Schedulability of Herschel/Planck Revisited using Statistical Model Checking

Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis

Department of Computer Science
Aalborg University

October 17, 2012

Outline

Herschel-Planck Scientific Mission at ESA

- Attitude and Orbit Control System software.
- Terma A/S: Steen Ulrik Palm, Jan Storbank Pedersen, Poul Hougaard.

Satellite Architecture

ASW	Application software performs attitude and orbit control, handles tele-commands, fault detection isolation and recovery.
BSW	Basic software is responsible for low level communication and scheduling periodic events.
RTEMS	Real-time operating system, fixed priority preemptive scheduler.
Hardware	Single processor, a few communication buses, sensors and actuators.

Problem Statement

- Single CPU, fixed priority preemptive scheduler.
- Mixture of 32 tasks: periodic, sporadic with dependencies.
- Mixed resource sharing (make priorities dynamic):
  - BSW tasks use priority *inheritance* protocol.
  - ASW tasks use priority *ceiling* protocol.

At Terma A/S:

- 1 out of 4 configurations *could not be proved schedulable* using schedulability analysis by Alan Burns.
- Neither simulation nor execution show any problems.

At Aalborg:

- The techniques are conservative at assuming worst case.
- *Hypothesis: model more details and achieve more accurate analysis using symbolic reachability and simulations.*

Progress Summary

ISoLA 2010:

- Detailed task model with both resource sharing protocols.
- Deterministic behavior assuming exact execution times.
- Verification memory reduction using *sweep-line* method.
- No deadline violation found.
- Estimated *response* and *blocking* times.

ISoLA 2012:

- Remodelled priorities using *broadcast channels*.
- Relaxed execution times to *[BCET,WCET]*.
- Full state space exploration, some deadline violations.
- Used UPPAAL SMC to show some non-schedulability.
- Extra: sporadic tasks break schedulability even for WCET.

Approach: combination of Symbolic and Statistical

Symbolic analysis:

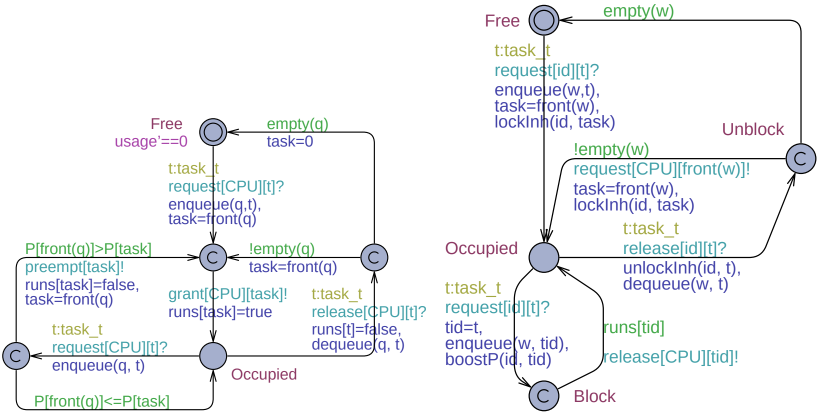
- Preemptive scheduler requires *stop-watches*.
- Exact reachability of stop-watch automata is *undecidable*.
- UPPAAL provides *over-approximation* for stop-watches.
- $\Rightarrow$ symbolic analysis may give spurious errors, but still suitable for *proving safety/schedulability*.

Statistical analysis:

- can show *presence of errors* but not absence.
- $\Rightarrow$ suitable for *disproving schedulability*.

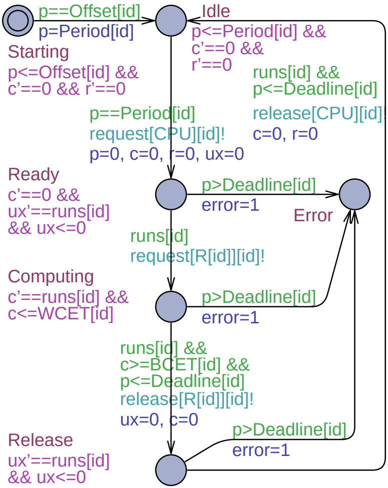
$f = \mathrm{BCET}/\mathrm{WCET}$:	0-71%	72-86%	87-89%	90-100%
Symbolic MC:	maybe	maybe	n/a	**Safe**
Statistical MC:	**Unsafe**	maybe	maybe	maybe

Conceptual Example: Scheduler and Resource

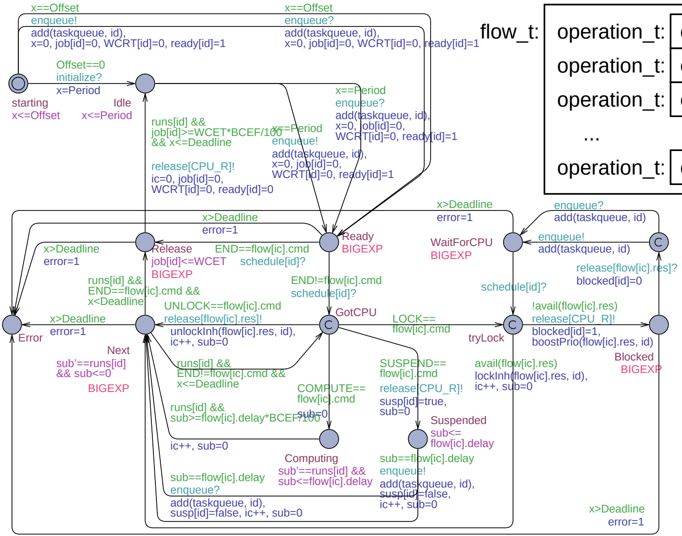


Conceptual Example: Task Model

```
const int NRTASK = 3; // # of tasks
const int NRRES = 1; // # of resources
typedef int [1, NRTASK] task_t;
typedef int [1, NRRES] res_t;
const int f=80; // fraction of WCET, in %
 int Period[task_t]   = {  100,  100, 100 };
 int Offset[task_t]   = {   20,    0,  10 };
 int WCET[task_t]     = {   15,   25,  40 };
 int BCET[task_t]     = { WCET[1]*f/100,
     WCET[1]*f/100, WCET[1]*f/100 };
 int Deadline[task_t] = {   20,   40,  70 };
res_t  R[task_t]      = {    1,    1,   1 };
 int P[task_t] = { 3, 2, 1 }; // priorities
bool runs[task_t] = { 0, 0, 0 };
bool error = false; // global variable
```



Satellite Software Task Template



flow_t:

operation_t:	optype_t	resid_t	time_t
operation_t:	optype_t	resid_t	time_t
operation_t:	optype_t	resid_t	time_t
...	...	...	...
operation_t:	optype_t	resid_t	time_t

Primary Functions Flow in UPPAAL

```
const ASWFlow_t PF_f = { // Primary Functions:
    { LOCK,    Icb_R, 0 },                 //  0) ------ Data processing
    { COMPUTE, CPU_R, 1600-1200 }, // 1) computing with Icb_R
    { SUSPEND, CPU_R, 1200 },      //  2)   suspended with Icb_R
    { UNLOCK, Icb_R, 0 },              //  3)
    { COMPUTE, CPU_R, 20577-(1600-1200) }, // 4) computing w/o Icb_R
    { COMPUTE, CPU_R, 3440 },      //  5) ------ Guidance
    { LOCK,    Sgm_R, 0 },             //  6) ------ Attitude determination
    { COMPUTE, CPU_R, 1218-121 }, // 7) computing with Sgm_R
    { SUSPEND, CPU_R, 121 },       //  8)   suspended with Sgm_R
    { UNLOCK, Sgm_R, 0 },              //  9)
    { COMPUTE, CPU_R, 3751-(1218-121) }, //10) computing w/o Sgm_R
    { COMPUTE, CPU_R, 42 },         //11) ------ Perform extra checks
    { LOCK,    PmReq_R,0 },           //12) ------ SCM controller
    { COMPUTE, CPU_R, 3300-1650 }, //13) computing with PmReq_R
    { SUSPEND, CPU_R, 1650 },       //14)   suspended with PmReq_R
    { UNLOCK, PmReq_R, 0 },           //15)
    { COMPUTE, CPU_R, 3479-(3300-1650) },//16) comp. w/o PmReq_R
    { COMPUTE, CPU_R, 2752 },       //17) ------ Command RWL
    { END,     CPU_R, 0 }               //18) finished
};
```

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

Verification Resources

A☐ not error

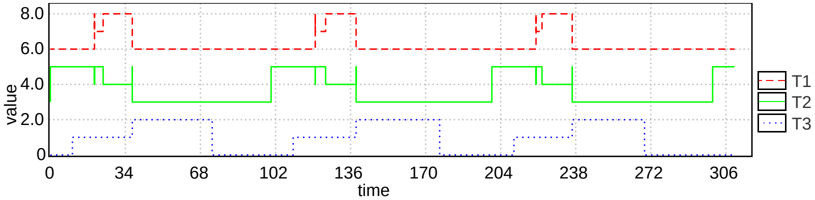
limit cycle	$f = 100\%$ states	mem	time	$f = 95\%$ states	mem	time
1	0.001	51.2	1.47	0.5	83.0	15:03
2	0.003	53.7	2.45	0.8	96.8	27:00
4	0.005	54.5	4.62	1.5	97.2	48:02
8	0.010	54.7	8.48	2.8	97.8	1:28:45
16	0.020	55.3	16.11	5.4	112.0	2:45:52
∞	0.196	58.8	2:39.64	52.7	553.9	27:05:07

limit cycle	$f = 90\%$ states	mem	time	$f = 86\%$ states	mem	time
1	1.5	124.1	1:22:43	3.3	186.9	6:39:47
2	2.4	139.7	2:09:15	5.3	198.7	9:14:59
4	4.4	138.3	3:48:40	9.2	274.6	14:12:57
8	9.1	156.5	8:38:42	18.2	364.6	28:35:32
16	17.8	176.0	16:42:05	35.4	520.4	44:06:57
∞	181.9	1682.2	147:23:25	**pos.unsafe**		99:07:56

ID	Task	Specification			WCRT			
		Period	WCET	Deadline	Terma	$f = 100\%$	$f = 95\%$	$f = 90\%$
1	RTEMS_RTC	10.000	0.013	1.000	0.050	0.013	0.013	0.013
2	AswSync_SyncPulseIsr	250.000	0.070	1.000	0.120	0.083	0.083	0.083
3	Hk_SamplerIsr	125.000	0.070	1.000	0.120	0.070	0.070	0.070
4	SwCyc_CycStartIsr	250.000	0.200	1.000	0.320	0.103	0.103	0.103
5	SwCyc_CycEndIsr	250.000	0.100	1.000	0.220	0.113	0.113	0.113
6	Rt1553_Isr	15.625	0.070	1.000	0.290	0.173	0.173	0.173
7	Bc1553_Isr	20.000	0.070	1.000	0.360	0.243	0.243	0.243
8	Spw_Isr	39.000	0.070	2.000	0.430	0.313	0.313	0.313
9	Obdh_Isr	250.000	0.070	2.000	0.500	0.383	0.383	0.383
10	RtSdb_P_1	15.625	0.150	15.625	4.330	0.533	0.533	0.533
11	RtSdb_P_2	125.000	0.400	15.625	4.870	0.933	0.933	0.933
12	RtSdb_P_3	250.000	0.170	15.625	5.110	1.103	1.103	1.103
13	(no task, this ID is reserved for priority ceiling)							
14	**FdirEvents**	250.000	5.000	230.220	7.180	5.553	5.553	5.553
15	**NominalEvents_1**	250.000	0.720	230.220	7.900	6.273	6.273	6.273
16	**MainCycle**	250.000	0.400	230.220	8.370	6.273	6.273	6.273
17	HkSampler_P_2	125.000	0.500	62.500	11.960	5.380	7.350	8.153
18	HkSampler_P_1	250.000	6.000	62.500	18.460	11.615	13.653	14.153
19	Acb_P	250.000	6.000	50.000	24.680	6.473	6.473	6.473
20	IoCyc_P	250.000	3.000	50.000	27.820	9.473	9.473	9.473
21	**PrimaryF**	250.000	34.050	**59.600**	**65.47**	**54.115**	**56.382**	**58.586**
22	**RCSControlF**	250.000	4.070	239.600	76.040	53.994	56.943	58.095
23	Obt_P	1000.000	1.100	100.000	74.720	2.503	2.513	2.523
24	Hk_P	250.000	2.750	250.000	6.800	4.953	4.963	4.973
25	StsMon_P	250.000	3.300	125.000	85.050	17.863	27.935	28.086
26	TmGen_P	250.000	4.860	250.000	77.650	9.813	9.823	9.833
27	Sgm_P	250.000	4.020	250.000	18.680	14.796	14.880	14.973
28	TcRouter_P	250.000	0.500	250.000	19.310	11.896	11.906	14.442
29	Cmd_P	250.000	14.000	250.000	114.920	94.346	99.607	101.563
30	**NominalEvents_2**	250.000	1.780	230.220	102.760	65.177	69.612	72.235
31	**SecondaryF_1**	250.000	20.960	189.600	141.550	110.666	114.921	122.140
32	**SecondaryF_2**	250.000	39.690	230.220	204.050	154.556	162.177	165.103
33	Bkgnd_P	250.000	0.200	250.000	154.090	15.046	139.712	147.160

SMC: Simulating Conceptual Model

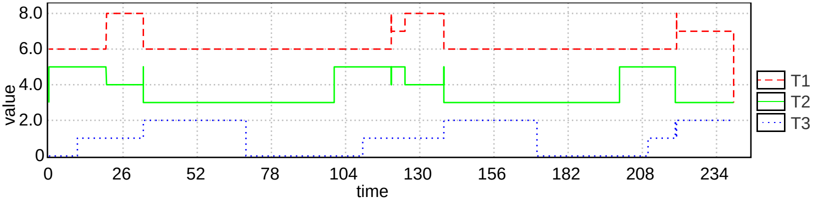
```
simulate 1000 [<=300] {
   (T(1).Ready+T(1).Computing+T(1).Release+runs[1]-2*T(1)
   (T(2).Ready+T(2).Computing+T(2).Release+runs[2]-2*T(1)
   (T(3).Ready+T(3).Computing+T(3).Release+runs[3]-2*T(1)
} :1: error
```



Normal run using $f = 80\%$.

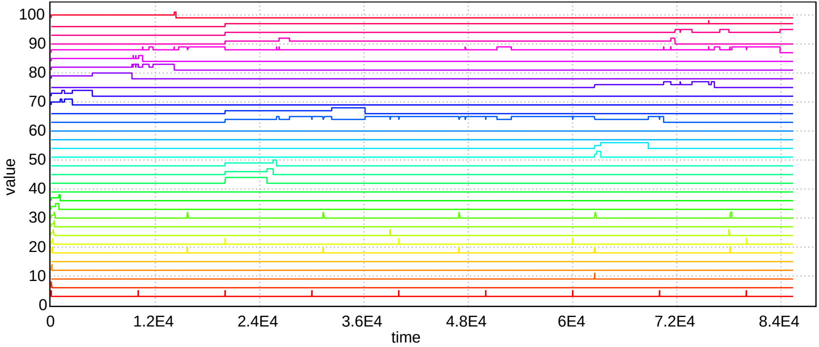
SMC: Simulating Conceptual Model

```
simulate 1000 [<=300] {
  (T(1).Ready+T(1).Computing+T(1).Release+runs[1]-2*T(1)
  (T(2).Ready+T(2).Computing+T(2).Release+runs[2]-2*T(1)
  (T(3).Ready+T(3).Computing+T(3).Release+runs[3]-2*T(1)
} :1: error
```

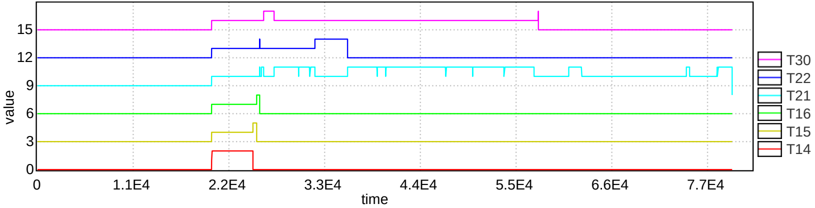


Failed run using $f = 79\%$.

Successful simulation run with $f = 90\%$:



Deadline violation with $f = 50\%$:



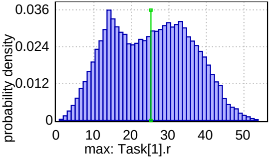
SMC of Herschel Model

```
Pr[<=LIMIT*250000](<> error)
```

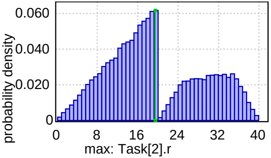
Limit cycles	f %	SMC parameters α	ε	Total traces, #	Error traces #	Probability	Earliest Error cycle	offset	Verification time
1	0	0.0100	0.005	105967	1928	0.018194	0	79600.0	1:58:06
1	50	0.0100	0.005	105967	753	0.007106	0	79600.0	2:00:52
1	60	0.0100	0.005	105967	13	0.000123	0	79778.3	2:01:18
1	62	0.0005	0.002	1036757	34	0.000033	0	79616.4	19:52:22
160	63	0.0100	0.05	1060	177	0.166981	0	81531.6	2:47:03
160	64	0.0100	0.05	1060	118	0.111321	1	79803.0	2:55:13
160	65	0.0500	0.05	738	57	0.077236	3	79648.0	2:06:55
160	66	0.0100	0.05	1060	60	0.056604	2	82504.0	2:62:44
160	67	0.0100	0.05	1060	26	0.024528	1	79789.0	2:64:20
160	68	0.0100	0.05	1060	3	0.002830	67	81000.0	2:67:08
640	69	0.0100	0.05	1060	8	0.007547	114	80000.0	12:23:00
640	70	0.0100	0.05	1060	3	0.002830	6	88070.0	12:30:49
1280	71	0.0100	0.05	1060	2	0.001887	458	80000.0	25:19:35

SMC: Response Times in Conceptual Model (f=0%)

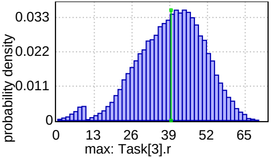
```
E[<=200; 50000] (max: T(1).r)
E[<=200; 50000] (max: T(2).r)
E[<=200; 50000] (max: T(3).r)
```



(a) Task T1.



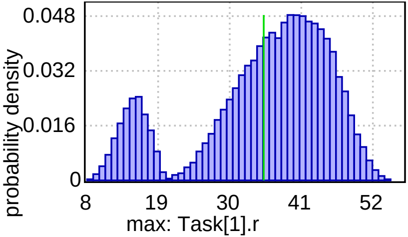
(b) Task T2.



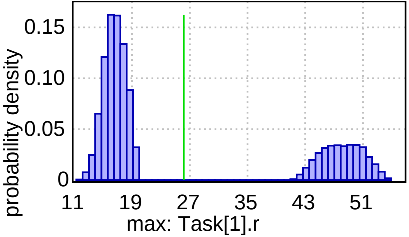
(c) Task T3.

f=0% (BCET=0), T1 violates deadline at 20.

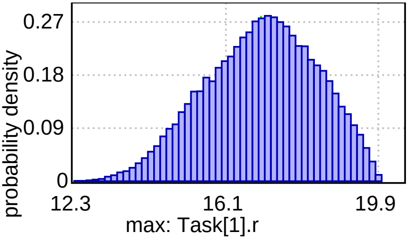
SMC: Response Times of T1



$f = 50\%$ (not safe).



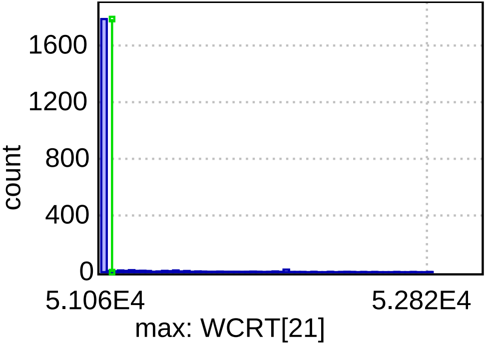
$f = 79\%$ (not safe).



$f = 80\%$ (seem OK).

Estimating WCRT for Herschel

`E[<=LIMIT*250000; 2000](max: WCRT[21])`



Plot for $f = \mathrm{BCET}/\mathrm{WCET} = 90\%$

Conclusions

- Model-based development allows more details formalized.
- MC for schedulability: UPPAAL is special
    - symbolic semantics for dense time
    - stop-watches (and much more in SMC)
    - clock difference diagrams (CDD vs. DBM)
    - sweep-line method
    - stochastic semantics for SMC
    - visual modeling & feedback
- Takes more memory and time, but OK for fixed systems.
- Sporadic tasks only in SMC for now.

Summary of Techniques Used

- Modeling:
  - Timed automata with clocks to express time constraints.
  - Stop-watches to track task progress.
  - Functions to implement resource sharing protocols.
  - Data structures to specify sequences of task operations.
- Symbolic model checking:
  - Exhaustive exploration of entire model state space.
  - Verification memory saving via sweep-line & CDD.
  - WCRT estimation using supremum query.
  - Schedule simulation and visualization with Gantt chart.
- Statistical model checking:
  - A lot of bounded concrete runs (disproving schedulability)
  - WCRT estimation via probability density over clock values.
  - Trace visualization via simulate query.

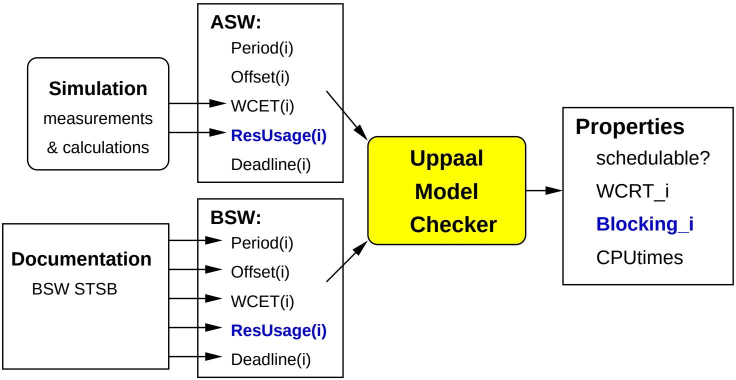
[all of the above is implemented in UPPAAL at uppaal.org]

Thank You
for attention

Gantt Chart Declaration

```
gantt {
  T(i:taskid_t):
    (ready[i] && !runs[i]) -> 1,// green: ready
    (ready[i] && runs[i]) -> 2, // blue:  running
    (blocked[i]) -> 0,          // red:   blocked
    susp[i] -> 9;               // cyan:  suspended
  R(i:resid_t):
    (owner[i]>0 && runs[owner[i]]) -> 2,  // blue: locked and actively used
    (owner[i]>0 && !runs[owner[i]] && !susp[owner[i]]) -> 1, // green: locked
          but preempted
    (owner[i]>0 && susp[owner[i]]) -> 9;  // cyan: locked and suspended
}
```

Sweep-Line Method via Progress Measure




```
const int CYCLE = 250*1000;
const int CYCLELIMIT = 3;
 int cycle = 1;
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
system Scheduler, Bkgnd_P < secondF_2 < secondF_1 < NominalEvents_2 <
    Cmd_P < TcRouter_P < Sgm_P < TmGen_P < StsMon_P < Hk_P <
    Obt_P < rCSControlF < primaryF < IoCyc_P < Acb_P < HkSampler_P_1 <
```