**Міністерство освіти і науки України**

**Харківський національний автомобільно - дорожній університет (ХНАДУ)**

# ЗБІРНИК НАУКОВИХ ПРАЦЬ

**за матеріалами**

**Міжнародної науково-практичної Internet-конференції**

**«МОДЕЛЮВАННЯ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В НАУЦІ, ТЕХНІЦІ ТА ОСВІТІ»**

**21-22 листопада 2018 року**

**Харків, ХНАДУ 2018**

**УДК 004.415+004.94**

# MODELING SOFTWARE FOR INDUSTRY 4.0

**Lizárraga M. L.(1), Buelna A.(1), Díaz-Ramirez A.(1), Amaro-Ortega V.(1), Kostikova M. V.(2), González-Navarro F. F.(3), Werner F.(4), Burtseva L.(3)**

**(1)Tecnológico Nacional de México/IT de Mexicali, Mexico,**
**(2)Kharkiv National Automobile and Highway University, Kharkiv, Ukraine,**
**(3)Universidad Autónoma de Baja California, Mexicali, Mexico**
**(4)Otto-von-Guericke University Magdeburg, Germany**

**Abstract.** Cyber-physical systems in Industry 4.0 are comprised of a computational core, a communication core, and a physical system. The computational core has strict timing restrictions and represents a Real-time system. In this paper, the modeling of the computational core is discussed.

**Key words:** Industry 4.0, Cyber-physical system, Real-time system, Scheduling, Earliest Deadline First policy, Rate Monotonic policy.

The fourth industrial revolution, known as Industry 4.0 (I40), refers to a set of technologies aimed to create the concept of smart factories. A Cyber-physical system (CPS) is considered to be a core technology for I40. CPSs are smart networked systems with embedded sensors, processors and actuators, which are designed to sense and interact with the physical world (including the human users). They support real-time and guarantee performance in safety-critical applications. One of the main features of a CPS is that it has strict timing restrictions, which must be satisfied, since otherwise, the results may be catastrophic. A system, which requires a complete assignment of the resources and provides functioning in a timely manner, is referred to as a Real-time system (RTS). I40 environment involves modeling software for the computational core of a CPS, which is in fact an RTS. The design of such systems permits an opportune response and the execution of a task within predefined time constraints. From a functional viewpoint, an RTS is a computer system, which is dedicated to monitoring of a process or to control the tasks.

The RTSs implemented in traditional hardware are employed in a wide range of applications. They are embedded into devices of common use, which are known nowadays as the Internet of Things (IoT), as well as into communication devices, such as mobile phones and computers. These systems have been a subject of great interest due to the trend towards the automation of quotidian use systems and applications. Examples are self-driving cars, autonomous airplanes, sensors and robots to care the elderly, to mention a few.

An RTS is conformed of a set of applications that request access to processors and resources and is commonly comprised of tasks, or threads, where each task is subjected to a series of temporary restrictions. The number of processors limits the maximum number of tasks that can be executed simultaneously on a computer. In a monoprocessor system, only one task can be executed at a time; any other task has to wait until the processor is free and can be re-scheduled. Therefore, it is necessary to define which task has to be next at each time instant on each processor. The scheduling algorithms for this assignment are also called planning policies.

In the planning context, a task in an RTS represents a set of related jobs that provide some functions of a system. Every job in a task is released periodically, sporadically, or aperiodically. In the periodic task model, the arrival of the jobs of a task is strictly periodic, separated by a fixed interval of time, called period. Among the RTS models, the most popular is the periodic task model.

The majority of the restrictions imposed to an RTS are expressed by the task release times, execution times, and most importantly, by the deadlines. In an RTS, the maximum possible time necessary to obtain a response must be less than or equal to the task deadline. The value that the task contributes to the system depends on the fulfillment of the deadline, starting at the activation moment. The deadlines can be classified into: soft deadlines and hard deadlines. Accordingly to the task's deadlines, an RTS can be: 1) soft RTS: missing a task deadline produces a performance degradation, but the tasks may continue the execution; 2) hard RTS: missing a task deadline is not acceptable due to possible catastrophic consequences.

An RTS is executed under a Real-time operating system (RTOS), and a specific Application Programming Interface (API). An RTOS is integrated into several modules that together allow the applications to interact with the hardware, is in the same manner as a general-purpose operating system. An RTS must be designed in such a manner that the accomplishment of the timing restrictions of the tasks that comprise the system are assured by the selection and the use of an RTOS and real-time scheduling algorithms. The correct choice of an RTOS is a fundamental aspect in the design of an RTS.

One of the most popular general purpose operating systems is Linux, which is employed by the academic community and companies for the development and execution of applications in diverse fields, such as control, computation, health, military and space, to name a few. However, Linux scheduling policies offer some level of timing guarantees for the soft RTSs, but they are not sufficient for systems with hard real-time constraints. To use Linux as a base for an RTOS, some modification to its kernel must be made.

Generally, there exist two approaches to allow Linux to provide a hard real-time support: 1) at the hypervisor level and 2) at the OS scheduler level. The first approach allows the coexistence of both, an RTOS and a generic OS, where the first one has a higher priority comparing with a non-RTOS. In contrast, in the OS scheduler level, the real-time capabilities are provided using multiple scheduling classes, where each class has several scheduling policies, being a real-time scheduling class of the highest priority. This approach is employed by several projects, both as commercial and as open source ones. One of them is the RT-Preempt, which includes free open source patches.

Almost all existing RTOSs provide two priority-based scheduling policies: First-In First-Out (FIFO) and Round Robin (RR). Liu and Layland [1] introduced the RM and EDF policies for scheduling periodic tasks in hard RTSs. The RM algorithm assigns priorities inversely proportional to the task periods; it is an optimal static priority assignment policy. It can be mapped through the FIFO policy. The EDF policy assigns the highest priority to the job with the earliest deadline; it is an optimal dynamic priority assignment policy. The EDF algorithm is not available in most of the existing hard RTOSs. However, EDF gets a better utilization of the available processor capacity, which allows the execution of more tasks in the same processor.

An RTOS provides three important functions to attend the tasks: 1) scheduling, 2) dispatching, and 3) intercommunication and synchronization. To model and implement an RTS, the Portable Operating System Interface (POSIX) is commonly used. It offers many advantages, such as portability and standardization of the application development. It is aimed for software compatibility between UNIX-like OSs, such as Linux, defining an API.

The modeling of software for hard RTS (i. e., for I40) requires that the designer defines the parameters of the system task: deadlines, periods, worst-execution times, among others. Also, a scheduling policy must be selected, and a feasibility tests must be applied, in order to verify that every temporal restriction is satisfied. Application of a feasibility test is crucial to know whether the schedule provided by the selected policy is feasible. A schedule is feasible if it respects the deadline of all jobs. An optimal scheduling policy always generates a feasible schedule for a hard RTS. This problem is commonly known as the feasibility test. Baruah et al. [2] proved that the feasibility test in uniprocessor RTSs is a co-NP-complete problem in the strong sense for non-trivial computational models.

Liu and Layland [1] demonstrated that RM is an optimal policy for the static priority assignment. The authors showed that the RM policy is able to schedule any periodic task set τ with implicit deadlines (periods are equal to the deadlines) if the total utilization of the processor satisfies $U_\tau \leq \ln 2 \approx 0.6931$, where $U_\tau$ is given as follows: $U_\tau = \sum_{i=1}^{n} C_i / T_i$ . Here $C_i$ is the execution time of task $i$ in the worst-case (WCET); $T_i$ is the size of the activation period.

A necessary and sufficient (exact) test for the RM policy was proposed by Lehoczky et al. [3]. It considers the processor utilization by the periodic task set as a function of time at a critical instant. Liu and Layland [1] introduced an exact test of the EDF feasibility for any periodic task set and proved the optimality of the EDF dynamic algorithm for uniprocessor architectures. A periodic task set is schedulable under the EDF policy if and only if $U_\tau \leq 1$.

A prototype of a controlled system was designed to illustrate a CPS system. This system was composed of three threads. It considers a periodic task (thread) that controls the appearance of any external object within a predefined range, and two independent tasks that perform calculations in the background. Such a system can be interpreted as a mobile robot, which checks periodically whether an object makes obstacles in his trajectory. If an object is detected, it turns an LED on immediately. Conversely, when the object is no longer detected, the LED is turned off. These actions are equivalent to the re-adjusting of the robot's trajectory, within a specific timing window, to avoid a collision. The exact schedulability test of Lehoczky et al. [3] was used to verify the schedulability of the task set using the RM scheduling policy. The test showed that one of the three threads was not capable to reach the execution time before the corresponding deadline even in the first activation. However, the value of the total processor utilization implicated that the task set can be still correctly scheduled using the EDF policy.

The next test was performed under the EDF policy. It showed that: 1) all the task's jobs were fulfilled by the respective deadlines; 2) the non-real-time tasks, which were grouped in Idle, were processed only when the real-time tasks did not require to be executed; 3) the schedule is similar in two investigated hyperperiods.

Some metrics obtained for the first hyperperiod displayed that: 1) The required time to switch between two tasks (context switch) with EDF is larger than with RM; 2) Each time a task needs to be activated, EDF uses more time than RM; 3) There is a double number of preemption points using the RM policy. Nevertheless, despite the FIFO policy requires less time to accomplish context switches, the times of the EDF algorithm are minimized during the execution because it does not produce such a quantity of preemptions.

As future work, we plan to implement the RTS using an embedded system as the hardware platform.

## References

1. Liu C. L. Scheduling algorithms for multi-programming in a hard-real-time environment / C. L. Liu, J. W. Layland // Journal of the ACM. – 1973. – Vol. 20. – No 1. – pp. 46 – 61.

2. Baruah S. K. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor / S. K. Baruah, L. E. Rosier, R. R. Howell // Real-Time Systems. – 1990. – Vol. 2. – No 4. – pp. 301 – 324.

3. Lehoczky J. The Rate Monotonic scheduling algorithm: exact characterization and average case behavior / J. Lehoczky, L. Sha, Y. Ding // Proceedings of the Symposium on Real Time Systems, Santa Monica, CA, USA. – 1989. – pp. 166 – 171.