Formal Verification of User-Level Real-Time Property Patterns

Ning Ge\(^1\)  Marc Pantel\(^2\)  Silvano Dal Zilio\(^3\)

\(^1\)Beihang University, China
\(^2\)University of Toulouse - IRIT/INPT, France
\(^3\)LAAS-CNRS, France

TASE’17
13 - 15 September 2017, Sophia Antipolis, France
Real-Time Requirements

Time Petri Nets & TINA

Real-Time Property Patterns

Elementary Observers for the Verification

Conclusion
Outline

1. Real-Time Requirements
2. Time Petri Nets & TINA
3. Real-Time Property Patterns
4. Elementary Observers for the Verification
5. Conclusion
Real-Time Requirements

Commonly required during the development of concurrent systems
- worst case execution time
- worst case traversal time
- state time duration
- schedulability
- etc.

Core issues for engineers using formal methods
- Ergonomy (user oriented expressiveness) of temporal logics
- Scalability of model checking
Property Pattern Approach
User level expressiveness and Scalability

- Ease the use of formal methods by providing reusable solutions
- Decompose complex problems to simpler ones, with a lower complexity
- Decrease the verification cost
- Qualitative patterns proposed by Dwyer cover 90% temporal requirements.
- Quantitative patterns are extended by Konrad.
Hierarchy of Time Property Patterns

- **Type**
  - Classification by Dwyer
    - **Qualitative**
      - Occurrence
      - Order
    - **Quantitative**
      - Duration
      - Periodic
      - Precedence

- **Catalog**
  - **Absence**
  - **Existence**
  - **Precedence**
  - **Chain Precedence**
  - **Minimum Duration**
  - **Bounded Recurrence**
  - **Maximum Duration**
  - **Bounded Invariance**
  - **Bounded Response**

- **Pattern**
  - Universality
  - Bounded Existence
  - Response
  - Chain Response
Property-Driven Approach
User level expressiveness and Scalability

Principle
The formal activities in the development process are based on the purpose of property-verification-ease.

Experiments by B. Combemale
Verification of structural and temporal properties in xSPEM models.
Needs for more scalable methods to verify quantitative properties.

Steps of our work
1. Characterizing properties.
2. Characterizing observable states and events.
3. Expressing real-time properties: property patterns.
4. Defining denotational semantics to Time Petri Net (TPN) + observers and reachability assertions.
5. Reducing state space: property-specific reduction for TPN.
Challenge & Property-Driven Verification Framework

1. Architecture/Behavior Mapping

2. Real-Time Property Patterns

3. Observer TPN Generation

4. TPN Reduction

5. Feedback Generation

- UML Real-Time Software Model
  - System Model
    - Architecture Model
    - Behavior Model
  - Real-Time Requirement

- Real-Time Property Specification

- Verification Result Computation
  - Tag Property Pattern Result Interpretation
  - TPN Model Checking

- Reduced Observer TPN
  - Reachability Assertions

- Tag Property Pattern Result

- Property Pattern Result

- Iteration Tag
Challenge & Property-Driven Verification Framework

Real-Time Requirements

UML Real-Time Software Model
- System Model
  - Architecture Model
  - Behavior Model
- Real-Time Requirement

Architecture/ Behavior Mapping
- TPN

1. Architecture/ Behavior Mapping
2. Real-Time Property Patterns
3. Observer TPN Generation
4. Observer TPN
5. Real-Time Property Specification

Feedback Generation
- Real-Time Property Verification Result

Verification Result Computation

Iteration Tag
- Tag Property Pattern Result Interpretation
- Reduced Observer TPN
- Reachability Assertions
- TPN Model Checking

N. Ge, M. Pantel and S. Dal Zilio
Challenge & Property-Driven Verification Framework

1. Architecture/Behavior Mapping

2. Real-Time Property Specification

3. Verification Result Computation

4. TPN Reduction

5. Feedback Generation

---

N. Ge, M. Pantel and S. Dal Zilio
User-Level Property Patterns Verification
TASE'2017, 13-15 Sept. 2017
Challenge & Property-Driven Verification Framework

1. Architecture/Behavior Mapping
   - UML Real-Time Software Model
     - System Model
       - Architecture Model
       - Behavior Model
     - Real-Time Requirement
   - Real-Time Property Patterns
   - TPN Generation

2. Real-Time Property Specification
   - Observer TPN
   - Observer TPN Generation
   - TPN Reduction
     - Reduced Observer TPN

3. Verification Result Computation
   - Verification Result
   - Feedback Generation
   - Real-Time Property Verification Result

4. Tag Property Pattern Result Interpretation
   - Property Pattern Result
   - Tag Property Pattern Result
   - TPN Model Checking

5. Iteration Tag
   - TAG
Challenge & Property-Driven Verification Framework

1. Architecture/Behavior Mapping
2. Real-Time Property Patterns
3. Observer TPN Generation
4. TPN Reduction
5. Feedback Generation

- UML Real-Time Software Model
  - System Model
    - Architecture Model
    - Behavior Model
  - Real-Time Requirement
- Observer TPN
- Real-Time Property Specification
- Verification Result Computation
- Reduced Observer TPN
- Reachability Assertions
- TPN Model Checking
Challenge & Property-Driven Verification Framework
Challenge & Property-Driven Verification Framework
Real-Time Requirements

Challenge & Property-Driven Verification Framework

UML Real-Time Software Model
- System Model
  - Architecture Model
  - Behavior Model
- Real-Time Requirement

1. Architecture/Behavior Mapping
   - TPN
2. Real-Time Property Patterns
   - Real-Time Property Specification
3. Verification Result Computation
   - TPN Reduction
   - Reduced Observer TPN
   - Reachability Assertions
4. TPN Model Checking
   - Tag Property Pattern Result
5. Feedback Generation

Property Pattern Result
- Tag Property Pattern Result Interpretation
- TPN Model Checking
- Reduced Observer TPN
- Reachability Assertions

Iteration Tag
- Tag Property Pattern Result
- Observer TPN Generation
- Observer TPN
- TPN Reduction
- Reduced Observer TPN
- Reachability Assertions
Outline

1. Real-Time Requirements
2. Time Petri Nets & TINA
3. Real-Time Property Patterns
4. Elementary Observers for the Verification
5. Conclusion
**Time Petri Net**

![Time Petri Net Diagram]

**TINA toolset**

- Analyze $\mu$-calculus, LTL, CTL properties in TPN.
- Integrate state space abstraction techniques (preserving different types of properties), on-the-fly model checking.
- Data manipulation (**tts**): variable definition and modification.
Outline

1 Real-Time Requirements
2 Time Petri Nets & TINA
3 Real-Time Property Patterns
4 Elementary Observers for the Verification
5 Conclusion
Real-Time Property Specification System

Exist A After B Within [bct, wct]

<table>
<thead>
<tr>
<th>Operator</th>
<th>Occurrence</th>
<th>Basic predicate</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absent</td>
<td>$B$</td>
<td>$A \land B$</td>
<td>global</td>
</tr>
<tr>
<td>or</td>
<td>Exist</td>
<td>$B$</td>
<td>between ($B + bct$) and ($B + wct$)</td>
</tr>
</tbody>
</table>
Outline

1. Real-Time Requirements
2. Time Petri Nets & TINA
3. Real-Time Property Patterns
4. Elementary Observers for the Verification
5. Conclusion
Design Principle of TPN/TTS Observers - Structure

Observer Structure

- A TPN/TTS observer is associated to the system through its arcs, joined at the transitions labelled $T_A$ and $T_B$ in components A & B.
- A TTS observer for state-based properties is not composed with the system but simply put in parallel (an operation usually referred to as free product).
Soundness here means that

- An observer should not impact the system’s behavior by introducing extra semantics or removing original semantics
- Observers preserve time divergence, meaning that an observer should not be able to stop the evolution of time (introducing some kind of time deadlocks).

Our approach

- By interacting with the system only by its transitions
- The observers work in a ”read-only” mode, guaranteed by the design ”linked from TPN transitions”.
Design Principle of TPN/TTS Observers - Efficiency

Principles

- A system with integrated observers should be able to generate state class graphs with a high-level abstraction (i.e. marking abstraction for TINA)
- The generating state space of a single observer shall be as small as possible.
- The checking of each property pattern shall be independent to promote parallel computation.
Elementary Observers - Basic Event Modifiers

Event modifier: an atomic element, or a composite one. Example: \( t \) unit of time after event \( E^{i-k} \).
- \( E^i \) (the \( i^{th} \) occurrence of event \( E \));
- \( E^{i-k} \) (the event delayed \( k \) times from the current event \( E^i \));
- \( E^{i-k} + t \) (the event delayed \( t \) unit of time from current event \( E^{i-k} \)).

Generic observer structure for event modifiers

![Observer Structure of Event Modifiers](image)

Figure: Observer Structure of Event Modifiers
Elementary Observers - Basic Predicates

- The specification of basic predicates relies on events and states.
- An event can be a single event modifiers or a composition of several event modifiers.
- A set of basic predicates used by our property patterns has been defined.
- The generic TPN structure of predicate observers is defined as Fig. 3.
- The transition $E_M$ is an event, and the predicate is assessed using the observer and a set of $mmc$ assertions.

Figure: Predicate Observer Pattern
Elementary Observers - Basic Scope Modifiers

Basic scope modifiers include

- Global
- Before $E_i$
- After $E_i$,
- Between $E_a$ and $E_b$
- Others are compositions of the basic ones.
Elementary Observers for the Verification

Elementary Observers - Occurrence Modifiers

Assume in the state class graph

- \( P \): set of states that match the predicate,
- \( S \): set of states that match the scope,
- \( P \land S \): set of states that match both the predicate and the scope.

**Occurrence**

- **Exist** *Predicate in Scope*: \( \left\{ \begin{array}{ll} P \land S \neq \emptyset & \text{if } S \neq \emptyset; \\ True & \text{if } S = \emptyset. \end{array} \right. 

- **Absent** *Predicate in Scope*: \( P \land S = \emptyset \)

- **Always** *Predicate in Scope*: \( P \land S = S \)
Example of Real-Time Property Verification

Two concurrent processes are modeled in TPN. Both execute only once. The target property $P$ is *Always* $E_A$ *After* $E_B$ *Within* $[1, 2]$.

Figure: Observer-based Verification Example
Example of Real-Time Property Verification

The scope *Between* $E_B + 1$ and $E_B + 2$ is observed by using a composite observer with three parts:

- **obs**$_1$ for event modifier $E_B + 1$,
- **obs**$_2$ for event modifier $E_B + 2$,
- **obs**$_3$ for scope modifier *Between* $E_B + 1$ and $E_B + 2$

Figure: TPN Observers for the Example
Example of Real-Time Property Verification

Figure: Reachability Graph of Verification Example

Very good scalability with Avionics IMA-AFDX systems (GALS – Globally Asynchronous Locally Synchronous)
Outline

1. Real-Time Requirements
2. Time Petri Nets & TINA
3. Real-Time Property Patterns
4. Elementary Observers for the Verification
5. Conclusion
Conclusion

Existing pattern systems
- target expressiveness of real-time requirements
- leave the verification related issues to the users
- Do not guarantee the efficiency of verification

Our pattern system
- Atomic patterns corresponding to elementary observers
- End-user real-time requirements are compositions of these patterns
- Automatic generation of composite observers for composite properties
- Positive outcome on the verification cost
- Integrated in our UML-MARTE real-time verification framework

Future work
- Still more experiments
- Mechanisation and proof of correctness
Thanks for your attention!