Timing Analysis of Embedded Systems using Model Checking

Vallabh R. Anwikar and Purandar Bhaduri

Dept. of Computer Science & Engineering
IIT Guwahati, India
pbhaduri@iitg.ernet.in

18th International Conference on Real-Time and Network Systems
Toulouse, France
November 4-5, 2010

Outline

1. Introduction
2. Background: Timed Automata
3. Model of Preemptable Tasks
4. Explicit-Time Model Checking
5. Conclusion

Introduction

- Embedded control systems are often distributed with a shared bus for communication.
  - automotive
  - aerospace
- Distributed real-time embedded systems
  - Tasks run on processors, communicate through messages.
  - Tasks: Fixed priority preemptive scheduling.
  - Messages: Bus access protocol (*e.g.*, FPNPS, TDMA, *etc.*).
  - Accurate timing analysis a challenging task.

Timing Analysis

- Existing approaches
  1. Extensions of Classical Schedulability Theory
     - Holistic Scheduling
     - SymTA/S
  2. Real-Time Calculus
  3. Model Checking
- The first two approaches are too approximate and therefore pessimistic.
- Timed Automata
  - Suffer from state space explosion.
  - Cannot model preemption accurately.
- Goal: Test the limits of timed automata based analysis using:
  - A novel approach due to Waszniowski *et al.*, 2005 to approximately model preemption in timed automata.
  - A generalized task model for preemptable tasks.

Related Work

- Modeling preemption accurately requires stopwatches.
  - Reachability for stopwatch automata is undecidable. [Krcál *et al.*, 2004]
- Preemption in timed automata with approximation:
  - Method proposed by Madl *et al.*, 2009
    - Approximates stopwatch automata using timed automata.
    - Discretizes clocks by introducing 'checkpoints' to store execution time before preemption.
    - Constructs a generalized task model implementing the approach in the DREAM Tool.
  - Method proposed by Waszniowski *et al.*, 2005
    - Approximates the clock value by nearest lower and upper integers.
    - No generalized task model as in case of Madl *et al.*

Related Work (cont.) – More Recent Approaches

- UPPAAL 4.1 [David *et al.*, 2010] has added stopwatches, with a zone based approximation algorithm for reachability.
- Approach using Calendar Automata and discrete time by Rajeev *et al.*, 2010.

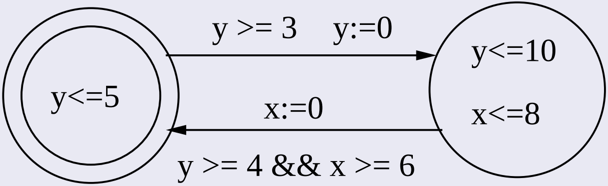
Contribution

- Constructed a generalized task model based on Waszniowski's method.
- Performed case studies applying this method.
- Compared with method proposed in DREAM in terms of time taken.
- Experimented with explicit-time approach for timing analysis.
- Compared explicit-time results with implicit-time results.

Timed Automata (Alur *et al.*, 1994)

- **Timed Automaton**: A timed automaton over set of actions *Act* and set of clocks C is a tuple $\langle L, l_0, E, I, V \rangle$ where
  - L is a finite set of *locations*
  - l_0 is the *initial location*
  - $E \subseteq L \times \Psi(C) \times Act \times 2^C \times L$ is the set of edges. When $\langle l, g, a, r, l' \rangle \in E$, we write $l \xrightarrow{g,a,r} l'$
  - $I : L \longrightarrow \Psi(C)$ is a function which assigns a clock constraint called *invariant* to a location
  - $V : L \rightarrow 2^{AP}$ is a a function which for each location assigns a set of atomic propositions that hold in the location

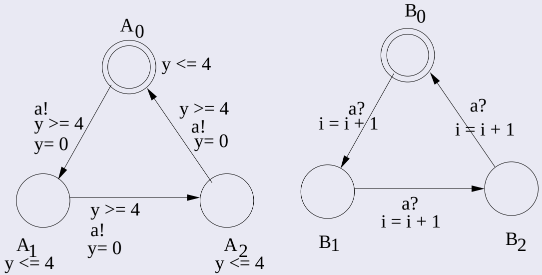
Timed Automaton Example



UPPAAL Tool

- Tool for modeling, validation and verification of real-time systems modeled as networks of timed automata.
- Timed automata are extended with bounded integers, arrays *etc.*
- Real valued clock variables are used for measuring time.
- Supports communication using synchronization and shared variables.

UPPAAL Example



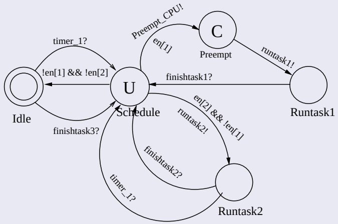
Timed Automata Models used in Verification

- TA model for a distributed real-time system includes:
  - Scheduler model
  - Preemptable task model
  - Message model

Scheduler Model (Madl *et al.*, 2009)

- For fixed priority preemptive scheduling.
- *Task1* has higher priority than Task2.
- *Task1* is released by *timer_1* while *Task2* is released by the completion of *Task3*
- The guard *en[1]* indicates that *Task1* is enabled.
- Whenever a higher priority task is scheduled, the *Preempt* signal is broadcast

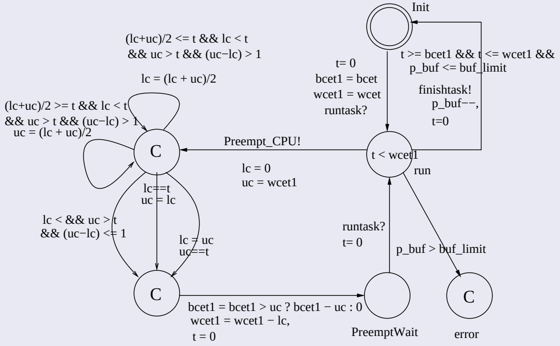
Scheduler Model in UPPAAL



Preemptable Task Model

- Approximates the elapsed execution time by using a bisection algorithm to obtain:
  - nearest lower integer bound *lc*, and
  - nearest upper integer bound *uc*.

Preemptable Task Model in UPPAAL



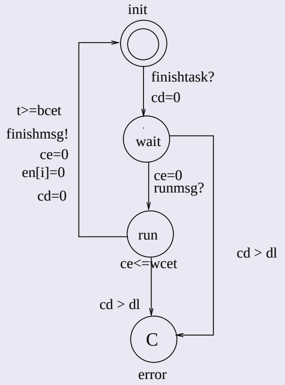
Over-approximation in Handling Preemption (Waszniowski *et al.*, 2005)

- Clock value c is approximated to closest upper and lower integers uc and lc
- $BCET_{new} := BCET - uc$
- $WCET_{new} := WCET - lc$
- $BCET_{new} \leq BCET_{Real}$
- $WCET_{new} \geq WCET_{Real}$
- Real behavior $\subseteq$ Modeled behavior

Message Model

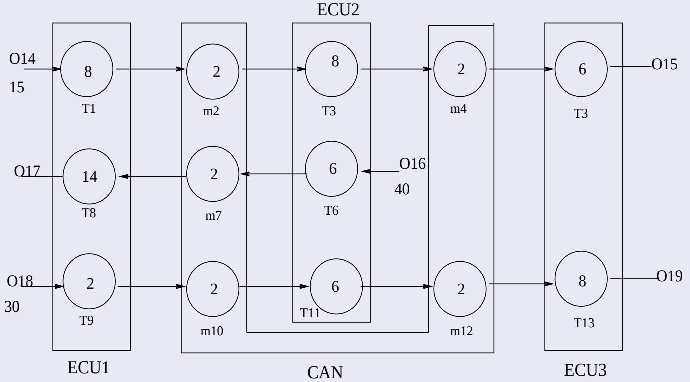
- Model of messages in the system.
- Execution time represents transmission time of message.
- Non-preemptive, *i.e.*, higher priority message waits for lower priority message on the bus.
- Clocks *cd* and *ce* model deadline and transmission time of the message.

Message Model in UPPAAL



Case Study 2 Using UPPAAL

Application containing CAN bus (di Natale *et al.*, 2007)



Application containing CAN bus

- Time taken by a message to reach an actuator from a sensor is called the *end-to-end latency*.
- Important design parameter and has to be within a certain *limit*.
- Multiple active chains in the system.
- Preemptive scheduling for tasks mapped on the ECUs, and Non-preemptive for messages
- Array of clocks used for modeling each active chain.
- Problem faced with the DREAM tool.

Results for Case Study 2: CAN Bus Application

- Traditional methods considers blocking of lower priority tasks by higher priority tasks (*critical instant*):
  - in reality such scenario may never occur in the system.
- Model checking is more accurate
  - Explores each and every execution path of the system.

Chain	UPPAAL	Real -Time Calculus
$O_{14} - O_{15}$	28	32
$O_{16} - O_{17}$	50	60
$O_{18} - O_{19}$	110	210

Table: Worst case latencies of three task chains

Implicit-Time and Explicit-Time Model Checking

Implicit-Time Approach

- Formalisms are extended with time *e.g.*, Timed automata, Timed Petri Nets
- LTL, CTL need extension for handling timed automata specific properties
- Specialized data structures representing clock variables *e.g.*, Differences Bounded Matrices.

Implicit-Time and Explicit-Time Model Checking (cont.)

Explicit-Time Approach

- A global integer variable is used for modeling time.
- Variable is incremented/decremented showing passage of time.
- We lose continuous semantics of time.
- According to Henzinger *et al.*, 1992, integer time verification is sound for
  - Time-bounded invariance
  - Time-bounded response
- Timing bounds are expressed via the use of
  - *Countdown Timer*
  - *Countup Timer*
  - *Expiration Timer*

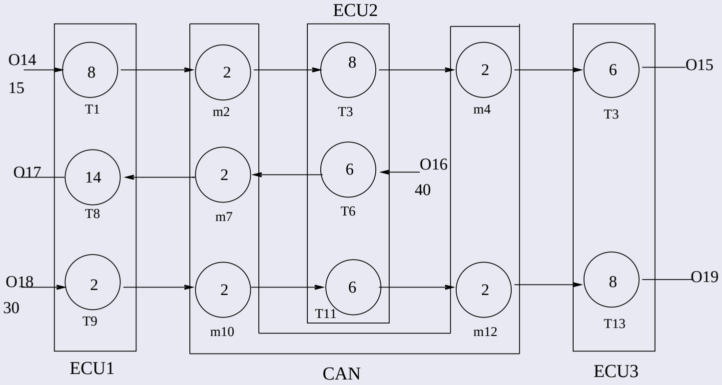
Advantages of Explicit-Time Approach

Advantages

- We can use model checkers like SPIN, SMV *etc.*, with easier learning curves.
- Easier to model preemption as we can store the current time value.

Case Study 2 Using SPIN

Application containing CAN bus



Handling Preemption in SPIN

PROMELA fragment

```
active proctype()
{
start: do
::atomic
{
  ((Proc_i ? [eval(id)])) --> exe_i = rem_i
   if  :: expire(exe_i);
          Proc_i ?  eval(id);
          rem_i = n;
          (runid == -1) --> Proc_j !! id;
       :: !((Proc_i ? [eval[id]))
            --> rem_i = exe_i; goto start;
  fi;
}
  od;
}
```

Results of Case Study 2 Using SPIN

Chain	UPPAAL	SPIN
$O_{14} - O_{15}$	28	28
$O_{16} - O_{17}$	50	55
$O_{18} - O_{19}$	110	120

Table: Worst case latencies of three task chains

- Results obtained with SPIN are comparable with that of UPPAAL.
- Modeling with SPIN is much easier than in UPPAAL, but ...
- Requires more memory and time.

.

Final Remarks

- Constructed a general task model for handling a preemptive task based on Waszniowski's method.
- Significant improvement as compared to real-time calculus and holistic scheduling.
- Our task model performs faster than method used in DREAM tool.
- Tried *explicit-time approach* for analyzing real-time systems.
- Observed that they do not perform much worse than implicit-time approach, but require significantly more memory.

Future Work

- Compare with UPPAAL 4.1 (stopwatches) based analysis and the Calendar Automata based method of Rajeev *et al.*, 2010.
- Try out bigger case studies for comparing the various approaches.
- Try to handle the state space explosion problem by symbolic approaches, model reduction, abstraction, *etc.*

Thank You...