Schedulability Analysis Using Uppaal: Herschel–Planck Case Study

Ruba ALMahasneh

Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, 2 Magyar tudósok krt., Budapest 1117, Hungary,
Email :mahasnehr@gsuite.tmit.bme.hu

- This proposed modeling framework for performing schedulability analysis by using Uppaal real-time model-checker .
- The frame work is inspired by a case study where schedulability analysis of a satellite system is performed.
- The framework assumes a single CPU hardware where a fixed priority preemptive scheduler is used in a combination with two resource sharing protocols and in addition voluntary task suspension is considered.

Goal of schedulability analysis

- **The goal of schedulability analysis is to check whether all tasks finish before their deadline.**
- Traditional approaches provide generic frameworks which assume worst case scenario where worst case execution time and blocking times are estimated and then worst case response times are calculated and compared w.r.t. deadlines. -pessimistic approach
- The idea this method is to base the schedulability analysis on a system model with more details, taking into account specifics of individual tasks. In particular this will allow a safe but far less pessimistic schedulability analysis to be settled using real-time model checking.

- The Herschel–Planck mission consists of two satellites: Herschel and Planck. The satellites have different scientific objectives and thus the sensor and actuator configurations differ, *but both satellites share the same computational architecture*

Model architecture

- Single processor, real-time operating system
- (RTEMS), basic software layer (BSW) and application software (ASW).

- Terma A/S has performed an extended worst case response time analysis
- **The goal of the study is to show that ASW tasks and BSW tasks are schedulable on a single processor with no deadline violations.**

- The tasks are carefully picked and timings aggregated and processed by **Terma** tool Schedule which performs worst case response time analysis as described and the **outcome** is *processor utilization and worst case response times (WCRT) for each task*. The system is schedulable when for every task i WCRTi is less than its Deadlinei, i.e. the task always finishes before its deadline.
- We use the index i as a **global** task identifier.

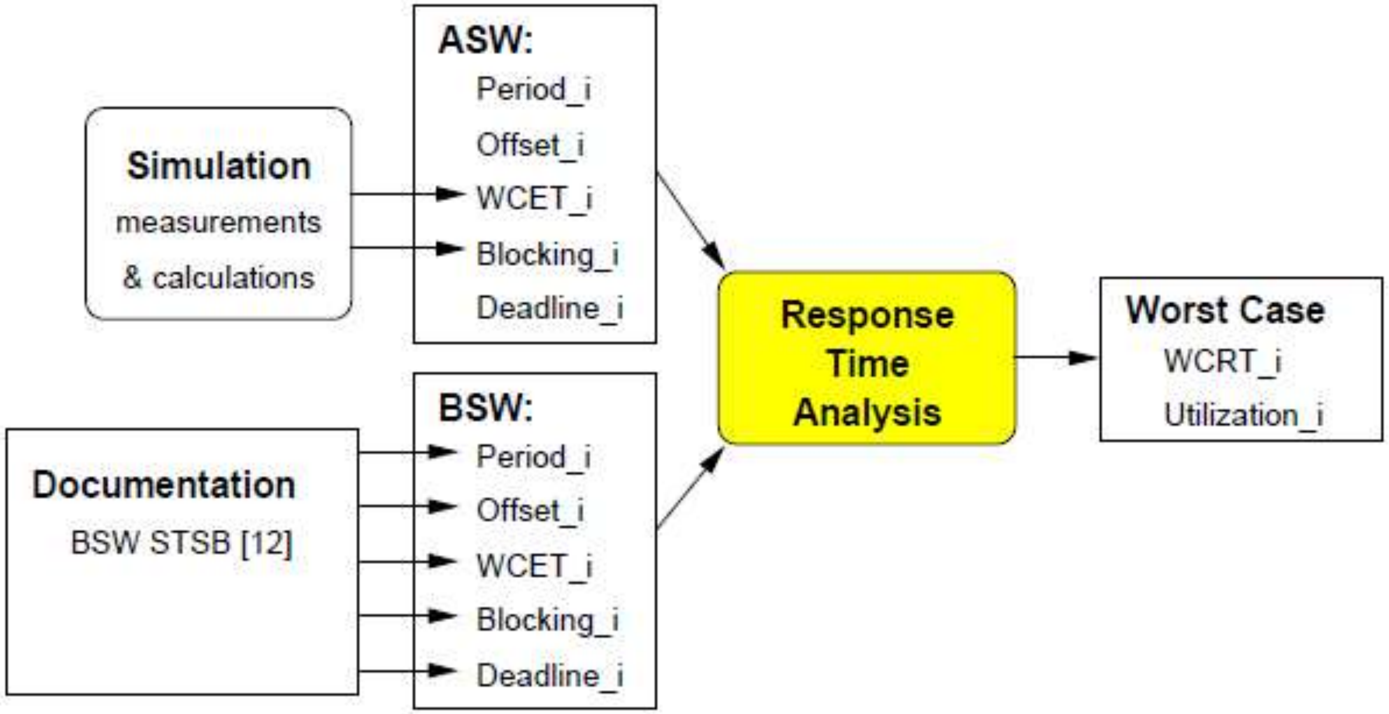


Fig. 1: Work-flow of schedulability analysis [10].

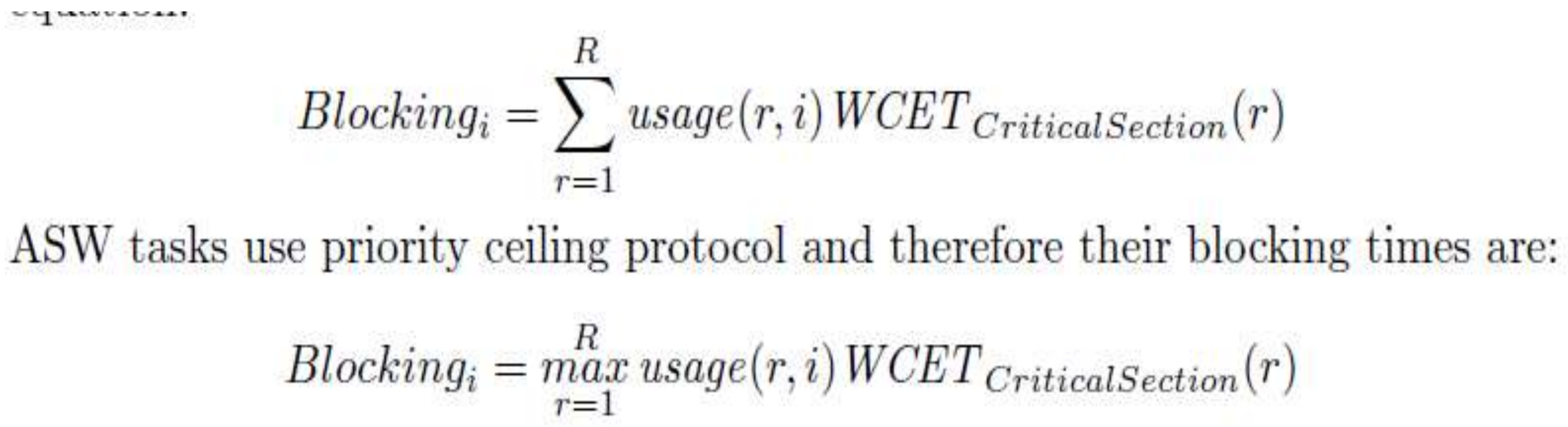
$$WCRT_i = \max_q Response_i(q)$$

To prove effectiveness of the theory

- In response time analysis, in order to prove schedulability it is enough to calculate worst case response times (WCRT) for every task and compare it with its deadline: if for every task the WCRT does not exceed the correspond deadline the system is schedulable.

$$WCRT_i = \max_q Response_i(q)$$

Blockingi denotes the blocking time when task i is waiting for a shared resource being occupied by a lower priority task. Blocking times calculation is specific to the resource sharing protocol used.



$$Blocking_i = \sum_{r=1}^{R} usage(r,i)\, WCET_{CriticalSection}(r)$$

ASW tasks use priority ceiling protocol and therefore their blocking times are:

$$Blocking_i = \max_{r=1}^{R} usage(r,i)\, WCET_{CriticalSection}(r)$$

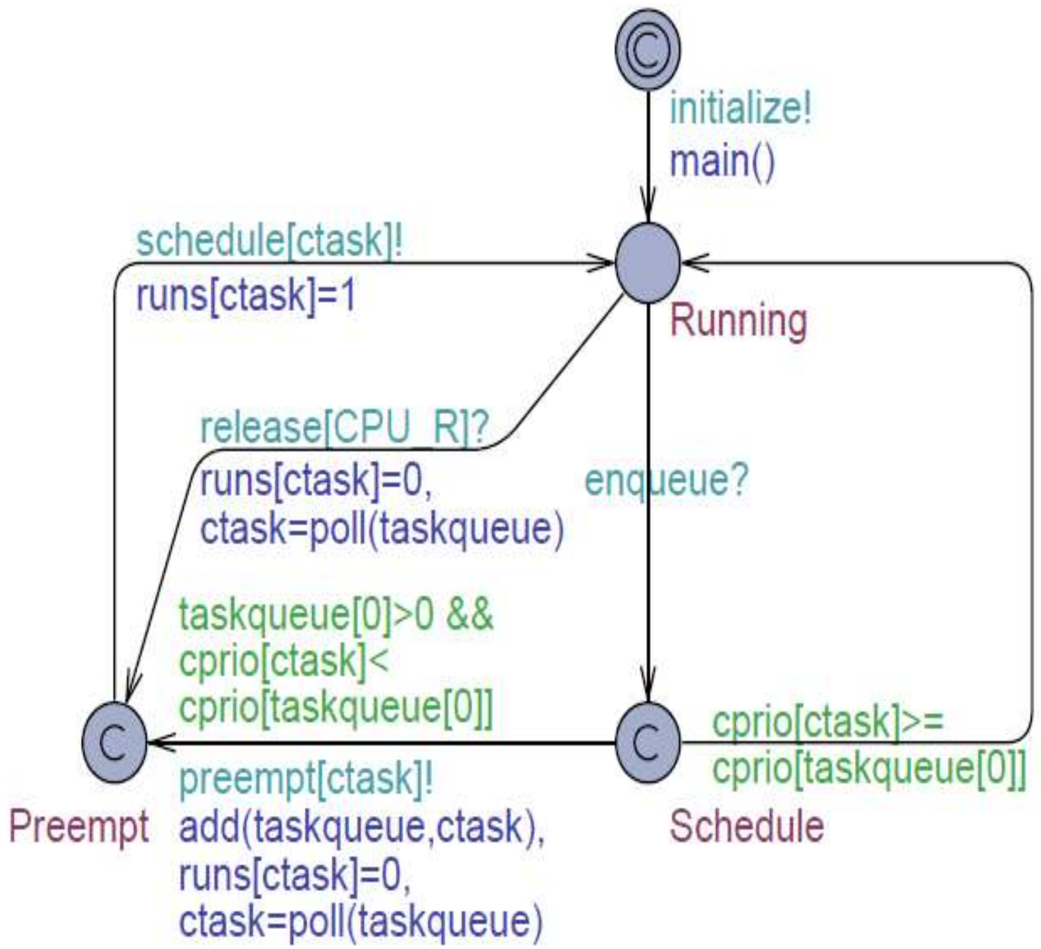
Some BSW tasks are periodic and some sporadic, but we simplify the model by considering all BSWtasks as **periodic**. ASWtasks are started by periodic task MainCycle.

Priority ceiling protocol

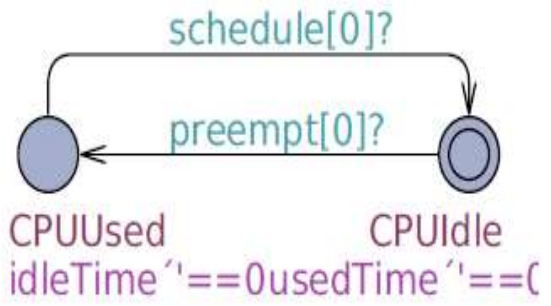
- It is a synchronization protocol for shared resources to avoid unbounded priority inversion and mutual deadlock due to wrong nesting of critical sections

From schedulability to reachability

- The main idea is to translate schedulability analysis problem into a reachability problem for timed automata and use the real-time model-checker Uppaal to find worst case blocking and response times and check whether all the deadlines are met.



(a) Template for CPU scheduler.



(b) Idle task model.

Processor Scheduler

- In the beginning **Scheduler initializes** the system (computes the current task priorities by computing default priority based on id and starts the tasks with zero offset) and in location *Running*
- waits for tasks to become **ready** or **current task to release the CPU resource.**
- When some task becomes ready, it adds itself to the task queue and signals on enqueue channel, thus moving Scheduler to location Schedule.
- **From location Schedule, the Scheduler compares the priority of a current task cprio[ctask] with highest priority in the queue cprio[taskqueue[0]] and either returns to**
- **Running (nothing to reschedule) or preempts the current task ctask, puts it into taskqueue and schedules the highest priority task from taskqueue.**

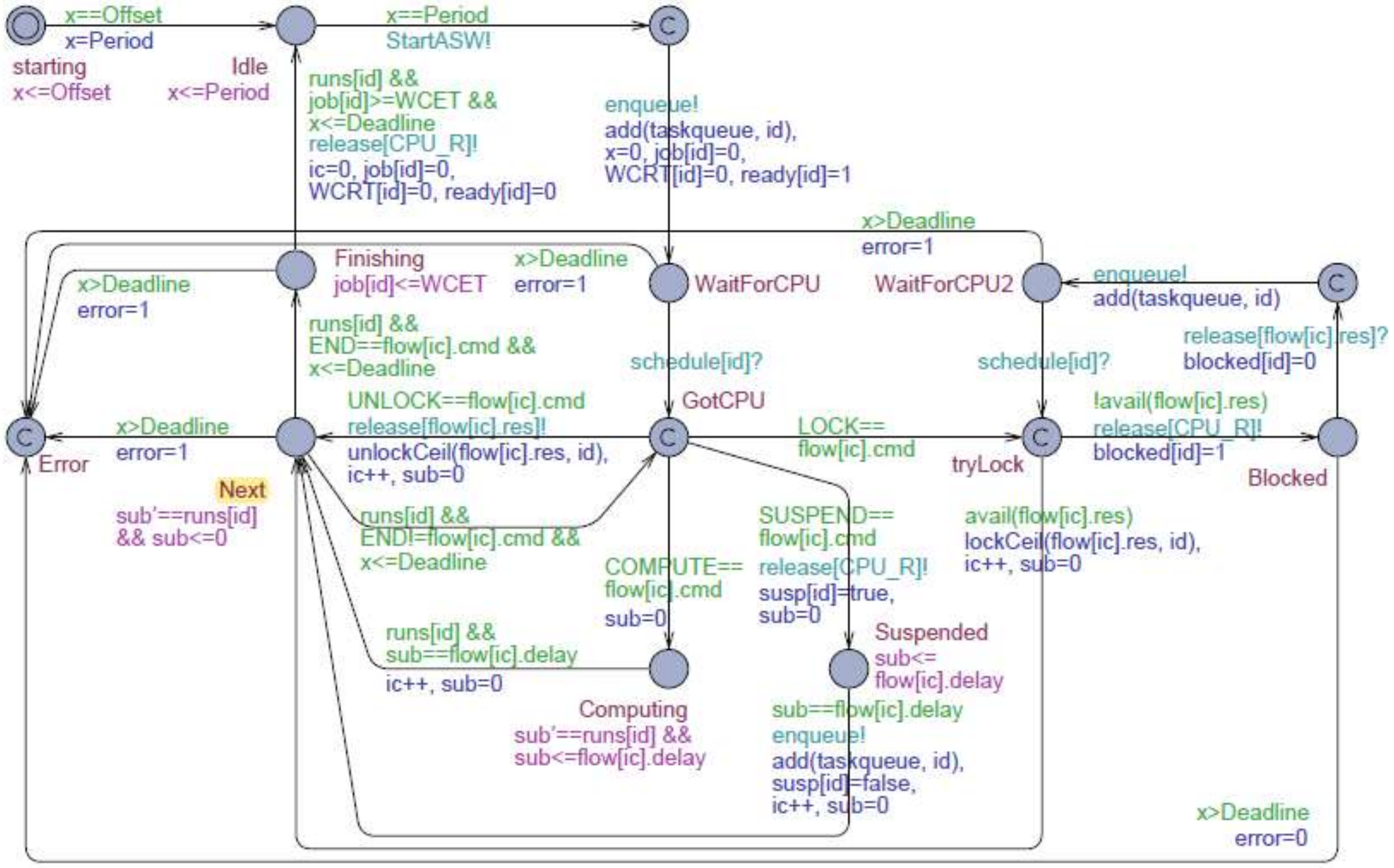
- The processor is released by a signal **release[CPU R],** in which case the Scheduler pulls the highest priority task from taskqueue and optionally notifies it with broadcast synchronisation on channel schedule (the sending is performed always in non-blocking way as receivers may ignore broadcast synchronisations).
- The taskqueue always contains at least one ready task

IdleTask.

- Figure 2b
- shows how IdleTask reacts to Scheduler events and computes the CPU usage time
- with stopwatch usedTime, the total CPU load is then calculated as

$$\frac{\text{usedTime}}{\text{globalTime}}$$

MainCycle


x==Offset
x=Period
starting
x<=Offset
Idle
x<=Period
x==Period
StartASW!
runs[id] &&
job[id]>=WCET &&
x<=Deadline
release[CPU_R]!
ic=0, job[id]=0,
WCRT[id]=0, ready[id]=0
enqueue!
add(taskqueue, id),
x=0, job[id]=0,
WCRT[id]=0, ready[id]=1
x>Deadline
error=1
Finishing
job[id]<=WCET
x>Deadline
error=1
WaitForCPU
WaitForCPU2
enqueue!
add(taskqueue, id)
x>Deadline
error=1
runs[id] &&
END==flow[ic].cmd &&
x<=Deadline
schedule[id]?
schedule[id]?
release[flow[ic].res]?
blocked[id]=0
UNLOCK==flow[ic].cmd
release[flow[ic].res]!
unlockCeil(flow[ic].res, id),
ic++, sub=0
GotCPU
LOCK==
flow[ic].cmd
!avail(flow[ic].res)
release[CPU_R]!
blocked[id]=1
x>Deadline
error=1
Error
Next
sub'==runs[id]
&& sub<=0
tryLock
Blocked
runs[id] &&
END!=flow[ic].cmd &&
x<=Deadline
SUSPEND==
flow[ic].cmd
release[CPU_R]!
susp[id]=true,
sub=0
avail(flow[ic].res)
lockCeil(flow[ic].res, id),
ic++, sub=0
COMPUTE==
flow[ic].cmd
sub=0
runs[id] &&
sub==flow[ic].delay
ic++, sub=0
Suspended
sub<=
flow[ic].delay
Computing
sub'==runs[id] &&
sub<=flow[ic].delay
sub==flow[ic].delay
enqueue!
add(taskqueue, id),
susp[id]=false,
ic++, sub=0
x>Deadline
error=0

Listing 1.2: Resource locking based on two different protocols.

```
/** Check if the resource is available: */
bool avail(resid_t res) { return (owner[res]==0); }
/** Lock the resource based on priority ceiling protocol: */
void lockCeil(resid_t res, taskid_t task) {
    owner[res] = task; // mark resource occupied by task
    cprio[task] = ceiling[res]; // assume the priority of resource
}
/** Unlock the resource based on priority ceiling protocol: */
void unlockCeil(resid_t res, taskid_t task){
    owner[res] = 0; // mark resource as released
    cprio[task] = def_prio(task); // return to default priority
}
/** Boost the priority of resource owner based on priority inheritance protocol: */
void boostPrio(resid_t res, taskid_t task) {
    if (cprio[owner[res]] <= def_prio(task)) {
        cprio[owner[res]] = def_prio(task)+1;
        sort(taskqueue);
    }
}
```

- At first **MainCycle** waits for Offset time to elapse/pass and moves to location Idle by setting the <u>*clock x to Period*</u>. Then the process is forced to leave Idle location <u>**immediately**</u>, to signal other ASW tasks, add itself to the ready task queue and arrive to location **WaitForCPU**. When **MainCycle** receives notification from **scheduler** it moves to location **GotCPU** and starts processing <u>*commands from the flow array*</u>. There are four types of commands:

Types of Available commands

- 1. **LOCK** is executed from location tryLock where the process attempts to acquire the resource. if the resource is not available and **retries** by adding itself to the processor queue again when resource is released. It continues to location **Next** by locking the resource if the resource is available.
- 2. **UNLOCK** simply releases the resource and moves on to location Next.

Types of Available commands

3. **SUSPEND** *releases the processor* for specified **amount of time** (suspend time) adds itself to the queue and moves to location Next. The task progress clock job[id] is not increasing but the response measurement clock WCRT[id] is.

That's why we can see **release[CPU_R]**

4. **COMPUTE** makes the task stay in location Computing for **at least the specified duration** of pure running time, i.e. the clock sub is stopped whenever the task is preempted and runs[id] is set to 0. Once the required amount of CPU time is consumed, the process moves on to location Next.

Types of Available commands

- From location **Next**, the process **is forced by runs[id] invariant to continue with the next operation**: if it is not the END and it is running, then it moves back to
- **GotCPU** to process next operation, and it moves to Finishing if it's the **END**. In **Finishing** location the process consumed **CPU** for the remaining time so that
- complete **WCET** is exhausted and then it moves back to **Idle**. From locations **Next** and Finishing the outgoing edges are constrained to check whether the deadline has been reached since the last task release (when x was set to 0), and edges force the process into Error ***location if x > Deadline.***

Conclusion

- Uppaal model-checker can be applied for schedulability analysis of a system with single CPU, fixed priorities preemptive scheduler, mixture of periodic tasks and tasks with dependencies, and mixed resource sharing protocols.
- Worst case response times (WCRT), blocking times and CPU utilization are estimated and checked model-checker
- We can model patterns use stopwatches in a simple and intuitive way
- A break-through in verification scalability for large systems (more than 30 tasks) is achieved by employing sweep-line method.
- Even better control over verification resources can be achieved by carefully designing progress measure.