2006-1854: REAL-TIME SYSTEMS SCHEDULING TOOL DEVELOPMENT

Daniel Ghiringhelli, Monmouth University

Daniel Ghiringhelli is completing his Masters in Software Engineering from Monmouth University, West Long Branch, NJ. He received his BS in Computer Science from Stevens Institute of Technology in Hoboken NJ in August, 2002. His research interests include ubiquitous computing, home theater system automation, software and network security.

Jiacun Wang, Monmouth University

Jiacun Wang received the PhD in computer engineering from Nanjing University of Science and Technology (NUST), China, in 1991. He is currently an associate professor of the software engineering department at Monmouth University, West Long Branch, New Jersey, USA. From January 2001 to August 2004, he was a member of scientific staff with Nortel Networks in Richardson, Texas. Prior to joining Nortel, he was a research associate of the School of Computer Science, Florida International University (FIU) at Miami. Prior to joining FIU, he was an associate professor at NUST. His research interests include software engineering, discrete event systems, formal methods, wireless networking, and real-time distributed systems. He authored Timed Petri Nets: Theory and Application (Norwell, MA: Kluwer, 1998), and published more than 50 research papers in journals and conferences. He is an editor of IEEE Transactions on Systems, Man and Cybernetics, Part C, and has served as a program committee member for many international conferences. Dr. Wang is a senior member of the IEEE.

Real-Time Systems Scheduling Tool Development

Abstract

This paper presents a real-time system (RTS) scheduling tool which implements some of the most popular scheduling algorithms. It allows users to specify real-time tasks in a RTS, then evaluates task scheduleability and plots the simulated schedules. It can be used by both instructors and students of RTS classes.

1. Introduction

Real-time systems are those systems that are required to respond to an external event in some timely manner[6,8]. An RTS is always subjected to timing constraints. The timing constraints imposed on hard real-time systems are absolute and must be satisfied. Therefore, special consideration needs to be taken when designing how a real-time system will execute all the possible tasks it may be assigned over any period of time.

RTS scheduling is a critical, and perhaps the most important part in teaching a real-time system class. Dozens of well-known scheduling algorithms are introduced in many textbooks[7,4], but can sometimes be difficult to explain with just words and graphs. It is desirable that, given a set of real-time task specifications, we have a dedicated software tool to carry out scheduleability analysis and schedule generation. However, such tools are not yet available. Commercial real-time operating system products, such as VxWorks[9], provide logic instruments with a graphical view of schedules, but they only work with a fully developed RTS - not a RTS model in terms of task specifications.

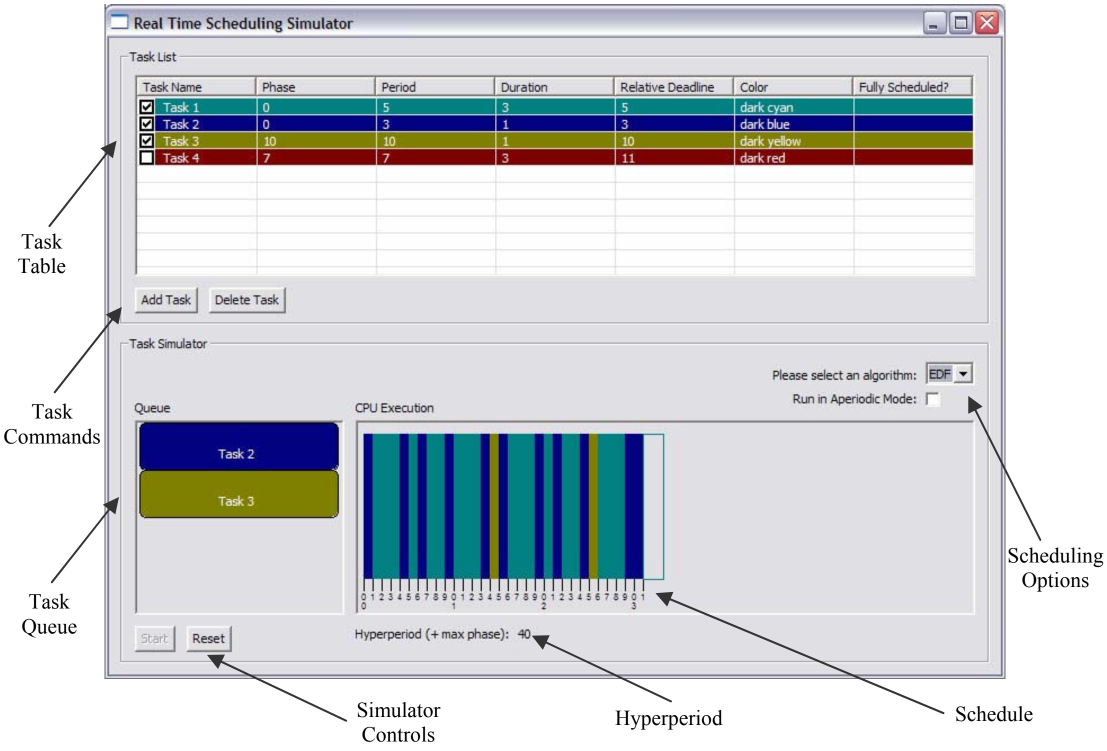
This paper presents a RTS scheduling tool newly developed at the Software Engineering Department of Monmouth University. The tool implements some of the most popular RTS scheduling algorithms, such as Earliest Deadline First (EDF), Least Slack Time First (LST), and Deadline Monotonic (DM). The tool evaluates RTS task scheduleability and plots the simulated scheduling results. It also comes with a very friendly user interface which allows users to easily (1) add or delete tasks/jobs, (2) specify or modify the real-time parameters of tasks/jobs, (3) select tasks/jobs for scheduling, and (4) select scheduling algorithms. For periodic tasks, the tool automatically calculates the hyperperiod and plots the schedule for one hyperperiod. As jobs are pre-empted and move to a waiting state, the tool displays the jobs in a waiting queue over the entire scheduling simulation.

The tool is developed primarily for education purpose. It is helpful for both instructors and students: First, it allows instructors to illustrate as many scheduling examples as possible during the limited class hours by simply changing the RTS tasks/jobs set or their real-time parameters. Second, it helps students to understand how the scheduling process proceeds with a given scheduling algorithm. Third, but not least, it allows students to compare and contrast the scheduling results of a given RTS model with different scheduling algorithms. As students are assigned various scheduling exercises, the RTS scheduling tool helps to reinforce the scheduling algorithm concepts they have learned by providing the correct schedules and the schedules' properties which the students can independently analyze. The tool was written in Java and provides a framework for the expansion of additional algorithms and features.

The paper is organized as follows: An overview of the tool functionalities and interface is given in Section 2. The implemented scheduling algorithms are introduced in Section 3. In Section 4, the high-level design of the tool is presented. Section 5 concludes the paper.

2. Tool Overview

The graphical user interface (GUI) of the RTS Scheduling Tool was designed to give users constant live feedback of the underlying scheduling simulation. It was built upon the Standard Widget Toolkit (SWT) framework developed by the Eclipse Foundation (www.eclipse.org) which provides a native windowing support for desktop Java applications.


Real Time Scheduling Simulator

Task List

Task Name	Phase	Period	Duration	Relative Deadline	Color	Fully Scheduled?
☑ Task 1	0	5	3	5	dark cyan	
☑ Task 2	0	3	1	3	dark blue	
☑ Task 3	10	10	1	10	dark yellow	
☐ Task 4	7	7	3	11	dark red	

Task Simulator

Please select an algorithm: EDF

Run in Aperiodic Mode: ☐

Hyperperiod (+ max phase): 40

Figure 1 Tool GUI.

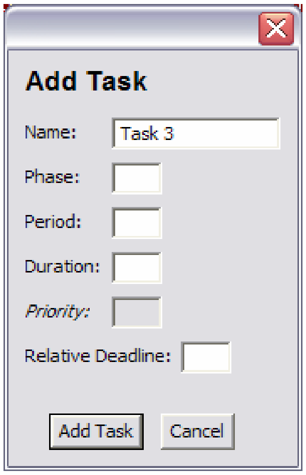
The main GUI for the RTS scheduling tool is shown in Figure 1 and has the following key features:

- **Task Table** – The task table lists all tasks that are available to be selected for scheduling simulation. Within the task table, individual task items can be modified by double-clicking its value. The user can edit a task's name, phase, period, duration, relative deadline and color. The "Fully Scheduled?" field is a read-only field that is updated when a simulation completes. This column answers the question: "Did this task meet all its deadlines and is therefore scheduleable?" Finally, there is a checkbox next to each task which indicates whether or not the task will be scheduled in the next simulation run.

- **Task Commands** – The task commands allow for the addition of new tasks and deletion of existing tasks.
- **Scheduling Options** – The scheduling options allow the user to select what type of scheduling algorithm to execute during the simulation run. There is also an "Aperiodic Mode" checkbox that turns on or off periodic scheduling. If this box is checked, then only the first job from each task will be scheduled.
- **Task Queue** – The queue dynamically displays any tasks that are blocked by the currently executing task.
- **Schedule (CPU Execution Canvas)** – The execution canvas displays each task's execution history (opaque) as well as the currently executing task (transparent).
- **Hyperperiod** – The hyperperiod is calculated for every simulation run and is shown below the CPU execution canvas. The simulation runs until the current time reaches the calculated hyperperiod.

Simulating the scheduling of a set of tasks in the RTS scheduling tool requires two primary steps: configuring the task set via the task pane and selecting the simulation algorithms. To add a new task, the "Add Task" button can be clicked which displays the dialog box shown in Figure 2. Each task's properties (Name, Phase, Period, Duration and Relative Deadline) can be customized to any non-negative integer.

Once the task has been added to the task table, the user can double-click any field to edit the value. Colors can also be selected from the color drop-down box. The "Fully Scheduled?" column is read-only. To delete a task, the user can select the task and click the delete button. To enable the task for simulation, the user checks the check-box next to the task name.



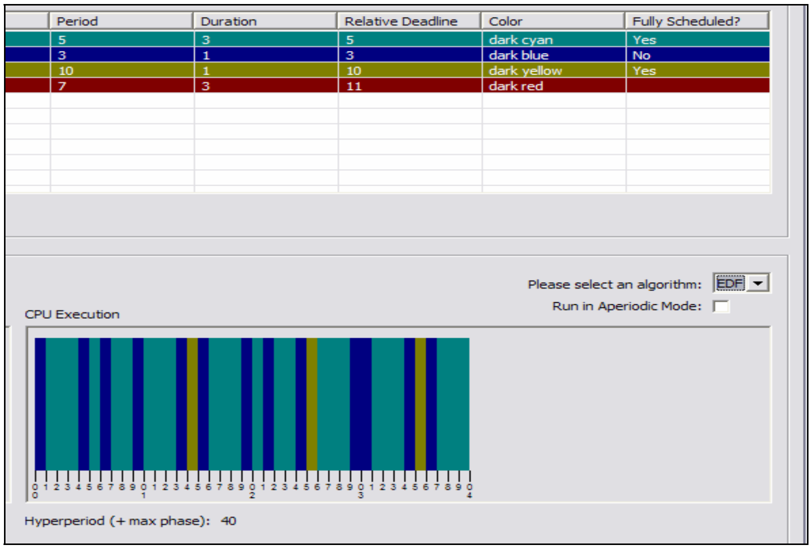
Add Task

Name: Task 3

Figure 2 Add task.

After tasks are configured, the user can then select the algorithm from the Task Simulator pane. The periodic/aperiodic options can also be selected from the checkbox next to the "Run in Periodic Mode" option. Aperiodic tasks will simulate one-time jobs that do not arrive periodically. Thus each task's period is ignored. Finally, the user can click "Start" to begin the simulation. Once the simulation reaches the calculated hyperperiod, it automatically terminates. The "Reset" button can be used at any time to clear the CPU Execution Canvas and associated Queue.

Once the hyperperiod expires, the simulator calculates the overall scheduleability of each task. In the sample run shown in Figure 3, the 2nd task in dark blue is marked as not being schedulable. Analysis of the CPU execution history reveals that the periodic scheduling of Task 2 at time t=39 was not scheduled, thus Task 2 was outstanding when the hyperperiod expired.



	Period	Duration	Relative Deadline	Color	Fully Scheduled?
	5	3	5	dark cyan	Yes
	3	1	3	dark blue	No
	10	1	10	dark yellow	Yes
	7	3	11	dark red	

Please select an algorithm: EDF

Run in Aperiodic Mode: □

CPU Execution

Hyperperiod (+ max phase): 40

Figure 3 Simulation complete.

3. Scheduling Algorithms

The RTS scheduling tool currently supports the simulation of three primary real-time scheduling algorithms known as the Deadline Monotonic (DM) algorithm, sometimes referred to as the Fixed Priority Scheduling algorithm, the Earliest Deadline First (EDF) algorithm, and the Least-Slack Time First (LST) algorithm. DM, EDF and LST are also most widely used algorithms for periodic tasks, while EDF and LSF are two well-known algorithms illustrating priority-driven one-time jobs scheduling. Periodic tasks are specified by the following parameters:

- *Phase*: The release time of the first job of a task.
- *Period*: The amount of time between the arrivals of two consecutive jobs of a task.
- *Duration*: The execution time of a job in a task.
- *Relative Deadline*: The job deadline relative to release time.

For one-time jobs, we only need to know their phases, durations and relative deadlines.

Deadline Monotonic (*DM*)

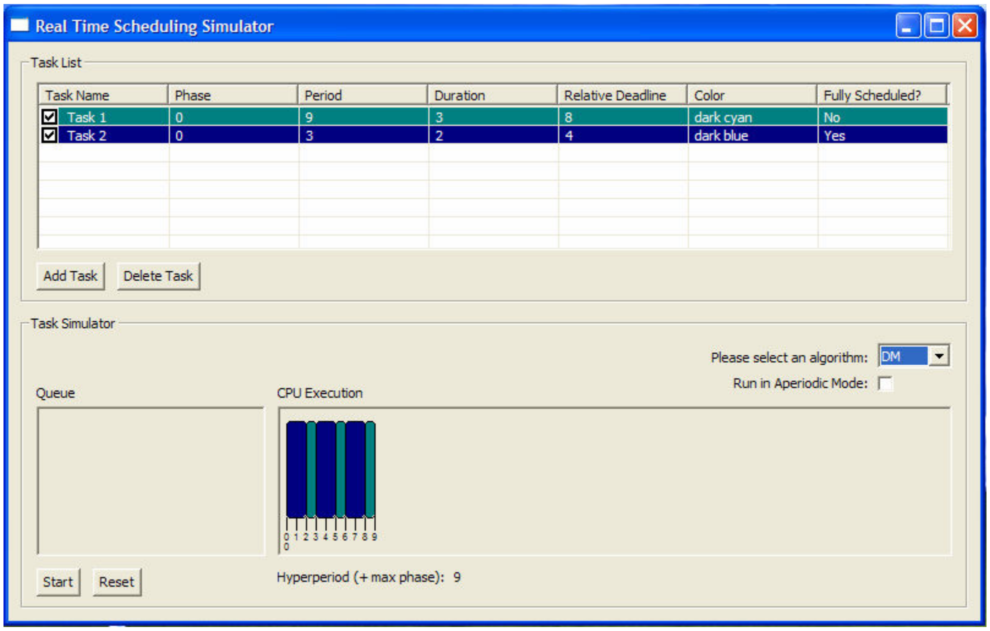
The DM algorithm is one of the simplest scheduling algorithms to implement[1,2]. It is considered a static metric algorithm because it operates solely over pre-computed priority values. In practice, a real-time system may have a separate program (or human analyst) that calculates task priorities prior to execution. These priorities are then saved for the DM scheduler to reference at run-time.

The DM scheduler always schedules the highest priority task (the one with the lowest relative deadline) that is available at any given time. The DM scheduler will pre-empt any lower priority task until the higher priority task completes execution. This simple static comparison provides a lightweight scheduler that requires little CPU overhead.

Consider the following tasks:

- Task 1: Phase = 0, Period = 9, Duration = 3, Relative Deadline = 8
- Task 2: Phase = 0, Period = 3, Duration = 2, Relative Deadline = 4

The RTS scheduling tool generates the schedule shown in Figure 4, which illustrates that Task 1 is not schedulable. In the DM scheduler, it can be seen that Task 2 will always pre-empt Task 1, at any given time. Thus at time t=0, t=3, t=6 (since Task 2's period = 3), Task 2 is immediately scheduled. Consider the time t=6. At this moment, Task 1 has executed for 2 seconds and requires just 1 additional second to complete execution before it's deadline at t=8. Conversely, Task 2 has yet to execute (for the new instance) and its deadline is at t=9. It is important to note that in this scenario, the DM scheduler will cause Task 1 to miss its deadline, while it would have been possible for both tasks to complete before their deadline.



Task Name	Phase	Period	Duration	Relative Deadline	Color	Fully Scheduled?
☑ Task 1	0	9	3	8	dark cyan	No
☑ Task 2	0	3	2	4	dark blue	Yes

Please select an algorithm: DM

Run in Aperiodic Mode: □

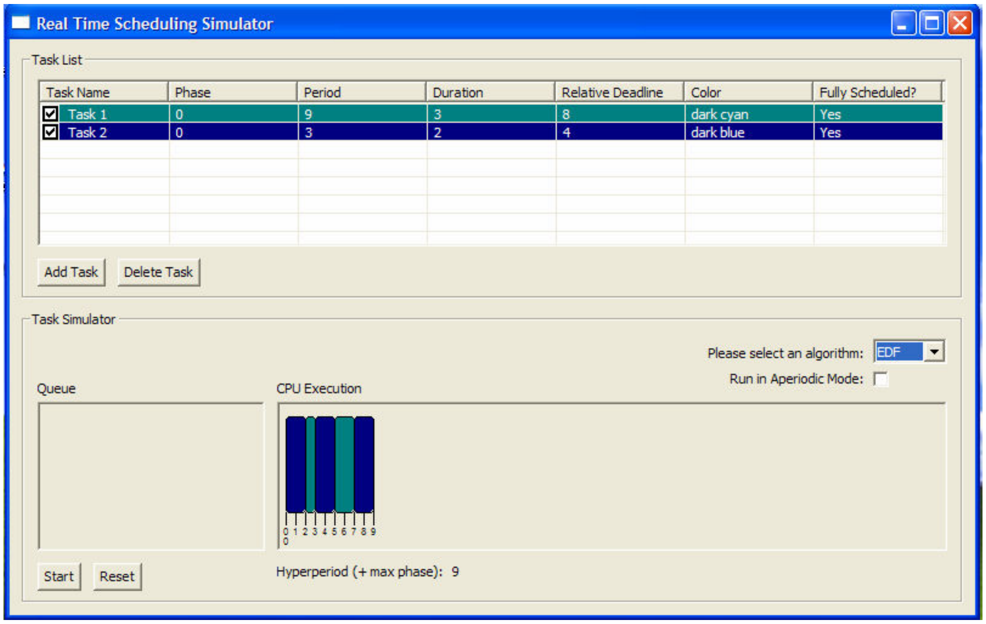
Hyperperiod (+ max phase): 9

Figure 4 DM schedule of tasks (0, 9, 3 8) and (0, 3, 2, 4).

Earliest Deadline First (*EDF*)

The EDF scheduler was designed to address some of the short-comings of the DM scheduler[3]. Since the DM algorithm is limited to static task properties, it has no way of detecting the scenario illustrated above. The EDF scheduler provides more optimal CPU utilization through the dynamic prioritization of tasks. EDF can always schedule any set of tasks that are schedulable by the DM algorithm. As new tasks arrive, the EDF algorithm will always schedule the task whose deadline is *soonest*. Therefore, as long as the utilization of the process set is less than the total capacity of the processor then all deadlines will be met[4].

Revisiting the scenario described above reveals that both Task 1 and Task 2 are schedulable as seen in Figure 5. At time t=6, Task 1 is not pre-empted since its deadline (t=8) is dynamically compared to the next deadline of Task 2 (t=9) and found to be nearer than Task 2. Thus Task 1 completes at t=7 and Task 2 completes before the hyperperiod expires (t=9).



Task Name	Phase	Period	Duration	Relative Deadline	Color	Fully Scheduled?
☑ Task 1	0	9	3	8	dark cyan	Yes
☑ Task 2	0	3	2	4	dark blue	Yes

Please select an algorithm: EDF

Run in Aperiodic Mode: □

Hyperperiod (+ max phase): 9

Figure 5 EDF schedule of tasks (0, 9, 3 8) and (0, 3, 2, 4).

Despite the theoretical improvements EDF algorithms offer over DM algorithms, in practice, DM schedulers are the most common method used in real-time systems. This can be attributed to a number of factors[4]:

- DM is more simple (and easier) to implement on a RTS then EDF. Additionally, since all priority-processing is done statically offline, the RTS resources do not require this additional overhead at run-time.
- DM can be considered more stable and predictable. If the RTS is overloaded, where a set of tasks will certainly miss their deadlines, then the DM scheduler guarantees that lower priority tasks will be the first to miss their deadlines. However, the EDF scheduler makes no such guarantee and can result in critical tasks missing deadlines before routine tasks.
- The EDF focuses purely on deadline, but deadline is not the only metric that can determine priority. The static "deadlines" used by DM can incorporate other factors as well.

Least Slack Time First (*LST*)

Also called *Minimum Laxity First* (MLF) algorithm, the LST algorithm assigns priorities to jobs based on their slacks: the smaller the slack, the higher the priority. LST is also optimal. The difference between EDF and LST is, EDF algorithm does not require any knowledge of the execution times of jobs, but the LST algorithm does[5].

Hyperperiod and Scheduleability

The RTS Scheduling Tool calculates the hyperperiod for the given task set before every simulation. The hyperperiod is fundamentally the least common multiple for all the task periods plus the maximum phase of all the tasks. When the "Start" button on the simulator is clicked, the hyperperiod is calculated and displayed below the CPU Execution Canvas. A task is said to be schedulable if every periodic instance of the task is fully completed on-time before the expiration of the hyperperiod. If a task is blocked in the queue at hyperperiod expiration or missed any deadline during the simulation, it is not fully schedulable for the given task set.

4. Tool Design

The RTS scheduling tool was designed to run as a stand-alone desktop GUI application that can be easily ported between various operating systems. Thus the Java programming language was a natural fit. Java's open specification and extensive library support provides an extremely capable platform on which to develop robust applications. The Java Virtual Machine (JVM) is designed to run on top of all common operating system platforms and thus provides easy portability of source code. RTS Scheduling Tool was developed on top of the Java 2 Platform, Standard Edition (J2SE) 5.0 version of java. It has been tested against Sun Microsystems' distribution of J2SE Runtime Environment (JRE) version 5.0.

The Java Development Kit (JDK) distribution comes standard with the Java Swing package, however RTS Scheduling Tool was developed on the SWT framework distributed by the Eclipse Foundation (www.eclipse.org) as part of the core eclipse platform. This development tradeoff was made due to the native operating system support provided by SWT. SWT is an open source widget toolkit for java designed to provide efficient, portable access to the user-interface facilities of the operating system on which it is implemented (Ref: SWT). Conversely, Java Swing does not provide the same platform abstraction but instead programmatically generates cross-platform widgets not necessarily native to the target operating system.

The RTS Scheduling Tool was developed on Windows XP Professional using the Eclipse 3.0 Integrated Development Environment (IDE). The IDE provides native support for JUnit testing and ant build facilities. The RTS Scheduling Tool is distributed via an executable jar which can be generated at build time and simply executed by double-clicking.

The RTS Scheduling Tool follows the Model View Controller (MVC) architecture as shown in Figure 6. The MVC architecture is a commonly used design pattern that provides a clean separation between the logical functions of application components. The MVC architecture in general divides the system into 3 components:

- *Model*: Maintains and stores all the data associated with the application and performs system functions over the data.

- *View*: Abstracts the data from the model for feedback to the user; passes user actions to the controller for logical processing.
- *Controller*: Responsible for the application logic and coordination between the view and the model.

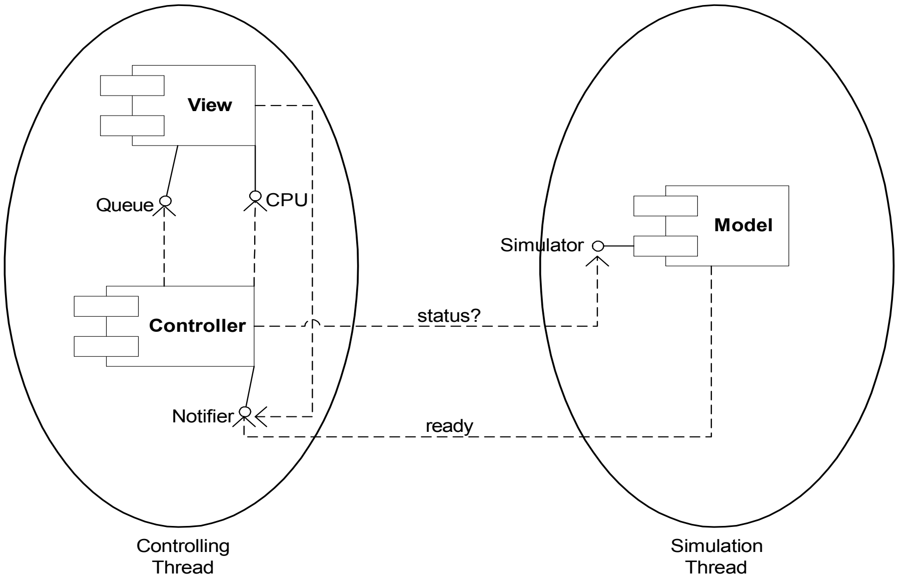


Figure 6 RTS scheduling tool model view Controller architecture.

It is not uncommon for the view to interact directly with the model for synchronization of data. However, the MVC architecture of the RTS Scheduling Tool intentionally separated the dependence between the view and model to enable a two-threaded system interaction interfacing directly between the model and controller. SWT constraints prevent the initiation of data transfer from a daughter thread to its parent thread. As seen in Figure , the Controlling Thread (parent) must therefore poll the simulator for change of status updates. To improve performance, this interaction occurs only when the model alerts the controller of a status change by setting a ready flag. When this flag is set, the controller will perform a *deep* query of the model and subsequently update the view.

The RTS Scheduling Tool has a relatively simple package layout for the source as shown in Figure 7. The `rts` (real-time scheduler) package is the root package of the system that contains the application entry point as well as the primary controller classes. The run-time event loop occurs in this package. The `rts.algorithms` package contains the abstract algorithm class and all the sub-classes that inherit from the abstract algorithm class. The algorithm package is responsible for all scheduling decisions made by the simulator.

The `rts.model` package provides the underlying simulation data elements used by the system. For example, all data pertaining to tasks are stored in the `rts.model` package. This package also contains the simulation loop that is executed by the simulation thread described in

Figure 7. Finally, the `rts.gui` package provides the GUI controls used to display simulation data. These controls inherit from the open source Eclipse SWT library.

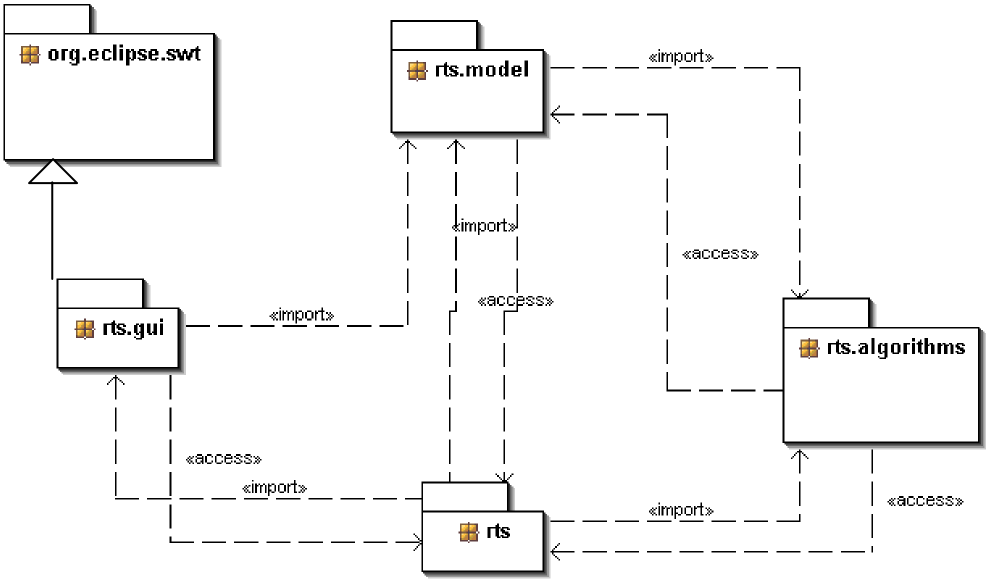


Figure 7 RTS scheduling tool package view.

5. Concluding Remarks

The RTS scheduling tool is a fully operational RTS task scheduler that allows for the analysis of various real-time scheduling algorithms. It was designed primarily as a courseware for RTS classes. We asked a few of students who took RTS course in the past to try the tool, and the feedback from them was quite positive. We will use this tool in our RTS Specification and Design course this fall.

With a user-friendly graphical user interface and a stable underlying simulator, the RTS scheduling tool provides a scalable architecture for future enhancements. Its development upon open-source technology, tools and utilities further provides a maintainable base of support.

To say that the RTS Scheduling Tool is completely operational does not preclude its potential for some upgrades and modifications. Below is a list of items that have yet to be fully supported, but have limited functionality in its current state:

- *GUI Validation*: Basic task parameters are verified (no negative numbers or characters), but the logical correctness of these values are not validated by the system. It may be desirable to provide the user with a warning message if they select values that are logically incorrect, e.g. a period that is shorter than a task's duration.
- *Simulation Pause/Restart Capability*: The initial design planned for this capability however it was not fully implemented in the final release. It may be nice to have at some point.

- *Extended Algorithm Support*: This capability exists however it just needs to be written. Adding additional algorithm support will only increase the usefulness of the tool.
- *Application Installer:* The RTS Scheduling Tool would benefit from an installer that would automatically check the client's system for required dependencies, such as JRE 1.4 or later and the SWT library.

Bibliography

1. P. Altenbernd, Deadline-monotonic Software Scheduling for the Co-synthesis of Parallel Hard Real-time Systems, *European Design and Test Conference*, 1995.
2. N. C. Audsley, A. Burns, M. F. Richardson and A. J. Wellings, Hard Real-Time Scheduling: The Deadline Monotonic Approach, *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, GA, USA, May 1991.
3. T. P. Baker, Multiprocessor EDF and Deadline Monotonic Schedulability Analysis, *24th IEEE International Real-Time Systems Symposium*, 2003.
4. A. Burns and A. Wellings, *Real-Time Systems and Programming Languages*, England: Pearson Education Limited, 2001.
5. R.I. Davis, K.W. Tindell and A. Burns, Scheduling Slack Time in Fixed Priority Pre-emptive Systems, *Proceedings Real-Time Systems Symposium*, pp. 222-231, December 1993.
6. P. A. Laplante, *Real-Time Systems Design and Analysis: An Engineer's Handbook*, 2nd Edition, New York: IEEE Press, 1997.
7. Q. Li and C. Yao, *Real-Time Concepts for Embedded Systems*, San Franscisco: CMP Books, 2003.
8. J. Liu, *Real-Time Systems*, Prentice Hall, 2000.
9. Wind, *WxWorks Programmer's Guide*, Wind River Systems, Inc., 2002.