Relaxing Scheduling Constraints for Systems with Logical Execution Time Specifications

embedded Software & Systems Research Center

Patricia Derler, Stefan Resmerita

UNIVERSITÄT SALZBURG

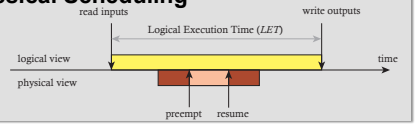
Logical Execution Time

Values available at the beginning of the LET are used for the computation and outputs are made available at the end of the LET.

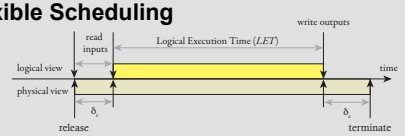
- time- and value-determinism, i.e. the same input produces the same output at the same points in time
- Specified at the model level independent from the functionality
- Execution = Simulation

e.g. Giotto[1], TDL[2], HTL[3], FTOS[4] …

Classical Scheduling



Flexible Scheduling



Runtime System

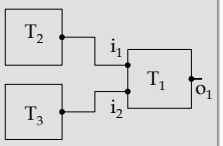
For every time-triggered task T:

- Update outputs of T at the end of the LET of T
- Update input ports of T
  - →from sensors at the beginning of the LET of T
  - →from other tasks any time except during the LET of T
    - from other TT tasks at the closest LET end of those tasks
    - from other ET tasks at the end of their execution
- Release tasks for execution

Advantages

- Enlarging the search space for feasible schedules that achieve time-safety. In particular, an application that is declared **unschedulable** under the classical constraints may be declared time-safe by using the relaxed constraints.
- If the system has **event-triggered** tasks executed in the background, then the **response times** of such events can be shorter with the relaxed constraints.
- If the system has high priority event tasks that may preempt time-triggered tasks, using the relaxed constraints reduces the risk of **missing LET deadlines** or reduces the number of missed LET deadlines.

Black-box view:



White-box view:

Syntactical information: Minimum execution time between release and reading inputs/writing outputs and task termination

```
T_Implementation() {
    initialize();
    readState();
    ...
    v1 = i1 + ...
    ...
    v2 = i2 + ...
    ...
    o1 = ...
    computeState();
}
```

(δ_{min1}, δ_{min2}, δ_{min3})

Inspired by PTIDES [5]

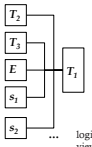
Semantical information: Conditional execution of code, hidden timing information

```
T_Implementation() {
    counter++;
    ...
    if (counter % 2
            == 0) {
        ...
        v2 = i2 + ...
        ...
    }
    ...
}
```

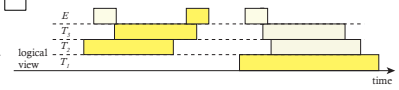
Scheduling Parameters: Minimum preemption time by higher priority tasks

Timing Predictability of Inputs: Time-triggered tasks produce outputs at LET-ends
Event-triggered tasks have minimum execution times and minimum interarrival times

Scheduling of a Task T_1



A Task T_1 reads from two time-triggered tasks T_2 and T_3, from an event-triggered task E and sensors, e.g. s_1 and s_2.



Definitions:

P^U … set of all ports that read from a task U

P^S … set of all ports that read from sensors

for the i^{th} execution of a task T

$t^i_{LS}(T)$ … start of the LET

$t^i_{LE}(T)$ … end of the LET

$t^i_r(T)$ … release time

$\delta_{min}(T, p)$ … minimum execution time between releasing the task and reading inputs from port p

Static Scheduling Bounds

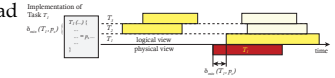
$$t^{\rho,i}_{Le}(U,T) = \begin{cases} 0 & \text{, if } t^0_{Le}(U) > t^i_{Ls}(T) \\ \max_{j=0,1,\dots} \left\{ t^j_{Le}(U) \;\middle|\; t^j_{Le}(U) \le t^i_{Ls}(T) \right\} & \\ & \text{otherwise} \end{cases}$$

$$\delta_{min}(T, P^U) = \begin{cases} \min\{\delta_{min}(T,p) | p \in P^U(T)\} & \text{, if } P^U(T) \neq \emptyset \\ +\infty & \text{, otherwise} \end{cases}$$

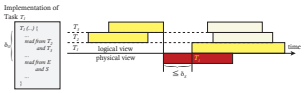
$$t^i_r(T) = \max \Big\{ 0,\; t^i_{Ls}(T) - \delta_{min}(T, P^S),\; \max_{U \in \mathbf{T}} \left\{ t^{\rho,i}_{Le}(U,T) - \delta_{min}(T, P^U) \right\} \Big\}$$

T_1 can start execution before the start of the LET.

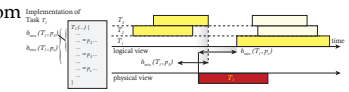
1. No inputs are read before the LET start



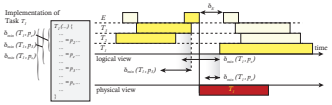
2. Start task after TT tasks finished execution



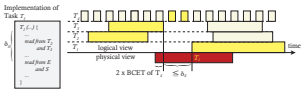
3. Read inputs from TT tasks after they finished execution



4. Consider event interarrival times



5. Consider preemption



Dynamic Scheduling Bounds

A task can be released at $t^i_r(T)$ if $\forall E \in \mathbf{E}_{\rightarrow T}$

(C1) the current time is at least $t^i_{r_{min}}(T)$ and

(C1a) the last time t when an $E \in \mathbf{E}_{\rightarrow T}$ produced an output before $t^i_{Ls}(T)$ is such that $t + \delta(E) > t^i_{Ls}(T)$

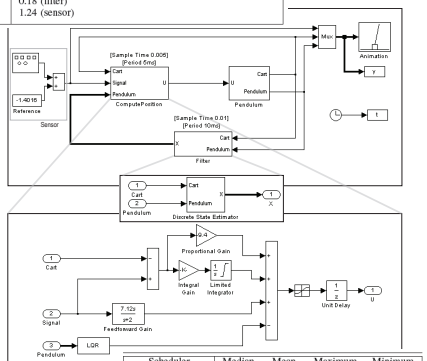
or

(C1b) $\forall E \in \mathbf{E}_{\rightarrow T}\ t^i_r(T) + \delta_{min}(T, P^E) \ge t^i_{Ls}(T)$ or

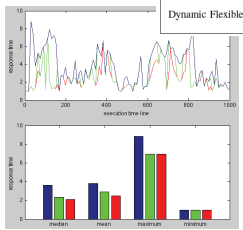
(C2) the current time is equal to or exceeded $t^i_{r_{max}}(T)$.

Example: Inverted Pendulum

Property	Task 1	Task 2	Task 3
Name	Computation Task	Filter Task	Sensor Task
Type	TT	TT	ET
Priority	1	2	3
LET	5	4	n/a
ϕ	3	0	n/a
π	10ms	5ms	n/a
δ	n/a	n/a	6/10
ET	1.58/3.26	1.72/2.25	0.95/2.22
δ_{min}	0.82 (any input) 0.18 (filter) 1.24 (sensor)	0.31 (any input)	



Scheduler	Median	Mean	Maximum	Minimum
Classical	3.6365	3.8216	8.8440	0.9673
Static Flexible	2.3372 (64%)	2.9071 (76%)	6.9318 (78%)	0.9673
Dynamic Flexible	2.1194 (58%)	2.4961 (65%)	6.9318 (78%)	0.9673



Further Work

- Flexible Scheduling with modes
- Analysis of the overhead introduced by flexible scheduling
- Implementation

References

[1] T. A. Henzinger, C. M. Kirsch, M. A. A. Sanvido, and W. Pree. From control models to real-time code using giotto. Control Systems Magazine, IEEE, 2003.
[2] Joseph Templ. Tdl - timing definition language 1.5 specication.
[3]Arkadeb Ghosal, Alberto Sangiovanni-Vincentelli, Christoph M. Kirsch, Thomas A. Henzinger, and Daniel Iercan. A hierarchical coordination language for interacting real-time tasks. In EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software, pages 132{141, New York, NY, USA, 2006. ACM.
[5] Jia Zou, Slobodan Matic, Edward A. Lee, Thomas Huining Feng, Patricia Derler. Execution Strategies for PTIDES, a Programming Model for Distributed Embedded Systems, to appear in RTAS, 2009.