Specification Patterns for Time-Related Properties

Volker Gruhn, Ralf Laue
University of Leipzig, Germany, Chair of Applied Telematics / e-Business[*]
{gruhn,laue}@ebus.informatik.uni-leipzig.de

Abstract

We present a pattern system for property specification. It extends the existing patterns identified in [4] which allow to reason about occurrence and order of events, but not about time conditions. Introducing time-related patterns allows the specification of real-time requirements.

The paper is limited to 3 pages. Therefore it contains only basic ideas. The details can be found in [9].

Keywords: *patterns, formal specification, timed model checking, verification*

1 Introduction

Often the persons who have to specify time-related requirements (for example business analysts who have to specify deadlines and other time-related conditions in business process models) are not familiar with existing formalisms and regard it as too difficult to use timed temporal logics. On the other hand, model checking tools that can be used to verify the correctness of a system often require specifications given as temporal logics formulas.

Property specification patterns were successfully used to bridge this gap between practitioners and model checking tools. However, the existing pattern system does not yet consider information about time. We present a catalog of patterns for time-related requirements.

2 Untimed Specification Patterns

Dwyer and his colleagues collected 555 specifications and found that 92% of them matched one of the patterns from their pattern system for property specification[4]. This pattern system enables people who are not experts in temporal logic to read and write formal specifications in a variety of formalisms. With the help of this system, properties like "Event A must be followed by event B" can be expressed. The pattern system does, however, not include timed properties like "Event A must be followed by event B within k time units". The pattern system is explained more deeply in [4] and on `patterns.projects.cis.ksu.edu`.

A property specification consists of a pattern (which describes *what* must occur) and a *scope*, which describes *when* the pattern must hold.

The patterns are as follows (In an event-based formalism, capital letters stand for events):
Absence: P never occurs.
Universality: P occurs throughout a scope.
Existence: P must occur sometime.
Bounded Existence: P must occur at least / exactly or at most k times.
Precedence: P must always be preceded by Q.
Response: P must always be followed by Q.
Chain Precedence / Chain Response: A sequence $P_1, \ldots P_n$ must always be preceded / followed by a sequence $Q_1, \ldots, Q_m$.

Scopes define, *when* the above patterns must hold:
global: The pattern must hold during the complete system execution.
before: The pattern must hold up to an event X.
after: The pattern must hold after the occurrence of an event X.
between: The pattern must hold from the occurrence of X to the occurrence of Y.
until: The same as "between", but the pattern must hold even if Y never occurs.

2.1 Adding Time Information

We express properties using an event-based formalism. This allows us to write something like "the point of time, when event P occurs", abbreviated by t(P). We use terms like $t(P) \pm k$ for "k time units after/before the occurrence of P".

We have added information about time to the following elements of the specification patterns:

1. **The events** (see section 5): Instead of just specify-

[*]The Chair of Applied Telematics / e-Business is endowed by Deutsche Telekom AG

ing that "X occurs", we consider events like "X occurs twice in n time units".

2. **The properties** (see section 4): We want to be able to specify properties like time-bounded Response ("P must always be followed by Q within k time units").

3. **The scopes** (see section 6): We want to delimit the period of validity for a pattern by scopes like "after t(P)+k".

3 Timed Observer Automata

We use the concept of timed observer automata (observers)[2] to describe the desired system behavior. Intuitively, observer automata run in parallel with the model under verification. They reach a certain state if and only if some property can be violated in the model. Synchronization labels are used for synchronizing the model under verification with observer automata: If "something interesting" occurs in the model, the observer automata can react immediately.

The majority of our patterns deal with safety properties. In order to prove that such a property is true, it is sufficient to check that the observer cannot reach some location(s), which we call *error locations*.

For liveness properties (like "every occurrence of P is followed by an occurrence of Q"), reasoning about infinite runs is necessary. As usual, we use *acceptance* conditions for this purpose: Some locations in the observer TA are marked as accepting locations. A counterexample is detected, if there is a non-Zeno run entering an accepting location infinitely often.

4 The Patterns

In [9], we have published the time-related patterns with their observers. The timed patterns include:

- time-bounded existence (something must occur within k time units)

- time-bounded response (P must be followed by Q, and $t(Q) \bowtie t(P) + k$ or $t(Q) \bowtie t(T) \pm k$(where T is some external event and $\bowtie \in \{\leq, \geq\}$). For $t(Q) \geq t(P) + k$, we distinguish two subcases: Either events Q that occur "too early" (i.e. before t(P)+k) are ignored or such events are regarded as a violation of the specification.

- precedence pattern "Q enables P after a delay"

- precedence pattern "Q enables P for k time units"

5 (Combined) Events

Using synchronization labels, a TA can "observe" the occurrence of an event in another TA and react in some way if the event occurs and synchronization can take place. However, often we want to react to "observations" which are related to more than one event or to the time when it occurs. To handle such "combined events" like "P occurs n times" or "both P and Q occur within a time span of no more than k time units" , we construct a *reporting TA* which can take certain transitions if and only if the combined event happens. These transitions can be labeled with a new synchronization label `M!`. In this way the reporting TA can signalize the combined event in the same way as "simple" events are signalized.

[9] lists reporting TA for these combined events:
chains (sequences) of events
time-bounded chains (sequences) of events
an event occurs **n times**
an event occurs **n times within k time units**
collections of events (for example events A, B and C occur, regardless of the order between these events)
time-bounded collections of events
non-occurrence of some event in a given time span

6 Scopes

Let **A** be the observer TA for some property. It can observe whether the property holds *globally*, i.e. during the entire execution of the model. **A** can be modified in order to check the property *over a given scope*. With the modified TA **A'**, we can check the validity of a property before, after and until t(D), and we can also deal with scopes before, after and until $t(D) \pm k$, where k is an integer. This allows us to write specifications like "Something must happen within at least 10 time units after the system has been started".

7 Implementation

The general procedure for implementing observers to check a property in a model-checking tool is as follows: The reporting TA and the observers must be scheduled together with the model under verification in a round-robin manner such that each step of the model is followed by a step of each reporting TA and a step of each observer without letting time pass. Hereby, it is necessary to pay attention to the hierarchy of synchronization labels: The TA whose run depends on events reported by other TA must be scheduled *after* the TA that can report something to them. Usually, this means that after each step of the model under verification, each reporting TA has the chance to evolve and finally the observers are scheduled.

8 Related Work

The original pattern catalog was introduced in [4], a survey of property specifications was published in [5]. [6] specifies time conditions using sentences in structured English, which is a pattern-like system. Later, the authors found that this approach was not suitable for their research project, so they started to use RT-OCL as an alternative approach[7].

The use of TA for specifying temporal properties is quite common[11][3]. [3] states that "automata based notations turned out to be simpler than most logics for describing sequences of events".

[8], [7], [10] and others[1] use timed UML models to specify a system and its properties. UML sequence diagrams, OCL constraints or UML state machines (acting as observers) serve as property specification language.

In the papers mentioned so far, the observer TA have to be constructed by hand, even if the translation of the model itself into the input language of a model checker can be done automatically. [1] introduces a visual language to specify real-time requirements and a tool that translates these requirements into the input language of the model checker Kronos. The user has still to learn a new notation, but the visual language is much easier to understand than other formalisms. However, we see a major drawback in the way how properties have to be specified: The user has to graphically describe the scenarios which *violate* the requirements.

9 Conclusions and Directions for Future Research

We believe that the proposed pattern system helps to specify time-related properties for model checking. Because the system is most useful if the observers are generated automatically in the input language of existing model checking tools, we will develop tool-support for this task.

We believe that the vast majority of real-world specifications are instances of patterns in our system. We should, however, evaluate the completeness of our pattern system by surveying an appropriate number of real-world specifications. If necessary, the pattern system will be updated as a consequence of this study.

References

[1] A. Alfonso, V. A. Braberman, N. Kicillof, and A. Olivero. Visual timed event scenarios. In *26th International Conference on Software Engineering (ICSE 2004)*, pages 168–177. IEEE Computer Society, 2004.

[2] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, pages 322–335. Springer-Verlag, 1990.

[3] V. A. Braberman and M. Felder. Verification of real-time designs: Combining scheduling theory with automatic formal verification. In *ESEC / SIGSOFT FSE*, pages 494–510, 1999.

[4] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In *FMSP '98: Proceedings of the second workshop on Formal methods in software practice*, pages 7–15. ACM Press, 1998.

[5] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. of the 21st international conference on Software engineering*, pages 411–420. IEEE Computer Society Press, 1999.

[6] S. Flake, W. Müller, and J. Ruf. Structured english for model checking specification. In *GI-Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen in Frankfurt*, Berlin, 2000. VDE Verlag.

[7] S. Flake, U. Pape, J. Ruf, and W. Müller. Specification and formal verification of temporal properties of production automation systems. In *Integration of Software Specification Techniques for Applications in Engineering*, volume 3147 of *LNCS*. Springer Verlag, 2004.

[8] S. Graf, I. Ober, and I. Ober. Model checking of UML models via a mapping to communicating extended timed automata. In S. Graf and L. Mounier, editors, *Proceedings of SPIN'04 Workshop, Barcelona, Spain*, volume 2989 of *LNCS*. Springer, April 2004.

[9] V. Gruhn and R. Laue. Patterns for timed property specification. In *3rd Int. Workshop on Quantitative Aspects of Programming Languages (QAPL 05), Edinburgh, Scotland, April 2005, to appear*, 2005.

[10] A. Knapp, S. Merz, and C. Rauh. Model checking - timed UML state machines and collaborations. In *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 395–416. Springer-Verlag, 2002.

[11] I. Ober and A. Kerbrat. Verification of quantitative temporal properties of SDL specifications. In *SDL '01: Proceedings of the 10th International SDL Forum Copenhagen on Meeting UML*, pages 182–202. Springer-Verlag, 2001.

[1] [8] mentions a number of other papers in the bibliography.