From Timed Models to Timed Implementations

Martin De Wulf, Laurent Doyen, J.-F. Raskin
Université Libre de Bruxelles
Motivations: Embedded Systems

- Embedded Systems
  - ... are difficult to develop (concurrency, real-time, continuous environment, ...).
  - ... are often safety critical (or at least, failure is expensive).

Need for good design practices
Model-based development and Verification

- Make a model of the environment:
  Environment
- Make clear the control objective:
  Bad
- Make a model of the control strategy:
  Controller
- Verify:
  Does Controller || Environment avoid Bad?

(There are programs to do that)
Model-based development and Verification

- Formalisms Used:
  Timed and Hybrid Automata and Reachability Analysis

Example: Water level in a tank \((x \text{ clock}, y \text{ level})\)

Environment

\[
\begin{align*}
A & : y \leq 10 \\ & \dot{y} \in [0, 1]
\end{align*}
\]

\[
\begin{align*}
B & : y = 10 \\ & \dot{y} \in [0, 1]
\end{align*}
\]

\[
\begin{align*}
C & : x \leq 2 \\ & \dot{y} \in [0, 1]
\end{align*}
\]

\[
\begin{align*}
F & : x \leq 2 \\ & \dot{y} \in [-2, -1]
\end{align*}
\]

\[
\begin{align*}
E & : \dot{y} \in [-2, -1]
\end{align*}
\]

\[
\begin{align*}
D & : y \geq 5 \\ & \dot{y} \in [-2, -1]
\end{align*}
\]

Bad: \(y < 1 \lor y > 12\)
\[ F \dot{y} \in [-2, -1] \]
\[ x \leq 2 \]

\[ E \dot{y} \in [-2, -1] \]
\[ y = 5 \]

\[ D \dot{y} \in [-2, -1] \]
\[ y \geq 5 \]

\[ A y \leq 10 \]
\[ \dot{y} \in [0, 1] \]

\[ B \dot{y} \in [0, 1] \]
\[ x := 0 \]

\[ C x \leq 2 \]
\[ \dot{y} \in [0, 1] \]

\[ H \]
\[ \text{Start} \]
\[ \text{Stop} \]

\[ z \leq 0 \]

\[ y < 1 \lor y > 12 \]
\[ \text{UNREACHABLE} \]
Good, but after verification?

- Classical approach:
  - code by hand
  - try to mimic as much as possible the model
  - but usually, no formal link
- Can we derive an implementation from a verified model?
  - Automatically
  - With properties preservation
The Big Picture

Timed Controller

Model for Verification Tools (Hytech or Uppaal)

Desired Properties

Code (BrickOS C)

OK?

Automatic Generation
Verification
Property Preservation
Goal

- **Transfer** of verified properties from models to code.
- **Type of models we consider:**
  - Controllers specified as timed automata with inputs and outputs
Timed automata are (in general) not implementable (in a formal sense)...

Why?

- Zenoness: $1/2$, $3/4$, $7/8$, ...

- No minimal bound between two transitions: $1/2, 1, 1+3/4, 2, 2+7/8, 3, ...$

- And more ...
• One can specify instantaneous response but not implement it.
More Problems

- Instantaneous synchronization between environment and controller is not implementable.

Environment

Classical controller
Not implementable
More Problems

• Models use **continuous clocks** and implementation uses **digital clocks with finite precision**

Classical controller
Not implementable

\[ x := 0 \quad \xrightarrow{\quad \text{V.S} \quad} \quad x \geq 3 \]

\[ x \leq 3 \]
Problems: Summary

- My controller strategy may be correct because:
  - it is zeno;
  - it acts faster and faster;
  - it reacts **instantaneously** to events and timeouts (synchrony hypothesis);
  - it uses **infinitely** precise clocks.
A possible solution...

- Give an alternative semantics to timed automata: Almost ASAP semantics.
  - enabled transitions of the controller become urgent only after $\Delta$ time units;
  - events from the environment are received by the controller within $\Delta$ time units;
  - guards are enlarged by $\Delta$.

where $\Delta$ is a parameter (modeling the performance of the hardware)
Intuition

- One can **specify** instantaneous response but **not** implement it.

Not implementable

```
\begin{align*}
x &:= 0 \\
x &\leq 0
\end{align*}
```

Solution: allow some delay

```
\begin{align*}
x &:= 0 \\
x &\leq \Delta
\end{align*}
```
More Intuition

- Instantaneous synchronization between environment and controller is not implementable.

Environment

Classical controller
Not implementable

Solution:
Uncouple event from perception by the controller

\[ a \leq \Delta \]
More Intuition

- Models use **continuous clocks** and implementation uses **digital clocks with finite precision**

Classical controller
Not implementable

Solution:
Slightly relax the constraints

\[
\begin{align*}
x := 0 & \quad x \geq 3 \\
x \leq 3 & \quad x \geq 3 - \Delta \\
x \leq 3 + \Delta & \quad x \leq 3 + \Delta
\end{align*}
\]
• AASAP semantics define a “tube” of strategies instead of a unique strategy in the ASAP semantics.
• This tube can be refined into an implementation while preserving safety properties.
Proof of implementability?

- We define an “implementation semantics” based on:

  - Time length of a loop: $\Delta_L$
  - Time between two clock ticks: $\Delta_P$
  - (+ possibly drift of the system clock: $\epsilon$)

Execution round:

1. Read System Clock
2. Update Sensor Values
3. Check all transitions and fire one if possible

Designed to convince of implementability
Proof of implementability?

Theorem:
For any timed controller, its AASAP semantics simulates (in the formal sense) its implementation semantics, provided that:

\[ \Delta > 3\Delta_L + 4\Delta_P \]

In this case, the implementation is guaranteed to preserve verified properties of the model, that is:

\[
\text{Controller}(\Delta) || \text{Environment} \text{ avoid Bad} \implies \\
\text{ControllerImp}(\Delta_L, \Delta_P) || \text{Environment avoid Bad}
\]
In practice?

- We wrote a program (Elastic) for verification using HyTech and Uppaal.
- Code generation for Brickos
**A case study**

The Philips Audio Control Protocol

**Sender**

**Receiver**

**1001100**

Timed Manchester encoding

Manchester encoding of 110100
Properties /requirements for the protocol:

- The receiver knows the length of a time slot but ignores when it begins;
- The receiver ignores length of the current bit string;
- Only UP signals can be perceived reliably;
- S/R uses (unsync.) digital clocks: there will be imprecision in sending and perceived receiving times;
- Sensors are polled every time slice: discrepancy between occurrence of UP events and detection.

... 3 first items should be dealt with by the logic of the protocol, 2 last items are related to robustness of the protocol: the AASAP-semantics deals with it.
Conclusion

Our semantics

- is the **first implementable semantics for timed automata**
- is **verifiable**, even for non-trivial case studies!
- guarantees **correct code** and not only correct idealized model!
- is **supported by a tool**!

In other words: we defined a complete methodology to go from a class of (real-)timed models to (real-)timed implementations, while ensuring the preservation of properties verified through model checking.