TIMED AUTOMATA APPROACH TO DISTRIBUTED AND FAULT TOLERANT SYSTEM VERIFICATION

Jan Krákora, Libor Waszniowski and Zdeněk Hanzálek

Czech Technical University in Prague
Centre for Applied Cybernetics, Department of Control Engineering
Karlovo nám. 13, 121 35 Prague 2, Czech Republic
{krakorj, xwasznio, hanzalek}@fel.cvut.cz

Typical Control Application

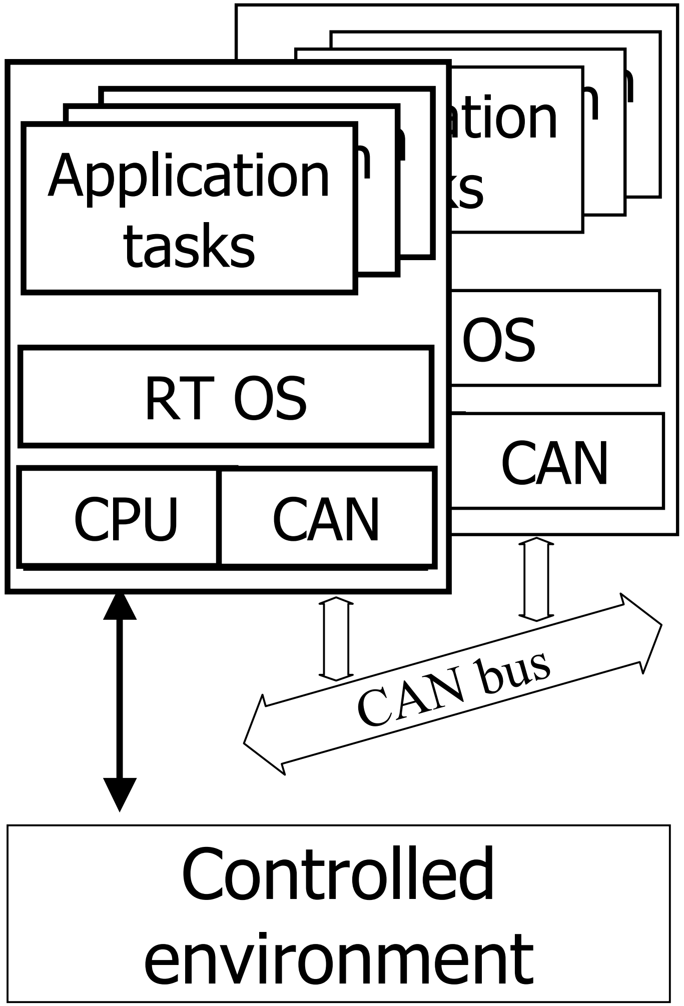
Objective:

to verify the system with respect to its specification

(unsafe state is avoided, end2end response times,…)

Approach:

- •Create fine grain model (timed automata),
- •Formalize specification (subset of temporal logic),
- •Use a Model Checking tool (UPPAAL)



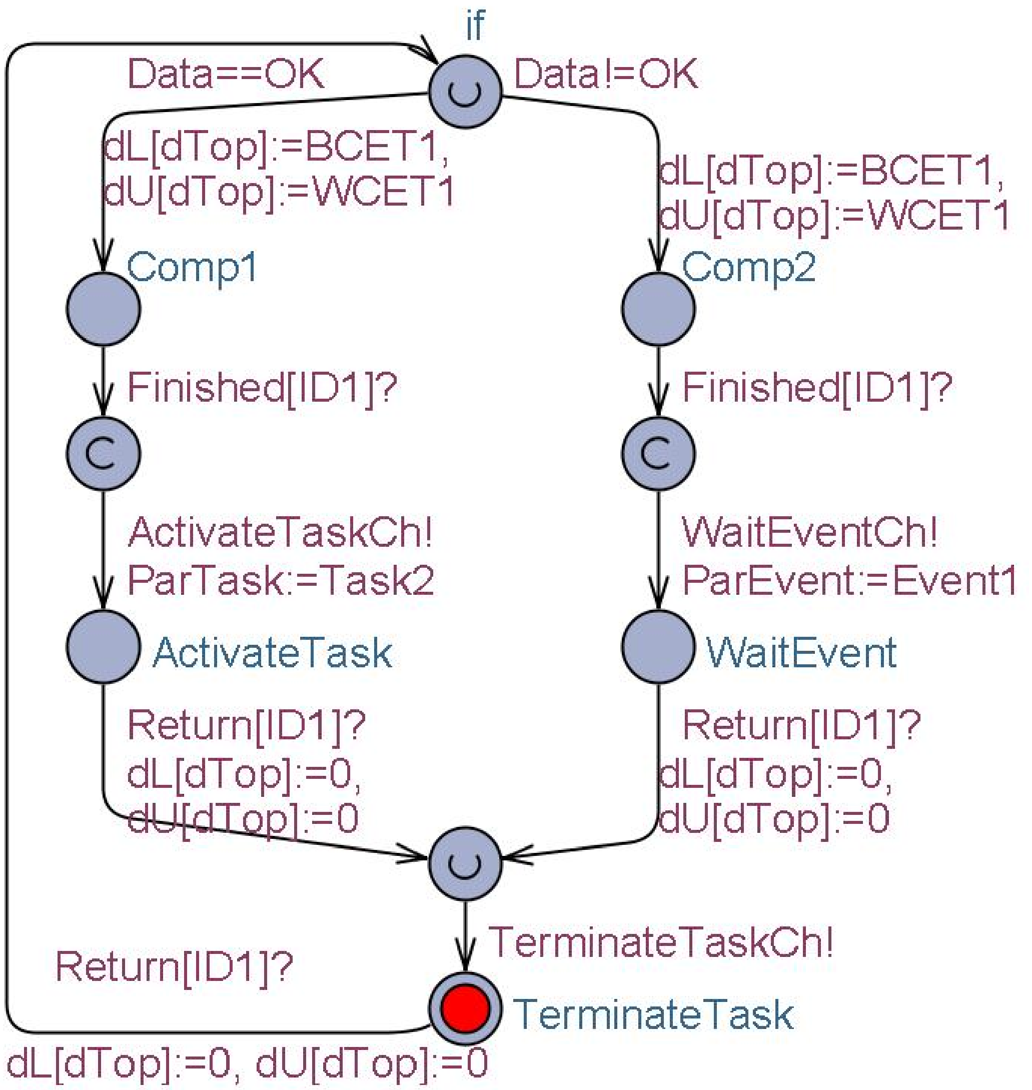
Fine Grain Model:

- ➢Tasks and ISR internal structure
- ➢OS services
- ➢Scheduling policy
- ➢Communication layer
- ➢Controlled environment

Example of Task Internal Structure

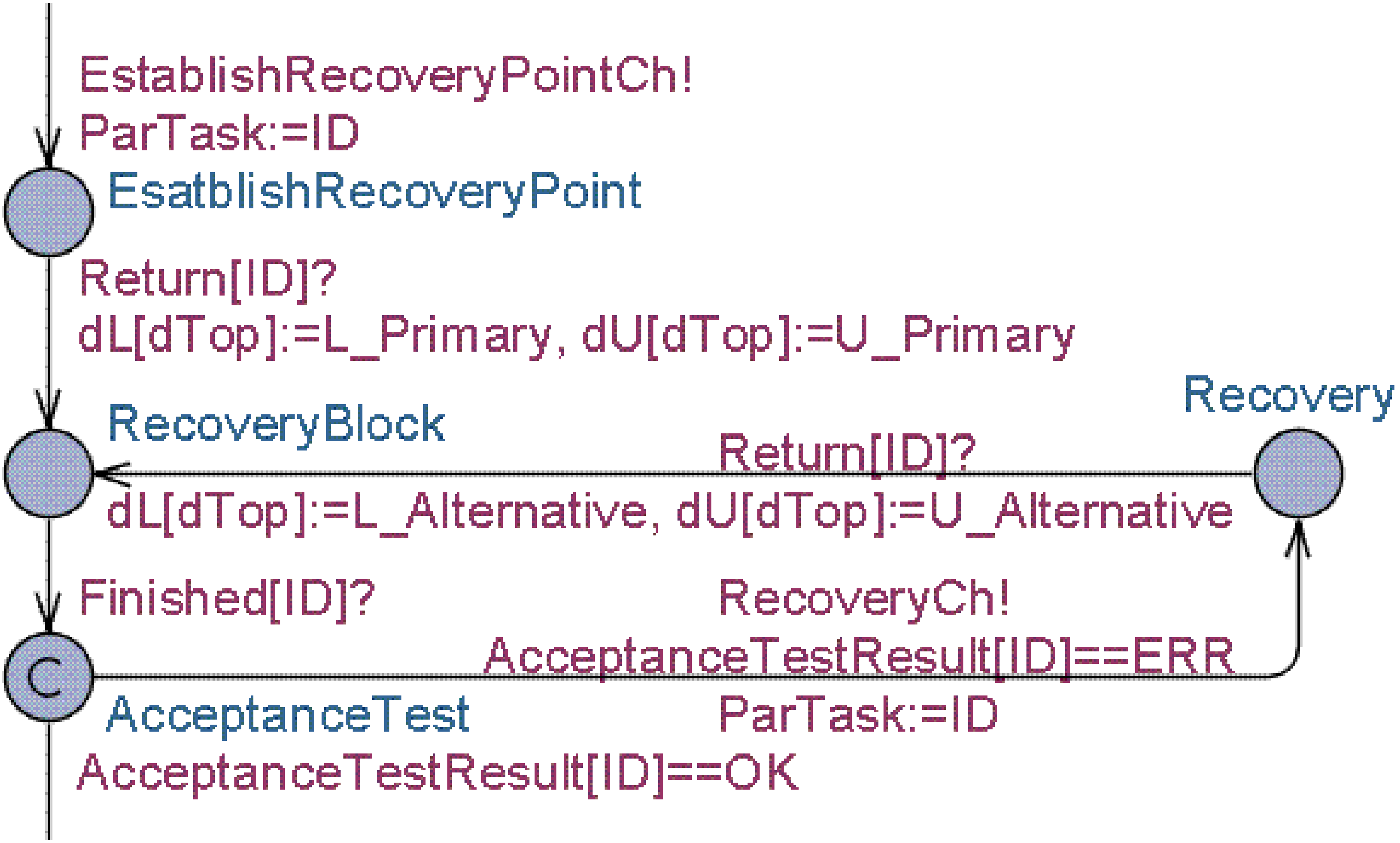
```
Task1()
{
 if (Data==OK)
   {
     Comp1;    // C∈⟨BCET1, WCET1⟩
     ActivateTask(Task2);
   }
 else
   {
     Comp2;    // C∈⟨BCET2, WCET2⟩
     WaitEvent(Event1);
   };
 TerminateTask();
}
```

Controlled environment



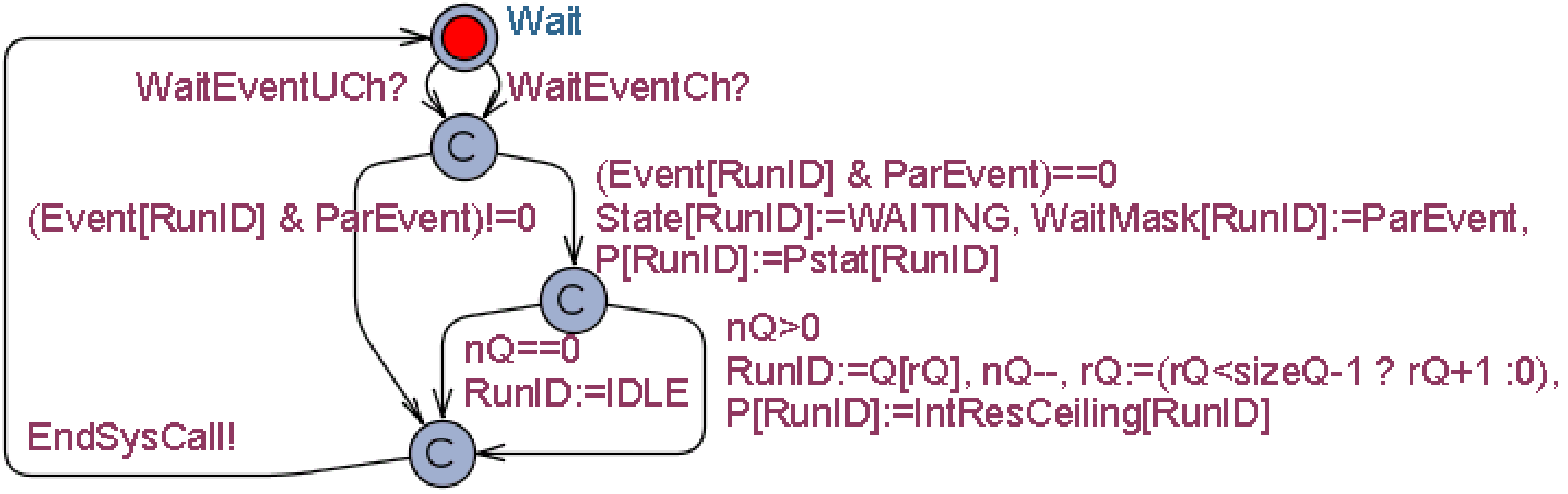
Example of Fault Tolerant Task – Recovery Blocks

```
Ensure (AcceptanceTestResult==OK)
by
   PrimaryBlock;
else by
   AlternativeBlock;
else Err;
```


EstablishRecoveryPointCh!
ParTask:=ID
EsatblishRecoveryPoint
Return[ID]?
dL[dTop]:=L_Primary, dU[dTop]:=U_Primary
RecoveryBlock
Recovery
Return[ID]?
dL[dTop]:=L_Alternative, dU[dTop]:=U_Alternative
Finished[ID]?
RecoveryCh!
AcceptanceTestResult[ID]==ERR
C
AcceptanceTest
ParTask:=ID
AcceptanceTestResult[ID]==OK

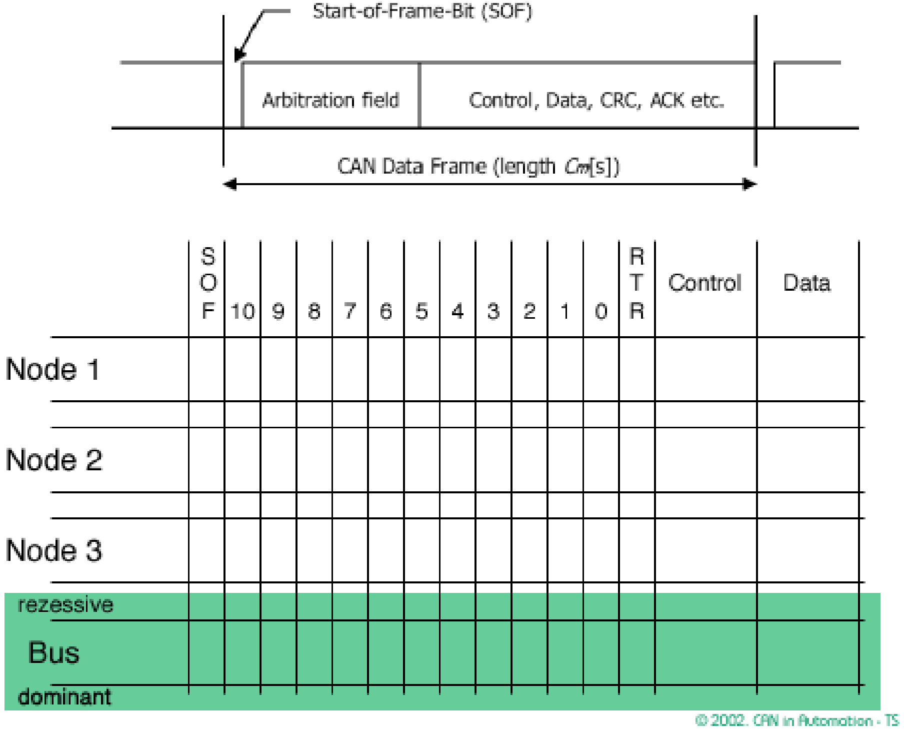
Example of OS Service Model - WaitEvent

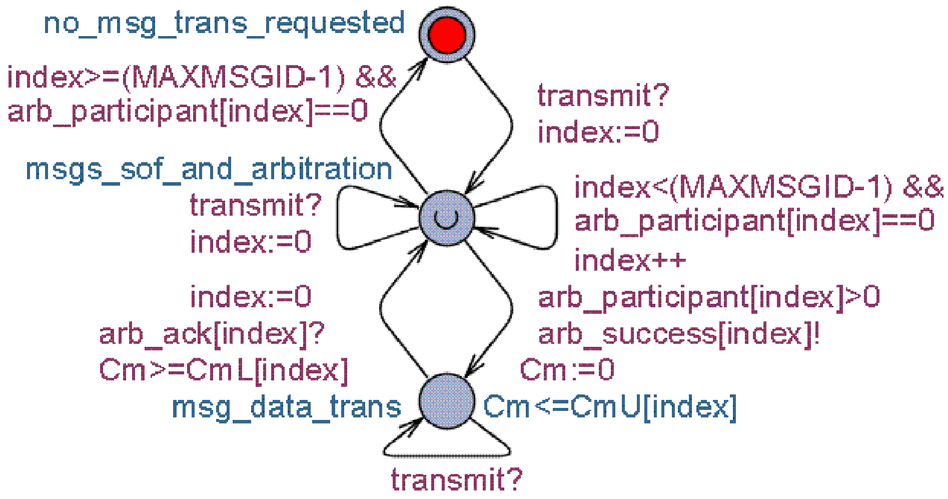
```
WaitEvent (Mask)
{
 if ((Event[RunID] & Mask) == 0)
 {
   State[RunID] := WAITING;
   WaitMask[RunID] := Mask;
   Release Internal Resource;
   RunID := Extract Top of ReadyQ;
   ContextSwitch;
   Get Internal Resource;
   State[RunID] := RUNNING;
 }
 return E_OK;
};
```


Wait
WaitEventUCh?
WaitEventCh?
C
(Event[RunID] & ParEvent)!=0
(Event[RunID] & ParEvent)==0
State[RunID]:=WAITING, WaitMask[RunID]:=ParEvent,
P[RunID]:=Pstat[RunID]
C
nQ==0
RunID:=IDLE
nQ>0
RunID:=Q[rQ], nQ--, rQ:=(rQ<sizeQ-1 ? rQ+1 :0),
P[RunID]:=IntResCeiling[RunID]
EndSysCall!
C

Example of Communication Layer -CAN

Arbitration technique





no_msg_trans_requested

index>=(MAXMSGID-1) &&
arb_participant[index]==0

transmit?
index:=0

msgs_sof_and_arbitration

transmit?
index:=0

index<(MAXMSGID-1) &&
arb_participant[index]==0
index++

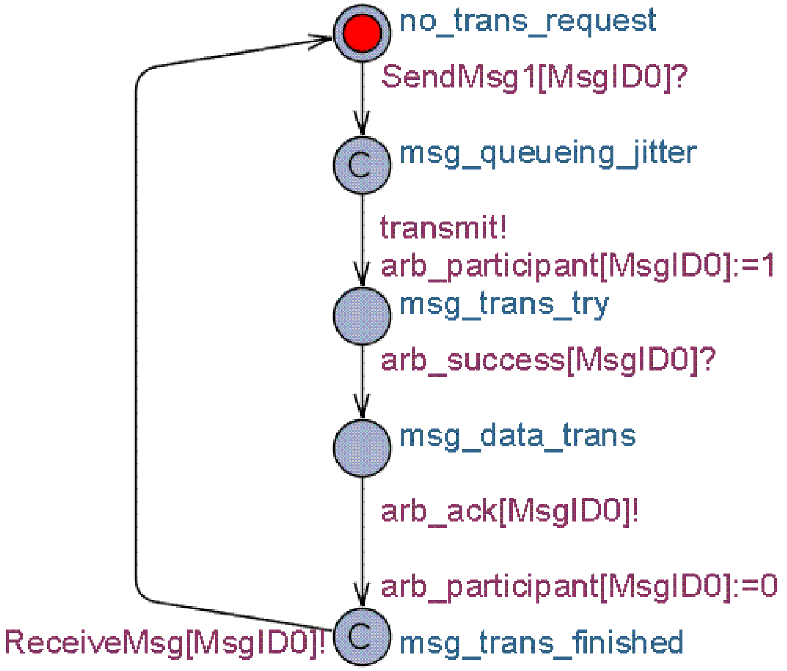
index:=0
arb_ack[index]?
Cm>=CmL[index]

arb_participant[index]>0
arb_success[index]!
Cm:=0

msg_data_trans

Cm<=CmU[index]

transmit?



no_trans_request

SendMsg1[MsgID0]?

msg_queueing_jitter

transmit!
arb_participant[MsgID0]:=1

msg_trans_try

arb_success[MsgID0]?

msg_data_trans

arb_ack[MsgID0]!

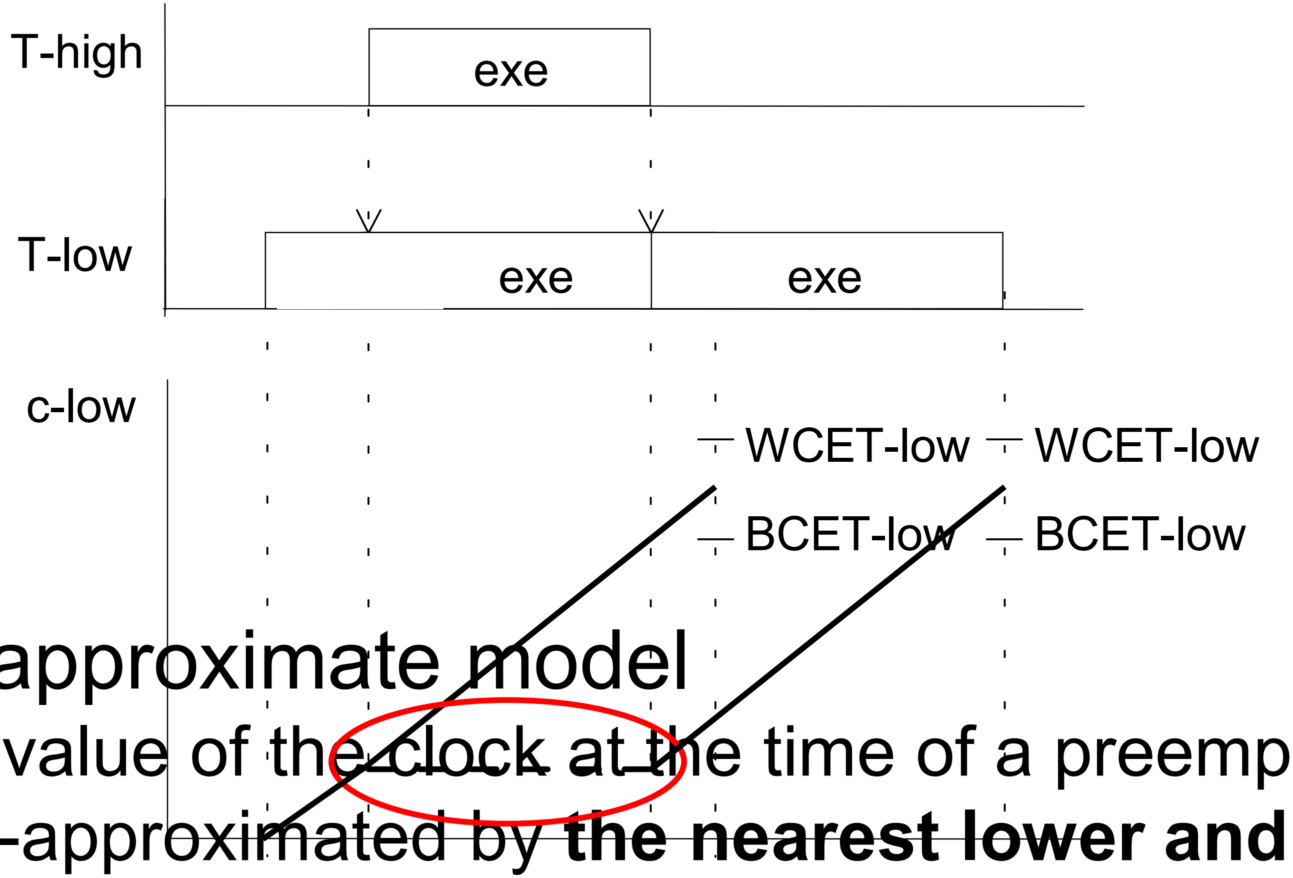
arb_participant[MsgID0]:=0

ReceiveMsg[MsgID0]!

msg_trans_finished

Transmission time
(given by message length and data ra

1st Essential Problem – Preemption



⇨ Over-approximate model

(the value of the clock at the time of a preemption is over-approximated by **the nearest lower and upper integer**)

In Time Automata, **clock variable** measuring process execution time **cannot be stopped** when preemption occurs (price paid for decidability of model checking probem).

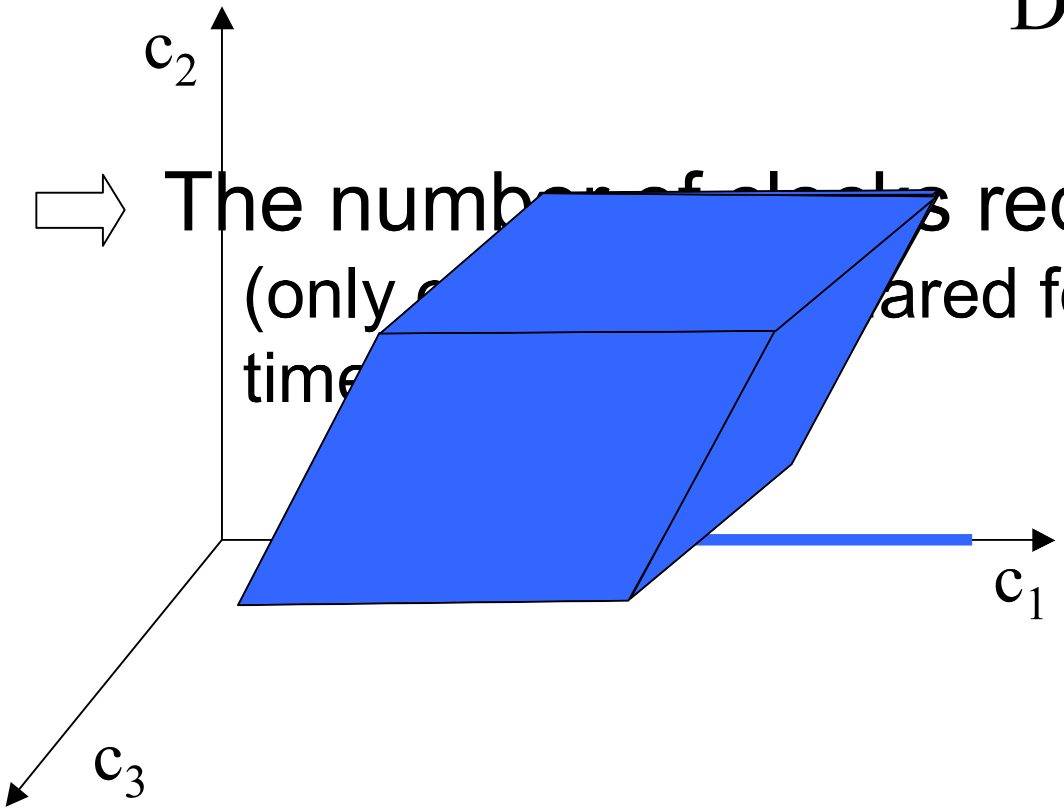
2nd Essential Problem – State-space explosion

The complexity of the model-checking verification **exponentially grows with the number of clocks**

⇨ The numb s reductio

(only c ared for mea on

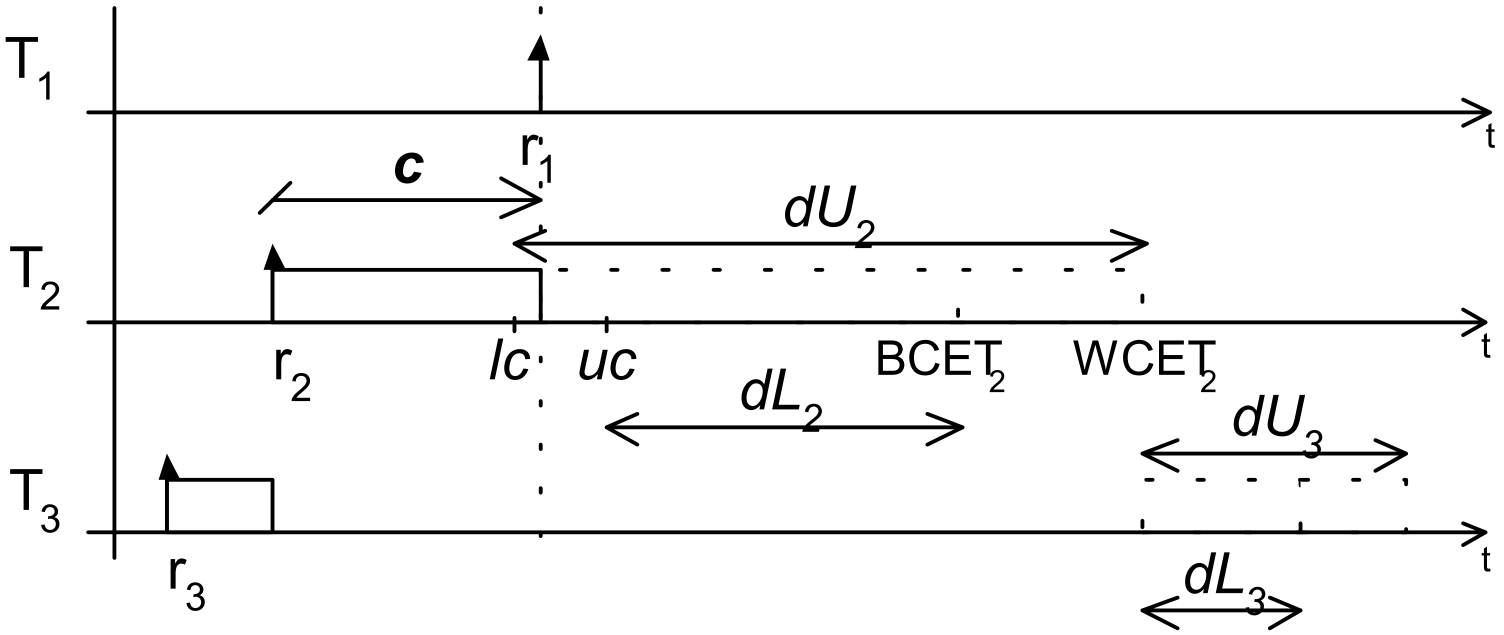
time



DBM

	c_0	c_1	c_2	c_3
c_0				
c_1				
c_2				
c_3				

Implementation



Δ-list

dL	dL_3	$BCET_2$	-	-
dU	dU_3	$WCET_2$	-	-

⇧ *dTop*

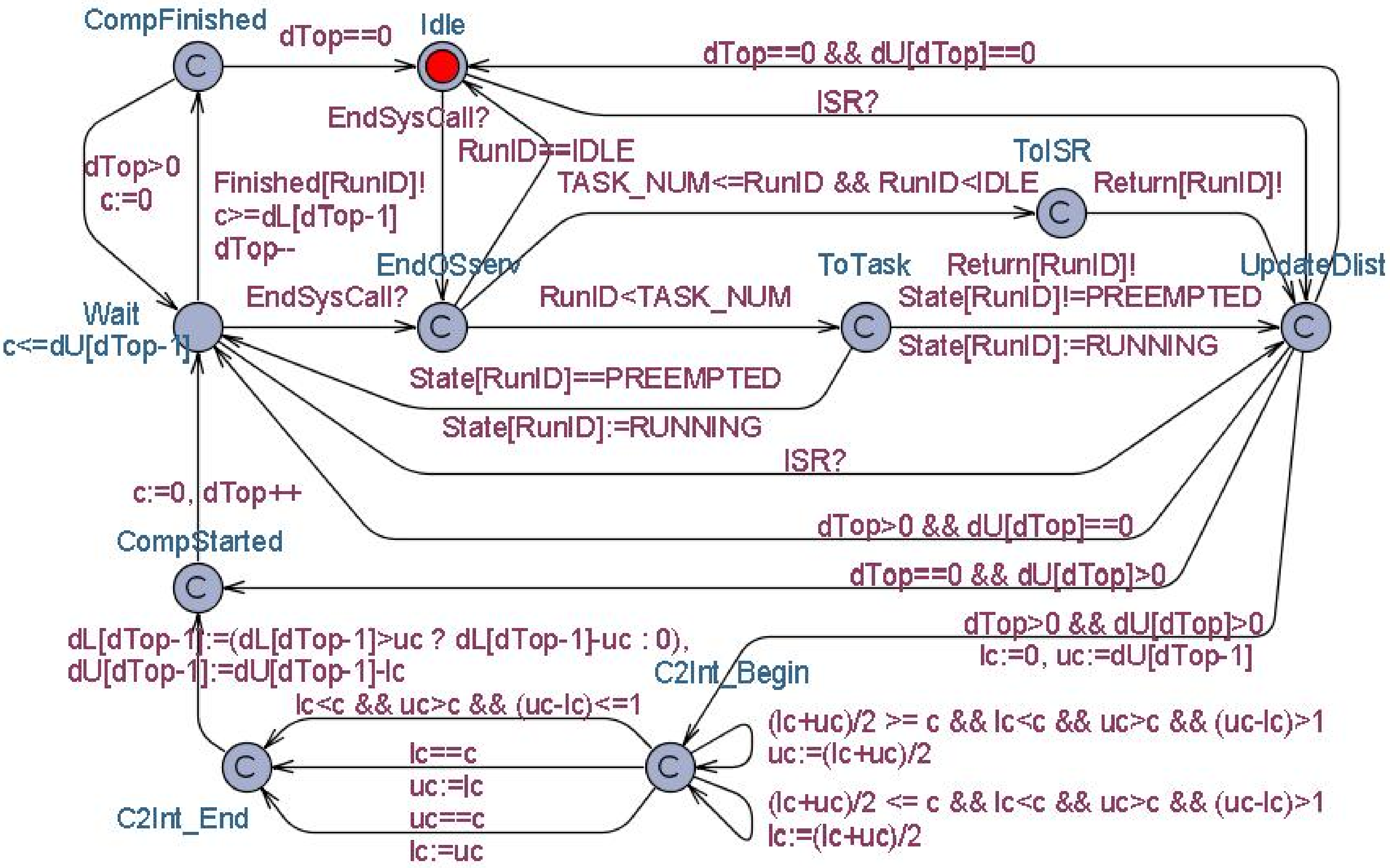
At time r_1 changed to

Δ-list

dL	dL_3	dL_2	$BCET_1$	-
dU	dU_3	dU_2	$WCET_1$	-

⇧ *dTop*

Timed automaton controlling preemption



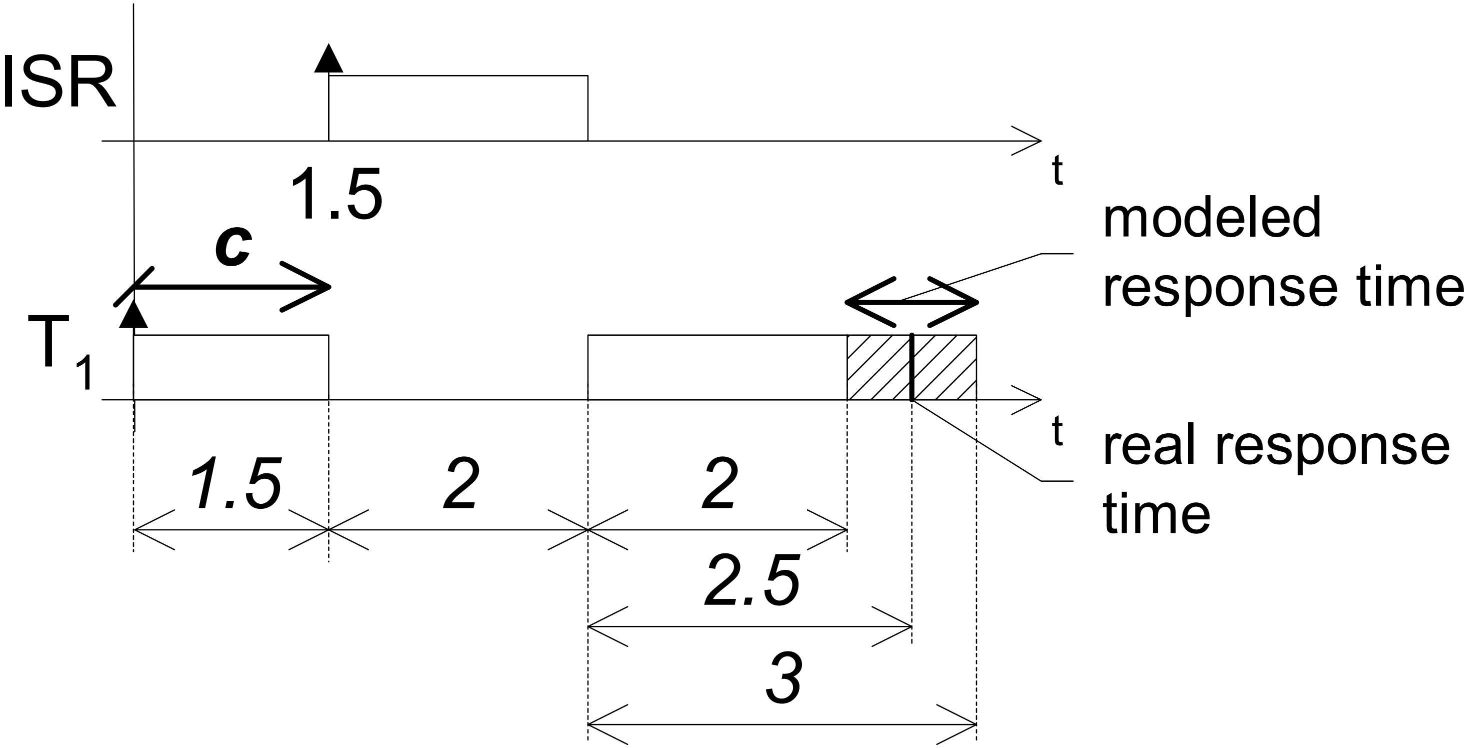
Over-approximation of the model

$IRQ \in (1, 2)$

$C_{ISR} = 2$

$r_{T1} = 0$

$C_1 = 4$

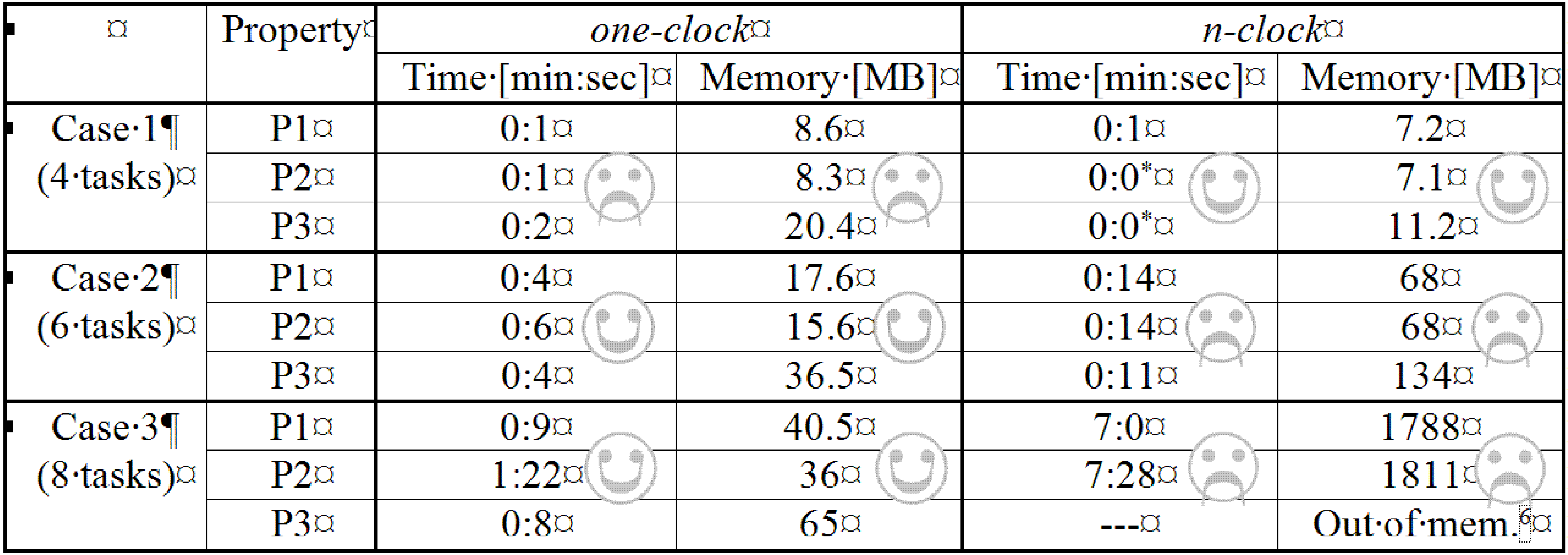


Real behavior $\subseteq$ Modeled behavior

Conclusion

- **model-checking approach** can be
  - used for **verification of distributed RT system** properties (we have designed model of *OSEK OS* services and *CAN*)
  - used for **verification of fault tolerant system** properties (we have developed model of *recovery blocks*)
  - easily changed or extended **by application developer**
- Over-approximation **preserves the most important properties**
  - *Safety properties* (“unsafe state is avoided”)
  - *Bounded liveness properties* (“Desirable state is reached in bounded time”)
- **drawback: high complexity** of model-checking limits size of verified application ()

Complexity



	Property	*one-clock*		*n-clock*	
		Time [min:sec]	Memory [MB]	Time [min:sec]	Memory [MB]
Case 1 (4 tasks)	P1	0:1	8.6	0:1	7.2
	P2	0:1 ☹	8.3 ☹	0:0* ☺	7.1 ☺
	P3	0:2	20.4	0:0*	11.2
Case 2 (6 tasks)	P1	0:4	17.6	0:14	68
	P2	0:6 ☺	15.6 ☺	0:14 ☹	68 ☹
	P3	0:4	36.5	0:11	134
Case 3 (8 tasks)	P1	0:9	40.5	7:0	1788
	P2	1:22 ☺	36 ☺	7:28 ☹	1811 ☹
	P3	0:8	65	---	Out of mem.[6]