hsSim: an Extensible Interoperable Object-Oriented n-Level Hierarchical Scheduling Simulator

João Pedro Craveiro, Rui Ormonde Silveira, and José Rufino
Universidade de Lisboa, Faculdade de Ciências, LaSIGE
Lisbon, Portugal
Email: jcraveiro@lasige.di.fc.ul.pt, rsilveira@lasige.di.fc.ul.pt, ruf@di.fc.ul.pt

***Abstract*—Hierarchical scheduling is a recent real-time scheduling topic. It is used to obtain temporal interference isolation in various scenarios, such as scheduling soft real-time aperiodic tasks along with hard real-time periodic tasks, as in mixed-criticality scenarios. Most theory and practice focuses on two-level hierarchies, with a root (global) scheduler managing resource contention by partitions (or scheduling servers), and a local scheduler in each partition/server to schedule the respective tasks. In this paper we describe the development of hsSim, an object-oriented hierarchical scheduling simulator supporting an arbitrary number of levels. With the goal of openness, extensibility and interoperability in mind, due care was put into the design, applying known design patterns where deemed advantageous. We demonstrate hsSim's interoperability potential with a case study involving the Grasp trace visualization toolset.**

I. Introduction

Hierarchical scheduling is a recent topic in the mature real-time scheduling theory discipline. In one of the first works on the topic, Abeni and Buttazo [1] present a hierarchical scheduling framework in which hard real-time tasks are scheduled along with a Constant Bandwidth Server (which, for the scheduler below, behaves exactly has the other periodic tasks); the CBS is, in turn, responsible for scheduling the jobs of aperiodic soft real-time tasks. Hierarchical scheduling is also used in mixed-criticality systems — systems providing multiple functionalities who differ *(i)* in how important ("critical") they are for the overall welfare of the system, and *(ii)* in the level of assurance of each one's mandatory certification [2]. A common approach to this certification issue of mixed-criticality systems is enforcing isolation through time and space partitioning (TSP) [3], such as in the ARINC 653 specification: tasks are separated into partitions, which are scheduled by a cyclic executive (according to a partition scheduling table — PST); in each partition's activity time windows, the respective tasks compete for scheduling based on a priority-based scheduling policy [4], [5]. Finally, hierarchical scheduling also sees application in the virtualization field, with nested virtualization for advanced security purposes evidencing the need to support an arbitrary number of hierarchy levels [6].

This work was developed in the followup of European Space Agency ITI project AIR-II (ARINC 653 In Space RTOS, http://air.di.fc.ul.pt). This work was partially supported by the EC, through project IST-FP7-STREP-288195 (KARYON, http://www.karyon-project.eu/), and by FCT, through the Multiannual and CMU|Portugal programs, and the Individual Doctoral Grant SFRH/BD/60193/2009. This work was partially supported by FCT/Égide (PESSOA programme), through the transnational cooperation project SAPIENT (Scheduling Analysis Principles and Tool for Time- and Space-Partitioned Systems, http://www.navigators.di.fc.ul.pt/wiki/Project:SAPIENT).

In this paper, we present the development of **hsSim** (pronounced *aitch-ess-sim*), a simulation tool for the hierarchical real-time scheduling research community. Due to this target community, we focus on supporting well-known and widely used models therein, such as the periodic task model and, in a near future, compositional analysis abstractions such as the periodic resource model [7]. hsSim pursues the goal of an open, reusable, extensible and interoperable tool. As such, we strived for a modular design and development methodology, aided by the careful application of the object-oriented paradigm and software design patterns. Design patterns are design decisions and solutions for recurring problems encountered in object-oriented systems. The employment of these design patterns caters to the goal of reusing successful past experience in software design [8]. To demonstrate the interoperability potential of hsSim, we describe a case study using Grasp, a trace visualization toolset [9].

Paper outline: In Section II, we present related work on scheduling simulation, including Grasp. In Section III, we briefly describe the system model we assume for the first iterations of hsSim's development. In Section IV, we detail the analysis and design steps taken, with emphasis on the main design patterns from whose application we took advantage. Section V describes the implementation and tests to the current version of hsSim, namely the interoperability case study based on the Grasp toolset. Finally, Section VI closes the paper with concluding remarks and ongoing work.

II. Related work

To the best of our knowledge, the current state of the art lacks a scheduling simulator for hierarchical scheduling frameworks with an arbitrary number of levels.

Cheddar [10] provides a set of scheduling algorithms and policies on which it is capable of performing feasibility tests and/or scheduling simulations for either uniprocessor or multiprocessor environments. In its latest versions, Cheddar already supports schedulability analysis of ARINC 653-like time- and space-partitioned (TSP) systems to some extent [10]. However, Cheddar presents some limitations in its current support thereto. A partition scheduling table (PST) is defined as an array of durations; besides presenting less usability, the current implementation limits the PST to having only one time window per partition per hyperperiod. Cheddar is designed to be extensible, and we are currently involved in a collaboration with the Cheddar team towards more complete hierarchical scheduling

capabilities, applying, among others, the principles and patterns identified in the present paper [11], [12].

Grasp [9] is a trace visualization toolset. It supports the visualization of multiprocessor hierarchical scheduling traces. Traces are recorded from the target system (by the Grasp Recorder or any other appropriate means) into Grasp's own script-like format, and displayed graphically by the Grasp Player. The Grasp toolset does not support simulation and supports only a two-level hierarchy, whereas hsSim simulates hierarchical systems with an arbitrary number of levels. However, the Grasp Player reads traces in a simple text-based format which can be recorded by other tools. In this paper, we demonstrate our tool's interoperability features by implementing the possibility to generate a Grasp trace.

CARTS [13] is an opensource compositional analysis tool for real-time systems, which automatically generates the components' resource interfaces; it does not perform simulation. CARTS relies strongly upon some of the authors' theoretical results (e.g., [14]) and, while implemented in Java, does not take advantage of the latter's object-oriented characteristics (inheritance, polymorphism, encapsulation — especially regarding the separation between domain and user interface). This makes it difficult to be extended and, as such, we chose to develop hsSim from scratch instead of modifying CARTS.

SPARTS [15] is a real-time scheduling simulation focused on power-aware scheduling algorithms. Its simulation engine is optimized by replacing cycle-step execution for an event-driven approach centered on the intervals between consecutive job releases. Hierarchical scheduling support is not mentioned.

MAST 2 [16] defines a model to describe the timing behaviour of real-time systems designed to be analyzable via schedulability analysis techniques; it is accompanied by a tool suite including schedulability analysis tools. MAST 2 introduces modelling elements for virtual resources, abstracting entities that rely on the resource reservation paradigm.

Finally, the Schesim [17] scheduling simulator supports hierarchical limited to two-levels. Simulation is based on models of the tasks' implementations, not on task abstractions such as the periodic/sporadic task models, making it useful for direct application, but not so for our target real-time scheduling theory research.

Regarding commercial tools, a mention is due to SymTA/S [1], a model-based timing analysis and optimization solution with support to ARINC 653. No specific mention is made to hierarchical scheduling, thus we can only assume it supports a two-level hierarchy. Due to its proprietary nature, we cannot fully assert its capabilities and it does not serve our purpose for open, reusable, extensible tools for academic/scientific research.

III. System model

In our model, tasks and partitions are both abstracted as synchronous periodic schedulable entities, characterized by the periodic task model — $\tau_i = \langle C_i, T_i, D_i \rangle$, respectively worst-case execution time (WCET), period, and relative deadline. Each task or partition generates a potentially infinite sequence of jobs (or activations) characterized as $J_{i,k} = \langle c'_{i,k}, r_{i,k}, d_{i,k} \rangle$, respectively remaining execution time, release time and absolute deadline time; we assume all tasks are released at the critical instant, i.e., $t = 0$. Sporadic tasks are not specifically addressed, but in our model we may abstract them as periodic tasks with periods identical to their minimum inter-arrival times.

[1] https://symtavision.com/symtas.html

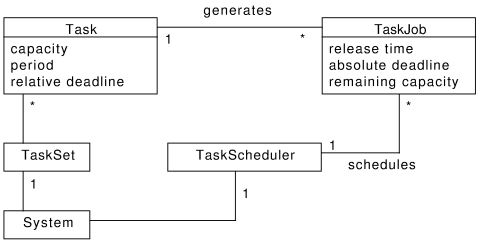

Fig. 1. Traditional 1-level system domain model

The system and the partitions have, each, their own scheduler to schedule their respective children schedulable entities. In traditional two-level hierarchical scheduling systems, such as those defined by the ARINC 653 specification [4], the system scheduler schedules partitions, whereas each partition's scheduler schedules the former's tasks. Here, we aim for a more generic model, where more levels can be composed (i.e., a partition can have children partitions, which in turn have tasks) [7], [14] and tasks can be coscheduled alongside partitions by the same scheduler [1] — hence the need to abstract tasks and partitions.

Finally, we assume scheduling over one unit-speed processor and that task/partition context switching times are either negligible or already accounted for in their temporal characteristics.

IV. Object-Oriented Analysis and Design (OOAD)

The implementation of a tool which we want to be flexible, extensible and more easily maintainable must be preceded by careful analysis and design. We will now document the main analysis and design steps and decisions taken, supported by Unified Modelling Language (UML) diagrams where deemed necessary. The overall design classes diagram is ommited due to paper length constraints.

A. Domain analysis

The traditional real-time system is flat (a 1-level hierarchy). The UML diagram for such a system's domain is pictured in Fig. 1. The system has a flat task set and a task scheduler.

A two-level hierarchical scheduling such as those corresponding to TSP systems [5] can be modelled as seen in Fig. 2. The system has a set of partitions and a root scheduler coordinating which partition is active at each instant. Each partition then has a set of tasks and a local scheduler to schedule the latters' jobs. This domain model strategy has two main drawbacks: it is hard-limited to two levels, and it only allows homogeneous levels (i. e., partitions and tasks cannot coexist at the same level).

B. n-level hierarchy: the Composite pattern

The *Composite* pattern is a design pattern that may be used when there is a need to represent part–whole hierarchies of

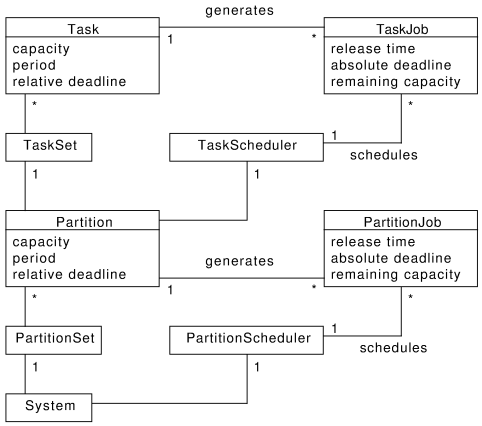

Fig. 2. 2-level hierarchical scheduling system domain model

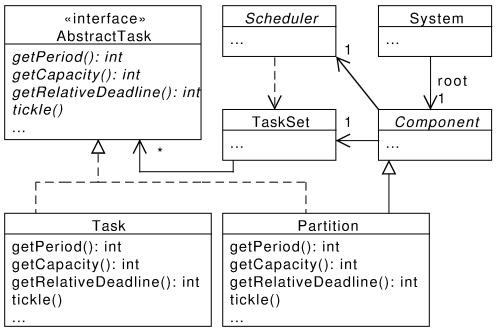

Fig. 3. n-level hierarchical scheduling system using the Composite pattern

objects, and allow clients to ignore the differences between composition of objects and individual objects [8]. Clients manipulate objects in the composition through a component interface, which abstracts individual objects and compositions.

Figure 3 shows the UML representation of the Composite pattern as applied to our domain. Applying this pattern to our model of a hierarchical scheduling framework allows breaking two limitations: the fixed number of levels in the hierarchy [7], [14], and the need for the hierarchy to be balanced [1]. The clients of the AbstractTask interface include schedulers, which will be able to schedule both tasks and partitions through a common interface, reducing implementation efforts and allowing extensions through new schedulable entities (e. g., servers). The application of this pattern triggered further refinements, such as making the TaskSet the System's and Partitions' AbstractTasks container, and merge TaskJobs and partition activations (PartitionJobs) under a generic Job abstraction.

C. Scheduling algorithm encapsulation: the Strategy pattern

The *Strategy* (or *Policy*) pattern is an appropriate solution to when we want to define a family of algorithms which should be interchangeable from the point of view of their clients [8]. In designing hsSim, we apply the Strategy pattern to encapsulate the different scheduling algorithms, as seen in Fig. 4. In the Scheduler abstract class, although we use a scheduling policy to initialize the JobQueue, we leave the obtention of

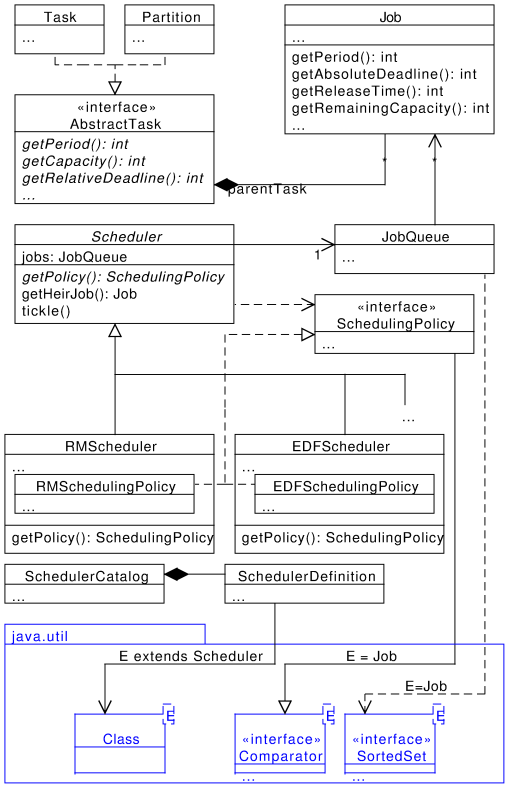

Fig. 4. Scheduling algorithm encapsulation with the Strategy pattern

the scheduling policy (the getPolicy() method) to the concrete scheduler classes; this is supported on another well-known design pattern, the Template Method [8].

The SchedulingPolicy interface extends Java's Comparator interface; this way, an instance of a subclass of SchedulingPolicy can be used to maintain the scheduler's job queue ordered in the manner appropriate for the scheduling algorithm being implemented.

The available strategies (scheduler types) are stored in a catalog, and more strategies can be loaded in runtime (provided the user interface gives a means to it). This is made possible by Java's native reflection capabilities.[2]

D. n-level hierarchy and polymorphism

Due to the design decisions regarding the Composite and Strategy patterns, most operations can implemented without having to account for which scheduler (or schedulers) are present, or for the structure and/or size of the hierarchy (partitions and tasks). Taking advantage of subtype polymorphism, we can invoke methods on Scheduler and AbstractTask references instances without knowing of which specific subtype thereof the instances are.

Let us see how this works with the scheduler tickle operation, which simulates the advance of system execution by one time unit. For the time being, we implement hsSim as a cycle-step execution simulator; an event-driven approach as the one seen

[2]Since we anticipated using the Java to implement hsSim, this and the following design decisions take explicit advantage from facilities provided by the Java libraries.

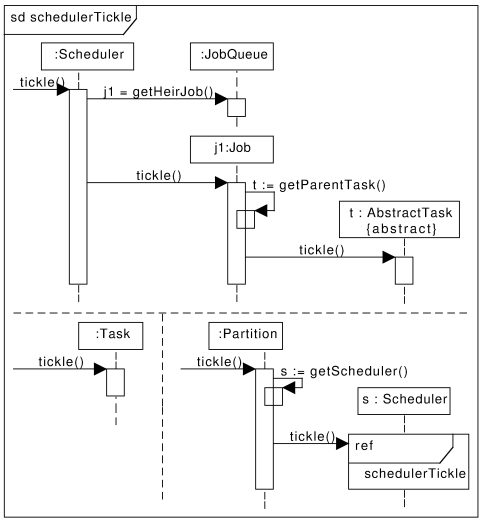

Fig. 5. Sequence diagram for the scheduler tickle operation

in SPARTS [15] is planned for future work (see Section VI). The UML sequence diagram modeling the interactions between objects in this operation is shown in Fig. 5. The hierarchical tickle process is started by invoking the tickle operation on the root scheduler without specific regard for what subtype of Scheduler it is; the right job to execute will be obtained because the scheduler's job queue is maintained accordingly ordered by an instance of an unknown SchedulingPolicy subtype. This job is then tickled, and in turns tickles its parent AbstractTask without knowing if it is a Task or a Partition. It is the job's parent's responsibility to invoke the right behaviour according to its type. If it is a Partition instance, this involves tickling its scheduler; this will cause an identical chain of polymorphic invocations to take place.

E. Decoupling the simulation from the simulated domain using the Observer and Visitor patterns

In hsSim, we want to decouple the simulation aspects (such as running the simulation and logging its occurrences) from the simulated domain itself. On the one hand, we want changes in the simulated domain (a system with partitions, tasks, jobs) to be externally known of, namely by one or more loggers, without the domain objects making specific assumptions about these loggers behaviour or interfaces. On the other hand, we want to be able to create new loggers without tightly coupling them to the domain objects or having to modify the latter. We found the Observer and Visitor patterns to be most appropriate to solve this specific problem.

The *Observer* pattern defines a publisher–subscriber dependency between objects, so that observers (subscribers) are notified automatically of the state changes of the subjects they have subscribed to. The subjects only have to disseminate their state changes to a vaguely known set of observers, in a way that is totally independent of how many observers there are and who they are — in the form of events. The *Visitor* pattern defines a way to represent an operation to be performed on an object hierarchy independently from the latter [8]. In few words, the

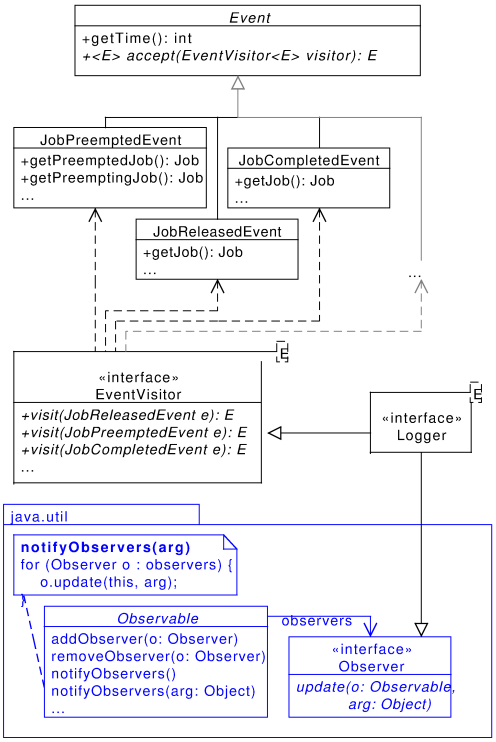

Fig. 6. Logger (Observer and Visitor patterns)

Observer pattern guides loggers in choosing from what domain objects they wish to receive events, and the Visitor pattern helps each logger define what to do with each kind of event.

Our application of these patterns in hsSim is pictured in Fig. 6. We take advantage from the simple Observer implementation provided by Java, with the Logger interface extending the Observer interface — thus obligating its subclasses' instances (the concrete loggers) to provide the method to be called to notify it of an event. The Logger interface also extends our EventVisitor interface, which defines methods to process each type of event. The visit methods are overloaded and every Event subclass provides the following method:

```
public <E> E accept(EventVisitor<E> visitor) {
    visitor.visit(this);
}
```

This way, when receiving an event e, a logger only has to invoke `((Event) e).accept(this)` to have the right visit method called.

We also apply the Observer pattern to establish a dependency between the system clock and the schedulers, so that the latter become aware of when to released their tasks' jobs.

V. Implementation and Use

For a first approach, we deemed necessary one scheduler for each of Carpenter et al.'s priority-based categories [18]: Rate Monotonic (RM) for fixed task priority, Earliest Deadline First (EDF) for fixed job priority, and Least Laxity First (LLF) for dynamic priority. We defined the following iteration strategy to have incrementally more complete prototypes:

1) get the core working with the three chosen schedulers (RM, EDF, LLF) and predefined input/output;

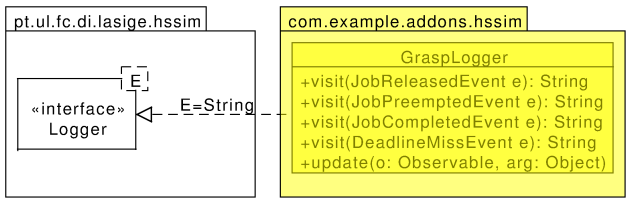

Fig. 7. GraspLogger implementing the Logger interface

Listing 1. GraspLogger excerpt (event notification reception)

```
public void update(Observer o, Object arg) {
    String result = ((Event) arg).accept(this);
    writeToFile(result);
}
```

2) implement a concrete logger extending the Logger interface (Section IV-E);
3) read the simulated scenario from an XML file;
4) implement a simple random scenario generator, with the option of either writing the generated scenario to an XML file or returning a System instance (and respective associated domain objects);
5) implement a text-based user interface for scenario data input and log visualization;
6) implement a graphical user interface for scenario data input, simulation metrics selection, and log visualization.

At the time of this paper's writing, we had completed the first three iterations, implementing both a plain-text logger and a logger which records traces in the format interpreted by the Grasp player [9]. We now describe the latter as a case study for hsSim's interoperability potential. XML reading is left out of this paper due to paper length restrictions.

A. Interoperability: a case study with Grasp

Because of our low coupling design for the event logging (Section IV-E), it is straightforward to trace the simulation to the format interpreted by Grasp Player [9]. The Grasp logger is implemented exactly as if it were developed by an external team, who could even only have access to the core of hsSim as a library. We implement the Logger generic interface, instantiating its type variable with the String type, since we want the processing (visit) of events to return the text to be added to the Grasp trace (Fig. 7).

The visit methods are invoked when an event notification is received, via the update method (Listing 1). Invocation is done indirectly through the accept method, so the right visit method is automatically selected and called.

Along the visit methods, we implement the following mapping between hsSim events and Grasp trace content:

- arrival of Job $J_{i,k}$ (of AbstractTask τ_i)
  1) if τ_i is a Partition, **serverReplenished** $\boldsymbol{\tau_i}$ $\boldsymbol{c'_{i,k}}$
  2) if τ_i is a Task, **jobArrived** $\boldsymbol{J_{i,k}}$ $\boldsymbol{\tau_i}$
- obtention of processor by Job $J_{p,q}$ (of AbstractTask τ_p)[3]

[3]The obtention of the processor from idle by a job is implemented as a JobPreemptedEvent with a null preempted job.

Listing 2. GraspLogger excerpt (visit of a job release event)

```
public String visit(JobReleasedEvent e) {
    Job job = e.getJob();
    StringBuilder sb = new StringBuilder();
    sb.append("plot␣");
    sb.append(Integer.toString(e.getTime());
    sb.append('␣');
    if (job.getParentTask() instanceof Task) {
        sb.append("jobArrived␣");
        sb.append(job.toStringId());
        sb.append('␣');
        sb.append(job.getParentTask().toStringId());
    } else { //Partition
        sb.append("serverReplenished␣");
        sb.append(j.getParentTask().toStringId());
        sb.append('␣');
        sb.append(j.getRemainingCapacity());
    }
    sb.append(System.getProperty("line.separator"));
    return sb.toString();
}
```

  1) if τ_p is a Partition, **serverResumed** $\boldsymbol{\tau_p}$ (followed by a **jobResumed** log of the task job the partition was last executing)
  2) if τ_p is a Task, **jobResumed** $\boldsymbol{J_{p,q}}$
- preemption of Job $J_{i,k}$ by Job $J_{p,q}$
  1) if τ_i, τ_p are Partitions, **serverPreempted** $\boldsymbol{\tau_i}$ (followed by a **jobPreempted** log of the job the preempted partition was last executing, **serverResumed** $\boldsymbol{\tau_p}$, and a **jobResumed** log of the task job the preempting partition was last executing)
  2) if τ_i, τ_p are Tasks, **jobPreempted** $\boldsymbol{J_{i,k}}$ **-target** $\boldsymbol{J_{p,q}}$ (followed by **jobResumed** $\boldsymbol{J_{p,q}}$)
- completion of Job $J_{i,k}$
  1) if τ_i is a Partition, **serverDepleted** $\boldsymbol{\tau_i}$
  2) if τ_i is a Task, **jobCompleted** $\boldsymbol{J_{i,k}}$

In Listing 2, we can see, as an example, the visit method responsible for processing a JobReleasedEvent event.

Although hsSim's core supports an arbitrary number of levels, we now show for the sake of simplicity an example using the current implementation of GraspLogger with a 2-level hierarchical setting.

B. Example test

The Rate Monotonic is used to schedule partitions and to schedules tasks in each partition. This examples is of a sample from the experiments of [19], and the timing characteristics of the partitions are derived from those of the respective tasks by applying Shin and Lee's periodic resource model [7]:

- partition 1 has a capacity of 16 over a period of 75 and contains 4 tasks ($\tau_i = \langle C_i, T_i, D_i \rangle$): $\tau_1 = \langle 46, 500, 500 \rangle$, $\tau_2 = \langle 71, 1000, 1000 \rangle$, $\tau_3 = \langle 25, 1000, 1000 \rangle$, and $\tau_4 = \langle 29, 2000, 2000 \rangle$;
- partition 2 has a capacity of 5 over a period of 25 and contains 2 tasks: $\tau_5 = \langle 32, 250, 250 \rangle$ and $\tau_6 = \langle 67, 1000, 1000 \rangle$;
- partition 3 has a capacity of 5 over a period of 25 and contains 3 tasks: $\tau_7 = \langle 27, 250, 250 \rangle$, $\tau_8 = \langle 109, 2000, 2000 \rangle$, and $\tau_9 = \langle 53, 2000, 2000 \rangle$.

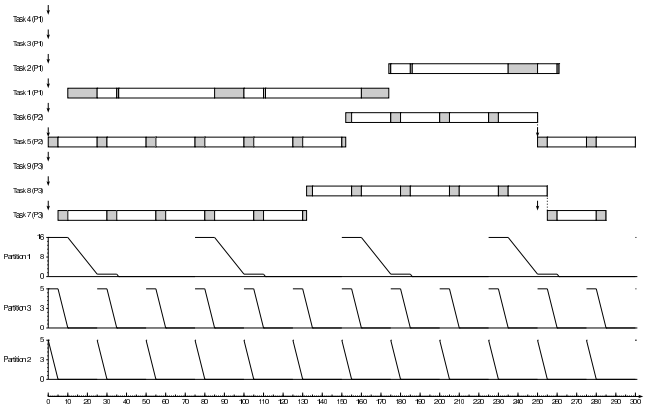

Fig. 8. Grasp Player output excerpt

For the trace generated by hsSim's GraspLogger (omitted due to paper length constraints), Grasp Player displays the graphical output partially pictured in Fig. 8. Partitions are traced as scheduling servers whose budget is consumed while it is active (independently of the execution of children tasks) and replenished at the beginning of each new period. The release, execution, preemption, and finishing of tasks are represented in a Gantt-like chart for each task.

VI. Conclusion and ongoing work

We have described the development of hsSim, an n-level hierarchical scheduling simulator. We emphasized the object-oriented analysis and design decisions, such as the careful application of design patterns, through which we pursued the purpose of yielding an open, reusable, extensible and interoperable tool. Applying the Composite and Strategy patterns allows implementing system operations independently from, respectively, the hierarchy's structure and size and the underlying scheduling algorithms; furthermore, the application of the Observer and Visitor patterns allows great flexibility to add new and diverse simulation loggers, since the simulation and logging aspects are decoupled from the domain concepts. We demonstrate this extensibility and interoperability features with a logger to trace the simulation to the format interpreted by Grasp Player.

Development continues after this submission, so more advances are expected in time for the WATERS 2012 event. Ongoing work includes opensourcing hsSim's code[4], and implementing more types of schedulers (namely those corresponding to scheduling servers, non-preemptive schedulers, cyclic executive and global/partitioned multiprocessor schedulers). Regarding multiprocessor support, much of these features' design (supporting homogeneous and heterogeneous multiprocessor platforms) is already done, but omitted from this paper. Further work planned for after supporting multiprocessor includes adding the option to recursively compute a partition's timing requirements from those of its children using compositional analysis [7], [20], [21] and paying further attention to the specificities of dealing with aperiodic/sporadic tasks.

We have an ongoing collaboration with the Cheddar team, and as such some of the advances made in the development of hsSim should be ported there to enhance its support to hierarchical scheduling and compositional analysis [10]–[12].

[4]http://code.google.com/p/hssim/

Acknowledgment

The authors would like to thank our SAPIENT project partners from Lab-STICC/UBO (Brest, France), especially Frank Singhoff, for useful discussions which helped improve the work presented in this paper.

References

[1] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *RTSS '98*, Madrid, Spain, 1998, pp. 4–13.

[2] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *IEEE Trans. Comput.*, 2011, to appear.

[3] J. Windsor and K. Hjortnaes, "Time and space partitioning in spacecraft avionics," in *SMC-IT 2009*, Jul. 2009, pp. 13–20.

[4] AEEC, "Avionics application software standard interface, part 1 - required services," Aeronautical Radio, Inc., ARINC Spec. 653P1-2, Mar. 2006.

[5] J. Rufino, J. Craveiro, and P. Verissimo, "Architecting robustness and timeliness in a new generation of aerospace systems," in *Architecting Dependable Systems VII*, ser. LNCS, A. Casimiro, R. de Lemos, and C. Gacek, Eds. Springer, Nov. 2010, vol. 6420, pp. 146–170.

[6] B. Kauer, P. Verissimo, and A. Bessani, "Recursive virtual machines for advanced security mechanisms," in *1st Int. Workshop on Dependability of Clouds, Data Centers and Virtual Computing Environments (DCDV 2011)*, Hong Kong, Jun. 2011.

[7] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS '03*, Cancun, Mexico, Dec. 2003.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1997.

[9] M. Holenderski, R. J. Bril, and J. J. Lukkien, "Grasp: Visualizing the behavior of hierarchical multiprocessor real-time systems," in *WATERS 2011*, Porto, Portugal, Jul. 2011.

[10] F. Singhoff, A. Plantec, P. Dissaux, and J. Legrand, "Investigating the usability of real-time scheduling theory with the Cheddar project," *Real-Time Syst.*, vol. 43, no. 3, pp. 259–295, Nov. 2009.

[11] J. Craveiro, J. Rufino, and F. Singhoff, "Architecture, mechanisms and scheduling analysis tool for multicore time- and space-partitioned systems," *ACM SIGBED Rev.*, vol. 8, no. 3, pp. 23–27, Sep. 2011, special issue of ECRTS 2011 WiP session, Porto, Portugal, July 2011.

[12] V. Gaudel, F. Singhoff, A. Plantec, S. Rubini, P. Dissaux, and J. Legrand, "An Ada design pattern recognition tool for AADL performance analysis," in *SIGAda '11*, Denver, CO, Nov. 2011, pp. 61–68.

[13] L. T. X. Phan, J. Lee, A. Easwaran, V. Ramaswamy, I. Lee, and O. Sokolsky, "CARTS: A tool for compositional analysis of real-time systems," in *3rd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS 2010)*, San Diego, CA, Nov. 2010.

[14] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using EDP resource models," in *RTSS '07*, Tucson, AZ, Dec. 2007.

[15] B. Nikolic, M. A. Awan, and S. M. Petters, "SPARTS: Simulator for Power Aware and Real-Time Systems," in *ICESS 2011*, Changsha, China, Nov. 2011.

[16] M. Gonzalez Harbour, J. J. Gutierrez Garcia, J. M. Drake Moyano, P. López Martínez, and J. C. Palencia Gutierrez, "Modeling distributed real-time systems with MAST 2," *J. Syst. Architect.*, 2012.

[17] Y. Matsubara, Y. Sano, S. Honda, and H. Takada, "An open-source flexible scheduling simulator for real-time applications," in *ISORC 2012*, Shenzhen, China, Apr. 2012.

[18] J. Carpenter, S. Funk, P. Holman, J. Anderson, and S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms," in *Handbook on Scheduling: Algorithms, Methods, and Models*, J. Y.-T. Leung, Ed. Chapman & Hall/CRC, 2004.

[19] J. P. Craveiro, J. Rosa, and J. Rufino, "Towards self-adaptive scheduling in time- and space-partitioned systems," in *RTSS 2011: WiP Session*, Vienna, Austria, Nov./Dec. 2011.

[20] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *ECRTS 2008*, Prague, Czech Rep., Jul. 2008.

[21] J. P. Craveiro and J. Rufino, "Heterogeneous multiprocessor compositional real-time scheduling," in *RTSOPS 2012*, Pisa, Italy, Jul. 2012.