Verification of Real Time Systems: Uppaal, Case Studies

Radek Pelánek

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

EVROPSKÁ UNIE MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY OP Vzdělávání pro konkurenceschopnost

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Uppaal Tool

- verification tool for real time systems
- based on timed automata
- UPPsala + AALborg university, academic tool
- widely used for teaching
- several industrial case studies
- `www.uppaal.org`

Extensions

- time extensions: location invariants, ‘diagonal’ constraints on clocks (comparison of two clocks)
- data: integer variables, C code
- concurrency: networks of timed automata, communication via handshake (without value passing)
- modeling aid: committed locations, urgent channels

Functionality of the Tool

modeling graphical tool for specification of timed automata, templates

simulation simulation of the model (manual, random)

verification verification of simple properties (restricted subset of Timed Computational Tree Logic), counterexamples can be simulated

Fischer's Protocol

- real-time protocol – correctness depends on timing assumptions
- simple, just 1 shared variable, arbitrary number of processes
- assumption: known upper bound D on reading/writing variable in shared memory
- each process has it's own timer (for delaying)

Fischer's Protocol

- `id` – shared variable, initialized -1
- each process has it's own timer (for delaying)
- for correctness it is necessary that $K > D$

```
Process i:
while (true) {
    <noncritical section>;
    while id != -1 do {}
    id := i;
    delay K;
    if (id = i) {
        <critical section>;
        id := -1;
    }
}
```

Fischer's Protocol: Demo

Notice:

- modeling: templates
- simulation: time zones
- verification: ability to find counterexample

Try:

- change the number of processes
- model has just one constant K, introduce two constants D, K
- let $K < D$ and find a run violating mutual exclusion

Bridge Puzzle

- 4 men, river, bridge, night, 1 flashlight
- at most 2 man on a bridge, flashlight necessary
- flashlight cannot be thrown
- men – different time to cross: 5 min, 10 min, 20 min, 25 min
- can they cross in 60 minutes?
- can they cross is less than 60 minutes?

Bridge Puzzle: Demo

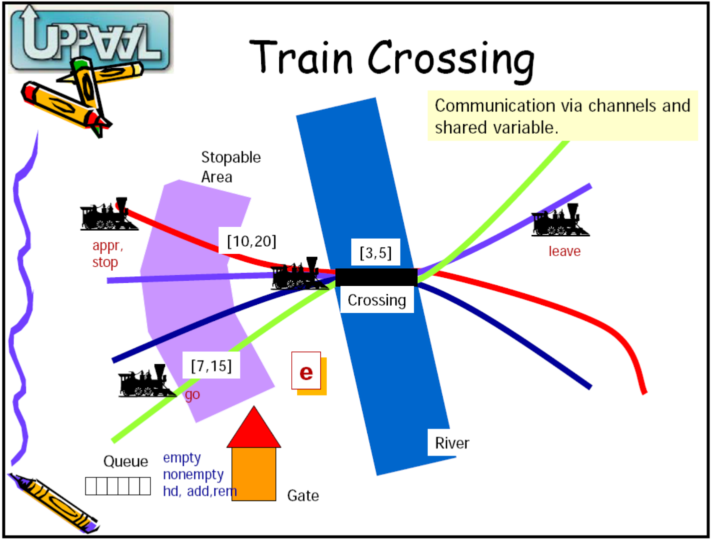
Notice:

- modeling: channels, synchronization
- simulation: message sequence chart
- verification: ability to find the fastest trace

Try:

- change the time to cross: 2 min, 3 min, 5 min, 8 min
- what is the minimum time to cross?

Train-Gate


Train Crossing
Communication via channels and shared variable.
Stopable Area
appr, stop
[10,20]
[3,5]
leave
Crossing
[7,15]
go
e
River
Queue
empty
nonempty
hd, add,rem
Gate

Train Crossing: Demo

Notice:

- modeling: C-code
- verification: types of properties

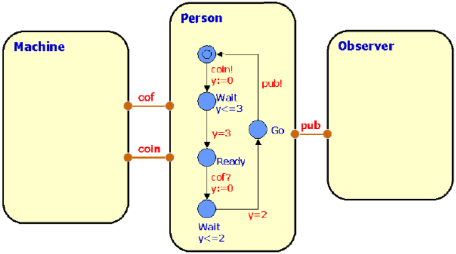
Concurrent Addition Puzzle

$$c := 1, x_0 := 0, x_1 := 0$$

$$\begin{array}{lcl} x_0 := c & & x_1 := c \\ x_0 := x_0 + c & \| & x_1 := x_1 + c \\ c := x_0 & & c := x_1 \end{array}$$

- construct model, hints:
  - one template for the two processes
  - `int x[2];`
- automatically find a run leading to $c = 23$

Coffe Machine



For model without "error states":

- committed locations "Start", "Go"
- urgent channel "cof"

Mutual Exclusion

- protocols:
  - Peterson's protocol: model without time
  - Alur and Taubenfeld's protocol: see handout
- property to check: mutual exclusion
- for each protocol make both correct and erroneous version

Peterson's Algorithm

- `flag[0]`, `flag[1]` (initialed to `false`) – meaning *I want to access CS*
- `turn` (initialized to 0) – used to resolve conflicts

```
Process 0:
while (true) {
   <noncritical section>;
   flag[0] := true;
   turn := 1;
   while flag[1] and
         turn = 1 do { };
   <critical section>;
   flag[0] := false;
}
```

```
Process 1:
while (true) {
   <noncritical section>;
   flag[1] := true;
   turn := 0;
   while flag[0] and
         turn = 0 do { };
   <critical section>;
   flag[1] := false;
}
```

Gear Controller

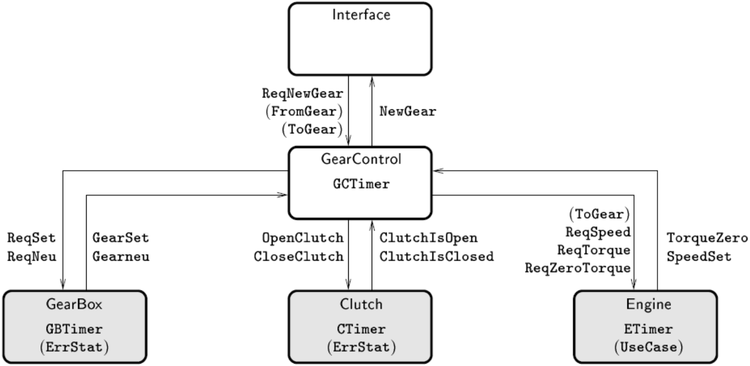
Formal Design and Analysis of a Gear Controller. M. Lindahl, P. Pettersson, W. Yi.

- component in the real-time embedded system that operates in a modern vehicle (specifically Mecel AB)
- the gear-requests from the driver are delivered over a communication network to the gear controller
- the controller implements the actual gear change by actuating the lower level components of the system, such as the clutch, the engine and the gear-box

Interface

- receives service requests, keeps information about the current status
- used by:
    - the driver using the gear stick
    - dedicated component implementing the gear change algorithm

Gear Controller



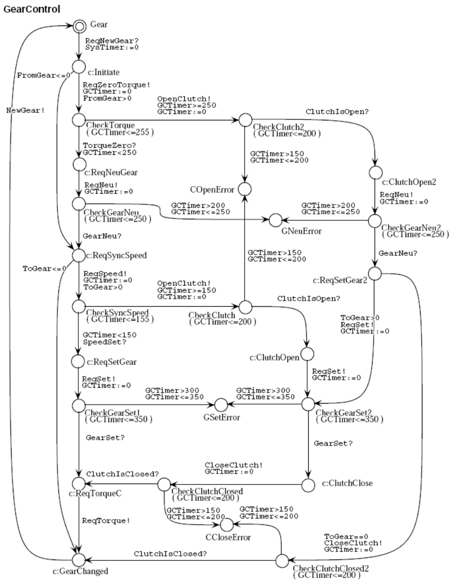
Functionality of the Controller

- gear change performed in five steps:
  1. accomplish zero torque
  2. release the current gear
  3. achieve synchronous speed
  4. set the new gear
  5. increase the engine torque back to previous level
- under difficult driving conditions: zero torque or synchronous speed not possible; then use the clutch

Timing Parameters

- setting/releasing of a gear by electrically controlled gear-box
- timeout for reaching the zero torque
- timeout for reaching synchronous speed
- time needed for opening/closing the clutch

Gear Controller



Requirements

- performance: a gear shift should be completed within 1.5 seconds, ...
- safety: controller detects and report errors if and only if clutch is not opened (closed) in time, ...
- functionality: it is possible to use all gears
- predictability: strict synchronization between components, e.g., when regulating torque, clutch should be closed, ...

Gear Controller

$$\mathsf{GearControl@Initiate} \leadsto_{\leq 1500} ((\mathtt{ErrStat} = 0) \Rightarrow \mathsf{GearControl@GearChanged})$$

$$\mathsf{GearControl@Initiate} \leadsto_{\leq 1000} ((\mathtt{ErrStat} = 0 \wedge \mathtt{UseCase} = 0) \Rightarrow \mathsf{GearControl@GearChanged})$$

$$\mathsf{Clutch@ErrorClose} \leadsto_{\leq 200} \mathsf{GearControl@CCloseError}$$

$$\mathsf{Clutch@ErrorOpen} \leadsto_{\leq 200} \mathsf{GearControl@COpenError}$$

$$\mathsf{GearBox@ErrorIdle} \leadsto_{\leq 350} \mathsf{GearControl@GSetError}$$

$$\mathsf{GearBox@ErrorNeu} \leadsto_{\leq 200} \mathsf{GearControl@GNeuError}$$

$$Inv\ (\mathsf{GearControl@CCloseError} \Rightarrow \mathsf{Clutch@ErrorClose})$$

$$Inv\ (\mathsf{GearControl@COpenError} \Rightarrow \mathsf{Clutch@ErrorOpen})$$

$$Inv\ (\mathsf{GearControl@GSetError} \Rightarrow \mathsf{GearBox@ErrorIdle})$$

$$Inv\ (\mathsf{GearControl@GNeuError} \Rightarrow \mathsf{GearBox@ErrorNeu})$$

$$Inv\ (\mathsf{Engine@ErrorSpeed} \Rightarrow \mathtt{ErrStat} \neq 0)$$

$$Inv\ (\mathsf{Engine@Torque} \Rightarrow \mathsf{Clutch@Closed})$$

$$\bigwedge_{i \in \{R,N,1,\ldots,5\}} Poss\ (\mathsf{Gear@Gear}_i)$$

$$\bigwedge_{i \in \{R,1,\ldots,5\}} Inv\ ((\mathsf{GearControl@Gear} \wedge \mathsf{Gear@Gear}_i) \Rightarrow \mathsf{Engine@Torque})$$

Power Control

Formal Verification of a Power Controller Using the Real-Time Model Checker Uppaal. K. Havelund, K. G. Larsen and A. Skou.

- real-time system for power-down control in audio/video components
- system is supposed to reside in an audio/video component and control (read from and write to) links to neighbor audio/video components such as TV, VCR and remote-control
- protocol used by audio/video company B&O
- design verified before implementation into products; several design errors were found

Power control

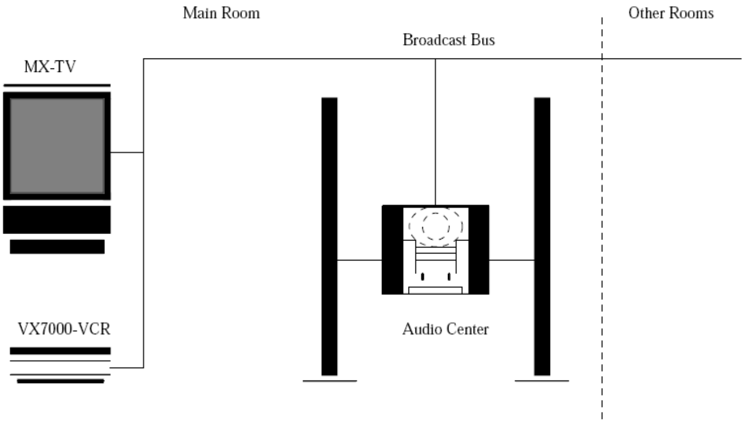


Fig. 1. Example B&O configuration.

Stand-by Mode

- components (processors) communicate via bus
- minimization of energy consumption $\Rightarrow$ stand-by mode
- valid data on bus $\Rightarrow$ leave stand-by mode
- entering (leaving) stand-by mode takes ap. 1ms, it is not atomic action
- purpose of protocol: switching to stand-by mode in consistent way

Power control

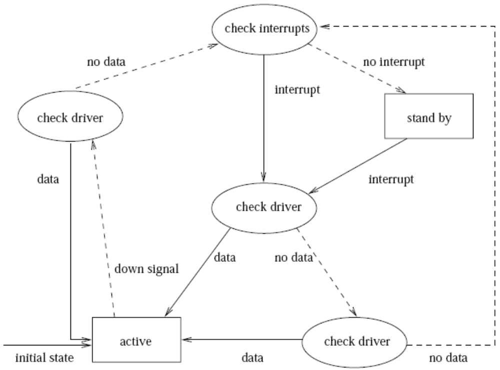


Fig. 3. Major protocol phases. The dotted lines indicate transitions leading towards power down. The full lines are leading towards power up. The two neighboring 'check driver' phases are necessary in order to be able to ignore noise from the communication lines.

Power control

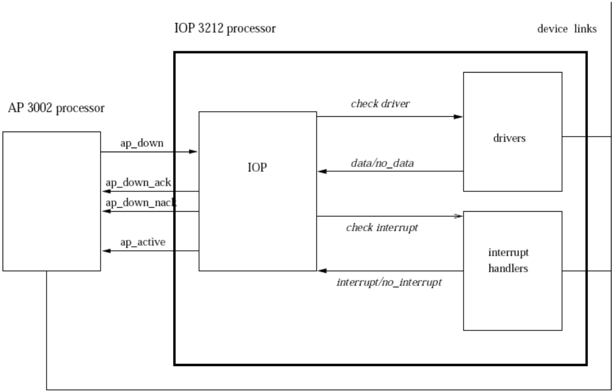
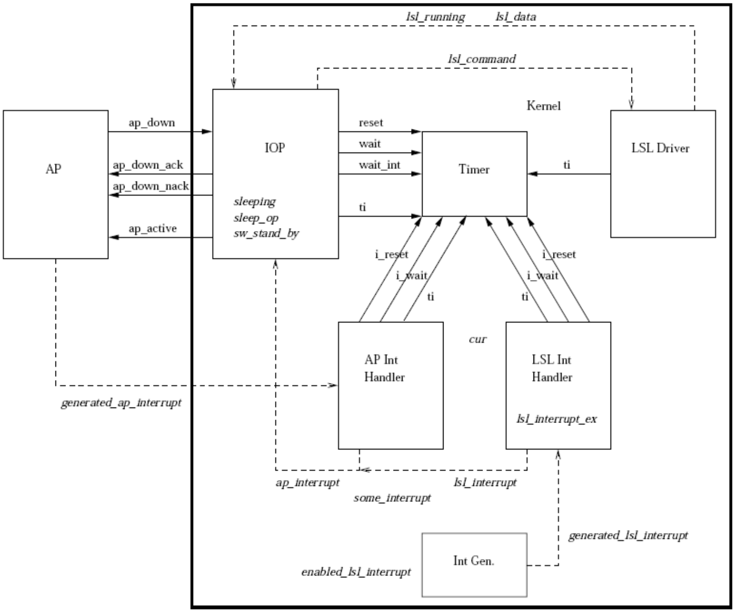
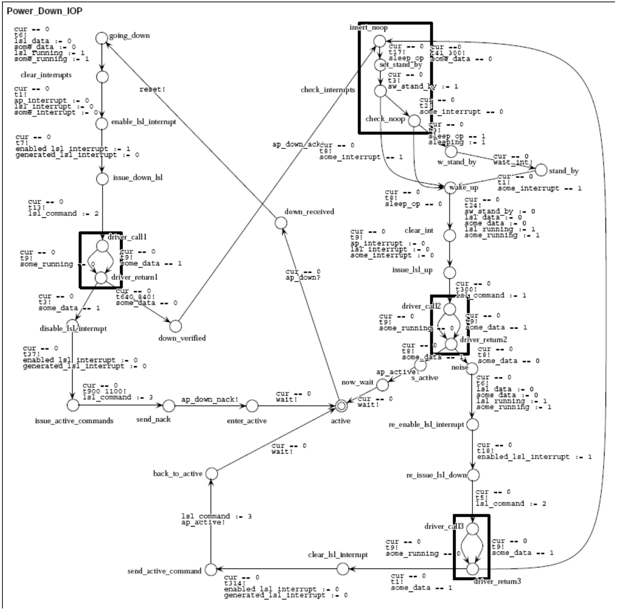


Fig. 2. Software architecture of the power down protocol. The protocol entity process (IOP) receives protocol commands (left arrows) from the drivers and interrupt handlers by issuing check commands (right arrows).

Power control



Power control

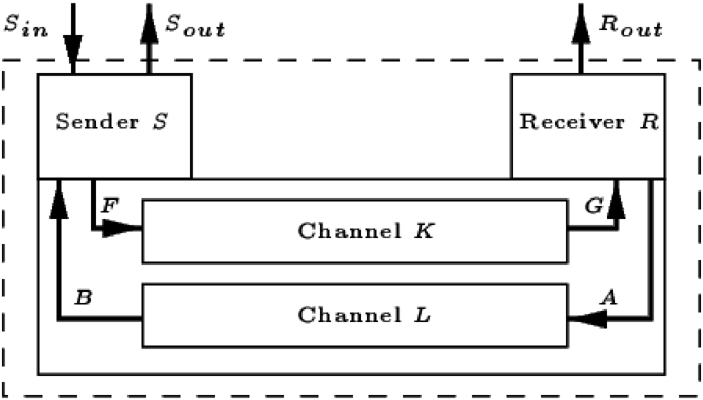


1. `sleeping` must not change from 0 to 1 while `sleep_op` has the value 0. *(The IOP must not go to sleep if there has been an interrupt – see Figure 10 for an explanation of these variables.)*
2. There must be a path from `active` to `stand_by` and vice versa. *(It must be possible for the IOP to switch between its two final states.)*
3. Every path from `active` to `noise` must pass through `stand_by` *(The IOP must have been asleep before reaching the* `noise` *state where it on its way up due to an interrupt discovers that the interrupt is "false", and hence caused by noise only.)*
4. The variable `sleeping` must not change from 0 to 1 while `lsl_interrupt` is 1 or `ap_interrupt` is 1 *(The IOP must not go to sleep as long as there is an untreated interrupt.)*
5. The shortest way from `driver_return1` to `driver_call2` does not take more than 1500 μs *(If the IOP on its way down verifies that the link is empty by calling the driver, and then immediately thereafter data arrive (an interrupt occurs) no more than 1500 μs must pass before the driver is called again.)*
6. The shortest way from `driver_return1` to `active` does not take more than 1500 μs *(If the IOP on its way down discovers data on the link by calling the driver, then no more than 1500 μs must pass before the IOP is active again.)*
7. The shortest way from `driver_return3` to `driver_call2` does not take more than 1500 μs *(Like 5, but in a different place in the protocol's execution.)*
8. The shortest way from `driver_return3` to `active` does not take more than 1500 μs *(Like 6, but in a different place in the protocol's execution.)*
9. If the last value of the variable `lsl_command` has been 1 or 3 (driver starting commands), then the value of `sleeping` must not change from 0 to 1 *(If the last command issued to the driver was a "start command", then the IOP must not go to sleep.)*

Bounded Retransmission Protocol

The Bounded Retransmission Protocol must be on time!, P. R. D'Argenio, J.P. Katoen, T.C. Ruys, J. Tretmans

- protocol goal: message transmission over unrealible medium
- based on the well-known alternating bit protocol
- allows only bounded number of retransmissions of each frame (piece of a file); timed specification
- protocol used in one of the Philips' products

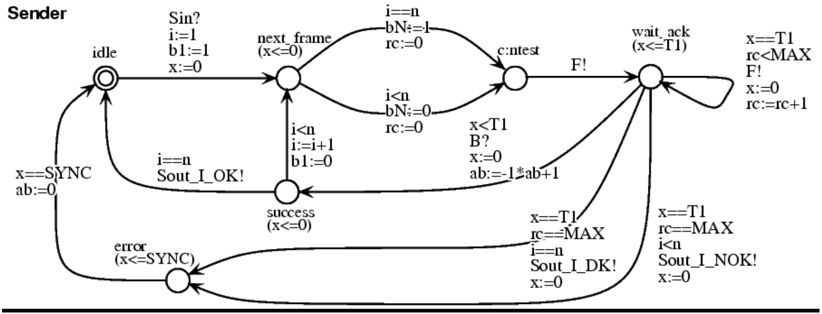


Schematic view of the BRP.

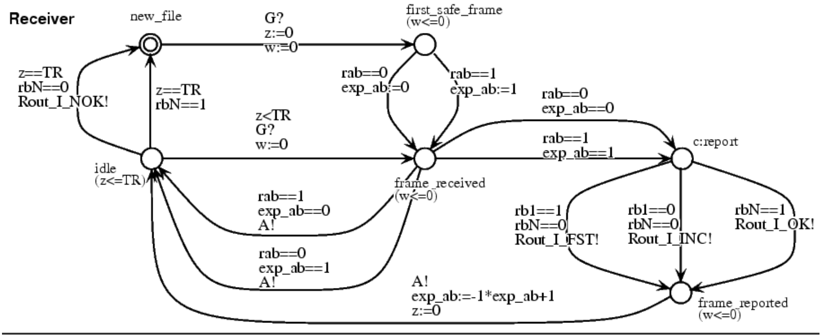
Timing Parameters

- sender's timeout (T1)
- receiver's timeout (T2)
- maximal delay in the channel (TD)
- synchronization time after failure (TR, SYNC)

Bounded Retransmission Protocol



Bounded Retransmission Protocol



Comment

using the tool it is possible to

- derive timing constraints necessary for the correctness of the protocol:

$$T1 > 2 \cdot TD, SYNC \geq TR \geq 2 \cdot MAX \cdot T1 + 3 \cdot TD$$

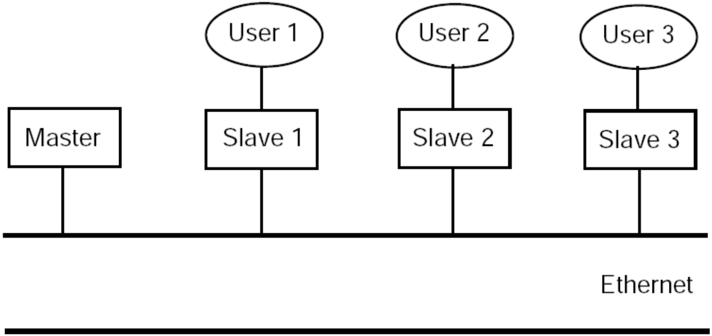
- verify that under these constraints the protocol satisfies formal specification

Collision Avoidance Protocol

Modelling and Analysis of a Collision Avoidance Protocol using SPIN and Uppaal. H. E. Jensen, K. G. Larsen, A. Skou

- several stations connected by an ethernet-like medium
- we assume basic protocol for error-free transmission
- on top of the basic protocol we built protocol for avoiding collisions (simultaneous transmission)
- basic ideas:
  - master station, assigns bus to slaves
  - delays (bus, receiving stations) have to be taken into account

Collision Avoidance Protocol



Goals of the Protocol

- collisions cannot occur
- transmitted data eventually reach their destination
- data which are received have been transmitted by a sender
- there is known upper bound on the transmission delay

the model is similar to the previous one

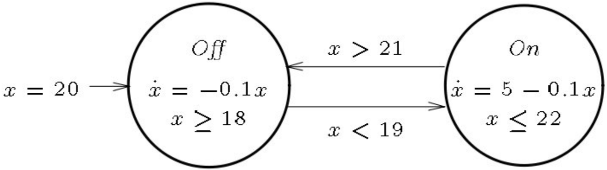
Extensions: Beyond Timed Automata ...

- probabilistic
- hybrid
- tasks (scheduling)
- games (controller synthesis)

Hybrid Automata

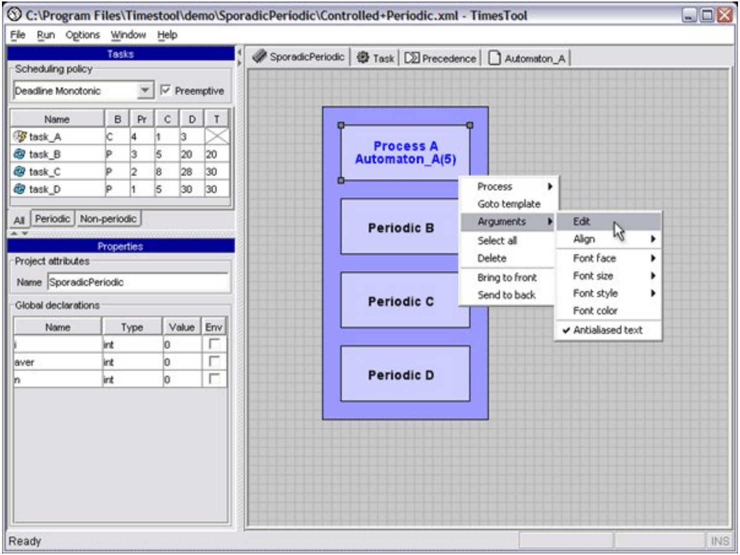
- embedded system modeling
- discrete control unit + continuous variables
- main difference wrt TA: change of continous variables does not need to be uniform (arbitrary function)
- used to model external physical processes, e.g. temperature

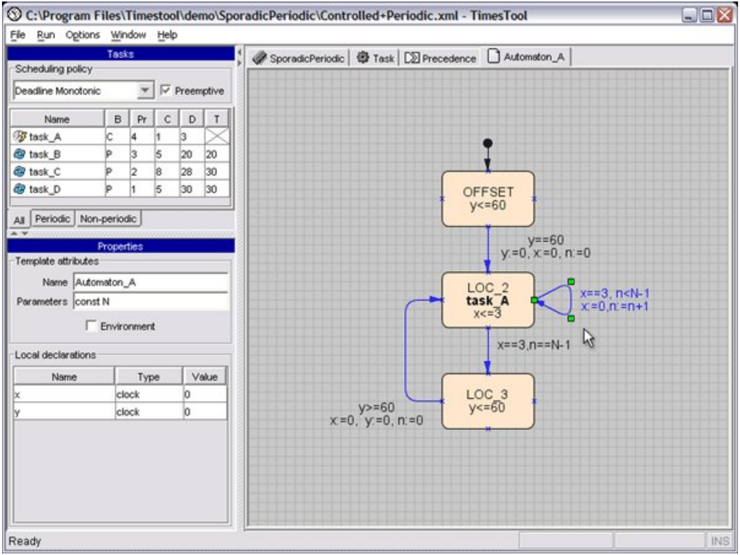
Hybrid Automata: Example

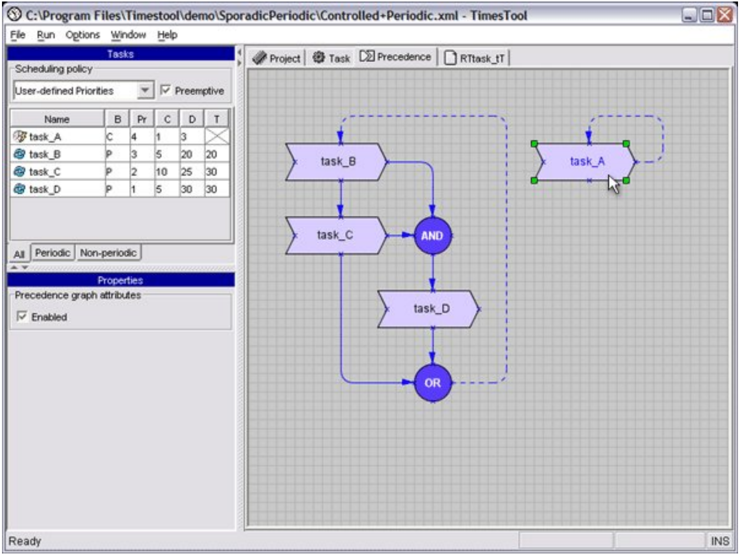


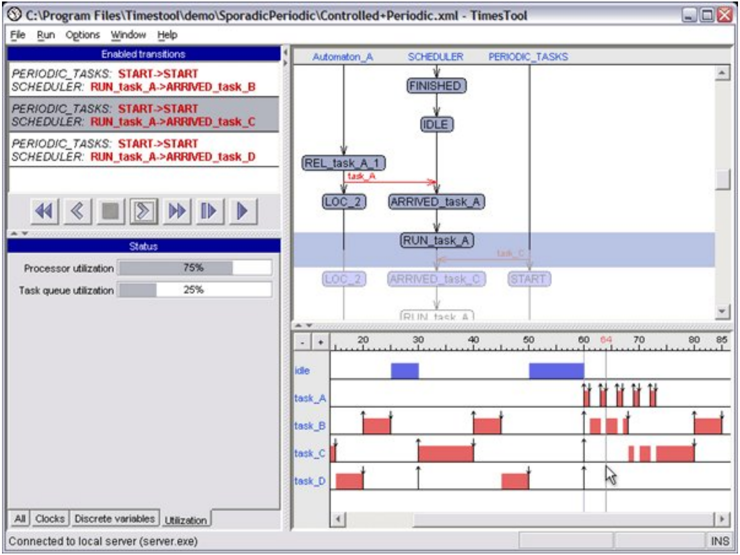
Automata with Tasks

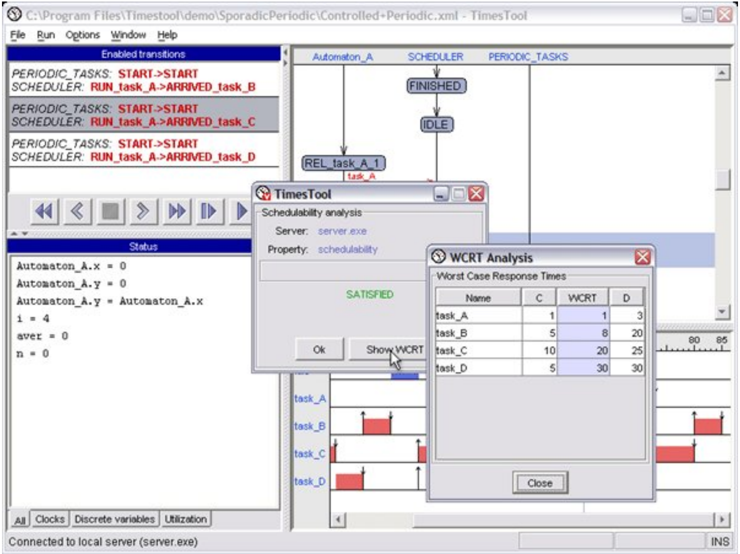
- Times – A Tool for Modeling and Implementation of Embedded Systems
- timed automata extended with tasks
- TA used to model task arrival patterns (more complex than simple periodicity)
- support for schedulability analysis, code generation


C:\Program Files\Timestool\demo\SporadicPeriodic\Controlled+Periodic.xml - TimesTool
File Run Options Window Help
Tasks
Scheduling policy
Deadline Monotonic
Preemptive
Name B Pr C D T
task_A C 4 1 3
task_B P 3 5 20 20
task_C P 2 8 28 30
task_D P 1 5 30 30
All Periodic Non-periodic
Properties
Project attributes
Name SporadicPeriodic
Global declarations
Name Type Value Env
i int 0
aver int 0
n int 0
SporadicPeriodic Task Precedence Automaton_A
Process A
Automaton_A(5)
Periodic B
Periodic C
Periodic D
Process
Goto template
Arguments
Select all
Delete
Bring to front
Send to back
Edit
Align
Font face
Font size
Font style
Font color
Antialiased text
Ready
INS


C:\Program Files\Timestool\demo\SporadicPeriodic\Controlled+Periodic.xml - TimesTool
File Run Options Window Help
Tasks
Scheduling policy
Deadline Monotonic
Preemptive
Name B Pr C D T
task_A C 4 1 3
task_B P 3 5 20 20
task_C P 2 8 28 30
task_D P 1 5 30 30
All Periodic Non-periodic
Properties
Template attributes
Name Automaton_A
Parameters const N
Environment
Local declarations
Name Type Value
x clock 0
y clock 0
SporadicPeriodic Task Precedence Automaton_A
OFFSET
y<=60
y==60
y:=0, x:=0, n:=0
LOC_2
task_A
x<=3
x==3, n<N-1
x:=0,n:=n+1
x==3,n==N-1
LOC_3
y<=60
y>=60
x:=0, y:=0, n:=0
Ready
INS


C:\Program Files\Timestool\demo\SporadicPeriodic\Controlled+Periodic.xml - TimesTool
File Run Options Window Help
Tasks
Scheduling policy
User-defined Priorities
Preemptive
Name B Pr C D T
task_A C 4 1 3
task_B P 3 5 20 20
task_C P 2 10 25 30
task_D P 1 5 30 30
All Periodic Non-periodic
Properties
Precedence graph attributes
Enabled
Project Task Precedence RTtask_tT
task_B
task_A
task_C
AND
task_D
OR
Ready
INS


C:\Program Files\Timestool\demo\SporadicPeriodic\Controlled+Periodic.xml - TimesTool
File Run Options Window Help
Enabled transitions
PERIODIC_TASKS: START->START
SCHEDULER: RUN_task_A->ARRIVED_task_B
PERIODIC_TASKS: START->START
SCHEDULER: RUN_task_A->ARRIVED_task_C
PERIODIC_TASKS: START->START
SCHEDULER: RUN_task_A->ARRIVED_task_D
Status
Processor utilization 75%
Task queue utilization 25%
All Clocks Discrete variables Utilization
Automaton_A SCHEDULER PERIODIC_TASKS
FINISHED
IDLE
REL_task_A_1
task_A
LOC_2
ARRIVED_task_A
RUN_task_A
task_C
LOC_2
ARRIVED_task_C
START
RUN_task_A
20 30 40 50 60 64 70 80 85
idle
task_A
task_B
task_C
task_D
Connected to local server (server.exe)
INS


C:\Program Files\Timestool\demo\SporadicPeriodic\Controlled+Periodic.xml - TimesTool
File Run Options Window Help
Enabled transitions
PERIODIC_TASKS: START->START
SCHEDULER: RUN_task_A->ARRIVED_task_B
PERIODIC_TASKS: START->START
SCHEDULER: RUN_task_A->ARRIVED_task_C
PERIODIC_TASKS: START->START
SCHEDULER: RUN_task_A->ARRIVED_task_D
Status
Automaton_A.x = 0
Automaton_A.y = 0
Automaton_A.y = Automaton_A.x
i = 4
aver = 0
n = 0
All Clocks Discrete variables Utilization
Connected to local server (server.exe)
INS
Automaton_A SCHEDULER PERIODIC_TASKS
FINISHED
IDLE
REL_task_A_1
TimesTool
Schedulability analysis
Server: server.exe
Property: schedulability
SATISFIED
Ok
Show WCRT
WCRT Analysis
Worst Case Response Times
Name C WCRT D
task_A 1 1 3
task_B 5 8 20
task_C 10 20 25
task_D 5 30 30
Close
80 85
task_A
task_B
task_C
task_D

Summary

- Uppaal tool: verification of real time models
- simple examples, main features of the tool
- overview of several realistic case studies
- extensions