

Amnell, T. and Fersman, E. and Mokrusin, L. and Pettersson, P. and Yi, W. Times (2002) - a tool for modeling and implementation of embedded systems. Proceeding of TACAS'02, vol. 2280 of LNCS, pp. 460-464.

Bate, I. and Burns, A. (1999) A Framework for Scheduling in Safety-Critical Embedded Control Systems. Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications, pp.467-475.

Campos, S.V.A and Clarke, E. (1999). Analysis and Verification of Real-Time Systems Using Quantitative Symbolic Algorithms. Journal of Software Tools for Technology Transfer, pp. 260-269

Kirsch, C. M. (2002). Principles of real-Time Programming. EMSOFT 2002, LNCS2491, pp. 61-75

Kollár, J., Václavík, P., Porubän, J.: The Classification of Programming Environments. Acta Universitatis Matthiae Belii, 10, 2003, pp. 51-64, ISBN 80-8055-662-8

Korousic-Seljac, B. (1994). Task Scheduling Policies for Real Time Systems. Microprocessors and Microsystems , vol 18, nr. 9, pp. 501-511

Laplante, Ph. A. (2000). Real-Time Systems Design and Analysis – An Engineer's Handbook – Second Edition, IEEE Computer Society Press.

Liu, J. (2000). Real-Time Systems. Prentice Hall

Richard, P. (2002). A Tool for Controlling Response Time in Real-Time Systems. TOOLS 2002, LNCS 2324, pp. 339-348

Spuri, M. And Stankovic, J. A. (1994). How to Integrate Precedence Constrains and Shared Resources in real-Time Scheduling. IEEE Transactions on Computers, vol 43, no. 12, pp. 1407-1412.

Stankovic, J. and Ramamritham, K. (1998). Deadline Scheduling for Real-time systems: EDF and Related Algorithm., Kluwer Academic Publishers.

Xu, J. (2003). On Inspection and Verification of Software with Timing Requirements, IEEE Transactions on Software Engineering, vol. 29, no. 8.