COMPUTATION MODEL FOR REAL-TIME CONTROL SYSTEMS DEVELOPMENT

Doina ZMARANDA, Gianina GABOR
Department of Computer Science, Faculty of Electrical Engineering and Information Technology,
University of Oradea, Universitatii 1, 410087 Oradea, tel. +40259408769, e-mail: {zdoina, gianina}@uoradea.ro

ABSTRACT

The use of modelling graphical tools can be very convenient for development of complex real-time control systems either at the specification level or at the design level. In this paper a framework for analysis multitasking real-time control systems based on the task computational model of the system is developed. Thus, the paper defines the computational model semantics and proposes further a software environment for analyzing and validating temporal behaviour of a real-time system which covers preliminary and detailed design development steps within the software life cycle.

Keywords: *real-time control system, event triggered computational model, schedulability analysis*

1. INTRODUCTION

1.1. General issues

Real-time control systems distinguish themselves by several specific characteristics, such as: they are embedded systems, built around particularly constraint architectures and, sometimes, severely limited processing resources; they are performing a large variety of monitoring and control operations, therefore their interaction with external environment represent an important part of the system; they are not stand-alone systems, they must respond to external stimuli that may come from a large variety of devices, such as sensors, and afterwards, they must process the data and responses must be sent in order to control the overall system based on the results; they are subject to critical requirements on timeliness execution.

It is well known that, within the domain of real-time critical systems, applications predominantly deal with control and regulation based upon sensors and actuators data. Thus, significant advantages can be achieved by viewing sensors and actuators as being both physical and logical. In this respect, processing becomes sensor/actuator driven as well as base time controlled. Thus, the control program acts like a reactive program, since it maintains an ongoing interaction with its environment and responds to dynamically changing inputs by producing corresponding outputs.

A harder but related demand on safety-critical real-time systems is determinism, that is, that execution time as well as the order of execution is predetermined. It is obvious that any priority based solution which permits dynamically mixed processing order will never be able to meet stringent determinism nor time interval reproducible requirements. For example, the well known probabilistic rate monotonic approach concerning an arbitrary mix of periodic processes can only be utilized to prove predictability and not determinism.

Increasing criticality and complexity of such real-time control systems leads to a challenge for dealing with in an integrated manner. Therefore, it is important to develop practical methodologies and strategies that could result in better solutions.

Generally, they are three conceptual levels in designing real-time control applications:

- *the framework level* – this emerged from the need of separate the concerns of logical an physical architecture of the system while ensuring that the latter will be recognized during the design process
- *computational model level* – needs to identify and characterize system's components and define system's model
- *realization level* – implies identification of methods and tools that enables construction of a feasible design

The first level is generally addressed by controlled process recommended standard practices, thus being imposed generally by external environment to which the control application is connected. Sometimes this level requires more support than it is usually provided.

The second level implies definitions of all system's components properties, as a fundamental prerequisite for further timing analysis as part of the design process. This second level provides information for the third level, realization level, which was supposed to verify the feasibility of the design and afterwards to enforce correct design against both functional and timing requirements.

Several development models were developed for real-time domain, and generally, they could be grouped into two categories: models that use event-triggered approach and models that use timed-triggered approach.

Event-triggered approach is dictated by the external environment. In the event-triggered based real-time systems, all communication and processing activities are initiated whenever a significant change of state, i.e., an event other than regular event of a clock tick, is noted. The signal of significant events is realized by the well-known interrupt mechanism. The event-triggered based systems require a scheduling strategy to achieve the appropriate software task that services the event [3].

In the time-triggered systems, all communication and processing activities are initiated at predetermined points in time. There is only one interrupt and that is the periodic clock interrupt, which partitions the continuous time into sequences of equally spaced granules [3].

If the main advantage of event-driven approach is flexibility and better resource utilization, the main advantage of time-driven approach is predictability. In the timed-triggered architecture, task activation is based on relevant state changes of external environment and takes place at predetermined points of time. For typical real-time monitoring and control application this is not

ISSN 1335-8243

a significant limitation, since it basically consists of periodic tasks that are enabled or not by a well defined logic. A timed-triggered activated task has a fixed activation period and can be scheduled off line without knowledge of future requests: therefore, while for timed triggered architecture a fixed activation period for tasks could be defined, for event triggered architecture at least a minimum inter-arrival period for activated task could be assessed. A main shortcoming for timed-triggered approach is that, in order to obtain a predictable real-time application, the worst case scenario must be known a-priori in order to be considered into the development process. This is not always the case; but, if it is possible, high responsive and predictable solution could be found. Of course, low resource utilization is the price that should be paid for obtaining predictability and determinism, but, for real-time safety critical applications, achieving more control on the validation and synthesis of control software is a much more important issue.

Besides the above shortcomings, there are also some advantages for using event triggered architectures: design effort for complex systems is lower than for the timed-triggered architecture approach and event-triggered approach exhibits better resource utilization.

For event-triggered approach, static scheduling provides highly deterministic and predictable behaviour that the dynamic one, under hard real-time constraints, but, it has a major disadvantage: its applications results in closed systems that are difficult to re-configure and maintain. This is against the design principle that imposes a flexible architecture that permits further reuse and reconfiguration. Dynamic scheduling is more flexible but, for real time control systems, it results in a non-deterministic solution.

1.2. Related work

For real-time applications, response time analysis represents an important step in system's model verification process, several tools being developed in this area [17], tools that are based on specific models [15]. The fundamental issue for currently developed models is the scheduling strategy and consequently, the schedulability analysis. Several scheduling approaches have been developed in the literature to address time-critical systems by constructing a pre-runtime scheduling for hard real-time systems that includes inter-task relations [22], by applying timed multitasking in distributed real-time environments [2]. Solutions for distributed environments are mainly focused on load balancing and optimization [6][7]. Obtaining an optimal scheduler is an important point of some models, by implementing a scheduler in such a way that the total execution time is minimized [4]. But, for real-time applications, were deterministic solutions should be obtained, scheduling of arbitrary tasks sets that provide a good, not necessary optimal solution could be a correct approach. Consequently, in our model optimality is not an issue. Moreover, because for several real-time control applications interaction with outside process is crucial, we consider that precedence constraints must also included into the model. It is well known that scheduling of a set of tasks with precedence constraints and dynamic activation can be solved in polynomial complexity time only if tasks are preemptable, as presented in [4]. The basic idea of the known approaches is to transform the initial task set into another task set by modifying some of the task timing parameters, that generally consist of release times and deadlines [1][3][8]. Our paper is based partly on this approach, by focusing only on release times not on deadlines. The reason for that lies in the idea that, in specific real-time control applications, due to their close interaction with outside environment, deadlines could not always be modified.

Several implementations of such approaches exist and they are starting from the premise that, even if for a set of tasks such release time modifications lead to a feasible solution, this should be followed by deadline modifications accordingly. Generally, existing algorithms for release and deadlines times transformations ensure that if there is a feasible schedule for the modified task set, the original one is also schedulable, viceversa being not true [9]. If deadlines are not modifiable, modifying only release times could have impact on task set schedulability, consequently further complex analysis should be made regarding the impact of such modifications to tasks deadlines [24]. In this idea, the framework developed in this paper permits simulation of a real-time task set behaviour that includes precedence constraints and analyze the impact on deadlines of these modifications. By developing a complex graphical simulator, deadline miss are clearly marked and schedulability of the arbitrary set of task is evaluated.

The paper has the following structure: section 2 presents the proposed model's semantics, theoretical background used for including precedence constraints into the model and issues related to model implementation; section 3 presents a description of the developed framework; section 4 illustrates framework's behaviour using different case studies; finally, section 5 presents some conclusions and future development possibilities.

2. MODEL DEVELOPMENT

2.1. Model's semantics

Our semantic model is based on top down approach decomposition method and on the concept of module that identifies the basic processing tasks (process) from the system. It is based on the generalized event-triggered approach where all implied tasks are periodic.

The concept of process or task is particularly useful since it allows modular representation of real-time system and also provides the basis for the relationship with other units that forms system's implementation.

The obtained computational model describes the structure of proposed program implementation in terms of containing tasks and forms the so called "Logical control part" level of the system. Definition of this structure requires prior extensively analysis; but, this procedure could not be feasible if it considers only the program components. During the process of analysis, interaction with outside process must be also considered. Therefore, definition of the model should consider also the "Technological part" of the modelled real-time control system, part that consists of all machines, production lines, drives and other. It includes the sensor units and

actuators necessary for the automation process and other tools for communications, command elements for manual actions, and other such things. Further the correct validation of the control program could take place only if program (that correspond to logical control) and controlled external environment (that corresponds to the technological part of the real-time controlled system) are integrated together into the model.

Consequently, there are several issues that should be considered when establishing if an operation should be modelled in a separate task or grouped with other transformations into one task, such as:

- *I/O dependency* – since the operation that should be implemented depend on input or output and must run at the speed of the I/O device, separate tasks must be used for such operations
- *user-interface dependency* – operations that depend on user interaction should be structured into separate tasks
- *periodic execution* – transformations that are executed regularly at predetermined intervals should be designed as separate tasks to facilitate scheduling; also, independent tasks must be used for periodic I/O activities
- *time-critical functions* - activities that have stringent response time requirements should be separated into tasks that permit priority scheduling; also, non time critical computations should be assigned low priority tasks references.

Consequently, after identifying possible tasks, the model structure could be described according to Figure 1, where tasks that belong to the "Logical control" part together with tasks that are related to the external environment ("Technological part" - sensors and actuators) are identified.

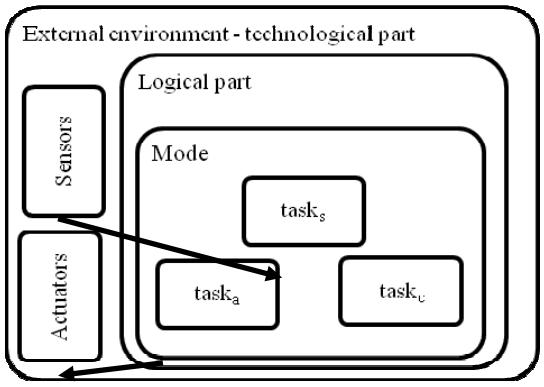

Fig. 1 Model example with different types of tasks

Sometimes, tasks could be grouped into several modes: a mode represents a set of periodically executed activities that could be included into the following categories: specific computations ($task_c$), actuators updates ($task_a$), sensors monitoring ($task_s$). If several modes are identified, tasks that are dealing with mode switches could be implemented. In the model semantics, task represents the basic element of functionality.

2.2. Model's implementation

Implementation of the model is usually described in means of tasks. Tasks are the basic units of work and the smallest entities that are treated as schedulable units by the operating system [10]. Every task could be defined as a set of parameters representing the attributes that distinguish it from other tasks.

Generally there are several types of parameters: functional parameters that describe the task's relationship with other processes which are defined in the system's specifications; temporal parameters that concern the temporal behaviour of the task; interconnection parameters that describe how the task is connected to other tasks; resource requirement parameters that describe the number and kind of resources used by the task [18].

The selection of values of these parameters depends on many factors, being generally related with the type of task: task assigned with monitoring function that will obtain input data from the system environment; tasks with a processing function based on input data captured by the monitoring tasks; tasks with a control function which receive information from processing tasks and perform some specific actions [21].

In real-time applications, temporal parameters are fundamental attributes and usually consist of the following [13]: *execution time* of task or computing time is the amount of time it takes to complete its execution; the *ready time* of a task is the earliest time at which the task is allowed to start its execution; particularly, a periodic task is one that becomes ready at constant interval of time; the *deadline time* or deadline of a task represents the latest time by which the task must finish its execution.

2.3. Integration of periodic and aperiodic tasks

When developing the task model, no assumption concerning the scheduling algorithm to be used should be made: it would be interesting to make the scheduling assessment from different point of views, for example taking into consideration both periodic and aperiodic tasks (most existing scheduling algorithms for hard real-time systems being either periodic or aperiodic, but not both [15] [13]).

In real-time systems, periodic tasks typically arise from sensor data or control loops, while aperiodic tasks generally arise from operator actions or aperiodic events. However, in practice, most real-time systems require an integrated and consistent approach, suitable for scheduling periodic tasks that have hard deadlines along with aperiodic tasks (generally represented by alerts, which require guaranteed response time). The problem of jointing during scheduling both types of tasks has several implications, and, consequently, needs a specific approach [15]: generally, the simplest approach is to construct a special periodic task, a so called polling task PT, with a relatively high priority T_{PT}, which is used to provide service to aperiodic tasks arrivals [20].

This special polling task is ready to run at the start of its period and services pending or arriving aperiodic tasks over the interval from the start of its period until C_{PT} units later (where C_{PT} represents task's computation time).

The PT is subject to preemption by higher priority tasks, until either it exhausts its execution time or there is no aperiodic work left to be executed. In the latter case, it loses any unused execution time and is unavailable to service aperiodic tasks until the start of the next period T_{PT}.

Using this approach, the PT is scheduled as if it is a periodic task with period T_{PT}. This assures a consistent approach and permits constructing a model based only on periodic tasks. The chosen priority for the PT is given relative to the other tasks priorities: the PT can execute at any priority level, but, assigning to the PT the highest priority allows one to guarantee that deadline for aperiodic alerts are met [20].

2.4. Integration of precedence relations between tasks

Taking into consideration precedence relations during scheduling arise a major problem: to check consistency between the release times of the tasks and the deadlines, within a given partial order.

Generally, the problem could be formulated in the following way: given a partial order denoted by < (on the two tasks T_i and T_j) the release times for these tasks (denoted by R_i and R_j), and the deadlines (denoted by D_i and D_j) are consistent with the partial order if [18]:

$$T_i < T_j \Rightarrow R_i \leq Rj \text{ and } D_i \leq D_j \quad (1)$$

where generally we use R_i and D_i to denote release time and deadline respectively for the task T_i.

According to the current practice, a task T_i is ready at time t and may be processed if $R_i \leq t$. Also, the release time and deadline are satisfied (the schedule being thus feasible) if the start time of the task is greater or equal to its release time and its completion time is less than or equal to its deadline. Consequently, task T_i is eligible to execution only if the current time is greater than or equal to its release time [18].

A consequence of the above relation is the idea that behind the consistency of a partial order stay an enforcement of precedent constrains by modifying release times and deadline in order to obey relation (1). If such enforcement is possible, this gives the initial premises that a scheduling algorithm will also obey to the precedence constrains imposed initially. Based on these precedence relations, a relative order between tasks should be established, if necessary by modifying releases times for tasks in accordance with the given partial order.

Implementation of the above concepts requires several steps:

- given a set of processes, characterized by deadlines, period, computation time and release time and precedence relations between them, a directed acyclic graph should be constructed to model partial order among tasks: there is an arc from node i to node j if and only if $T_i < T_j$
- sort the tasks in topological order using topologic sorting algorithm; if algorithm has no solution then the given precedence relations are not consistent and initial data about tasks must be changed
- if the algorithm has several solutions, then all possibilities will be retained and further analyzed
- verify the consistency between the release times and deadlines with the previously determined partial order, according to the assertion formulated in (1)
- if situations where relation (1) is not accomplished are identified, modify releases times and deadlines for tasks in accordance with the given partial order
- analyze the impact of the modifications through the schedulability analysis by simulation using the developed tool from paragraph 2.

The characteristic modifications should be made without affecting tasks initial timing constraints; in our approach, thus implying modification of release times with a late value, if this leads to consistency with the imposed precedence order. So, the new release time for a task T_i can be calculated by the following steps:

- if all tasks have been processed, stop; otherwise, processing the tasks should be made in direct topological order
- for the following task T_i the release time is calculated based on the release time of the previous task T_{i-1} in the above determined topological order, according to the following relation:

$$R_i = \max (R_i , R_{i-1} + C_{i-1}) \quad (2)$$

where C_{i-1} is the worse computation time for predecessor task T_{i-1}, and R_i and R_{i-1} are the release times for tasks T_i and T_{i-1} respectively.

3. TOOL STRUCTURE AND IMPLEMENTATION

The main goal of the developed software tool is to facilitate the design of multitasking real-time applications. The tool permits easy description of temporal behaviour of the application's task set based on task time characteristics that are initially given. Moreover, the tool allows the user to choose a scheduling policy and to verify the compliance of his design relatively to the imposed specifications (deadlines and precedence) by performing a simulation and by making schedulability analysis.

The main features of the analysis tool are the following: a graphical user interface for introducing tasks parameters, such as: deadline, execution time, priority; 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 (Rate Monotonic, Earliest Deadline First, Least Slack Time First, First In First Out) and non-periodic tasks (Round Robin, Shortest Job First, First Come First Served, Highest Response Ratio Next, Shortest Remaining Time Next) [11][19]. It includes a simulator that shows a graphical representation of the generated trace of execution for the set of tasks, according to the chosen scheduling algorithm. The framework structure is presented in Figure 2.

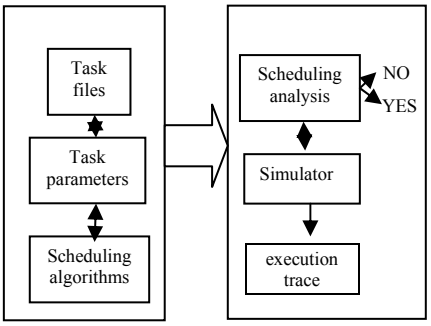

Fig. 2 Framework structure

The developed tool uses the general task specification model for describing each task T_i and the timing characteristics are given using the following parameters [12]:

- task T_i identifier ID_i and/or name (unique) N_i – defines a task uniquely;
- task T_i execution or computing time C_i – the execution time of a task can vary within an interval $[C_{Bi}, C_{wi}]$ where C_{Bi} and C_{wi} 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 modelling the timing behaviour 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 behaviour model, and results must be compared; another approach could analyze the timing behavior starting from the idea of variable execution times [23].
- task deadline D_i – is a typical task constraint in real-time systems and is 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 T_i arrives, it should finish within D_i time units [16].
- task period T_{iper} – for periodic tasks, task period T_{iper} is considered to be equal to task deadline D_i
- task ready (arrival) time R_i – represents the moment of time when the task T_i is ready for execution.
- precedence constrains - generally, in a real-time system, tasks are not running independently and 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.

The tool was implemented in Visual C++ and it is divided into two parts: a specification part and an analysis part.

In the specification part, user models the analyzed real-time system, by given all initial data about the set of tasks.

In the analysis part, the framework provides a simulator for simulation of tasks execution according to the chosen scheduling policy. Based on the simulation, the given real-time system is analyzed from the schedulability point of view [13].

The framework implements the most used scheduling algorithms, grouped into two categories: periodic and aperiodic. Between of periodic tasks, one is usually a polling task for processing aperiodic events, (as stated in chapter 2.3) system's designer should decide about its priority according to system's initial specifications. From the tool's point of view, this task is viewed in a similar way as the other tasks. This approach allows to mix periodic and aperiodic tasks during scheduling, a common requirement in real-time systems [16].

Using this simulation, the points when tasks arrive, or when they are suspended or completed could be easily observed. Also, when a task misses out its deadline, this situation is marked in the simulation view using a red cross. By simulation user can validate the dynamic behaviour 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 and a message to the user is displayed, indicating if 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, the framework could be extended 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.

4. RESULTS

The paper defines a computational model for real-time control systems development based on general event-triggered approach. Temporal behaviour of the model were simulated using a specific developed framework that permits system's temporal behaviour and schedulability analysis by simulation. In the following examples, the framework is used to carry on analysis of release time modifications impact due to precedence constrains to task deadlines.

Table 1 Task set without precedence constraints: R = task ready time; C = task computing time; T_{per} = task period; D = task deadline

ID	Name	R	C	T_{per}	D
1	Proces1 (T_1)	0	10	20	80
2	Proces2 (T_2)	0	25	50	50

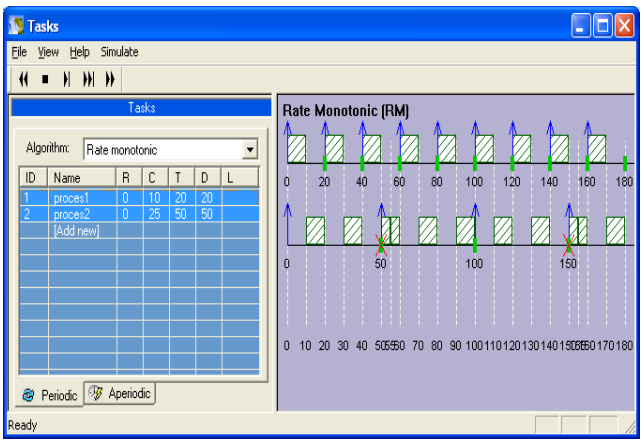

Fig. 3 Non-schedulable set of tasks

Tool's functionality was tested using different case studies, where each task is presented as a sequence of components, each with a different execution time and deadline. For example, the task set presented in Table 1 is not schedulable using the rate monotonic scheduling algorithm, and the simulator indicates the non-schedulability of the task set according to Figure 3. However, the task set is schedulable with Earliest Deadline First (EDF) scheduling algorithm [14], according to Figure 4.

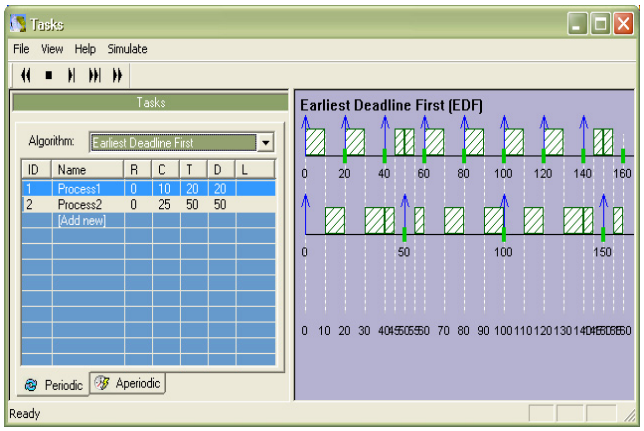

Fig. 4 Same set of tasks schedulable with EDF

Table 2 Task set with precedence constraints: R = task ready time; C = task computing time; T_{per} = task period; D = task deadline

ID	Name	R	C	T_{per}	D	Precendence constraints
1	Proces1 (T_1)	0	10	80	80	
2	Proces2 (T_2)	20	20	80	80	after Proces1
3	Proces3 (T_3)	10	10	80	80	after Proces4
4	Proces4 (T_4)	0	20	80	80	after Proces2

If we model the task set presented in Table 2, without considering precedence relationships between them, the resulted simulation is presented in Figure 5.

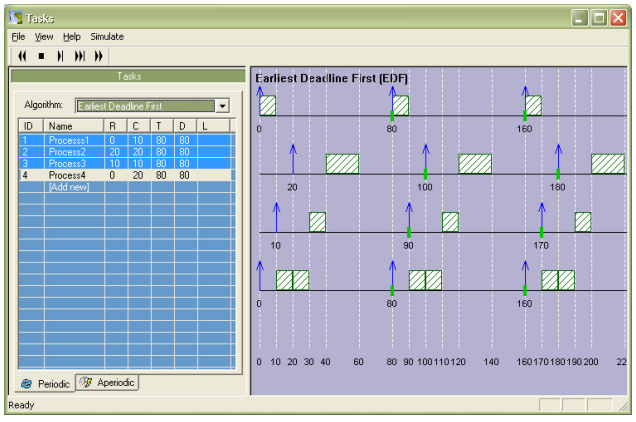

Fig. 5 Schedulable set of tasks using EDF without precedence constraints

But, when considering also precedence relationships between tasks, for example by adding some precedence constraints for the task set (as presented in Table 2), we have to enforce scheduling algorithm to take this constraints into consideration and provide a feasible scheduling, if it is possible.

According to the relation (1), we have the following:

$T_1 < T_2 \Rightarrow (R_1 = 0) \leq (R_2 = 20) \Rightarrow$ true;
$T_4 < T_3 \Rightarrow (R_4 = 0) \leq (R_3 = 10) \Rightarrow$ true;
$T_2 < T_4 \Rightarrow (R_2 = 20) \leq (R_4 = 0) \Rightarrow$ false;

Consequently, in the case of tasks T_2 and T_4 there is an inconsistency between the imposed release times and precedence requirements. So, after establishing the topological order that satisfies the imposed precedence relations (in this case, this is: T_1 , T_2 , T_4 , T_3), modification of release times should be done according to (2):

$R_2 = \max (R_2 , R_1 + C_1) = \max (20, 0+10) = 20$
$R_4 = \max (R_4 , R_2 + C_2) = \max (0, 20+20) = 40$
$R_3 = \max (R_3 , R_4 + C_4) = \max (10, 40+20) = 60$

The resulting release times according assures the imposed precedence relations between tasks. The result is illustrated in Figure 6, showing that, for this example, there is a feasible schedule that takes into consideration also the precedence constraints.

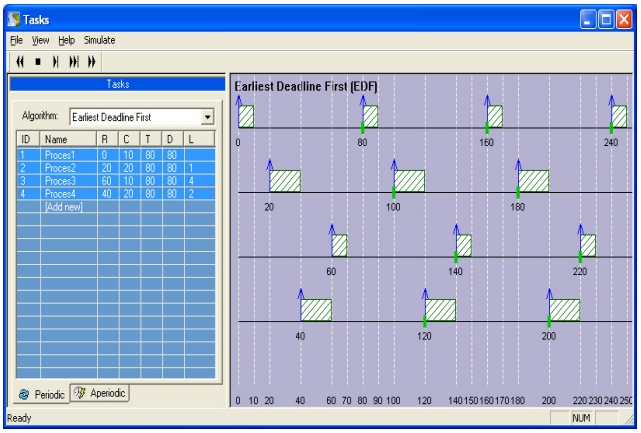

Fig. 6 Schedulable set of tasks with precedence constraints, after release time modifications

Such analysis should be done also for other scheduling algorithms; the above discussion could be extended also for deadline modifications too.

But, for this situations several additional problems arises for periodic algorithms, where task period is assumed to be equal to task period.

For the time being, the tool computes and verifies if there is a solution for the topological order problem and, if yes, signals the existing inconsistencies for releases times.

Then the release times could be modified by user manually, through the application interface. Further implementation and development of the tool is intended to automatically modify releases times for processes; this could be done by implementing the algorithm that calculates release times based on variable execution times that are given initially. Also, analysis of several scheduling possibilities, and, if possible, choosing the optimal one is intended to be implemented in the future.

5. CONCLUSIONS

When modeling real-time systems behavior, practical assumption should often be made, in order to reduce model complexity. Therefore, many models that are currently used were limited from different point of views. In this paper we present a method for development computational models for real-time control systems

together with a tool that allows implementing the model based on tasks and perform schedulability analysis. By using the results given by the tool framework, temporal behavior of modeled real-time system could be assessed prior to their actual implementation.

The framework is based on the assumption that the computational time of each constitutive task could be assessed. In practice this is not always the case; a further improvement will investigate possibilities of extending the application with analysis of impact of using variable execution times into the developed tool.

REFERENCES

[1] AMNELL, T., FERSMAN, E., MOKRUSIN, L., PETTERSSON, P., YI W., A tool for modeling and implementation of embedded systems. Proceeding of TACAS'02, vol. 2280 of LNCS, pp. 460-464, 2002

[2] ANGELOV C., BERTHING J., Distributed Timed Multitasking - A Model of Computation for Hard Real-Time Distributed Systems, IFIP International Federation for Information Processing, Volume 225, pp. 145-154, 2006

[3] BATE, I., BURNS, A., 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, 1999

[4] BUTTAZZO G., Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications, Second Edition, Springer Science +Bussiness media Inc., 419 pp, 2005

[5] CAMPOS, S.V.A., CLARKE, E., Analysis and Verification of Real-Time Systems Using Quantitative Symbolic Algorithms. Journal of Software Tools for Technology Transfer, pp. 260-269, 1999

[6] CHEJURI, C., KHANNA, S., ZHU, A., Algorithms for minimizing weighted flow time, Proceeding of ACM Symposium on Theory and Computing, 2001

[7] DU, J., LEUNG, J., Minimizing mean flow time with release time and deadline constrains, Proceedings of Real Time Systems Symposium, pp.24-32, 1998

[8] DUIN C. W., SLUIS E. V., On the complexity of adjacent resource scheduling, Journal of Scheduling, 9(1), pp. 49-62, 2006

[9] HLADIK P.E., DEPLANCHE A.M., An extention of Holistic Schedulability Analysis for Precedence Relations in Multiprocessor Systems, 15th Euromicro Work In Progress on Real-Time Systems, 2003

[10] KOLLAR, J., PORUBAN, J., VACLAVIK, P., Time and memory profile of a process functional program, Acta Polytechnica Hungarica, vol. 3, no. 2, pp.27-40, ISSN 1785-8860, 2006

[11] KOROUSIC-SELJAC, B., Task Scheduling Policies for Real Time Systems. Microprocessors and Microsystems , vol 18, nr. 9, pp. 501-511, 1994

[12] KIRCH, C., Principles of Real-Time Programming, EMSOFT02, LNCS 2491, Springer-Verlag Berlin, 2002

[13] LAPLANTE PH. A., Real-Time Systems Design and Analysis – An Engineer's Handbook – Second Edition, IEEE Computer Society Press, 2000

[14] LEHOCZY, J. P., Fixed priority scheduling of periodic task sets with arbitrary deadlines, Proceedings 11th IEEE real-Time Systems Symposium, pp. 201-209, 1990

[15] LEUNG J. T., Handbook of Scheduling: Algorithms, Models and Performance Analysis, Chapman and Hall/CRC, 2004

[16] LIU, J., Real-Time Systems. Prentice Hall, 2000

[17] RICHARD, P., A Tool for Controlling Response Time in Real-Time Systems. TOOLS 2002, LNCS 2324, pp. 339-348, 2002

[18] SPURI, M., STANKOVIC, J. A., How to Integrate Precedence Constrains and Shared Resources in real-Time Scheduling. IEEE Transactions on Computers, vol 43, no. 12, pp. 1407-1412, 1994

[19] STANKOVIC, J. AND RAMAMRITHAM, K., Deadline Scheduling for Real-time systems: EDF and Related Algorithm., Kluwer Academic Publishers, 1998

[20] STROSNIDER, J. K., LEHOCZKY, J. P., SHA L., The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard real-Time Environments, IEEE Transactions on Computers, vol. 44, no. 1, 1995

[21] SVRCEK W., MAHONEY D., YOUNG B., A Real-Time Approach to Process Control, John Wiley & Sons, 2000

[22] TAVARES E., MACIEL P., SILVA B., OLIVEIRA M.O., Hard real-time tasks scheduling considering voltage scaling, precedence and exclusion relations, Information Processing Letters, Volume 108, Issue 2, pp. 50-59, 2008

[23] XU, J., On Inspection and Verification of Software with Timing Requirements, IEEE Transactions on Software Engineering, vol. 29, no. 8, 2003

[24] WANG L., ZHAO M., ZHENG Z., WU Z., End-to-End Worst-Case Response Time Analysis for Hard Real-Time Distributed Systems, SAFECOMP: Computer safety, reliability,and security International conference N°24, Fredrikstad , Norway, vol. 3688, pp. 233-245, 2005

Received April 15, 2009, accepted August 4, 2009

BIOGRAPHIES

Doina Zmaranda was born on 14.07.1967. In 1990 she graduated (MSc) at the department of Computers Science and Automation of the Faculty of Electrical Engineering at the Politechnical University "Traian Vuia" Timisoara, Romania. She defended her PhD in the field of Computer

Science in 2001; her thesis title was "Considerations regarding design and implementation of real-time applications using Programmable Logic Controllers (PLCs)". Since 1990 she is working within the Department of Computer Science, Faculty of Electrical Engineering and Information Technology, University of Oradea, Romania. Her scientific research is focusing on real-time application development and programming. In addition, she also investigates questions related with the reliability of complex control systems.

Gianina Gabor was born on 5.01.1963. In 1986 she graduated (MSc) at the department of Automation and Computers Science of the Faculty of Electrical Engineering from the Politechnical University "Traian Vuia" Timisoara, Romania. She defended her PhD in the field of Automatic Control in 2005; her thesis title was "Contributions to the availability of control systems study with aplications for the control system of a geothermal power plant". Since 1993 she is working within the Department of Computer Science, Faculty of Electrical Engineering and Information Technology, University of Oradea, Romania. Her scientific research is focusing on reliability and availability of complex systems, control systems, modelling and simulation. She also investigates questions related with real-time systems.