Abstracting Timing Information in UML State Charts via Temporal Ordering and LOTOS

Valentin Chimisliu
Institut für Softwaretechnologie
Technische Universität Graz
Graz, Austria

Franz Wotawa
Institut für Softwaretechnologie
Technische Universität Graz
Graz, Austria

Motivation

- Introduction of academic model based test case generation tools into industry
- Hide the user unfriendly formal language
- Use of a widely accepted modeling language
- Abstract concrete timing information using temporal ordering

- UML State Charts
- LOTOS
- Model Description
- Model Transformation
- Future Work
- Conclusions

UML State Charts

- UML behavioral diagrams
- Describe reactions of a system to external stimuli
- Hierarchical structures
- Composed of states, pseudostates and transitions

UML State Charts

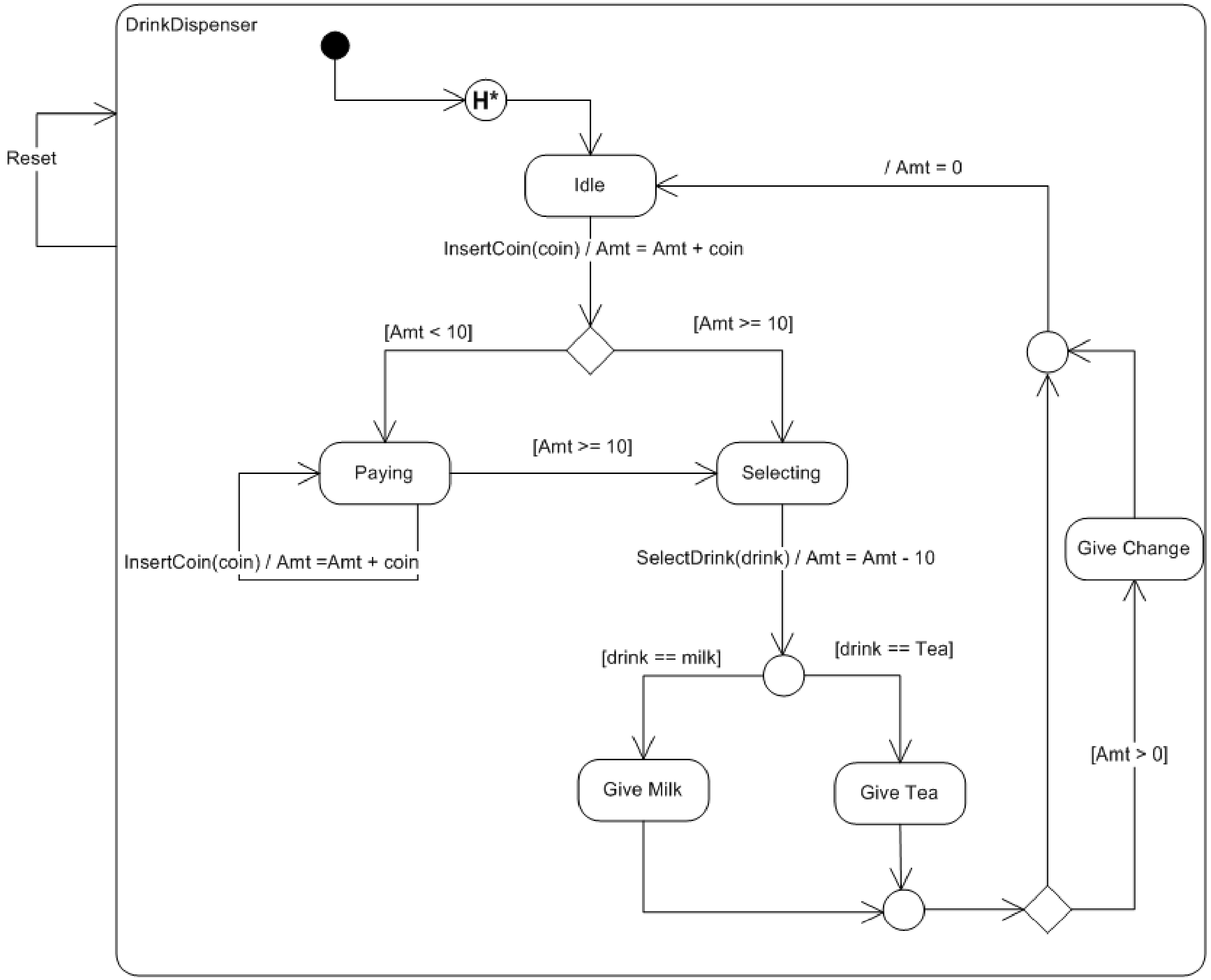
- Atomic execution
- Timeouts
- Run-to-Completion step
- Asynchronous communication

UML State Charts

$$SC = (S, T, V, i_{ps0})$$

- States: $S = S_s \cup S_{ps} \cup S_c$
- Pseudostates: $S_{ps} = I_{ps} \cup J \cup E_{en} \cup E_{ex} \cup C \cup H$
- Transitions: $T \subseteq S \times L \times S$, $L \subseteq E \times G \times A$
- Variables: V
- Initial State: i_{ps0}

UML State Charts



- UML State Charts
- LOTOS
- Model Description
- Model Transformation
- Future Work
- Conclusions

LOTOS

- **ISO** standard
- Based on process algebra and on the abstract data type language **ACT ONE**.
- describes a system as a set of interacting communicating processes
- process behavior is described through behavioral expressions
- processes communicate through event gates

LOTOS

Specification structure

```
“specification” name_spec [gate list]
  (parameter list) : functionality

type definitions

“behavior”
  behavior expression
“where”
  type definitions
  process definitions
“endspec”
```

LOTOS

- Some LOTOS operators:
  - **“;”** - **action prefix -** startCar;DRIVE.
  - “**[ ]**” - **choice** - (CarStarts;DRIVE) **[ ]** (CarBroken; WALK)
  - **“||”** – **full synchronization** - (a;b;c;X)**||**(a;b;Y) includes a;b…
  - **“|||”** – **interleaving** - (a;b;c;P) ||| (x;y;T) includes a;x;y;b;c...
  - **“|[< gates >]|”** - **partial synchronization** - (a;b;c;P) |[b]| (b;y;T) includes a;b;y;c... and a;b;c;y... .
  - “**stop**” - **inaction** - a system that can not show any action.

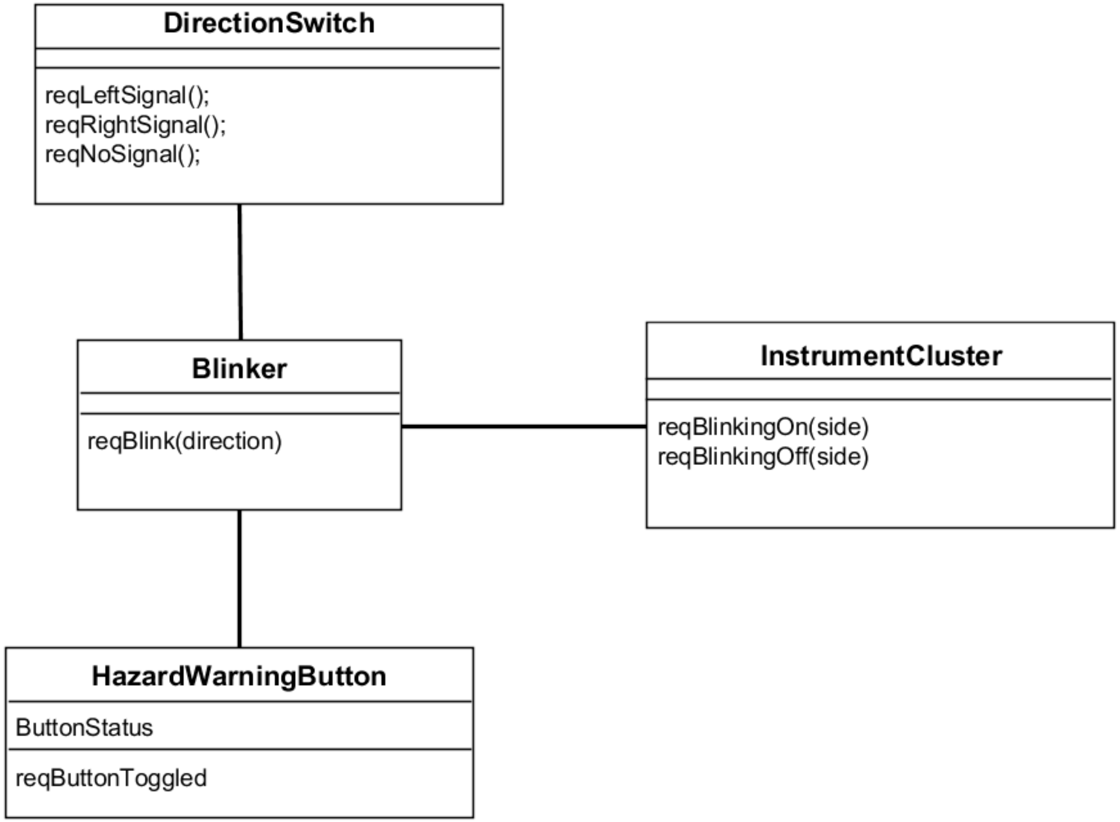
- UML State Charts
- LOTOS
- Model Description
- Model Transformation
- Future Work
- Conclusions

Model Description

- Class diagram
- Each class has attached a statechart
- Classes communicate through data carrying events
- Asynchronous communication
- FIFO queue

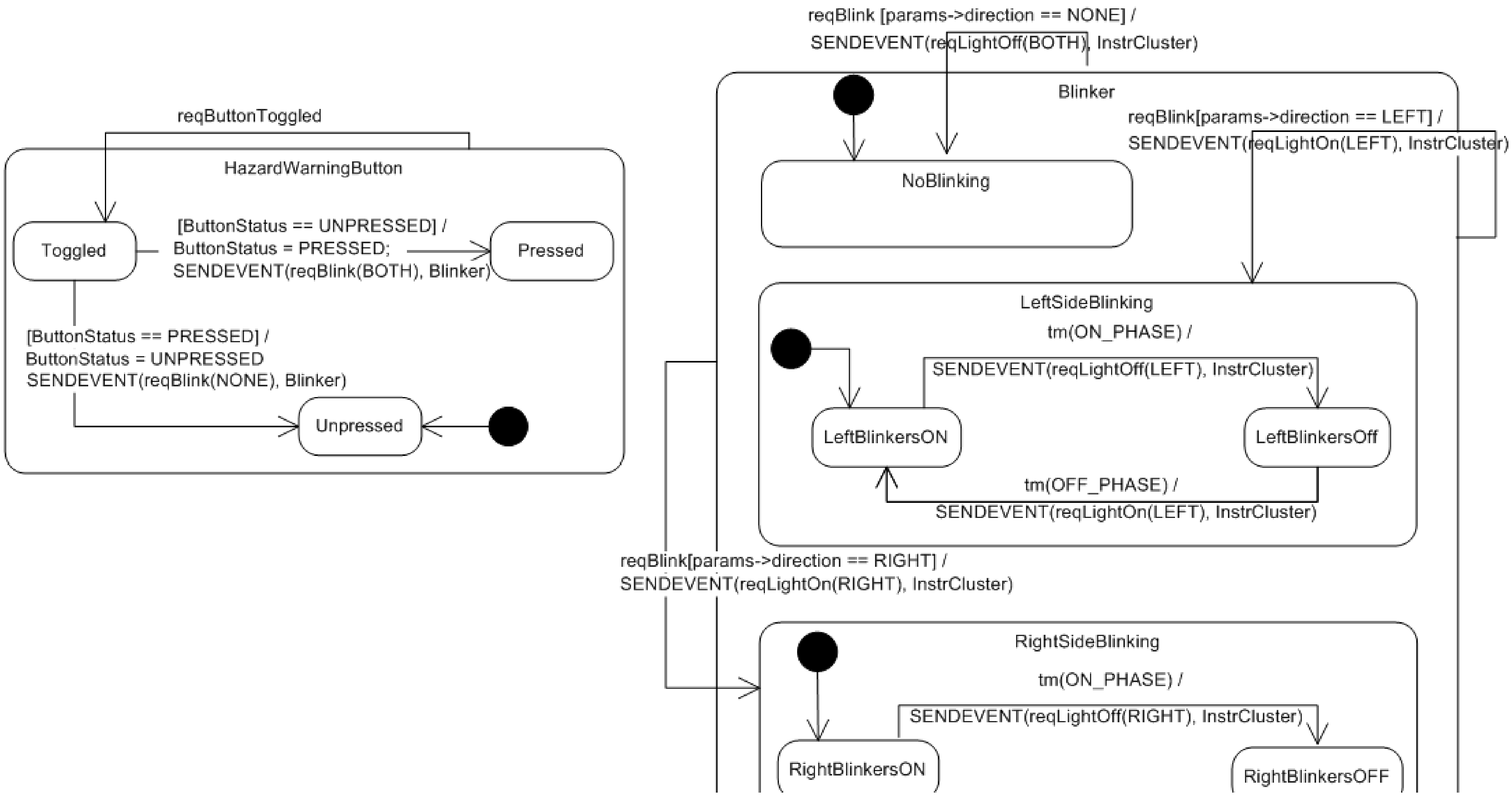
Direction Indication

System Structure



Direction Indication

System Behavior



- UML State Charts
- LOTOS
- Model Description
- **Model Transformation**
- Future Work
- Conclusions

Model Transformation

- Flattened Statechart ➔ LOTOS process
- UML States ➔ LOTOS processes
- Variables ➔ Process parameters
- Transitions ➔ Behavioral expressions
  - triggering events ➔ gates offerings
  - assignment statements ➔ process parameter operations
  - generation of events ➔ gates offerings
- Asynchronous communication➔ FIFO event queue

Model Transformation

System Structure in LOTOS

(HazardWarningButton[alfaGate, inGate, toQueue,outGate,time]
(UNPRESSED, tmQueue_nil) **|||**

Blinker[alfaGate, inGate, toQueue, outGate, time]

(tmQueue_nil) **|||**

InstrumentCluster[alfaGate, inGate, toQueue, outGate, time]

(tmQueue_nil) **|||**)

||

Synchronization[alfaGate, inGate, toQueue, outGate, time]

(true, true, true, true, tmQueue_nil)

Model Transformation

Main LOTOS Abstract Data Types

- **ComModels** – identifies the communicating models (UML class)
- **Transition** – represents properties of UML transitions (id, containing model, etc.)
- **Event** – the events used for communication
- **EventQueue** – used to store events generated during run to completion steps
- **TmEvent** – represents the timeout event
- **TimeQueue** – container for the timeout events ready to elapse

Model Transformation

Time Queue Abstract Data Type

type TIMEQUEUE is TMEVENTS, ComModels

sorts

TmEvent

opns

t_nil (*! **constructor** *) : -> TmQueue

add (*! **constructor** *) : TmEvent, TmQueue -> TmQueue

min: TmQueue -> TmEvent

pop_min : TmQueue -> TmQueue

decr_queue : Nat, TmQueue -> TmQueue

pop_ModelTrans: Model, TmQueue -> TmQueue

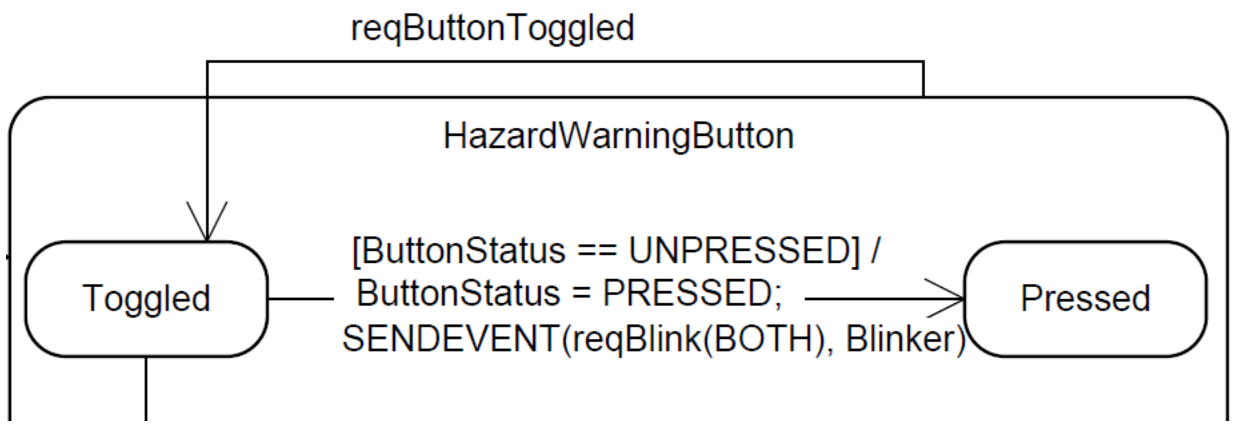
popTrans: Transition, TmQueue -> TmQueue

Model Transformation

States and Variables

Control Gate | Input Gate | Event Communication Gate | Output Gate | Time Gate

process Toggled [alfaGate, inGate, toQueue, outGate, time]
(ButtonStatus: Nat, tmQueue: TmQueue,ellapsedTime: Nat) : **noexit:=**



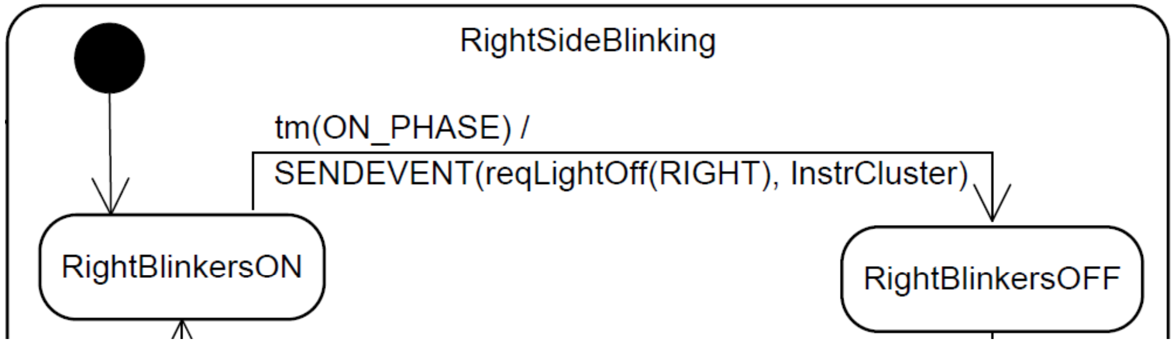
Model Transformation

Transitions

alfaGate !Blinker !true !min(tmQueue);

inGate !Blinker_T76 ?msg:TmEvent [(msg == getEvByTrans(Blinker_T76 , tmQueue))];
time ?tmt: Nat !itsTaster;
outGate !tmQueue !timeSynch !ellapsedTime ;
toQueue !reqLightOff(RIGHT, InstrCluster);

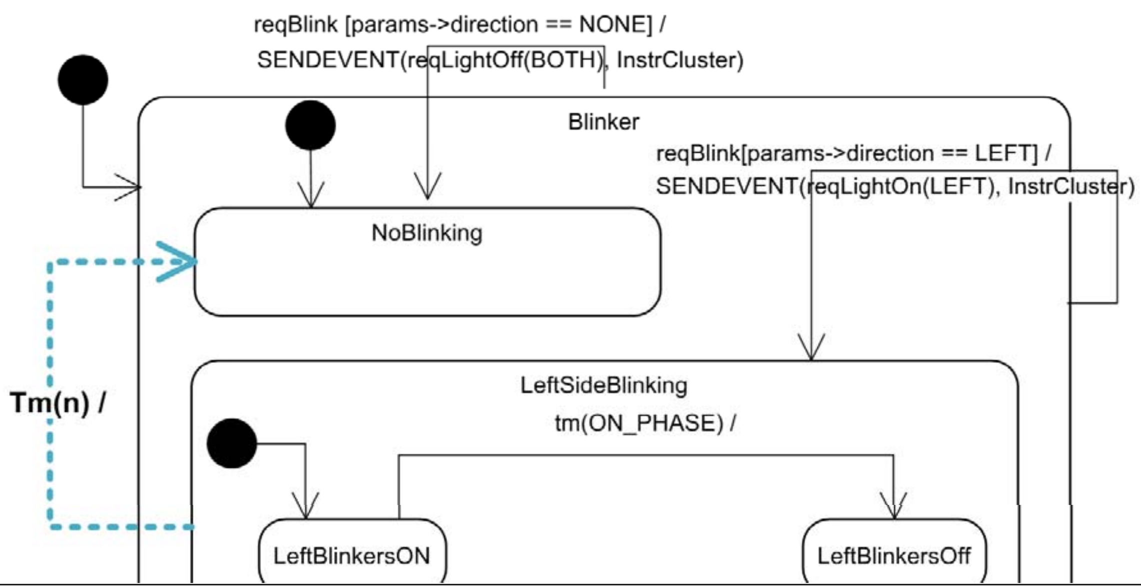
RightBlinkersOFF[alfaGate, inGate, toQueue, outGate, time]
add(tm(!Blinker_T78 , OFF_Phase), popTrans(Blinker_T76 ,tmQueue)))



Model Transformation

Transitions - Timing and Hierarchy

```
alfaGate !Blinker !true !min(tmQueue);
………….
RightBlinkersOFF[alfaGate, inGate, toQueue, outGate, time]
        (
        add(
                tm(!Blinker_T78 , OFF_Phase),
                (decr_queue(getParam_timeout(min(tmQueue)),popTrans(Blinker_T76 ,tmQueue)
            )
        )
```



Model Transformation

Communication and Timing

- Synchronization Process purposes:
  - Communication within the system
  - Timing control
  - Run to Completion Step
  - Atomic firing of transitions

Model Transformation

The Synchronization porcess

```
process Synchronization [alfaGate, inGate, toQueue, outGate, time]
    (stable_Blinker: Bool, stable_HazardWButton: Bool, ….. ,
    SyncTimeQueue: TmQueue):  noexit:=

    <get_models_into_stable_state>
    [ ]
    <consume_timeout_event>
    [ ]
    <consume_event>

endproc
```

Model Transformation

The Synchronization porcess

<get_models_into_stable_state>

- <fire_completion_trans>
- <empty_event_queue>
- **process** SYNCHRONIZATION[<comm_gates>](<synch_params>)

Model Transformation

The Synchronization porcess

<consume_timeout_event>

inGate ?tr:Transition ?msg:TmEvent [<tm_guard>];
<propagate_timing_info>

<tm_guard>:= not(t_empty(SyncTimeQueue))
and (msg == min(SyncTimeQueue))

<propagate_timing_info>:=
time !getParam_timeout(min(SyncTimeQueue)) !<model_id>;

- UML State Charts
- LOTOS
- Model Description
- Model Transformation
- Future Work
- Conclusions

Future Work

- application of verification and test case generation techniques
- Generation of test purposes directly on the statechart level
- Application of abstraction techniques

- UML State Charts
- LOTOS
- Model Description
- Model Transformation
- Future Work
- Conclusions

Conclusions

- Model transformation aimed at allowing the application of techniques like test case generation provided by academic tools to systems described using the widely industry accepted UML notation.

- Considered distributed timed control oriented systems using asynchronous communication
- Constructs for abstracting timing aspects
- Appropriate abstractions should be applied during the modeling process

THANK YOU !