The TIMES Tool Suite

Dott. Luigi Di Guglielmo

Prof. Tiziano Villa

University of Verona

Dep. Computer Science

Italy

Outline

- Introduction
- Scheduling Theory
- Scheduling in TIMES
- Modeling in TIMES
- TIMES tool suite overview
- Examples
- Exercise

Introduction (I)

- **TIMES** is a tool suite designed for modeling and schedulability analysis for embedded real-time systems
- The tool is developed by the Uppsala University (Sweden)
- It is appropriate for systems that can be described as a *set of* preemptive or non-preemptive *tasks* triggered periodically or sporadically by time or controlled by external events on clocks (guards)

Introduction (II)

- Given a system model consisting of:
  - A set of application tasks whose executions may be required to meet particular timing and resource constraints
  - A network of timed automata describing the task arrival patterns
  - A preemptive or non-preemptive scheduling policy
- TIMES will generate a scheduler and calculate the worst-case response times for the tasks
- The system model may also be validated using a model checker (i.e. Uppaal)

Scheduling Theory

- In classic scheduling theory, real time tasks are assumed to be periodic or sporadic
  - Periodic tasks arrive with fixed rates periodically
  - Sporadic tasks arrive with a minimal inter-arrival time
- Analysis based on this kind of arrival models often yields pessimistic results

Scheduling in TIMES (I)

- TIMES relaxes the stringent constraints on tasks arrival time by modeling tasks arrival patterns also using Timed Automata
- The proposed model is expressive enough to describe
  - Concurrency and synchronization
  - Periodic tasks
  - Sporadic tasks
  - Preemptive or non-preemptive tasks
  - Tasks precedence and their resource constraints

Scheduling in TIMES (II)

- An extension of the standard notion of schedulability to automata is required:
  - An automaton is schedulable if there exists a scheduling strategy such that all possible sequences of events accepted by the automaton are schedulable, i.e., all associated tasks can be computed within their deadlines

Modeling Real-time Systems in TIMES

- The two central concepts in TIMES are:
  - *Task*
  - *Scheduling policy*

Task

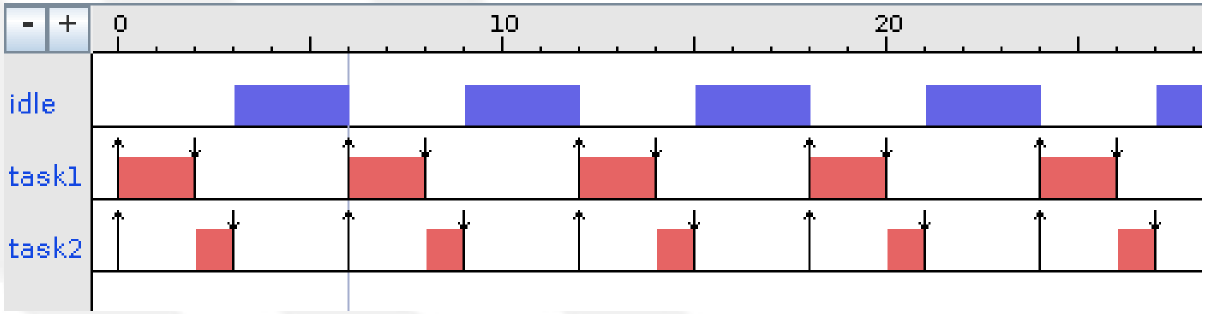
- A task is an executable program (e.g., a piece of C-code) characterized by
  - A *name*
  - A *priority*
  - A *worst case execution time* (WCET)
  - A *deadline*
    - Time point before which the task should complete its execution
  - A *behavior*
    - i.e., the task model

Task Model

- The task model defines how a task is released
  - *Periodically* with a fixed period
  - *Sporadically* with a given minimal inter-arrival time
  - *Controlled* by a particular automaton location
- Tasks execution depends on the scheduling policy adopted

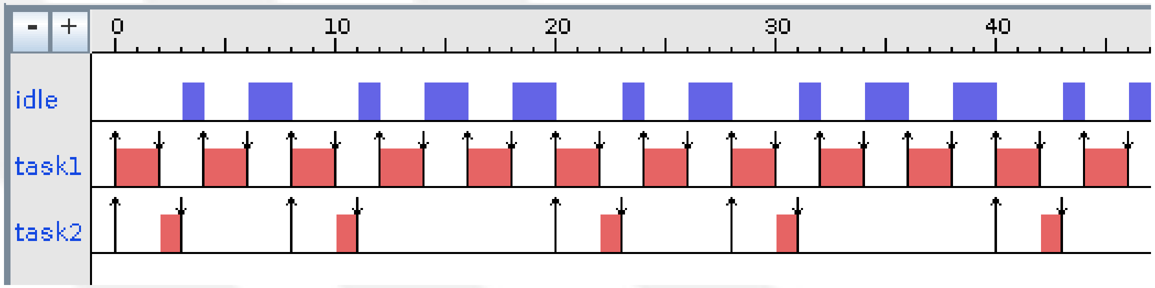
Periodic Task

- A periodic task is inserted with a fixed period into the queue of ready tasks



Sporadic Tasks

- A sporadic task is a particular kind of aperiodic task in real-time systems
- It is inserted into the queue of ready tasks considering a minimal inter-arrival time
- “I don't know how often this task will run, but assume it will be at least this *minimum inter-arrival time* between releases. If it tries to run more often that that, don't let it.”



Controlled Task

- A controlled task *T* can be used to label a location *L* of a timed automaton defined to model a particular release pattern
- When the automaton enters the location *L*, the task *T* is inserted into the queue of ready tasks

Scheduling policy

- Scheduling policies establishes how a task in the ready queue is chosen to be executed
  - Deadline monotonic
    - Task priorities are assigned according to deadlines, with the higher priority for the shorter deadline
  - Rate monotonic
    - Task priorities are assigned according to periods, with the higher priority for the shorter period
  - User-defined priorities
    - Task priorities are manually set
  - Earliest deadline first
    - Task priorities are computed at run-time
    - Whenever a scheduling event occurs (task finishes, new task released, etc.) the queue will be searched for the process closest to its deadline. This process is the next to be scheduled for execution.
  - First Come First Served
    - Tasks are executed in the same order they are inserted into the queue of ready tasks

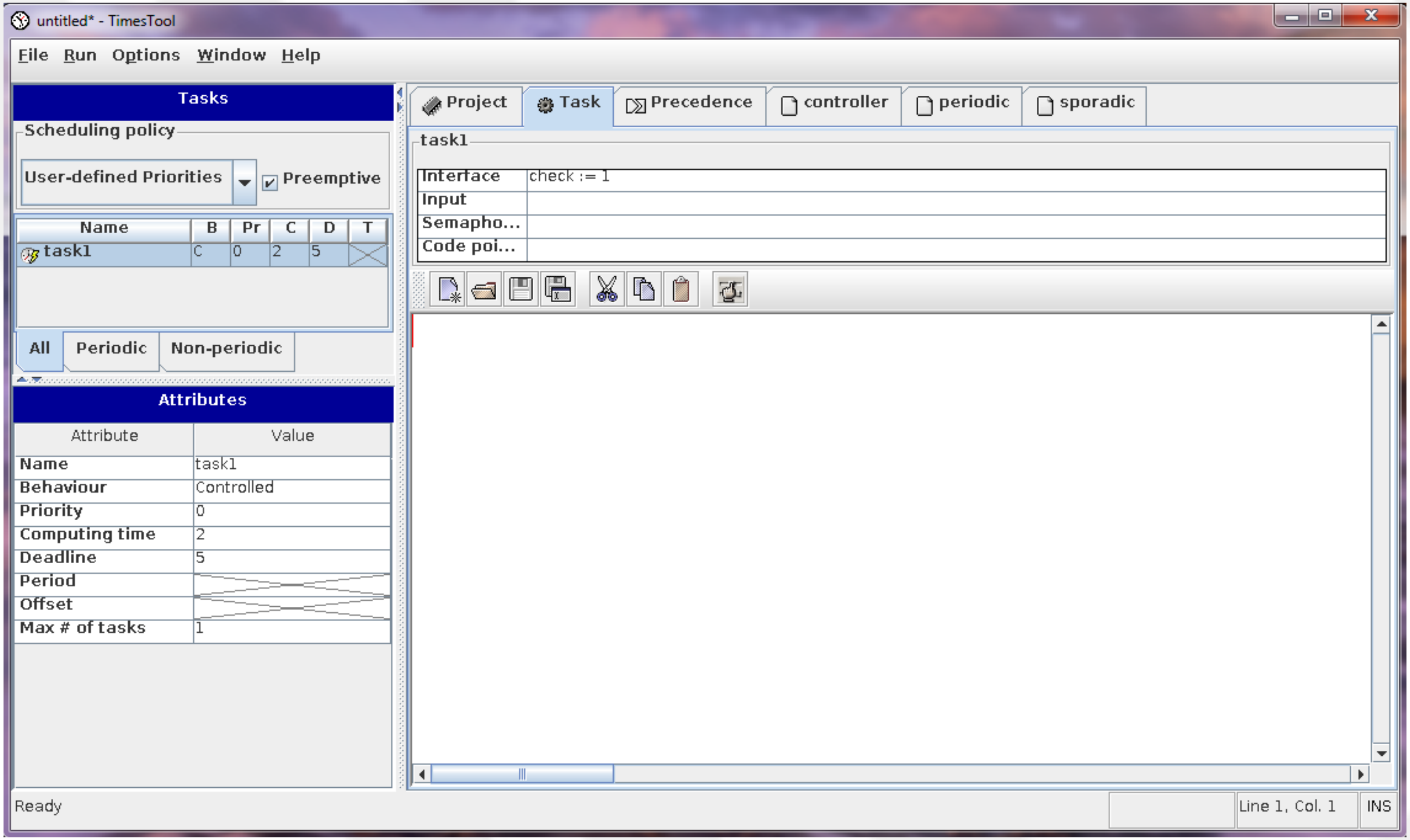
TIMES Engine

- Timed automata extended with tasks are translated into timed automata
- Periodic and sporadic tasks are modeled with timed automata
- Scheduling policies are modeled with timed automata
- The composed automaton is analyzed with Uppaal
  - Schedulability analysis
  - Property verification

Overview of the TIMES tool suite

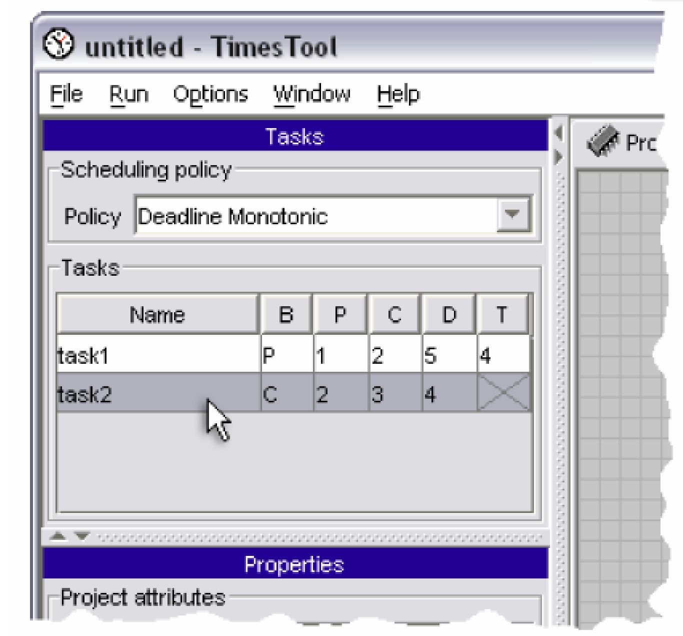
- A graphical editor
  - Modeling a system and the abstract behavior of its environment
- A simulator
  - Validating the dynamic behavior of the system by simulating the tasks according to parameters and scheduling policy
  - Showing a graphical representation of the execution trace
- A verifier
  - Checking schedulability analysis
    - Checking that reachable states are schedulable is equivalent to checking that tasks meet their deadlines
- Code generator
  - Generating automatically C-code for brickOS platform from the model

The Graphical Editor (I)

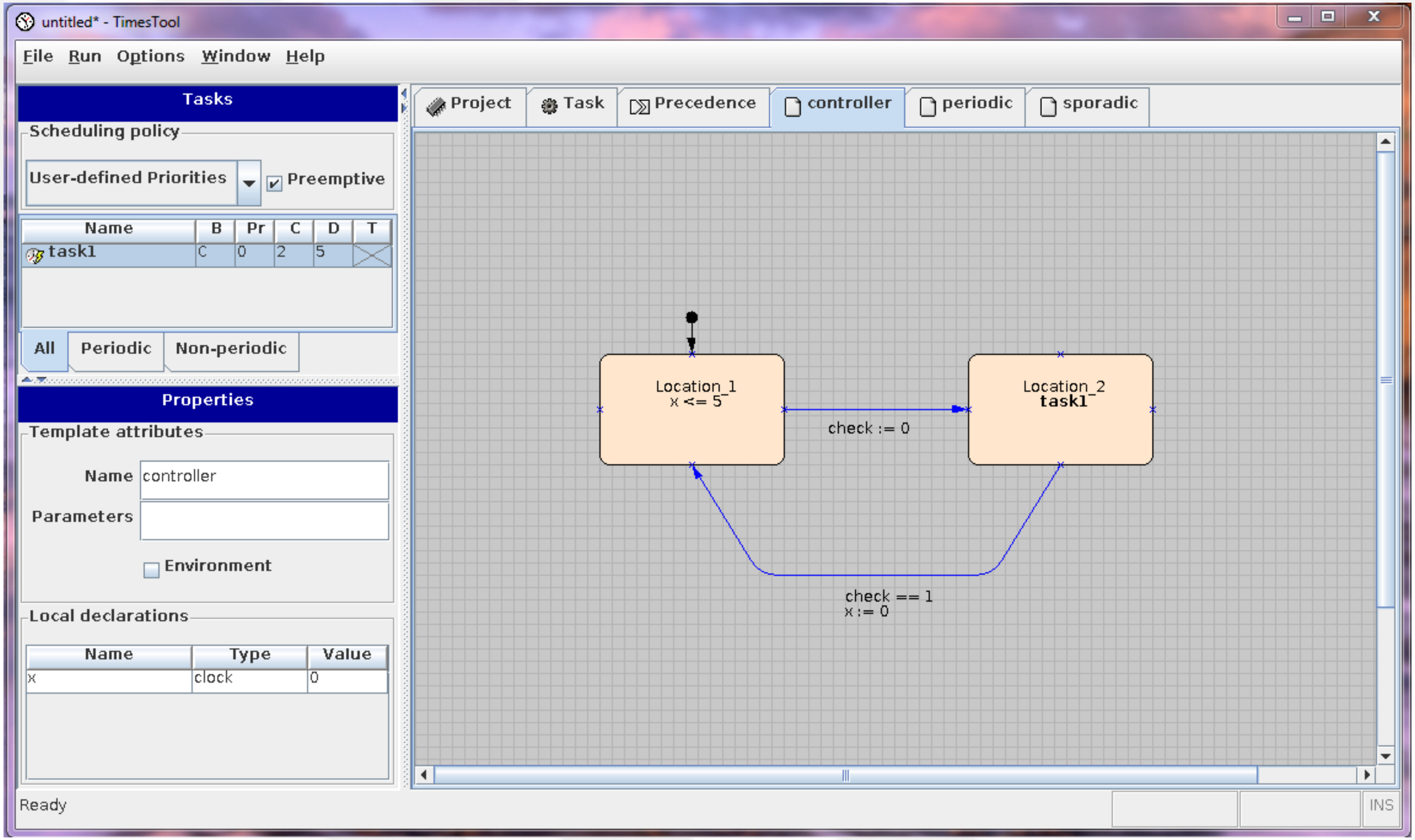


The Graphical Editor (II)

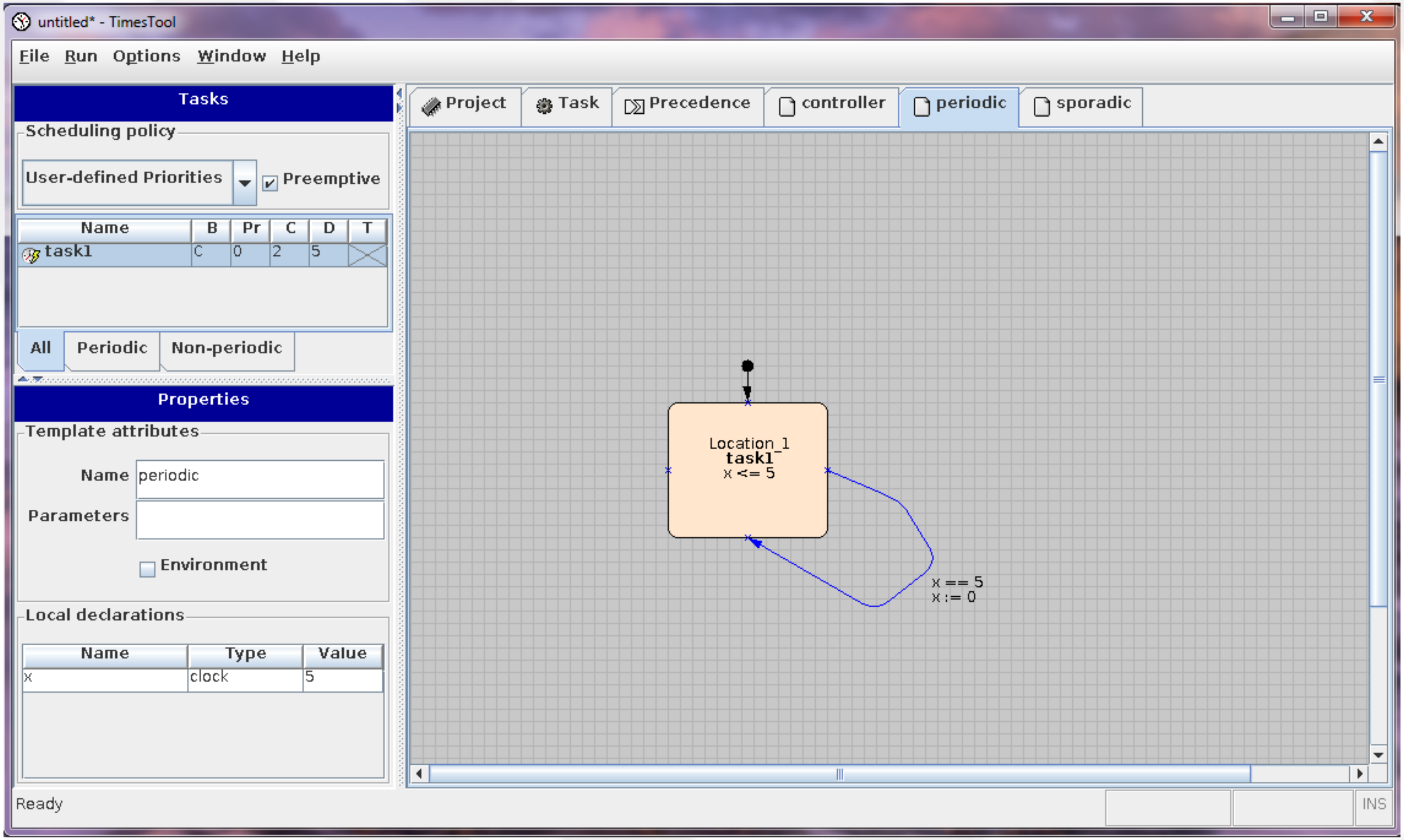
- Scheduling Policy:
  - Deadline monotonic
  - Rate monotonic
  - User-defined
- Task:
  - Name
    - The task name
  - Behavior (B)
    - Task model
      - C: controlled
      - P: periodic
      - S: sporadic
  - Priority (P)
    - The bigger the number, the higher the priority
  - Execution Time (C)
    - Time required for a task to complete its execution in the worst case
  - Deadline (D)
    - Task execution deadline
  - Period (T)
    - Time interval between releases of two sequential task instances for periodic tasks
    - Minimal inter-arrival time for sporadic tasks



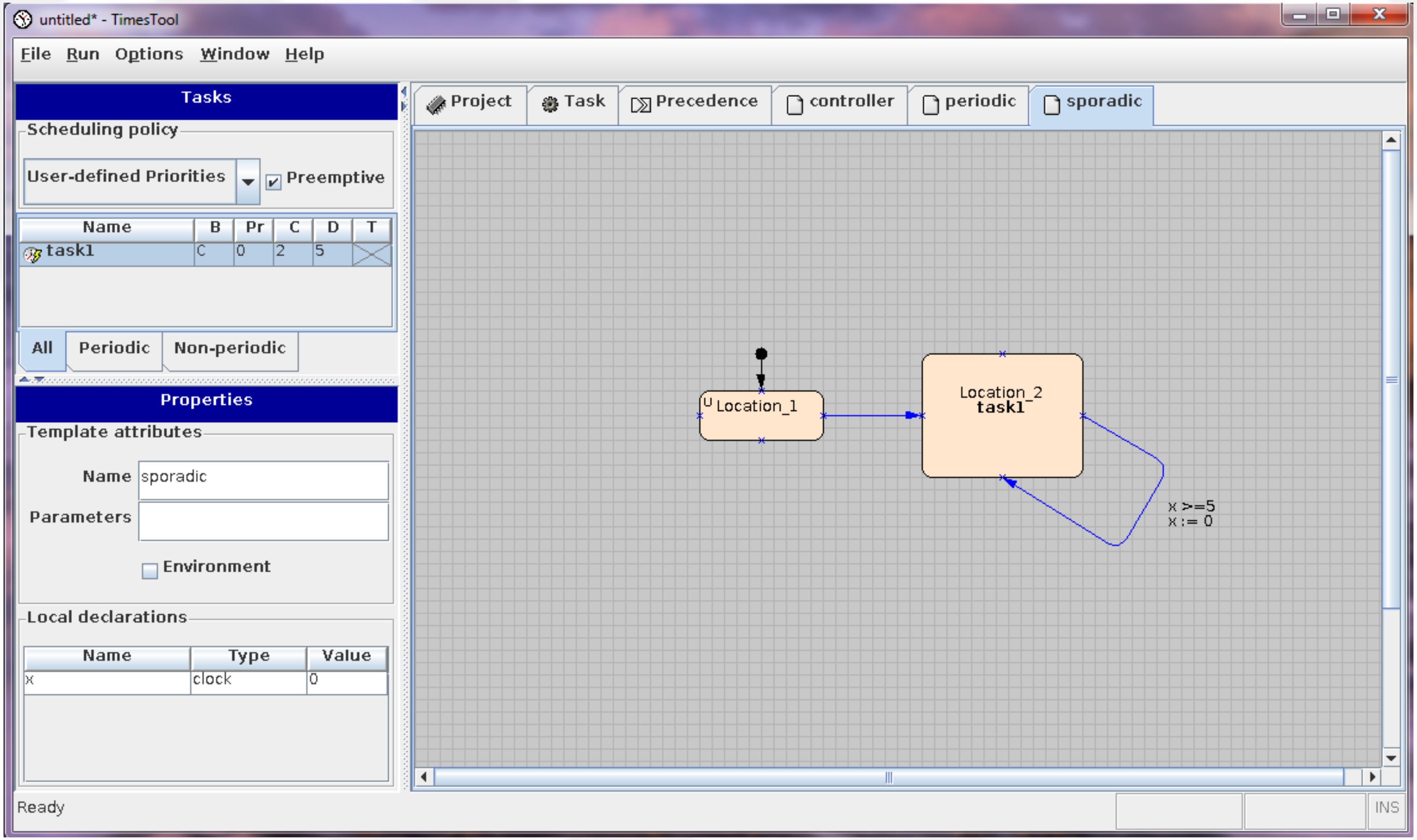
The Graphical Editor (III)



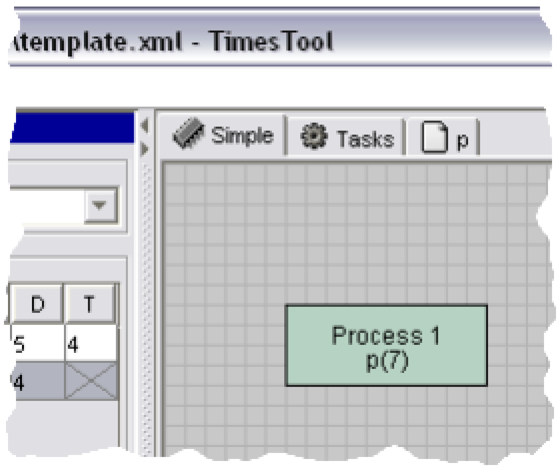
The Graphical Editor (IV)



The Graphical Editor (V)



The Graphical Editor (VI)

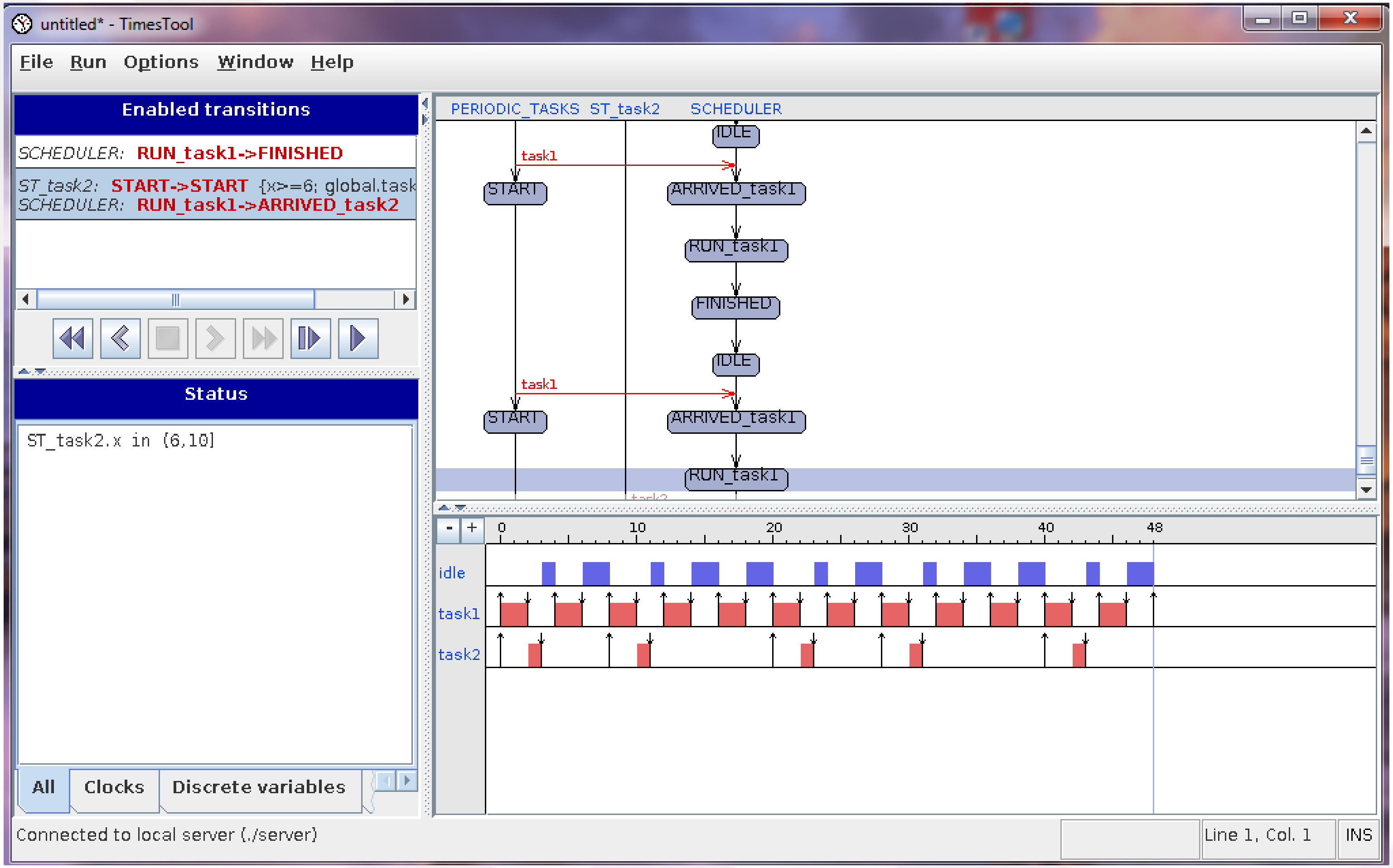


- Processes are instances of automaton templates
- Processes are characterized by:
  - The *name* of the instance
    - E.g., Process 1
  - The *identifier* of the template modeling the automaton
    - E.g., p()
  - The (Actual) parameters of the template
    - E.g., (7)

The Graphical Simulator (I)

- The simulator window is divided into 4 sections:
  - ***Enabled transitions*** show the transitions that the system can take from the given state
    - Step-by-step simulation
      - The user has to choose which transition execute
    - Random simulation
      - The transitions are randomly chosen
  - ***Watches*** report the values of clocks, variables as well as tasks and processor utilization factors
  - ***Message Sequence Chart*** shows the processes interaction
    - Current locations of processes
    - Inter-process synchronizations (performed via named channels)
  - ***Gantt Chart*** shows the timeline
    - Execution of tasks
    - Processor idle time

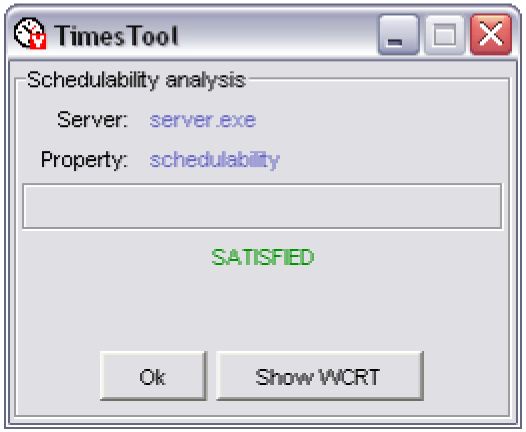
The Graphical Simulator (II)

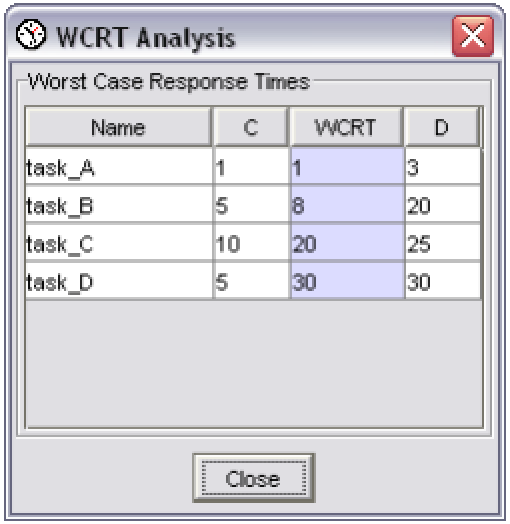


The Schedulability Analysis (I)

- The schedulability analysis establishes if the scheduling policy adopted satisfies the deadlines
  - If yes, the tool reports the worst-case response time (WCRT)
  - Otherwise, the tool returns a counterexample of the schedulability of the tasks

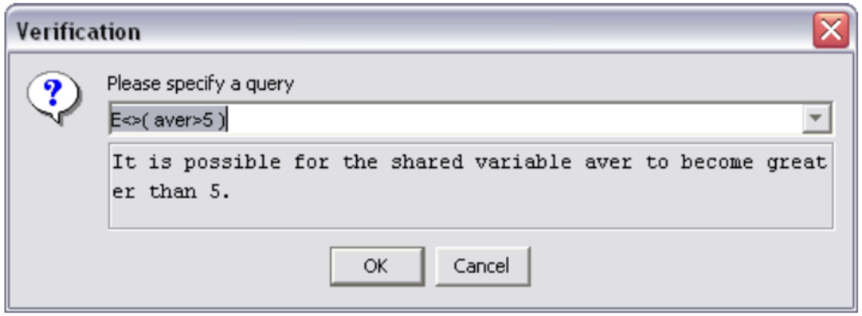
The Schedulability Analysis (II)

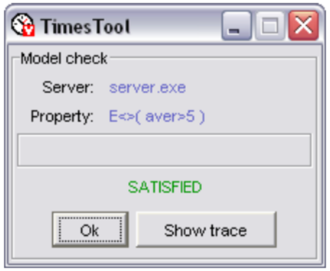




Verification in Times

- With the verification engine it is possible to check properties of the system specified by temporal formulas
- Times adopts the same Query Language of Uppaal
  - State formulae
    - E.g.,
      x == 7
  - Path formulae
    - E.g.,
      A[] not deadlock
      E<> (x > 10)





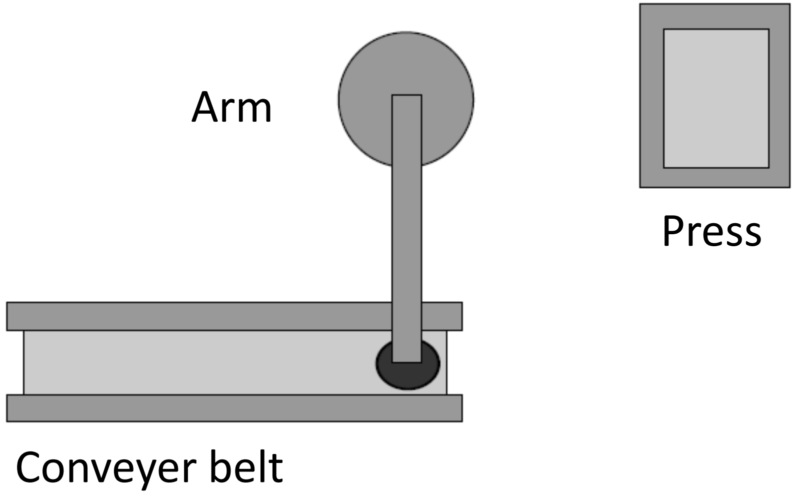
Automatic Code Generation in Times

- Times provides automatic generation of code from the system description
- Currently the only supported target is *Hitachi H8* processor of the LEGO Mindstorm RCX brick running *brickOS*

An industrial plant

EXAMPLE

Example (I)

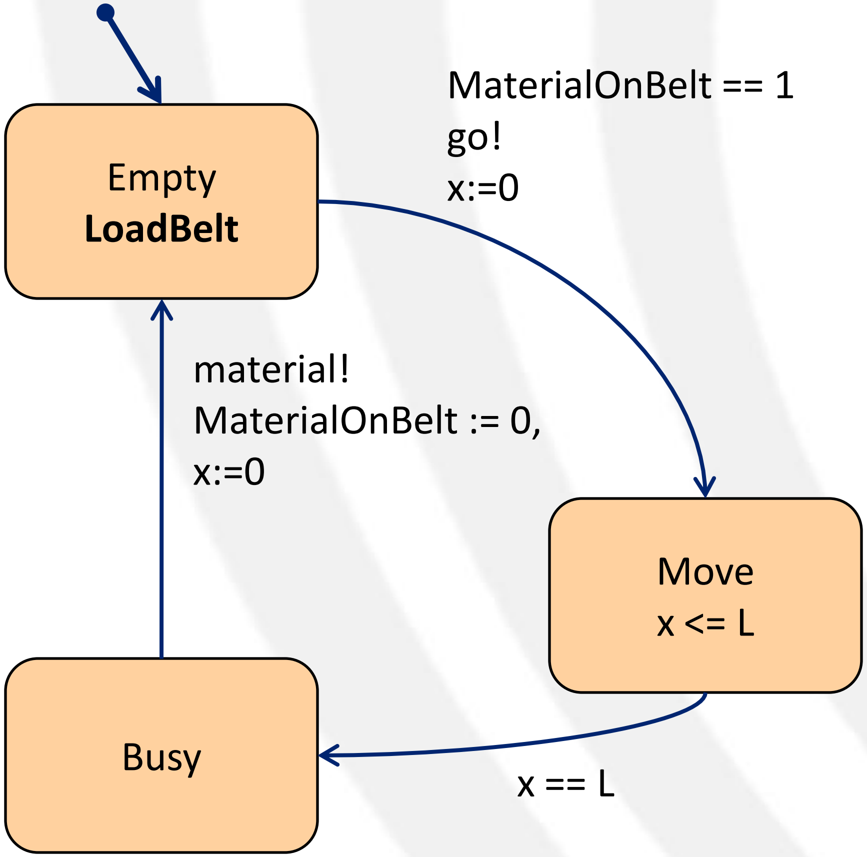


- A piece of material is loaded on the conveyer belt and moved towards the arm
- The arm takes the piece and move it into the press
- The press works the piece
- The controller is unique, with a single processor

Example (II)

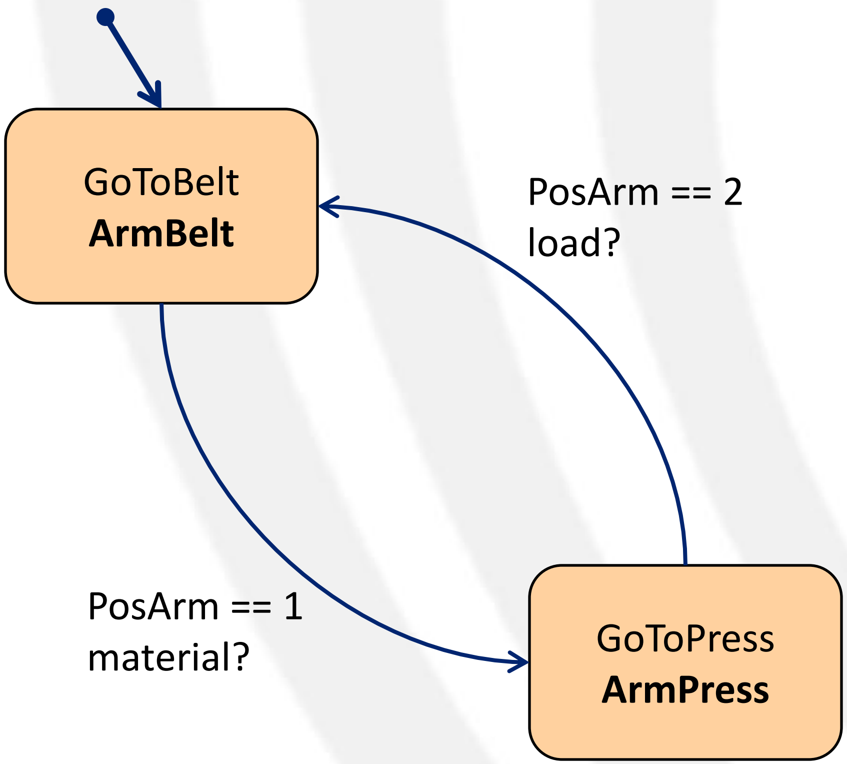
- LoadBelt:
  - Execution time: **1**
  - Deadline: **5**
  - Interface: **`MaterialOnBelt :=1`**
- ArmBelt
  - Execution time: **3**
  - Deadline: **10**
  - Interface: **`PosArm := 1`**
- ArmPress:
  - Execution time: **3**
  - Deadline: **10**
  - Interface: **`PosArm := 2`**
- LoadPress:
  - Execution time: **1**
  - Deadline: **5**
  - Interface: **`PressReady := 1`**

The Belt Automaton



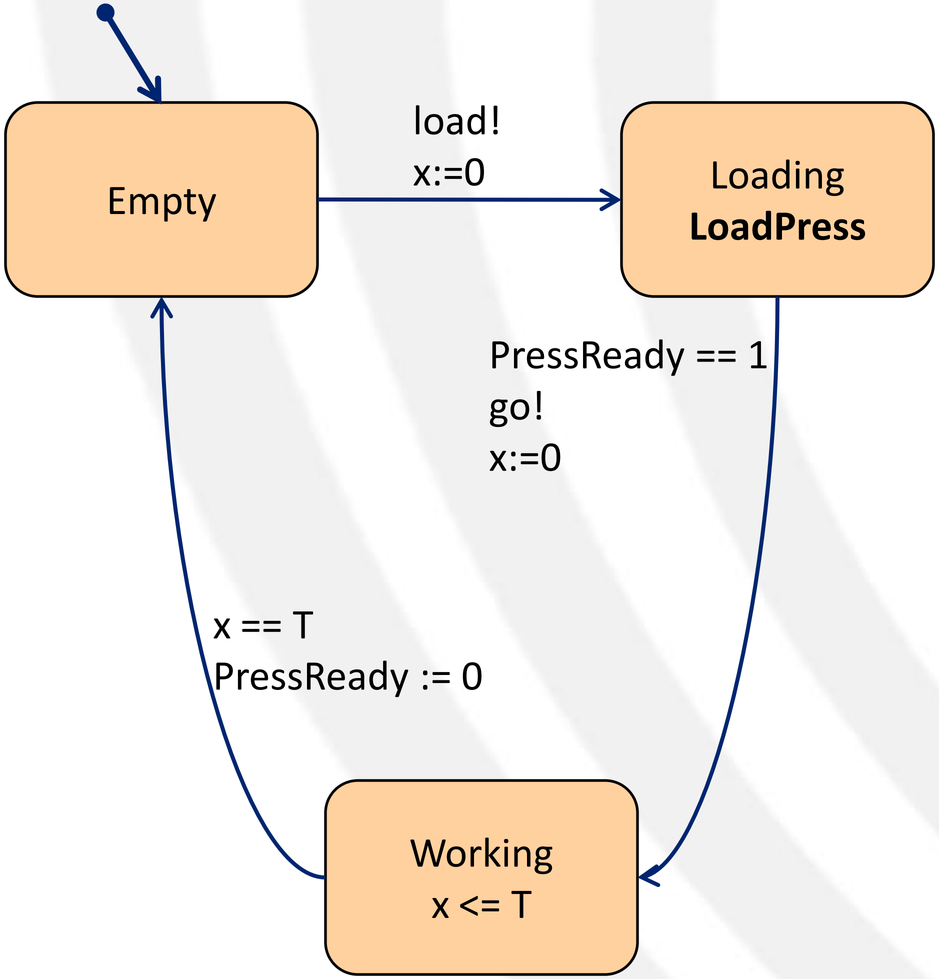
- Template:
  - `Belt`
- Parameters
  - `const L`
- Local variables:
  - `clock x`
- Global variables
  - `int MaterialOnBelt`
    `urgent chan material`
    `urgent chan go`

The Arm Automaton



- Template:
  - `Arm`
- Global variables
  - `int PosArm`
    `urgent chan material`
    `urgent chan load`

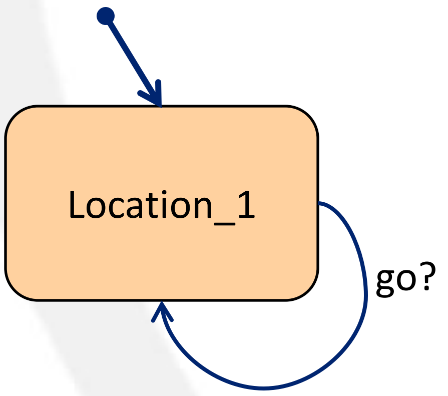
The Press Automaton



- Template:
  - `Press`
- Parameters
  - `const T`
- Local variables:
  - `clock x`
- Global variables
  - `int PressReady`
    `urgent chan load`
    `urgent chan go`

The Urgent Automaton

- This automaton enables *urgent* transitions



- The `go` label represents a global `urgent chan`
- Every urgent transition must be labebed with `go`!
- In this way the transition is performed as soon as possible

System Overview

- Into the Project window insert the following processes:

Ar Arm()	Pre Press(10)
Be Belt(12)	Urg Urgent()

- Processes are instances of templates previously defined
- The conveyer belt takes L = 12 time units to bring pieces
- The press takes T= 10 time units for working

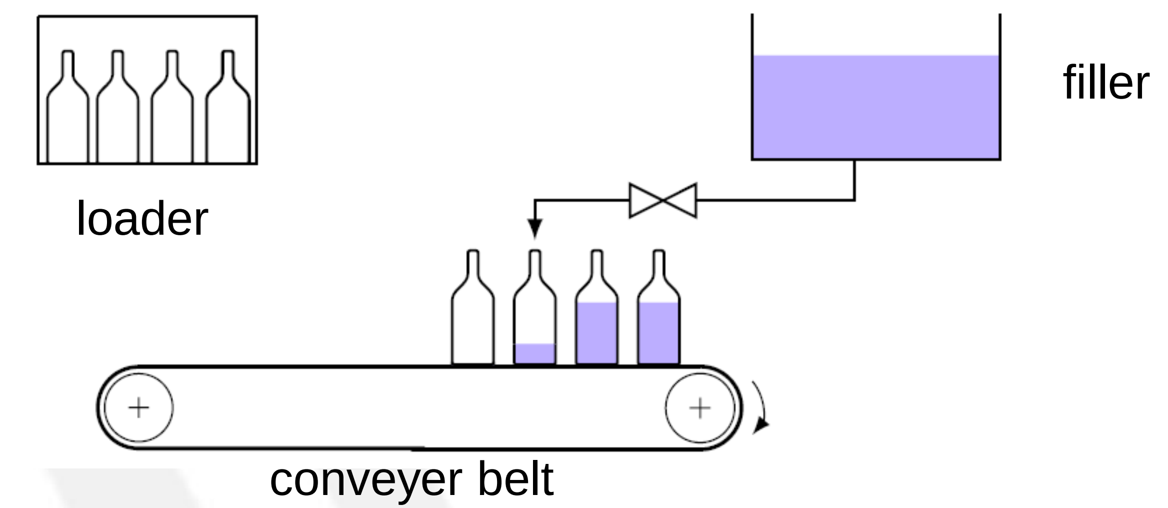
Exercise

- Implement the system with Times
- Perform the schedulability analysis with different policy
  - Which policies allows the schedulability of the system?
- Check the following properties:
  - Deadlocks do not occur
    - A[] not deadlock
  - Every piece stays on the belt at most 15 time units
    - A[] not (Be.x > 15)

The filling bottle system

EXERCISE

Bottle Filling System: Overview



- Bottles are loaded on the conveyer belt 4 a time
- The belt has length 6 and it brings bottle to the filler
- Bottles are filled one by one
- At the beginning there are 2 bottles on the belt

Bottle Filling System: Tasks

- LoadBottles
  - Execution time: **8**
  - Deadline: **20**
  - Interface: `Bottles := Bottles + 4`
- MoveBelt
  - Execution time: **1**
  - Deadline: **5**
  - Interface: `ReadyBottle := 1, Bottles := Bottles - 1`
- FillBottle
  - Execution time: **2**
  - Deadline: **5**
  - Interface: `FullBottle := 1`

Exercise

- Implement the system with Times
- Perform the schedulability analysis with different policy
  - Which policies allows the schedulability of the system?
- Design the system in order to satisfy the following specifications
  - Deadlocks do not occur
    - A[] not deadlock
  - The conveyer belt will never be empty
    - A[] Bottles > 0
  - The bottles on the belt are at most 6
    - A[] Bottles <= 6