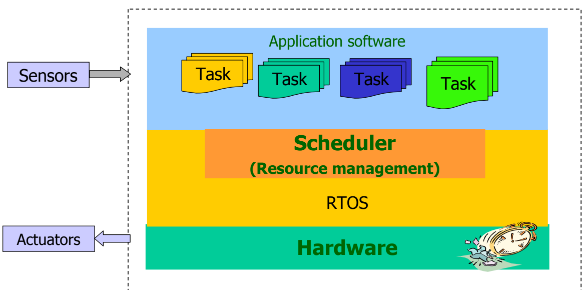
Schedulability Analysis of Timed Systems

with contributions from

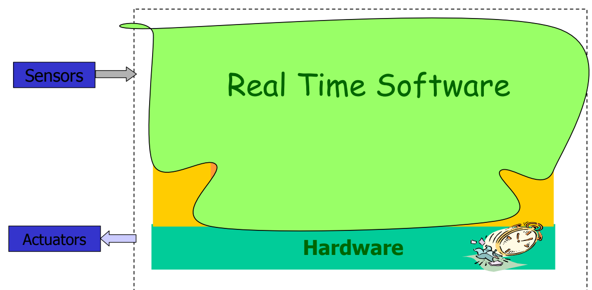
Tobias Amnell, Elena Fersma, John Håkansson,
Pavel Kracal, Leonid Mokrushine, Christer Nordström,
Paul Pettersson and Anders Wall

PROBLEM SETTING

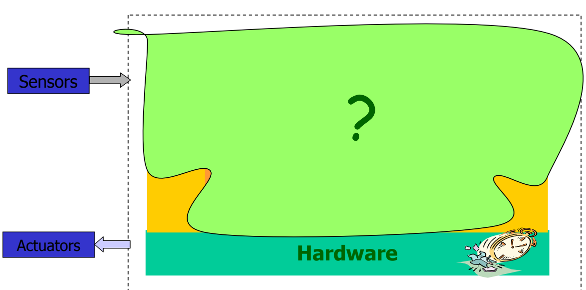
Real Time Systems



Real Time Systems



Real Time Systems

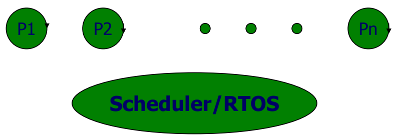


"Who is Who" in Timed Systems

- Real Time Scheduling [RTSS …]
  - Task models, Schedulability analysis
  - Real time operating systems
- Automata/logic-based methods [CAV, TACAS …]
  - (Timed) Automata, Petri Nets, Process Algebras …
  - Modelling, Model checking …
- (Real-Time) Programming Languages […]
  - Esterel, Signal, Lustre, Ada …
- … …

”Classic” Real Time Scheduling

Periodic tasks



well-developed techniques e.g. Rate-Monotonic Scheduling

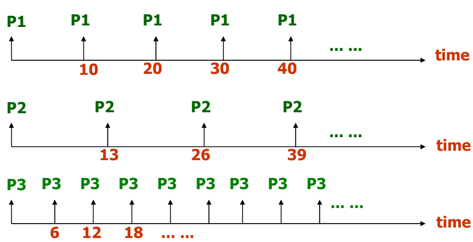
Rate-Monotonic Scheduling

- P1...Pn arrive at fixed rates
- Fixed Prioirity Order: higher frequency => higher priority
- Always run the task with highest priority (FPS)
- Schedulability can be checked by solving (or UB test):

$$Ri = Bi + Ci + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * Cj$$

If Ri <= Di for all tasks Pi, the system is schedulable

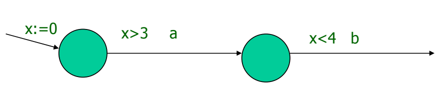
Arrival Rates (of Tasks): Periodic



In ”real life”

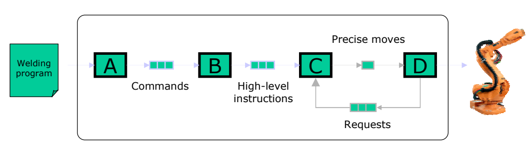
- tasks may share many resources (not only CPU time)
- tasks may have complex control stuctures and interactions
- tasks may not be that ”regular” (often non-periodic)

Task Arrival Patterns: Timed Traces



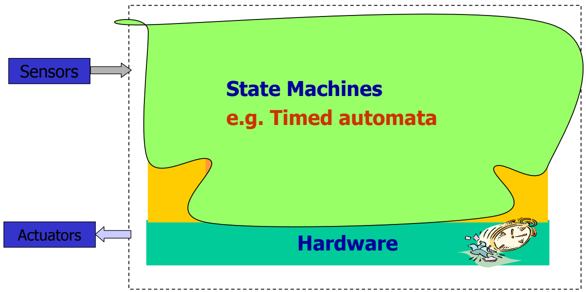
(3.3, a) (3.4, b),
(6.5, a),
(3.6, a) (3.9, b),
(3.14, a) (3.14159, b)
... ...

The ABB Robot Controller

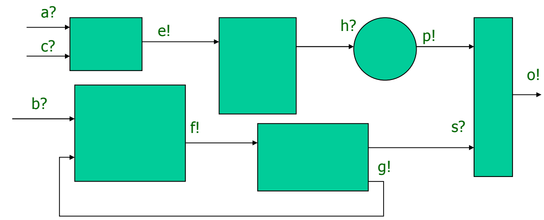


- ABB robot controller (2 500 000 loc)
- Real time tasks A,B,C,D
- Read inputs from channels write output to channels
- Task priority order D>C>B>A (FPS)
- Buffer overflow/underflow, WCRT

Automata-Based Approaches

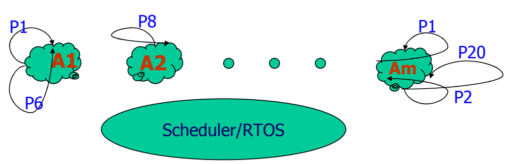


Networks of Real-Time Components



Automata-based Approaches

A controller = a set of timed automata accepting events trigger tasks Pi's



How to schedule tasks/automata? Worst-case response times?

Problems to solve

- Schedulability analysis: check
  (A1 || A2 || ... || An || Scheduler) satisfies K
  - A scheduler is given e.g. FPS, RMS, EDF etc
  - K is a requirement specifying e.g. safety
- Schedule synthesis: find X such that
  (A1 || A2 || ... || An || X) satisfies K

OUTLINE

- A Model for Timed Systems [1998]
  - Timed automata with tasks
- Schedulability and Decidability [TACAS 02,04]
  - Timed automata with bounded subtraction
- More Efficient Algoritms [TCS 06, IC 07]
  - Schedulability analysis using 2 clocks
  - (similar to Rate-Monotonic Scheduling)

Implemented in the TIMES tool

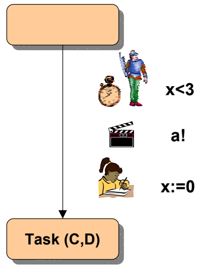
- TIMES Demo
- Current/Ongoing Work

The MODEL

(Timed Automata with Tasks)

Timed Automata with Tasks

- Events
  - Discrete Transitions
- Timing constraints
  - Clocks / Guards / Resets
  - Complex arrival rates
- Tasks
  - Asynchronous execution
  - WCET, Deadline
  - Scheduling policy
  - Precedence constraints
  - Resource constraints



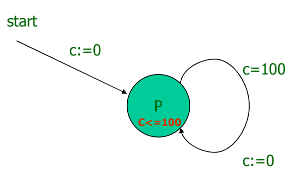
Tasks = Executable Programs (e.g. C, Java)

- Task parameters:
  - C: WCET
  - D: Relative Deadline
  - (other parameters for schedulling e.g. Priority)
- Task Interface:

```
Task P
{
v1 := F1(v1...vn)
...
vn := Fn(v1...vn)
}
```

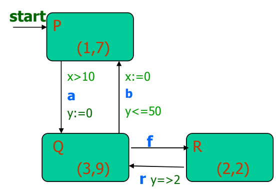
(a set of variables updated)

Example: periodic tasks



c: clock
P: task

Timed Automata with Tasks (Example)



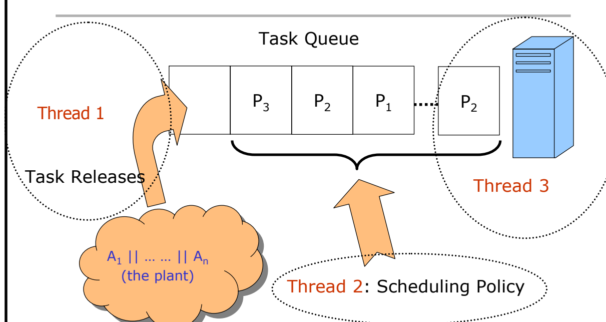
- Processor 1 (event handler)
  - Initially, P in the queue
  - Run-to-Completion/Stablization
    - Whenever a available and x>10, Q is put in the queue
    - Then
      - Whenever b available and y<=50, P is put in the queue
      - Whenever f available, R is put in the queue.
- Processor 2 (task handler)
  - Schedule and Compute tasks in the queue

P	P	Q	Q	Q	R	...	...	...

Timed Automata withTasks

- Assume a set of tasks Pr
- A timed automaton with tasks is a tuple: $<N,n_0,T,M>$
  - $<N,n_0,T>$ is a standard timed automaton
    - N is a set of nodes
    - n_0 is the initial node
    - $T \subseteq N \times (B(C) \times Act \times 2^C) \times N$ is the set of 'edges'
      - C is a set of clocks
      - Act is a set of actions
      - B(C) is the set of clock constraints e.g. X <10 etc
  - $M: N \rightarrow 2^{Pr}$ is a mapping which assigns each node a set of tasks

The Execution Platform

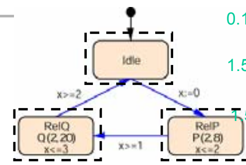


States/Configurations of automata

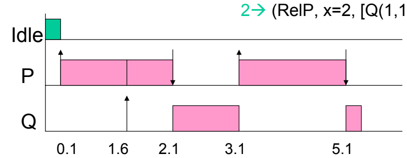
A **state** is a triple: (m, u, q)

- m — Location (node)
- u — clock assignment (valuation)
- q — task queue

Run of TAT



(Idle, x=0, [])
0.1→ (Idle, x=0.1, [])
→ (RelP, x=0, [P(2,8)])
1.5→ (RelP, x=1.5, [P(0.5,6.5)])
→ (RelQ, x=1.5, [P(0.5,6.5),Q(2,20)])
1.5→ (RelQ, x=3, [Q(1,18.5)])
→ (Idle, x=3, [Q(1,18.5)])
→ (RelP, x=0, [P(2,8),Q(1,18.5)])
2→ (RelP, x=2, [Q(1,16.5)])



Sch and Run

- **Sch** is a function sorting task queues according to a given scheduler e.g FPS,EDF,FIFO etc

  Example: EDF [P(2, 10), Q(4, 7)] = [Q(4, 7), P(2, 10)]

- **Run** is a function corresponding to running the first task of the queue for a given amount of time.

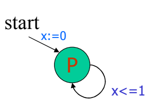
  Examples: Run(0.5, [Q(4, 7), P(2, 10)]) = [Q(3.5, 6.5), P(2, 9.5)]
  Run(5, [Q(4, 7), P(2, 10)]) = [P(1, 5)]

Semantics (as transition systems)

- States: <m,u,q>
  - m is a location
  - u is a clock assignment (valuation)
  - q is a queue of tasks (ready to run)
- Transitions:
  1. (m,u,q) −a→ (n, r(u), Sch[M(n)::q]) if (m) −g a r→ (n) & g(u)
  2. (m,u,q) −d→ (m, u+d, Run(d,q)) where d is a real

OBS: q is growing (by actions) and shrinking (by delays)

"Zenoness" = Non-Schedulability



P=(2,3)

Zeno: ∞ many P's may arrive within 1 time unit !

P(2,3)	P(2,3)	P(2,3)	P(2,3)	P(2,3)	...					..

But after 2 copies, the queue will be non-schedulable

SCHEDULABILITY

Schedulability of automata

a state is a triple: (m, u, q)

- m: location
- u: clock assignment
- q: task queue

•A state is schedulable if q is schedulable

•An automaton is schedulable if all reachable states are

Schedulability of Automata

Assume a scheduler Sch:

- A state (m,u,q) is schedulable with Sch if
  - $Sch(q)= [P_1(c_1,d_1)P_2(c_2,d_2)\ldots P_n(c_n,d_n)]$ and
  - $(c_1+\ldots+c_i)<=d_i$ for all $i<=n$ (i.e. all deadlines met)
- An automaton is schedulable with Sch if all its reachable states are schedulable
- An automaton is schedulable with a class of scheduling policies if it is schedulable with every Sch in the class.

DECIDABILITY

Schedulability Analysis (Non-preemptive scheduling)

FACT [1998]

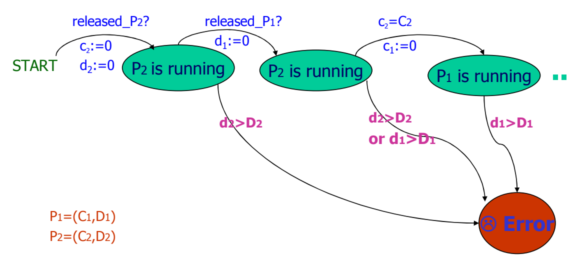
For Non-preemptive schedulers, the schedulability of an automaton can be checked by reachability analysis on ordinary timed automata.

Proof ideas (1):
Size of schedulable queues is bounded

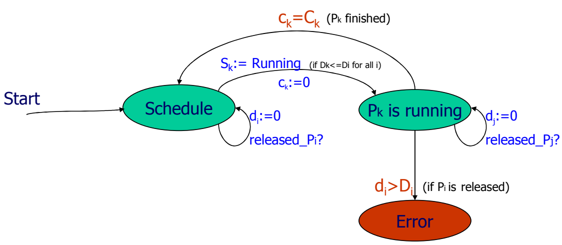
- The maximal number of instances of P_i in a schedulable queue is bounded by $M_i = \lceil D_i/C_i \rceil$
- The maximal size of schedulable queues is bounded by $M_1 + M_2+\ldots+M_n$
- To code the queue/scheduler, for each task instance, use 2 clocks:
  - c_i remembers the computing time
  - d_i remembers the deadline

• • •	(c_i,d_i)	• • •

Proof ideas (2):
The scheduler as an automaton



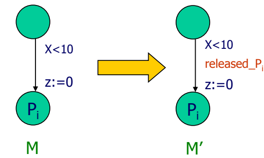
The scheduler automaton



SCHEDULER

Proof Ideas (3)

- Modify the original automaton M: adding 'release!' to inform the scheduler



- Check reactability of the error state for
  M' || SCHEDULER

How about preemptive scheduling?

- We may try the same ideas
  - Use clocks to remember computing times and deadlines
- BUT a running task may be stopped to run a more 'urgent' task
  - Thus we need stop-watches to remember "accumulated computing times"
- Then the schedulability problem is undecidable ?
- This is wrong !!

Decidability Result [TACAS 2002]

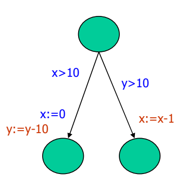
FACT

For Preemptive schedulers, the schedulability of an automaton can be checked by reachability analysis on Bounded Substraction Timed Automata (BSA).

NOTE
- Reachability for BSA is decidable
- Preemptive EDF is optimal; thus the general schedulability checking problem is decidable.

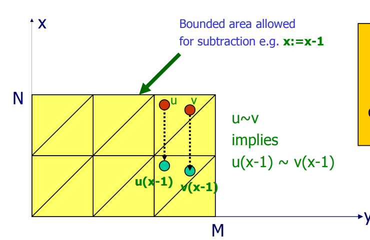
Timed automata with subtraction
i.e. Subtraction Automata, [McManis and Varaiya, CAV94]

- Subtraction automata are timed automata extended with subtraction on clocks
- That is, in addition to reset x:=0, it is also allowed to update a clock x with x:= x-n where n is a natural number



Bounded Subtraction Automata

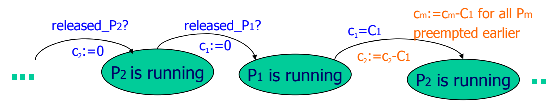
- A subtraction automaton is bounded if its clocks are non-negative and bounded with a maximal constant (or subtraction is only allowed in the bounded zone).



FACT:
Location Reachability checking is decidable!

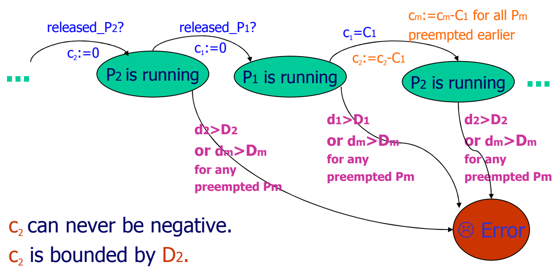
Schedulability Checking as a reachability problem for Bounded Subtraction Automata

Proof ideas (no stop but subtraction :-)



- Model the scheduler as a subtraction automaton
  - Do not stop the computing clock c_2 when a new task P_1 is released
  - Let c_2 for P_2 (preempted) run until the task P_1 (with higher priority) finishes, then perform $c_2 := c_2 - C_1$ (note: C_1 is the computing time for P_1).

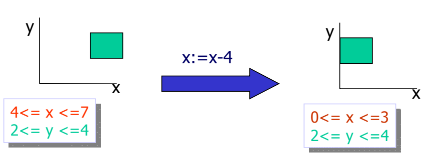
Proof ideas (clocks are bounded):



- c_2 can never be negative.
- c_2 is bounded by D_2.

END of proof

Schedulability analysis using DBM's



Subtraction on Clocks, added to DBM-library (UPPAAL)

Complexity

#clocks (needed)
= 2 x #instances (maximal number of schedulable task instances)
= 2 x $\sum_i \lceil Di/Ci \rceil$

This is a huge number in the worst case
But the run-time complexity is not so bad!

It works anyway !!!

- #active tasks in the queue is normally small, and the run-time complexity is only related to #active clocks
- If Too many active tasks in the queue (i.e. Too many active clocks), the check will stop sooner and report ”non-schedulable”
- AND the analysis can be done symbolically!

WE CAN DO BETTER ! [TACAS 03, TCS 06]

For fixed priority scheduling strategies (FPS),
we need only 2 clocks (and ordinary timed automata)!

The 2-CLOCK ENCODING

(for fixed-priority scheduling strategies)

Main Idea

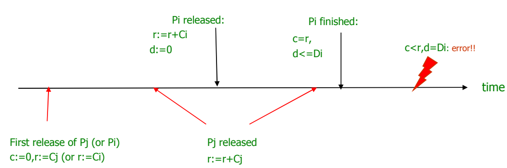
- Check the schedulability of tasks one by one according to priority order (highest priority first)
- This is similar to response time analysis in RMS

To code the queue/scheduler, we need:

- 1 integer variable for Pi:
  - r denotes the response time as in RMS
    (the total computing time needed before Pi finishes)
- 2 clocks for Pi:
  - c remembers the accumulated computing time
    (so much has been computed so far)
  - d remembers the ”deadline”

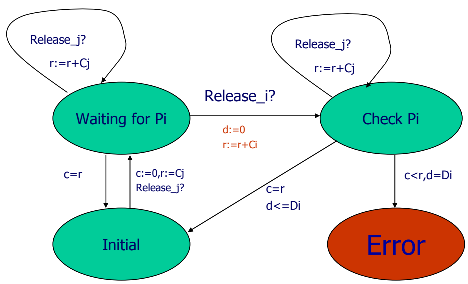
Intuition of the encoding: $Ri = Ci + \Sigma_{pri(Pj)>pri(Pi)} Cj$

– Assume: priority(Pj) > priority(Pi) and Pi is analyzed



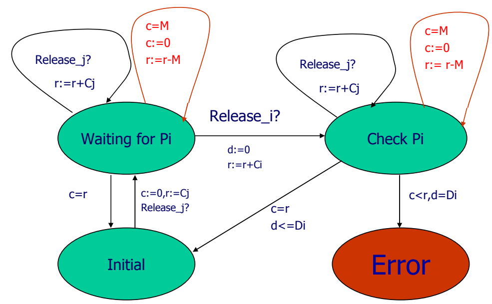
When Pi finishes, r = Ri

The "FPS scheduler": analyzing Pi



Note that it is not clear that c and r are bounded !

The "FPS scheduler": analyzing Pi
(we need the boundedness)



OBS: r-c is the only interesting info, so M can be any integer! Let M=Ci

c and r are bounded

- c is bounded by M
- r is bounded by r_{max} + Ci
  - Where r_{max} is the maximal value of r from previous analysis for all tasks Pj with higher priority

So the scheduler is a standard TA **END**

SUMMARY: Decidability

- For Non-preemptive schedulers, the problem can be solved using standard TA.
- For preemptive schedulers, the problem can be solved using BSA (Bounded Substraction Automata).
- For fixed-priority schedulers, the problem can be solved using TA with only 2 extra clocks – similar to the classic RMA technique (Rate-Monotonic Analysis).

Undecidability

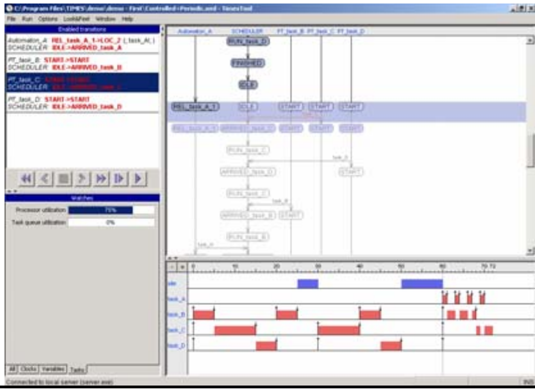
Unfortunately, the problem will be undecidable if the following conditions hold together:

1. Preemptive scheduling
2. Interval computation times
3. Feedback i.e. the finishing time of tasks may influence the release times of new tasks.

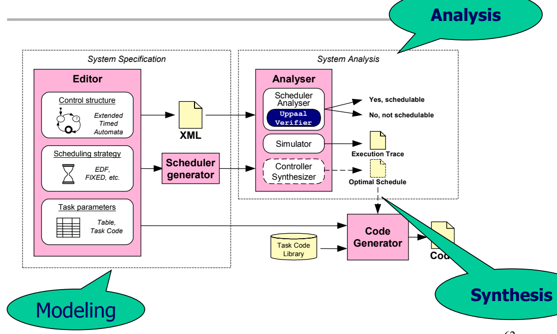
Conclusions/Remarks

- Unification of model-checking, real time scheduling, and synchronous programming: a unified model for timed systems (can express complex temporal and resource constraints).
- The first decidability result (and efficient algorithms) for preemptive scheduling in dense time models:
  - The analysis is symbolic (using DBM's in the UPPAAL tool)
- Implementation: **TIMES**

TIMES demo

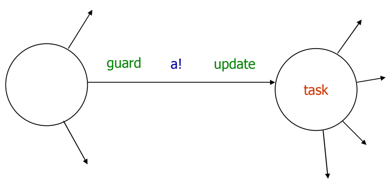


An Overview of TIMES



The INPUT LANGUAGE is very much like "guarded commands"

OBS: guard and update may contain data variables (integer, array)



- guard, update: "synchronous" computation which takes "no time" [CONCUR'04]
  - we adopt the synchronous hypothesis
- task: "asynchronous" computation whch takes time

Tasks = Executable Programs (e.g. C, Java)

- Task Type
  - Synchronous or Asynchronous
  - Non-Periodic (triggered by events) or Periodic
- Task parameters: C, D etc
  - C: Computing time and D: Relative Deadline
  - other parameters for schedulling e.g. priority, period
- Task Interface (variables updated 'atomically')
  - $X_i := F_i(X_1 \ldots X_n)$
- Tasks may have shared variables
  - with automata
  - with other tasks (priority ceiling protocols)
- Tasks with precedence constraints

Functionality/Features of TIMES

- GUI
  - modeling: automata with asynchronous tasks
  - editing, task library, visualization etc
- Simulation
  - symbolic execution as MSC's and Gant Charts
- Verification
  - all you do with UPPAAL
  - Schedulability analysis
- Code Generation

Code Generation in TIMES

- Run Time Systems
  - Event Handler, OS interrupt processing system or Polling
  - Task scheduler, generated from task parameters
- Application Tasks = threads (or processes)
  - Already there! (written in C)
  - Current version of TIMES support LegoOS

TIMES Demo


system.xml - TimesTool
System
Tasks
Task parameter
Start
Times