Observer Patterns for Real-Time Systems

Étienne André

Laboratoire d’Informatique de Paris Nord
Université Paris 13, Sorbonne Paris Cité, France
Context: Verifying Complex Timed Systems (1/2)

- Need for early bug detection
  - Bugs discovered when final testing: expensive
  - Need for a thorough modeling and verification phase
Context: Verifying Complex Timed Systems (2/2)

- Use formal methods

A **model** of the system

A **property** to be satisfied

| is unreachable |
Use formal methods

A model of the system

A property to be satisfied

Question: does the model of the system satisfy the property?
Context: Verifying Complex Timed Systems (2/2)

- Use formal methods

A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Yes

No

Counterexample
Context: Verifying Complex Timed Systems (2/2)

- Use formal methods

A model of the system

A property to be satisfied

Question: does the model of the system satisfy the property?

Yes

No

Counterexample
Specifying Properties for Real-Time Systems

  - Very expressive
  - Not easy to handle by non-experts
    - \( EF((EX.P)U(AG.Q)) \)
  - Expressiveness not often supported in full by tools
  - Requires complex model checking algorithms
  - Too expressive?

- **Observer-based properties** [Aceto et al., 1998b, Aceto et al., 1998a]
  - Quite expressive
  - Do not require complex algorithms
  - Not easy to handle by non-experts
  - Too expressive?
Outline

1. Why Observer Patterns?
2. Common Patterns
3. Conclusion and Perspectives
Goal

- Identify and express **common properties** met in the design and verification of complex real-time systems

- Define an associated formal **semantics**

- Shall be non-compositional
  - 😊 Avoids mistakes or ill-formed properties
  - 😞 Discards properties never or rarely met in practice
  - 😞 Non-complete
Patterns: Related Works

- **Design patterns** for software engineering [Gamma et al., 1995]
  - Non-original, non-compositional, non-complete
  - Can be inserted into freely written code

- **Timed automata patterns** [Dong et al., 2008]
  - Specification for real-time concurrent systems
  - Specify the model and not the property
  - Aim at completeness

- **Other patterns for real-time systems**
  - Dedication to scheduling problems [Khatib et al., 2001]
  - Specification using UML [Mekki et al., 2009]
  - Timed automata patterns generated from RTESMs [Han et al., 2013]
Why Observer Patterns?

Observers: All is Reachability

- **Observer**: additional subsystem monitoring the system behavior
  - Synchronization with the system events
  - Definition of bad / good states

- Reduction to **pure reachability** properties
  Here, we consider 2 main kinds of properties:
  1. Never reach the bad state
  2. Always end in a good state

- Advantages
  - Any tool implementing (non-)reachability can implement such patterns
Why Observer Patterns?

Contribution

Our Approach

- A **limited** set of non-compositional patterns
  - Commonly used in practice
  - Discards too complicated properties

- Expressed using an intuitive **English-like syntax**
  - Partially inspired by CASL-LTL
    - [Reggio et al., 2003, Choppy and Reggio, 2006]
  - Described using a full English sentence
  - Easy to understand by non-experts

- Associated with a **formal semantics**
  - Expressed using **observers**
  - Does not require complex algorithms (only reachability)
    - Implemented in both timed automata [Alur and Dill, 1994] and Stateful Timed CSP [Sun et al., 2013]
Outline

1. Why Observer Patterns?
2. Common Patterns
3. Conclusion and Perspectives
Action Precedence: Acyclic Version (1/2)

Syntax:

\[
if \ a_2 \ \text{then} \ a_1 \ \text{has happened before}
\]

English description:

“If \ a_2 \ \text{happens at least once}, then \ a_1 \ \text{has happened before the first occurrence of} \ a_2.”

- Typical use: avoid false positives for alarms
  - “An alarm must ring only if an intrusion has happened before”
- Does not mean that the intrusion will always lead to an alarm!
**Action Precedence: Acyclic Version (2/2)**

If $a_2$ then $a_1$ has happened before

Timed automaton observer:

```
\begin{itemize}
  \item $l_0 \rightarrow l_1$ for $a_1$
  \item $l_0 \rightarrow l_b$ for $a_2$
  \item $l_b \rightarrow l_1$ for $a_1, a_2$
\end{itemize}
```

**Stateful Timed CSP:**

\[ P_{obs} \overset{*}{=} (a_2[n_{bad} := T] \rightarrow P_S) \land (a_1 \rightarrow P_S) \]
Action Precedence: Cyclic Version

Syntax:

*every time* $a_2$ *then* $a_1$ *has happened before*

English description:

“Every time $a_2$ happens, then $a_1$ has happened before, since the last occurrence of $a_2$ (if any).”

Timed automaton:

![Timed automaton diagram]

Stateful Timed CSP:

$$P_{obs} \equiv P_1$$

$$P_1 \equiv (a_2\{v_{bad} := T\} \rightarrow P_S) \Box (a_1 \rightarrow P_2)$$

$$P_2 \equiv (a_1 \rightarrow P_2) \Box (a_2 \rightarrow P_1)$$
Action Precedence: Strict Cyclic Version

Syntax:

\textit{every time }a_2\textit{ then }a_1\textit{ has happened exactly once before}

English description:

“Every time }a_2\textit{ happens, then }a_1\textit{ has happened before, exactly once since the last occurrence of }a_2\textit{ (if any).”}

Timed automaton:

\begin{center}
\begin{tikzpicture}[auto, node distance=2cm, thick,->, main/.style={draw, circle, minimum size=1cm}]
  \node[main, fill=red!20] (l0) {$l_0$};
  \node[main, fill=red!20] (l1) [right of=l0] {$l_1$};
  \node[main, fill=red!20] (lb) [below of=l0] {$l_b$};
  \path (l0) edge [loop right] node {$a_1\, , \, a_2$} (l0)
                edge node {$a_2$} (l1)
  (l1) edge node {$a_1$} (lb)
  (lb) edge [loop right] node {$a_2$} (lb)
  (l1) edge [loop left] node {$a_1$} (l1);
\end{tikzpicture}
\end{center}

Stateful Timed CSP:

\begin{align*}
P_{obs} & \models P_1 \\
P_1 & \models (a_2\{v_{bad} := T\} \rightarrow P_S) \square (a_1 \rightarrow P_2) \\
P_2 & \models (a_1\{v_{bad} := T\} \rightarrow P_S) \square (a_2 \rightarrow P_1)
\end{align*}
Eventual Response: Acyclic Version (1/2)

Syntax:

\[
\text{if } a_1 \text{ then eventually } a_2
\]

English description:

“If \(a_1\) happens, then \(a_2\) eventually happens.”

- Typical use:
  - “Every time access is requested, then access is eventually granted”
  - “Every time an intrusion occurs, then the alarm eventually rings”

- Does not require \(a_1\) to happen!

- Typical liveness property\(^1\)

---

\(^1\) Cf. https://cs.nyu.edu/acsys/beyond-safety/liveness.htm
Eventual Response: Acyclic Version (2/2)

if $a_1$ then eventually $a_2$

Timed automaton:

Stateful Timed CSP:

$P_{obs} = (a_2 \rightarrow P_{obs}) \quad \square \quad (a_1\{v_{good} := F\} \rightarrow P_1)$

$P_1 = (a_1 \rightarrow P_1) \quad \square \quad (a_2\{v_{good} := T\} \rightarrow P_S)$
**Eventual Response: Acyclic Version (2/2)**

\[ \text{if } a_1 \text{ then eventually } a_2 \]

**Timed automaton:**

\[ \begin{array}{c}
    a_2 & \xrightarrow{a_1} & a_1, a_2 \\
    l_g & \xrightarrow{a_1} & l_1 & \xrightarrow{a_2} & l'_g
\end{array} \]

**Stateful Timed CSP:**

\[
\begin{align*}
    P_{obs} & \triangleq (a_2 \rightarrow P_{obs}) \ \square (a_1\{v_{\text{good}} := F\} \rightarrow P_1) \\
    P_1 & \triangleq (a_1 \rightarrow P_1) \ \square (a_2\{v_{\text{good}} := T\} \rightarrow P_S)
\end{align*}
\]

Also exists in “cyclic” and “strict cyclic” versions (cf. paper)
Action Before Deadline (1/2)

Syntax:
\[ a \text{ no later than } d \]

English description:
“\( a \) will happen no later than \( d \) units of time after the system start.”

- Typical use: signal changes in hardware verification
- Similar to the “deadline” pattern in [Dong et al., 2008]
Action Before Deadline (2/2)

\[ a \text{ no later than } d \]

\[ x_{obs} \leq d \]

\[ x_{obs} := 0 \]

\[ x_{obs} = d \]

\[ P_{obs} \models (a \rightarrow P_S) \text{ timeout}[d] \]

\[ ((e_{obs}\{v_{bad} := T\} \rightarrow P_S) \setminus \{e_{obs}\}) \]

Stateful Timed CSP:
Time-Bounded Action Precedence

Syntax:

\[ \text{if } a_2 \text{ then } a_1 \text{ has happened at most } d \text{ units of time before } \]

English description:

“If \( a_2 \) happens at least once, then \( a_1 \) has happened at most \( d \) units of time before the first occurrence of \( a_2 \).”

- Timed extension of the “Action precedence” pattern
- Also exists in “cyclic” and “strict cyclic” versions
Time-Bounded Response

Syntax:
if $a_1$ then eventually $a_2$ within $d$

English description:
“If $a_1$ happens, then $a_2$ will eventually happen within $d$ units of time.”

- Typical use:
  “If an intrusion occurs, then the alarm eventually rings within 2 seconds”

- Also known as time-bounded liveness [Behrmann et al., 2005]

- Timed extension of the “Eventual response” pattern

- Also exists in “cyclic” and “strict cyclic” versions
Sequence

Syntax:

\textit{sequence} \( a_1, \ldots, a_n \)

English description:

“Actions \( a_1, \ldots, a_n \) must occur in this order.”

- Imposes an order, but does require the actions to happen
- Also exists in “cyclic” version
## Use of the Patterns in a Set of Case Studies

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Nb</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-reachability</td>
<td>9</td>
<td>Prtc</td>
</tr>
<tr>
<td>Always end in good state</td>
<td>2</td>
<td>HW Prtc</td>
</tr>
<tr>
<td>Action precedence</td>
<td>2</td>
<td>HW</td>
</tr>
<tr>
<td>Eventual response</td>
<td>5</td>
<td>Prtc</td>
</tr>
<tr>
<td>Action before deadline</td>
<td>1</td>
<td>HW</td>
</tr>
<tr>
<td>T-bounded action precedence</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T-bounded response</td>
<td>3</td>
<td>HW Prtc Sched</td>
</tr>
<tr>
<td>Sequence</td>
<td>1</td>
<td>HW</td>
</tr>
<tr>
<td>Combinations</td>
<td>3</td>
<td>Prtc Sched</td>
</tr>
<tr>
<td>Others</td>
<td>2</td>
<td>Prtc</td>
</tr>
</tbody>
</table>

Based on 29 case studies (real-time systems)
Outline

1. Why Observer Patterns?
2. Common Patterns
3. Conclusion and Perspectives
Conclusion

- **Observer patterns**
  - Common properties
  - Intuitive syntax
  - Reachability-based formal semantics
  - Limited expressiveness

- Extends to **parameter synthesis**
  - Parameterized version of the patterns
    - E.g.: “if $a_1$ then eventually $a_2$ within $p$”, with $p$ a parameter
  - Implemented in IMITATOR [A., Fribourg, Kühne, Soulat, 2012]

- Extends to **state- and variables-based properties**?
  - So far: action-based
  - No theoretical problem for state and variables
    But what about synchronization?
Perspectives

- What about liveness?
  - Requires Büchi conditions in the observers

- Add a limited dose of compositionality
  - Combination of patterns
    - Example: “[sequence a, b, c] within 10 seconds”
      (Sequence + new “within” pattern)
  - Propose a bounded depth for well-formedness?

- Semantic equivalence between the implementation of the patterns in timed automata and Stateful Timed CSP respectively

- Reuse some of the most common existing “patterns”
  [Khatib et al., 2001, Mekki et al., 2009]

- Improve the English description
  - Take compositionality into account
Bibliography
References I

The power of reachability testing for timed automata.

Model checking via reachability testing for timed automata.

A theory of timed automata.

IMITATOR 2.5: A tool for analyzing robustness in scheduling problems.

*Principles of model checking.*
MIT Press.
References II

Beyond liveness: Efficient parameter synthesis for time bounded liveness.
In FORMATS’05, volume 3829 of Lecture Notes in Computer Science, pages 81–94.
Springer.

A formally grounded software specification method.

Timed automata patterns.

Design patterns: Elements of reusable object-oriented software.
Addison-Wesley Longman Publishing Co., Inc.

Han, F., Herrmann, P., and Le, H. (2013).
Modeling and verifying real-time properties of reactive systems.


Additional explanation
Explanations for the 4 pictures in the beginning

Allusion to the Northeast blackout (USA, 2003)
Computer bug
Consequences: 11 fatalities, huge cost
(Picture actually from the Sandy Hurricane, 2012)

Allusion to any plane crash
(Picture actually from the happy-ending US Airways Flight 1549, 2009)

Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
No fatalities
Computer bug: inaccurate finite element analysis modeling
(Picture actually from the Deepwater Horizon Offshore Drilling Platform)

Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
28 fatalities, hundreds of injured
Computer bug: software error (clock drift)
(Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)
Licensing
Source of the graphics used (1/2)

Title: Hurricane Sandy Blackout New York Skyline
Author: David Shankbone
Source: https://commons.wikimedia.org/wiki/File:Hurricane_Sandy_Blackout_New_York_Skyline.JPG
License: CC BY 3.0

Title: Miracle on the Hudson
Author: Janis Krum (cropped by Étienne André)
Source: https://secure.flickr.com/photos/davidwatts1978/3199405401/
License: CC BY 2.0

Title: Deepwater Horizon Offshore Drilling Platform on Fire
Author: ideum
Source: https://secure.flickr.com/photos/ideum/4711481781/
License: CC BY-SA 2.0

Title: DA-SC-88-01663
Author: imcomkorea
Source: https://secure.flickr.com/photos/imcomkorea/3017886760/
License: CC BY-NC-ND 2.0
Source of the graphics used (2/2)

Title: Smiley green alien big eyes (aaah)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

Title: Smiley green alien big eyes (cry)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain
License of this document

This presentation can be published, reused and modified under the terms of the license Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)

(LATEX source available on demand)

Author: Étienne André

https://creativecommons.org/licenses/by-sa/3.0/