Technical Report

SPARTS: Simulator for Power Aware and Real-Time Systems

Borislav Nikolic

Muhammad Ali Awan

Stefan M. Petters

HURRAY-TR-111101

Version:

Date: 11-07-2011

SPARTS: Simulator for Power Aware and Real-Time Systems

Borislav Nikolic, Muhammad Ali Awan, Stefan M. Petters

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

Real-time systems demand guaranteed and predictable run-time behaviour in order to ensure that no task has missed its deadline. Over the years we are witnessingan ever increasing demand for functionality enhancements in the embedded real-time systems. Along with the functionalities, the design itself grows more complex. Posed constraints, such as energy consumption, time, and space bounds, also require attention and proper handling. Additionally, efficient scheduling algorithms, as proven through analyses and simulations, often impose requirements that have significant run-time cost, specially in the context of multi-core systems. In order to further investigate the behaviour of such systems to quantify and compare these overheads involved, we have developed the SPARTS, a simulator of a generic embedded real-time device. The tasks in the simulator are described by externally visible parameters (e.g. minimum inter-arrival, sporadicity, WCET, BCET, etc.), rather than the code of the tasks. While our current implementation is primarily focused on our immediate needs in the area of power-aware scheduling, it is designed to be extensible to accommodate different task properties, scheduling algorithms and/or hardware models for the application in wide variety of simulations. The source code of the SPARTS is available for download at our website.

SPARTS: Simulator for Power Aware and Real-Time Systems

Borislav Nikolić Muhammad Ali Awan Stefan M. Petters
CISTER Research Unit, ISEP-IPP, Porto, Portugal
borni,maan,smp@isep.ipp.pt

***Abstract*—Real-time systems demand guaranteed and predictable run-time behaviour in order to ensure that no task has missed its deadline. Over the years we are witnessing an ever increasing demand for functionality enhancements in the embedded real-time systems. Along with the functionalities, the design itself grows more complex. Posed constraints, such as energy consumption, time, and space bounds, also require attention and proper handling. Additionally, efficient scheduling algorithms, as proven through analyses and simulations, often impose requirements that have significant run-time cost, specially in the context of multi-core systems.**

In order to further investigate the behaviour of such systems to quantify and compare these overheads involved, we have developed the SPARTS, a simulator of a generic embedded real-time device. The tasks in the simulator are described by externally visible parameters (e.g. minimum inter-arrival, sporadicity, WCET, BCET, etc.), rather than the code of the tasks. While our current implementation is primarily focused on our immediate needs in the area of power-aware scheduling, it is designed to be extensible to accommodate different task properties, scheduling algorithms and/or hardware models for the application in wide variety of simulations. The source code of the SPARTS is available for download at [1].

I. Introduction

Nowadays, there is an increasing need for embedded systems. Besides the demand for enhancing the functionalities, these systems also have to accommodate even more complex mechanisms of interaction with the environment. The most obvious examples are avionics, automotive electronics and mobile phones.

Some of these devices have to guarantee results within a given time. Such systems, which have to fulfil posed constraints, are called real-time systems. In these systems, the correctness of execution presents only one component of the correct operation, another one is the actual time when the result of the computation is available.

On top of these issues, real-time embedded systems often have limited or intermittent power supply. There are several possibilities to optimise the system consumption of the system. One is to use Dynamic Voltage and Frequency Scaling (DVFS), another is to use sleep states. This means that in the analysis of such systems, power/energy management must be properly considered.

This work was supported by the RePoMuC project, ref. FCOMP-01-0124-FEDER-015050, funded by FEDER funds through COMPETE (POFC - Operational Programme 'Thematic Factors of Competitiveness) and by National Funds (PT) through FCT - Portuguese Foundation for Science and Technology, and the RECOMP project, funded through the FCT under grant ref. ARTEMIS/0202/2009, as well as by the ARTEMIS Joint Undertaking, under grant agreement Nr. 10202.

Functionality enhancements in the applications for embedded real-time systems not only require changes within the software, but also pose significant requirements regarding needed hardware capacities. Also, in some cases it is needed to provide the integration of several applications with different criticality levels into one device. For instance, in modern cars safety critical applications are integrated with comfort functionality. One solution for arising problem is the introduction of multi-core devices in this domain.

In order to provide efficient execution of such applications and their tasks, specific algorithms for scheduling are needed. They are very diverse in terms of requirements and produced results. Given those, there is the question of evaluating the scheduling approaches and identifying the trade-offs involved in employing one strategy over another.

One approach in tackling this problem is to build the system model. It gives the possibility to hide unnecessary and negligible details, so only aspects of interest can be implemented and tested. This method is very fast and can provide the comparison between different scheduling approaches. SPARTS is a simulator that can model aforementioned different system behaviour aspects.

The contributions of this paper are as follows. We present the SPARTS - a slot based execution environment. The tool itself provides extensive flexibility in task-set generation for different scenarios and purposes. The task-sets can be used for schedulability tests as well as simulation purposes. The modular structure of the SPARTS allows easy development and integration of new scheduling algorithms for both, single and multi-core systems. The results of the simulations give indications about the performance and various overheads incurred by different schedulers (pre-emptions, energy consumptions, migrations for multi-cores, etc.) and can be used as a filter before exhaustive and exact analysis on a real system is performed. The SPARTS can be extended and adapted to fit the needs of the user in the area of interest.

The SPARTS performs the simulation in event-driven manner. Rather than doing cycle-step execution, the SPARTS works by looking backward into the interval between two consecutive job releases and calculates the execution without unnecessary cycle-level granularity. With this approach we save the computation and yet provide "correct" execution modelling. This allows to perform the simulations of large task-sets for long periods of time with high temporal efficiency.

The rest of the paper is organised as follows. Next section summarises related work. In Section III we introduce SPARTS and describe its modules. Section IV describes the software

architecture of the SPARTS with the focus on the extensibility and provides the measurements. Finally, Section V concludes the paper with the list of future-work activities.

II. Related Work

The work of De Vroey et al. [2] presents not only the simulation tool, but also the language for implementation of new scheduling mechanisms. The generic structure of the language requires significant effort to implement scheduling algorithms and also to extend the language to incorporate additional features, such as work-conserving algorithms. The tool is focused only on scheduling policies and is computationally very demanding, which brings constraints on both, simulation time and number of tasks that comprise the task-set.

Diaz et al. [3] proposed Realtss - a Simulator focused only on scheduling part. It incorporates several most common scheduling policies and includes the mechanism for resource sharing. However, the simulator has cumbersome mechanism for the specification of the task-sets into the system. The tool is only focused on periodic tasks and fails to cover sporadicity issues. Despite the fact that simulator allows easy implementation of new scheduling algorithms, it does not provide the mechanism for implementing ones that consider non-work-conserving behaviour (procrastination, sleep states etc.).

Kramp et al. [4] present a framework for implementation of new scheduling algorithms for the multi-cores. The framework executes in time-driven manner, which is computationally very expensive, even for task-sets with a small number of tasks: consulting scheduler, dispatcher and refreshing the ready queue on every tick of the CPUs in the simulation process.

The Cheddar developed by Singhoff et al. [5] consists of both, feasibility analyses and simulation engine. There are several feasibility analyses, such as analytically derived sufficient tests and observations of worst-case response-times for the tasks. However, these mechanisms are available only for small number of scheduling algorithms and cannot be applied for any newly devised ones. Also, this tool accommodates tasks with arbitrary deadlines but works only with periodic task-sets. The implementation of new scheduling algorithms requires significant effort. The simulation execution is computationally demanding.

Urunuela et al. in [6] give a description of multi-core simulation tool for scheduling evaluation called STORM. It comprises multiple types of tasks. Precedence constraints can be specified. Also, the STORM measures various characteristics apart from scheduling, such as energy consumption, CPU load, etc. However, as all previously described tools, this one also executes the simulation in time-driven manner, which is computationally demanding.

SymTA/S [7] presents a package specialised for scheduling analysis and optimizations applied in many areas, such as electronic control units, buses / networks and complete embedded real-time systems. Unlike SymTA/S which is generic, RapiTime [8] targets real-time, embedded applications. It collects execution traces to provide execution time measurement statistics and to allow off-line browsing through the execution history for debugging and optimisation purposes. It also helps in determining the worst-case execution time, which can be further used for optimisation purposes. However, both tools are commercial, so source code availability and eventual extensibility for academic purposes is questionable.

III. SPARTS

We firstly describe one concrete system model by which we adopted the simulator for our own research purposes. However, our approach with SPARTS is more general and allows accommodations of various system models for different purposes, where some existing parameters can be omitted due to irrelevance to the current cause, while some other, which we didn't introduce here, can be integrated.

A. System Model

The system Γ is represented as $\langle \tau, \alpha, \mu \rangle$. Where τ is task set, α is the scheduling algorithm and μ represents the number of CPU-s. The τ is defined as $\langle U, \ell \rangle$, where U is total utilization, while ℓ represents the number of the tasks. Single task is represented as τ_i and is described by $\langle T_i, D_i, C_i, \Theta_i \rangle$, which relate to minimum inter-arrival time, relative deadline, WCET, and task type respectively. Besides these parameters, which apply for every scheduling algorithm and form a generic base, we also implemented some of the features that surface only in one specific group of schedulers. For example, by A_i we denote the property called the *budget*, which is used in reservation-based frameworks. Also, for our own simulation purposes we assumed and used implicit deadlines for the tasks, precisely, both scheduling algorithms that we implemented and tested require task-sets with $D = T$ feature. However, SPARTS also accommodates tasks with arbitrary deadlines.

Each job is represented as $j_{i,m}$ and is described by $\langle r_{i,m}, d_{i,m}, \hat{c}_{i,m} \rangle$, corresponding to release time, absolute deadline, and actual execution time respectively. Apart from these common, scheduler-generic properties, jobs also contain the specific ones. One example is $a_{i,m}$, which presents corresponding remaining budget and is specific for the reservation based framework. Maximum sporadic delay and best-case execution time limits are represented as Δ and C^b and are fractions of minimum inter-arrival time and WCET, respectively. The presence of the budget property allow easy integration of reservation-based scheduling frameworks [9].

B. General Overview

Figure 1 summarises the general SPARTS structure. The Simulator is divided into four different parts: 1) Task-set Generator (TSG), 2) Job Generator (JG), 3) Job Sequencer (JS) and 4) Execution Environment (EE). The input parameters are delivered to the TSG, which creates the task-set. The generated task-set is passed into the JG and job instances for desired simulation time are produced. The JS orders the jobs by their release times and prepares a stream for the execution. Then, the EE executes the stream of jobs and collects required parameters for the reporting tool to do further analyses.

C. The Task-Set Generator

The basic purpose of the TSG is to give user an ability to craft different task-sets. A set of abstract parameters (total utilisation, distribution of tasks according to task types, minimum inter-arrival time ranges, randomisation level, etc.)

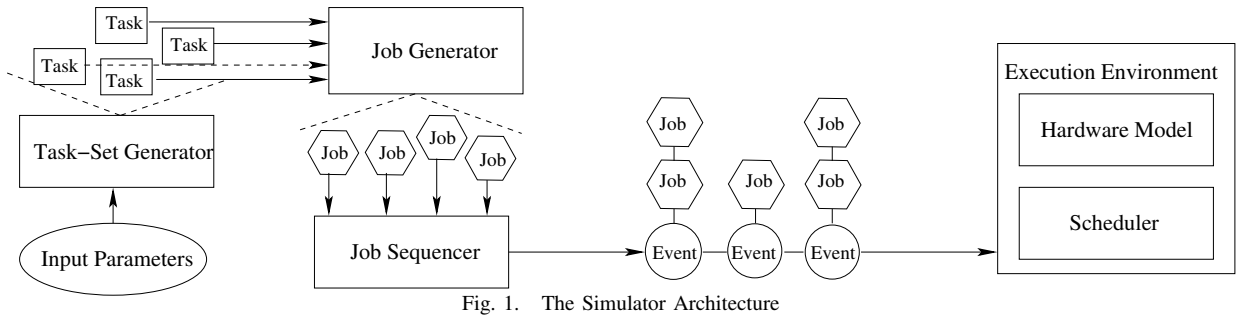

Fig. 1. The Simulator Architecture

form the global values for the task-set and are used to create concrete tasks. If greater control of task-set generation process is needed, fine-tuned creation is allowed through injection of concrete parameters into generating tasks (deadline, execution time, minimum inter-arrival, task type, etc.), as opposed to allowing the TSG to produce all the parameters by applying selected randomisation level. The purpose of use defines which of these two options is more convenient. Industry standard XML files are used to input the parameters for both of them. In case of the abstract input mechanism, the following tasks parameters are taken into consideration.

Number of tasks n is self explanatory.

System Utilisation presents the target utilisation of the task-set that will be generated as the output of this module.

Task Break Down allows distinction among tasks according to their needs or purposes. As explained in Section I, sometimes tasks of different types have to be incorporated into one task-set. This possibility allows separation and different treatment of those tasks by the scheduler. One such scheduling algorithm that handles different task types is Rate-Based Earliest Deadline (RBED) [9]. For our own simulation purposes, which focused on energy consumption properties of scheduling, we assumed one task break down option and it aligns with the classification of the aforementioned scheduler (hard real time (HRT), soft real time (SRT) and best effort (BE) tasks). However, neither the number of task types, nor their particular use are limited and only negligible effort is required to extend the existing mechanism for other uses/divisions. Thus, one possible breakdown is $type \in HRT, SRT, BE$. The total number of tasks and system utilisation are broken down between those types. For example, a break-down of $\{10, 20, 70\}$ would allocate 10% of the total number of tasks to HRT, 20% to SRT and 70% BE tasks. Similar break down is possible for the total utilisation. Each task is assigned individual utilisation U_i from the respective utilisation share of its type, where $\sum_{i \in HRT} U_i = U_{HRT}$. The same follows for the SRT and BE tasks. Nevertheless, $U_{HRT} + U_{SRT} + U_{BE} = U$.

Minimum Inter-Arrival TimeBounds presents the interval between the minimal and the maximal value for the inter-arrival time of a task. Again, this is scheduler specific feature by which a different treatment of task classes can be specified, depending on the needs of the user. This interval is defined to restrict the minimum inter-arrival time of the specific tasks within the specified interval. For example, we have chosen that BE tasks have longer period/minimum inter-arrival time than RT tasks. For each task a random number from given interval is selected and used as its minimum inter-arrival time.

Individual Task Utilisation U_i is calculated based on the following reasoning.

$$U_{indvar} \leq U_{rel} \leq 1 \tag{1}$$

$$U_{reltot} = \sum U_{rel} \tag{2}$$

$$U_i = \frac{U_{rel}}{U_{reltot}} \times U_{type} \tag{3}$$

U_{indvar} is a user defined non-negative variable less than or equal to 1. This variable allows the variances level in the individual utilisation of the tasks within the task type and hence controls the randomisation level. In order to equally distribute utilisation of any type among its tasks (no randomisation), set $U_{indvar} = 1$, which means $\forall i \in type, U_i = \frac{U_{type}}{n}$. With the minimum inter-arrival time T_i and the task utilisation of U_i the worst-case execution time is calculated as $C_i = U_i * T_i$.

Maximum Sporadic Delay Limit Δ places a limit for the sporadic delay for all tasks. This is expressed as a fraction of the minimum inter-arrival time. Each task is randomly assigned a sporadic delay Δ_i in the interval $[0; \Delta * T_i]$. Setting $\Delta = 0$ creates strictly periodic system.

BCET Limit f^b expresses the minimum best-case execution time for all tasks and is expressed as a fraction of the WCET C_i. Each task is assigned an individual BCET f_i^b in the interval $[f^b * C_i; C_i]$, where f^b gets the value between 0 and 1. Setting f^b to 1 means that all executions will be performed with WCET.

Similar to listed common parameters, additional, scheduling specific parameters can be specified as well. For example, reservation-based scheduling algorithms require budget dedicated to each task based on its type. Expected slack and/or borrowing intensity from future for each task are also configurable properties.

The automatic generation of task-sets with configurable randomisation level is helpful for testing the large number of task-sets that vary in the parameters of interest. However, in many cases it is desirable to test concrete, given task-sets. These might be corner cases, or the scenarios which reflect some real system. For this purpose the TSG provides an interface to load pre-defined concrete task-sets and forward that to the JG.

D. Jobs Generation

As previously described, the TSG is only concerned with creating a task-set, with a number of characteristics limiting, but not fully describing the individual characteristics of the jobs created from that particular task. In real systems, not only tasks vary in their requirements, the jobs of the same task also vary in behaviour depending on external system state and input

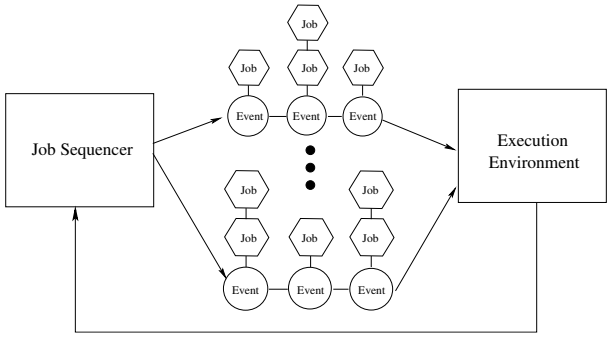

Fig. 2. The Job Sequencer output

parameters. We implemented this feature by varying the actual execution times and the sporadic delays of each job prior to their release, within the interval specified for each task. For that purpose, BCET limit f_i^b and sporadic delay limit Δ_i of the tasks are exploited.

The actual inter-arrival time of two successive jobs is generated using a random value in the interval $[0, \Delta_i]$ and adding it to the minimum inter-arrival time T_i of the task. Similarly, the actual execution time of a job is derived from the interval $[f_i^b * C_i, C_i]$. An obvious future extension would be to allow the use of execution-time profiles and sporadic-delay profiles in the process of job generation. Sporadic system model is shaped by the maximum sporadic delay limit Δ. As previously mentioned, by setting the value of $\Delta = 0$ we can create strictly periodic behaviour. There is also a combined approach, where with $\Delta = \Delta'$ only for the first job instance of a task, and $\Delta = 0$ for the rest we simulate periodic behaviour with specific initial offsets. Similarly, varying execution time of the jobs can be restricted to WCET by setting $f^b = 1$. In this case, the randomisation of the actual execution times is omitted. Analyses with this approach are very common in safety critical real-time systems.

Since SPARTS is event-driven and not tied to the cycles, it gives the possibility to choose the granularity of the system resolution. Logically, each job release is considered as a separate *event*. As we use a discrete notion of time, simultaneous releases are not uncommon. Therefore, jobs released at the same time are considered as different events with zero time difference. Although, the simulation process is optimised by merging the events with the identical temporal characteristics into one. For implementation purposes we decided that event merely presents a carrier of time, on top of which jobs with the same release time are concatenated. The reason for creating the events only for job releases and not for all the activities within the system (e.g. deadlines, pre-emptions) is twofold. Firstly, event streams produced in this way present the ordering of activities within the task-set, independent of concrete scheduler. With this in mind, the same event stream can be forwarded to different scheduler instances within the system for parallel execution. Secondly, event stream generation is recognised as the performance bottleneck of the simulation process. For efficiency purposes it is beneficial to split the process into two stages; in the event generation stage the events (job releases) are created and in the execution stage, the other scheduler specific system activities, such as deadlines and the pre-emptions, are injected.

E. Job Sequencer

The JS is the module that receives that jobs from the JG and arranges them in timely manner. Basically, it produces the *event stream* for the EE. The event stream may have some dependence on scheduling algorithm. For instance, for partitioned multi-core scheduling approach, multiple event streams will be produced for different cores, as shown in Figure 2. One the other hand, for single core and multi-core global scheduling algorithms, a single event stream will be created and utilised.

The event stream is generated for the specified time duration. Sometimes, the simulation time is long enough to be very demanding in both, computation and memory resources. Moreover, the behaviour of the tasks can also be very demanding in a very short time interval. To resolve this issue and assure good performance, the simulation time can be broken into smaller pieces, which would be sequentially introduced to the EE on demand. We call that time window a *horizon*. During specified horizon, jobs are generated for every task. Also, for every task the information are kept about the offset of the first next job instance that is out of the current horizon. When the events are generated for next horizon, these information are used. With this approach we maintain the flow of the relative offsets between the jobs of the same task belonging to different horizons.

While the JG produced the jobs with the relative time offsets, the JS allows injections of absolute offsets. This means that with multiple simultaneous releases, a critical instants can easily be generated (initial offset equals 0 for every task). The later allows, for instance, to analyse the longest busy period in the Earliest Deadline First (EDF) algorithm. If one is interested in observing the average behaviour, a random offset can be achieved by discarding the first job release of every task.

F. Execution Environment

EE consists of two mutually interacting modules, the Scheduler and the Execution Engine. The Scheduler consults selected scheduling algorithm and follows its policies. It manipulates the ready queue, selects next job for execution and, if appropriate, performs power management instructing the hardware to perform state changes. These decisions are passed to the Execution Engine through the method invocations. For instance, the next executing job is forwarded along with the command to execute it. Other possibility is to issue a command for going into particular sleep state (for the scheduling algorithms which utilise sleep states). Also, along with the commands, the duration of that particular command may be specified. We implemented this feature with the *Timers* mechanism. Timers present possible interrupts of the execution process, when something that might change the execution process happens, such as new arrivals, expirations of the deadlines, completions of the executions. Timers are set by the Scheduler. The Execution Engine simulates real execution by calculating elapsed time until the first timer would fire, decreasing the outstanding execution times of the jobs, going into or out of the sleep states, etc. Then the control is passed back to the Scheduler to decide future steps (e.g. select new job for execution (if any), set new timers) and gives the control back to the Execution Engine.

Additionally, during the execution process, if appropriate, the power model is consulted. As mentioned, current implementation includes sleep states which are utilised by both LC-EDF [10] and ERTH [11] schedulers but also many other power aware scheduling mechanisms which might be implemented. The power model provides the information about the energy consumption in the state which execution unit currently utilises. It calculates the overhead of transitions, such as going from active to sleep state or even switching between different sleep states, by taking into account power properties of each state. In order to completely cover power-aware scheduling area, we plan to extend the model to also embrace the DVFS methodology. This model of execution allows the incorporation of overheads of many types. Beyond power-related aspects, like the integration of the overheads of inter-process communication, operating system, and task dependencies are intended.

The advantage of the event driven manner of the EE is that it's not required to keep the track of the execution and the events simultaneously and thus enables a very clean and well performing implementation. Beyond the current bandwidth server implementations, other schedulers like FP can be easily implemented and are planned for future work to compare fixed-priority (FP) with the rate-based environment. Brief description of that process is given in Section IV-B.

The simulation of elapsed time since the last event allows easy extraction of various performance parameters such as number of pre-emptions, energy consumption, memory accesses, resource sharing etc. The EE can easily be extended for multi-core scheduling algorithms. For partitioned multi-core algorithms, multiple entities of execution engines can be instantiated to represent each core in parallel, where each instantiation has its own event stream to execute. However, in case of global multi-core scheduling algorithm, there exists one single event stream and therefore one ready queue from which every core selects the task for the execution.

G. Repeatability and Randomisation

In SPARTS we paid close attention to allow both, repeatability and randomisation. Existing random number generators are platform dependent (the same system call return different values in different operating systems). In order to allow platform independency, consistency of the results and repeatability, we implemented self-contained pseudo-random number generator with the uniform distribution [12] that is based on *Multiplicative Congruential Method*. Our current implementation uses uniform distribution, however, we plan to extend the simulator in order to utilise different distributions and use them where convenient for more realistic descriptions of different stochastic processes in the system (sporadicity, actual execution times, aperiodic arrivals, etc).

We implemented a two level randomisation approach for generating task and job parameters. With this we simulated the variances in the minimum inter-arrival time and the execution time. For instance, to derive the execution time, each task τ_i gets a random number y, selected within an initial range of $[C^b, 1] * C_i$. This randomly selected number y creates another range $[y, 1] * C_i$ to select the actual execution time $\hat{C}$ for each job $j_{i,m}$ of the task τ_i. Thus, tasks differ in the amount of variation of the execution times for individual jobs of a task. This approach gives the possibility to effectively control the influence of the randomisation on the simulation process and in the same time allows the reproduction of the same results with the same parameters, which is of paramount importance in post-simulation analysis.

IV. Use and Application

A. Input and Output Mechanisms

The simulation itself requires many input parameters. Our approach was to use *Convention over Configuration*. This means that all the parameters in the system already have default values, which can be explicitly overridden by the user. Some of the default parameters are located into the system XML files, while the other are embedded into the code. The users can change the parameters by accessing exposed XML files. At the start up, the SPARTS checks whether user altered some parameters and if so, uses them during the execution. In similar manner, the system provides output results in readable form through the CSV format.

B. Architecture and Extensibility

The simulator is developed in the *Java* programming language in the *Eclipse Framework*. Aforementioned parts (TSG, JG, JS, EE) are all different software modules that are communicating through defined interfaces. The purpose for introduction of the loose coupling mechanism was the assumption that some users may be interested in enhancing the functionality of one part of the system without any knowledge about the rest. This approach separates the responsibilities among the modules, so any potential extensions, in most cases, require the alteration of only one module.

C. Interfaces and Modularity

The modular design of the SPARTS also allows independent usage of its parts for different purposes. For that reason we provide brief description of the interfaces between the modules and potential separated use. This modular break-down allows easy invocation of desired scenarios for testing purposes, such as corner cases. Therefore, SPARTS not only provides the possibility to test the average case with the average task-set like the majority of the simulators do, but also to further investigate the behaviour of different scheduling mechanisms slightly below, above, or exactly at the threshold.

TSG provides the possibility to create large amounts of task-sets for substantial testing purposes. In this way, the TSG is used as a single module which is fed with the task-system properties. As an output the module generates and provides the task-sets. Populated objects can be passed to another application for analyses or can be saved into a file. These task-sets can be used for deriving a comparison and analysis of the acceptance ratios for different scheduling tests.

JG & JS can also act as stand-alone parts and can be initialised by the task-sets provided in the the files or from some other applications. As an output, theses modules produce the event streams ready for the execution. The streams can also be passed to another software for different analyses.

EE can solely be used apart from the rest of the SPARTS for execution purposes. The parameters for the execution can be

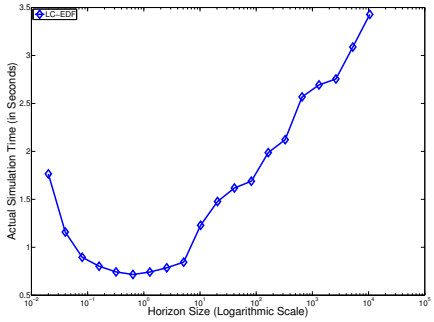

Fig. 3. Horizon size trade-off

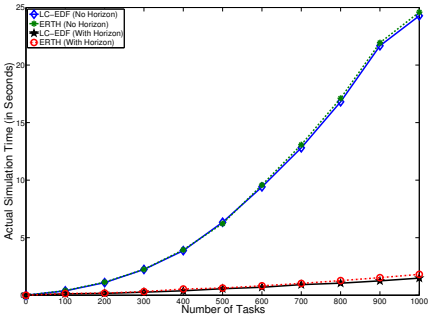

Fig. 4. Execution time with respect to task-set size

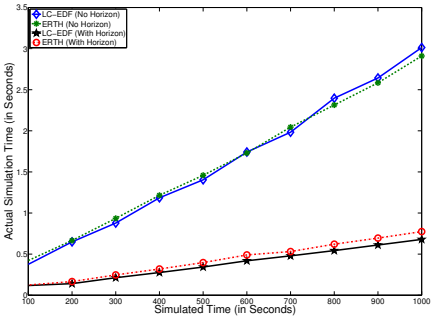

Fig. 5. Simulation time vs actual simulated time

provided by some other application. The execution itself is also closely tied to the reporting mechanism which can populate the statistics or the analysis findings in desired way.

D. Examples

In order to illustrate the performance of the SPARTS, we have simulated both LC-EDF [10] and ERTH [11] scheduling algorithms. We also included horizon mechanism for the performance comparison. The hardware used during the simulations consists of Intel Core 2 Duo CPU (3 GHz) with 2 GB of ram. Firstly, we analysed the trade-off when choosing the horizon length. The simulation assumed task-set size $|T| = 100$, $U = 75\%$ and the simulated period $t = 1000$ seconds. We varied the horizon length from $\frac{1}{50}$ of maximum inter-arrival time for HRT tasks to the size of the complete simulated period (the execution without the horizon mechanism). From the Figure 3 it is clear that small time-windows bring unnecessary overheads. The extension of the horizon gives better results, but from one point onwards, the simulation time again starts to increase, due to very large lists and queues as a direct result of the horizon length. We found for this particular set-up that the size of 0.6 of the maximum inter-arrival time for the HRT tasks gives the best results and we used it in the other experiments for comparison.

Figure 4 depicts the effect of the variation in the task-set size over the actual execution time for both algorithms, with and without the horizon mechanism. For this experiment we used $t = 100$, $U = 0.75$, and the $|T|$ is varied from 100 to 1000 tasks. The actual simulation time increases, as expected, with the number of tasks because the system has to evaluate denser event stream, as the number of events increases with an increase in the number of the tasks and vice versa. However, it is clear that simulations with the horizon mechanism outperform several times ones without it.

The effect of varying the execution time is studied in Figure 5. We fixed the $|T| = 100$ and $U = 75\%$. t is varied from 100 to 1000 seconds. The actual simulation time increases approximately linearly with the increase of the simulated time for the same $|T|$. The conclusion holds for both algorithms. The same reasoning about the performance of the simulations with and without the horizon mechanism can be drawn here.

V. Conclusions and Future Work

The SPARTS is under active development and is being extended. The features presented throughout the paper are ones that we have already implemented. However, as indicated in the individual sections, further extensions to this simulator are possible and desirable. For instance, the incorporation of resource demands within tasks and resource sharing mechanisms is intended. Besides run-time priorities, tasks may also have pre-defined static priorities. This allows implementation of fixed-priority and also some of dual-priority scheduling mechanisms [13]. From multi-core perspective, a task can have specific properties, such as favourable core, penalty for execution on unfavourable core, migration overhead etc.

The source-code of our current simulator is available on our website for downloading [1]. Along with it, a technical report with the in-detail explanations about our concrete implementation, and about the possibilities for the extensions can be found there. Finally, our primary goal is not only to develop the simulator for our own research needs and to share the results with the scientific community, but also to allow other interested parties to contribute, develop and extend different parts of the simulator that are areas of their own research.

References

[1] B. Nikolic, M. A. Awan, and S. M. Petters, "Simulator for power aware and real-time systems: Sparts," 2011. http://webpages.cister.isep.ipp.pt/~borni/.

[2] S. De Vroey, J. Goossens, and C. Hernalsteen, "A generic simulator of real-time scheduling algorithms," in *29th Simul. Symp. 1996*, pp. 242 –249, Apr 1996.

[3] A. Diaz, R. Batista, and O. Castro, "Realtss: a real-time scheduling simulator," in *4th Int. Conf. Electric. & Electron. Engin. (ICEEE 2007)*, pp. 165–168, Sep 5-7 2007.

[4] T. Kramp, M. Adrian, and R. Koster, "An open framework for real-time scheduling simulation," in *Int. WS Parall. & Distr. Processing*, pp. 766–772, 2000.

[5] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," in *ACM SIGAda international conference*, (New York, NY, USA), pp. 1–8, ACM, 2004.

[6] R. Urunuela, A. Deplanche, and Y. Trinquet, "Storm a simulation tool for real-time multiprocessor scheduling evaluation," in *Emerging Technologies & Factory Automation (ETFA), 2010 IEEE Conf.*, pp. 1 –8, Sep 2010.

[7] "Symta/s, Symtavision GmbH." http://www.symtavision.com/symtas.html.

[8] "Rapitime, Rapita Systems Ltd.." http://www.rapitasystems.com/products/RapiTime.

[9] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson, "Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes," in *24th RTSS*, (Cancun, Mexico), Dec 2003.

[10] Y.-H. Lee, K. Reddy, and C. Krishna, "Scheduling techniques for reducing leakage power in hard real-time systems," in *15th ECRTS*, pp. 105 – 112, jul. 2003.

[11] M. A. Awan and S. M. Petters, "Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems," in *23rd ECRTS*, 2011.

[12] P.-C. Wu, "Multiplicative, congruential random-number generators with multiplier $\pm 2k1 \pm 2k2$ and modulus $2p-1$," *TOMS*, vol. 23, pp. 255–265, June 1997.

[13] R. Davis and A. Wellings, "Dual priority scheduling," in *16th RTSS*, pp. 100 –109, Dec 1995.