Natural Interpretation of UML/MARTE Diagrams for System Requirements Specification

Aamir Mehmood Khan, Frédéric Mallet, Muhammad Rashid

Natural Interpretation of UML/MARTE Diagrams for System Requirements Specification

Aamir M. Khan
Dept. of Electrical & Computer Engg.
College of Engineering,
University of Buraimi, Oman.
Aamir.m@uob.edu.om

Frédéric Mallet
Université Nice Sophia Antipolis
I3S, UMR 7271 CNRS, INRIA
Sophia Antipolis, France.
Frederic.Mallet@unice.fr

Muhammad Rashid
Dept. of Computer Engg.,
College of Computer & Info. Systems,
Umm Al-Qura University, Saudi Arabia.
mfelahi@uqu.edu.sa

***Abstract*—To verify embedded systems early in the design stages, we need formal ways to requirements specification which can be as close as possible to natural language interpretation, away from the lower ESL/RTL levels. This paper proposes to contribute to the FSL (Formal Specification Level) by specifying natural language requirements graphically in the form of temporal patterns. Standard modeling artifacts like UML and MARTE are used to provide formal semantics of these graphical models allowing to eliminate ambiguity in specifications and automatic design verification at different abstraction levels using these patterns.**

***Keywords*—Graphical Properties, Temporal Patterns, FSL, UML, MARTE, CCSL, LTL, Modeling, Embedded Systems**

I. Introduction

Conventionally, the design of an embedded system starts with the system requirements specified by the requirements engineers. In the Electronic Design Automation (EDA) domain, these requirements are then used to implement an executable model and perform early validation. There is a huge gap between these requirements expressed in natural languages and the subsequent formal levels like TLM and RTL. To verify embedded systems we need to specify requirements formally. Requirements expressed in natural language are ambiguous and the ones expressed in ESL are too low-level to deal with design complexity. Intermediate levels like the Formal Specification Level (FSL) have been proposed to fill this gap [1], [2], [3] with models that are both close enough to requirements engineer concerns, and formal enough to allow further phases of automatic or semi-automatic generation and verification. This paper contributes to this effort at FSL.

To express our requirements, we want to use graphical representation as close as possible to what happens naturally. So we propose to use UML [4], a well-accepted modeling language, and its extensions to provide us a way to express time-related properties in a formal way. We use the MARTE profile [5] to extend UML to build timed and untimed requirements for real-time and embedded systems [6]. Additionally, MARTE proposes as an annex, the Clock Constraint Specification Language (CCSL) [7] to complement UML/MARTE modeling elements with timed or causal extensions. CCSL is also used to encode the semantics of UML/MARTE models and resolve potential semantic variation points [8]. In our proposal, CCSL is important to keep the requirements formal and executable.

We promote the use of graphical models over the use of temporal logics [9] formula. Indeed, while temporal logics, like LTL/CTL [10], are widely used in the later design stages in conjunction with model-checkers [11], they are not suitable to directly express high-level requirements at early stages. They are commonly rejected by requirements engineers [12], [13] despite the various attempts to alleviate the syntax with higher-level constructs like in PSL (Property Specification Language) [14]. So we endorse to loose the expressiveness of LTL if we have a clear and formal graphical way of defining properties.

This paper proposes graphical properties to represent various temporal patterns that occur frequently in systems. So for identifying these temporal patterns, we considered several examples like the famous stream boiler case study [15], [16], railway interlocking system [17] and the traffic light controller case study [18]. Working on these diverse examples to model behavioral properties in UML, we noted that several temporal patterns were repeated across examples. To express these patterns in UML, we need MARTE time model and the associated CCSL for representing temporal artifacts. The expressiveness of MARTE/CCSL alone is not enough to model such graphical patterns, so we propose some profile extension to the UML. Moreover, set of rules are defined for creating such patterns and a library of frequently occurring patterns is presented to model temporal behavior. Graphical temporal patterns are represented in UML using state machine and sequence diagrams. Stereotypes from the MARTE time model are used to constrain the time intervals in sequence diagrams and to express logical clocks and time units. State machine diagrams are extended with our custom built *Observation profile* which then helps to build *scenarios*, basic building block for the graphical patterns.

This paper contributes to the trend to build a Formal Specification Level as an intermediate level from natural-language requirements to code synthesis. Presented approach finds its worth by three characteristics that are not, to the best of our knowledge, used jointly in previous approaches. (1) A set of pre-defined primitive domain-specific property patterns, (2) A graphical UML/ MARTE formalism to capture the properties. Rather than having to rely on natural-language, the semantics of these graphical properties is given by a MARTE/CCSL specification, (3) Logical polychronous time [19] as a central powerful abstraction to capture both causal and

Partially funded by the National Science, Technology, and Innovation Plan (NSTIP) Saudi Arabia.

temporal constraints.

Rest of the paper is organized as follows. Section II discusses the related work and their differences with the proposed approach. Section III presents the state of the art about UML artifacts: state machine and sequence diagrams followed by the MARTE time model. Section IV introduces the Observation profile and its associated concepts like scenarios. Section V presents the main contribution of the paper in the form of library of temporal patterns. Then the section VI depicts the application of these temporal patterns on a traffic light controller case study. Finally section VII concludes the paper with a glimpse over the future possibilities.

II. Related Work

Various efforts have been made over the past two decades to bridge the gap between natural language and temporal logic. Initially property specification patterns [20] were proposed in the form of a library of predefined LTL formulae from where the users can pick their desired pattern to express the behavior. Later other works proposed the specification of temporal properties through graphical formalisms [21], [22], [23], and [24]. There have also been several attempts to encode temporal logics formula as UML diagrams [25], [26].

Property Sequence Charts (PSC) [27], [28] are presented as an extension to UML sequence diagrams to model well-known property specification patterns. Originally PSCs focused on the representation of order of events and lacked the support for the timed properties. But the later extension in the form of Timed PSC [25], [26] support the specification of timing requirements. PSCs mainly target LTL properties. The domain of expressiveness of CCSL is different from LTL and LTL-based languages, like PSL. CCSL can express different kind of properties than LTL [29]. Here we are interested to express properties on logical clocks for which LTL is not an obvious choice. Also LTL is not meant to express physical time properties, for which, this framework prefer the use of CCSL. Moreover, PSCs do not benefit from the popular embedded systems modeling and analysis profiles like MARTE. Rather than encoding formula with UML extensions, we propose to reuse UML/MARTE constructs to build a set of pre-defined property patterns pertinent for the domain addressed.

The work presented by Bellini and his colleagues is a review of the state of the art for real-time specification patterns, organizing them in a unified way. It presents a logic language, TILCO-X, which can be used to specify temporal constraints on intervals, again based on LTL. The work of Konrad et al. [30] is to express real-time properties with a facility to denote time-based constraints in real-time temporal logics MTL (Metric Temporal Logic) extending the LTL. Finally, another research work, DDPSL (Drag and Drop PSL) [31], presents a template library defining PSL formal properties using logical and temporal operators.

The research domain of *runtime verification* [32], [33] relies on lightweight formal verification techniques to check the correctness of the behavior of a system. Most works on such online monitoring algorithms focus on the LTL, CTL, MTL for expressing the temporal constraints [34], [35] where high expertise is required to correctly capture the properties to be verified. Moreover, specialized specification formalisms that fit the desired application domains for runtime verification are usually based on live sequence charts (LSCs) or on MSCs [13]. Another research work to mention here is about the generation of controller synthesis from CCSL specifications [36]. Mostly the main focus of runtime verification community is on the generation of efficient observers for online monitoring whereas our proposed framework targets the integration of observers in a complete work-flow. The focus here is on the presenting a natural way to model temporal behavior.

Another aspect of the related work is the advent of Formal Specification Level (FSL), still an informal level of representation. The focus of the FSL approach is to transform natural language descriptions directly into models bridging the design gap. On the other hand, our proposed framework targets the graphical representation of temporal properties replacing the need for textual specification of the system. *Natural language front-end* is a general trend to allow for a syntax-directed translation of concrete pattern instances to formulae of a temporal logic of choice, also used in [30] and [37]. Our framework approach is different from the FSL approach as we target the verification and validation of a subset of behavior rather than the complete system. Design engineers are usually well acquainted to UML diagrams (both state machine and sequence) and any graphical alternative to complex CCSL or LTL/CTL notations is more likely to get wide acceptance. Moreover, the use of MARTE profile allows to reuse the concepts of time and clocks to model timing requirements of embedded systems.

III. UML/MARTE Semantics

This paper proposes an approach to interpret the UML diagrams in a natural way. Hence this section introduces state of the art about the UML artifacts we consider: state machines and sequence diagrams. UML itself lacks the notion of time which is essentially required in modeling temporal patterns. So selected features from the MARTE time model are explained which are used to facilitate semantically sound representation of time.

A. UML *State of the Art*

UML *state machine diagrams* [38] provide a standardized way to model functional behavior of state-based systems. They provide behavior to an instance of a class (or object). Each state machine diagram basically consists of states an object can occupy and the transitions which make the object change from one state to another according to a set of well defined rules. Transitions are marked by guard conditions and an optional action. Among the set of state machine elements defined, only a small subset is used by the presented approach to represent graphical properties by giving specific semantics to that chosen subset. The presented approach specifies S as a set of finite states consisting of simple states and final states while the other states (like initial pseudo-states and choice pseudo-states) are not used at all. From the standard set of state machine elements, only the top region is used while all other set of regions like in state hierarchy are not considered. Finally, the state machine is considered to have a finite set of valid transitions.

UML sequence diagrams [38] allow describing interactions between system objects and actors of the environment. A

sequence diagram describes a specific interaction in terms of the set of participating objects and a sequence of messages they exchange as they unfold over time to effect the desired operation. Sequence diagrams represent a popular notation to specify scenarios of the activities in the form of intuitive graphical layout. They show the objects, their lifelines, and messages exchanged between the senders and the receivers. A sequence diagram specifies only a fragment of system behavior and the complete system behavior can be expressed by a set of sequence diagrams to specify all possible interactions during the object life cycle. It is useful especially for specifying systems with time-dependent functions such as real-time applications, and for modeling complex scenarios where time dependency plays an important role. Sequence diagrams consist of objects, events, messages and operations. *Objects* represent observable properties of their class(es). Object existence is depicted by an object box and its 'life-line'. A life-line is a vertical line that shows the existence of an object over a given period of time. An *event* is a specification of a significant occurrence having time and space existence. A *message* is a specification of a communication that conveys information among objects, or an object and its environment.

Just like the state machine diagrams, the proposed approach focuses on a subset of sequence diagram elements. Amongst the combined fragment elements, this work only uses the $consider$ fragment and sequence diagrams are not used in a hierarchical fashion. Moreover, *StateInvariant* are used in the scenarios to represent *ActivationState* similar to the state machines, showing the triggering of specific state event.

B. MARTE Time Model

The proposed approach uses concepts of clocks and time for which MARTE time model [5], [39] is utilized. MARTE time model provides a sufficiently expressive structure to represent time requirements of embedded systems. In MARTE, time can be physical viewed as dense or discretized, but it can also be logical related to user-defined clocks. Time may even be multiform, allowing different times to progress in a non-uniform fashion, and possibly independently to any (direct) reference to physical time. MARTE time model is a set of logical clocks and each clock can be represented by $\langle \mathcal{I}, \prec \rangle$, where $\mathcal{I}$ represents the set of instants and $\prec$ is the binary relation on $\mathcal{I}$.

Figure 1 presents a simplified view of MARTE Time sub-profile. The green elements are not part of the time sub-profile. At the heart of the profile, the stereotype *ClockType* extends the metaclass *Class* while the stereotype *Clock* extends metaclasses *InstanceSpecification* and *Property*. Clocks can appear in structural diagrams (like SysML block definition, internal block definition, or UML composite structure) to represent a family of possible behaviors. This clock association gives the ability to the model elements identifying precisely instants or duration. MARTE introduces *ClockConstraint* stereotype extending the metaclass *Constraint* through which a MARTE timed system can be specified. *TimedConstraint* is a constraint imposed on the occurrence of an event or on the duration of some execution, or even on the temporal distance between two events. The presented approach uses the *TimedConstraint* on the sequence diagram *DurationConstraint* elements, discussed further in the text later.

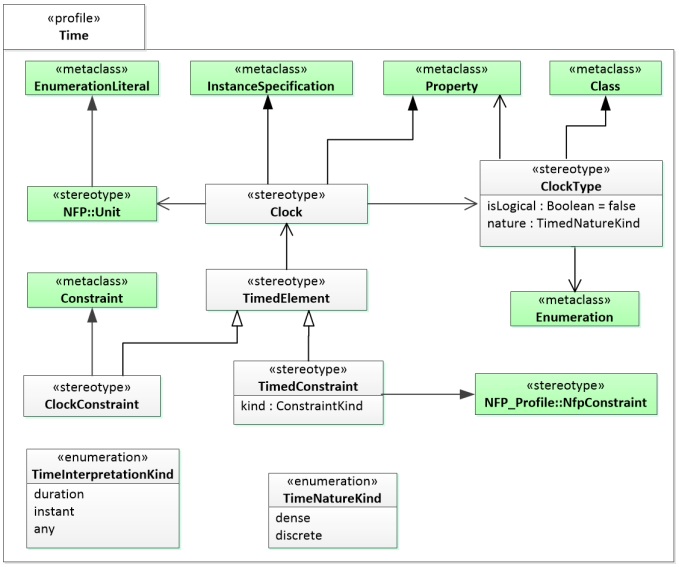


Fig. 1: Excerpt of MARTE Time Sub-profile

IV. Observation Profile

The combined features of UML and MARTE provide a base for developing timed-models but they still lack the specialized features to model graphical temporal patterns. This section presents the first contribution of this paper in the form of a UML profile extension targeting temporal patterns. We introduce the *Observation profile* which is designed to target the expressiveness of graphical temporal properties. This profile resides on top of the UML/MARTE to provide a structure for building some predefined patterns. The focus of the presented work is not the minimalist approach but rather the expressiveness of the property from the system designer's point-of-view. So when a property specifies occurrence of something at some point in time, then it is an event and it seems natural to represent such properties as logical clocks. But when the properties specify duration or interval (not a particular point in time), then the obvious choice of representation is state relations.

In the proposed approach, extended state machine diagrams are mainly used to represent the behavior patterns of a system. These diagrams provide a more natural and syntactically sound interpretation of graphical behavioral patterns of system states. Moreover, some of the system state/event relation patterns are mapped to sequence diagrams in this presented approach. State machines are used to model state-based relations represented graphically in the form of scenarios. Two basic scenarios are possible: negative and positive. These scenarios act as building blocks for more complex state patterns.

Positive scenarios model something that must happen under given conditions, as shown in Figure 2. Two types of stereotypes are used in this scenario. The *ObservationScenario* stereotype extends the UML state machine metaclass. It defines further three tagged values: ScenarioType, Consider and Ignore. *ScenarioType* is the basic type of the scenario. *Consider* contains the collection of all the events that are relevant to this scenario. It is just like the sensitivity list in SystemC, Verilog or VHDL. If the list of relevant events

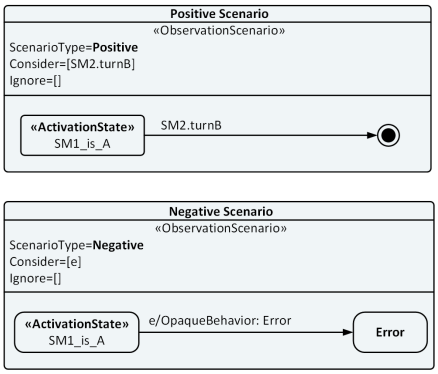


Fig. 2: Positive and Negative Scenarios

is large, then the list of events that are not relevant maybe modeled using the *Ignore*. Another stereotype applied to the scenario is the *ActivationState* stereotype. It extends the UML state metaclass and is used to identify the state that activates this particular scenario. It is active whenever the system is in a specific condition. The positive scenario expects an event to occur whenever the considered state is active. Failure occurs if the event does not occur. The scenario terminates normally if the desired event occurs. In Figure 2, the positive scenario represents the relation *State A starts State B*. Hence one state is ensuring to start another state at the same time, which is a very common behavior in systems.

Negative scenarios model something that must not happen under given conditions. So when the state machine is active, it checks for a particular trigger event that leads the system to a violation/error state shown in Figure 2. This type of properties can use model-checking to detect if the system under observation ever reaches an error state. The negative property in the figure models the relation *State A excludes Event e*, means the event e is not possible while the state is A. One other stereotype applied optionally to the states is the *Duration* stereotype used to model the delay in some case of temporal patterns (not shown here). Such a stereotype is only required for the state machines while the sequence diagrams utilize the existing features of MARTE TimedConstraint stereotype. Figure 3 shows the proposed Observation profile as a collection of all these presented stereotypes.

V. Proposed Temporal Patterns

The major contribution of this proposed approach is to provide a set of reusable generic graphical temporal patterns. For identifying the temporal properties of systems, we started by considering several examples like the famous stream boiler case study [15], [16], railway interlocking system [17] and the traffic light controller case study [18]. Working on these diverse examples to model behavioral properties in UML, we noted that several temporal patterns were repeated across different examples. These patterns collected across various examples were refined with some inspiration taken from the Allen's algebra targeting intervals [40]. This practice gave us a valuable collection of generic patterns divided into three major categories of behavioral relations that may exist in a system: state-state relations, state-event relations, and event-event relations. Researchers have already shown that constraints

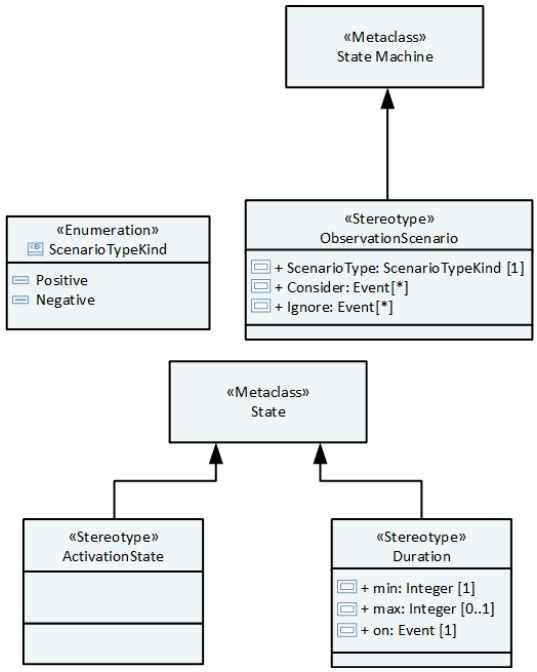


Fig. 3: Proposed Observation Profile

specified in CCSL are capable of modeling logical event-event temporal relations [39], [41]. These CCSL constraints can be represented graphically using SysML/MARTE models [42], [43], [44]. Temporal patterns for the other two categories of relations are:

State-State Relations: precedes, triggers, contains, starts, finishes, implies, forbids, and excludes.

State-Event Relations: excludes, triggers, forbids, contains and terminates.

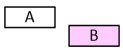
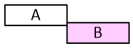
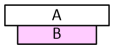
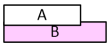
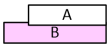
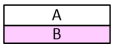
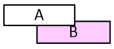
Next subsections discuss selected few of these temporal patterns. A detailed list of these patterns with their syntax and semantics is available online [45].

A. State-State Relations

Presented approach including the two basic scenarios can be used to formally model relations between two different states of a system. Allen's algebra [40] for intervals provides a base defining thirteen distinct, exhaustive and qualitative relations of two time intervals, depicted in Table I. From the comparative analysis of these relations, six primitive relations are extracted that can be applied to the state-based systems. Further two 'negation' relations are added, based on their usage and importance in the examples, to complete the set. The overlapping of states is not particularly interesting relation to dedicate a pattern for. But it can easily be modeled indirectly using the state-event relation 'A contains b_s' where b_s is the start event of state B.

Semantically a state can be considered similar to an interval. We use the nomenclature of using capital letters (A,B,...) to denote states and small letters (a,b,...) for events/clocks. Dot notation like SM1.turnA is also used throughout the text to show the specific events. Given a strict partial ordering

TABLE I: Allen's Algebra and Proposed Relations

No.	Graphic	Allen's Algebra for Intervals	State-State Relation	Symbol
1		A precedes B	A precedes B	$\leqslant$
2		B preceded-by A		
3		A meets B	A triggers B	$\models$
4		B met-by A		
5		A contains B	A contains B	$\supseteq$
6		B during A		
7		A starts B	A starts B	$\vdash$
8		B started-by A		
9		A finishes B	A finishes B	$\dashv$
10		B finished-by A		
11		A equals B	A implies B	$\Rightarrow$
12		A overlaps B	See the text	
13		B overlapped-by A		
			A forbids B	$\neg$
			A excludes B	#

$\mathbb{S} = \langle S, < \rangle$, a state in $\mathbb{S}$ is a pair $[a_s, a_f]$ such that $a_s, a_f \in$ S and $a_s < a_f$. Where a_s is the start and a_f is the end of the state interval. An event or point e belongs to a state interval $[a_s, a_f]$ if $a_s \leqslant e \leqslant a_f$ (both ends included).

Precedence is an important state property where the relation 'A precedes B' means the state A comes before the state B. It includes a delay/deadline clause to explicitly specify the duration between the termination of state A and the start of state B. This delayed version also be equated to the triggers state-event relation (e triggers A after [m,n] on clk). The unit of the duration is dependent on the level of abstraction that is target of the graphical specification.i.e, it can be physical clock, loosely timed clock, or logical clock. Deadline defers the evaluation of state A until some number of ticks of clk (or any other event) occur. The number of ticks of clk considered are dependent on the two parameter natural numbers min and max evaluated as:

- [0, n] means 'before n' ticks of clk
- [m, 0] means 'after m' ticks of clk
- [m, m] means 'exactly m' ticks of clk

Mathematically, given a partial ordering $\mathbb{S}$ having the states A ($[a_s, a_f]$) and B ($[b_s, b_f]$), a constant n and a clock clk, the equation

$$A \boxed{\leqslant} B \quad by\ [m, n]\ on\ clk$$

means $a_f \leqslant b_s$ and b_s occurs within the duration $a_f + \Delta$, where Δ is between m and n ticks of event clk. The last tick of clk coincides with the start the state B (i.e, b_s). Graphically, precedence is based on positive scenarios shown in Figure 4. The first state (State_is_A) is an activation state (shown by the stereotype) while the second state has the duration stereotype applied to specify the interval. The observation scenario gets active when the state machine SM1 is in A state. It then checks for the state exit (turnNotA) and expects the other state machine SM2 to be in state B within the specified time duration. If this behavior occurs as desired, then the scenario goes dormant till the next state activation occurs.

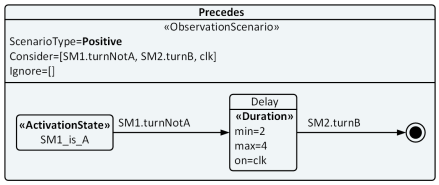


Fig. 4: A precedes B by [2, 4] on clk

Forbiddance is a negation property. Relation 'A forbids B' bars B to occur after state A occurs. It has another slightly different operator that works with events (e forbids A), discussed later on. Hence mathematically, given a partial ordering $\mathbb{S}$ having the states A ($[a_s, a_f]$) and B ($[b_s, b_f]$), the equation $A \boxed{\neg} B$ means $b_s \neq a_f$. Its graphical temporal pattern is shown in Figure 5. Scenario activates whenever SM1 is in state A and on exiting this state, the SM2 is expected to be not in state B (else violation occurs).

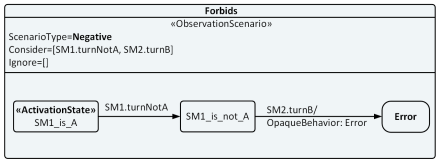


Fig. 5: A forbids B

Exclusion between two states restricts them to occur at the same time. Mathematically, given a partial ordering $\mathbb{S}$ having the states A ($[a_s, a_f]$) and B ($[b_s, b_f]$), the relation 'A excludes B' means that $b_f < a_s$ and $a_f < b_s$ for all instances of A and B. This relation can be decomposed into two basic state-event exclusion relations (shown without boxed symbols).

$$A \# bs\ and\ \ B \# as$$

Graphically, this temporal pattern is derived from the exclusion relation of state and event (discussed in the next sub-section). Two negative scenarios are used to model this behavior as shown in Figure 6. So during the particular state A for SM1, the event turnB is expected not to occur and vice versa for the other case.

B. State-Event Relations

The relations between the system states and events can mostly be modeled using the UML sequence diagrams which

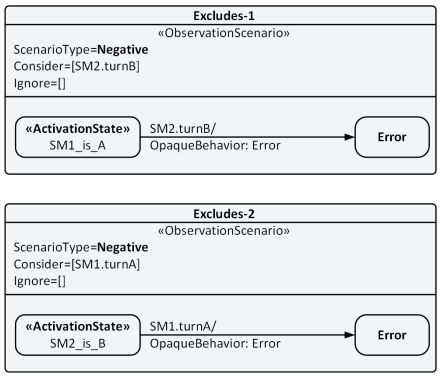


Fig. 6: A excludes B

suits modeling flow of events. The concept of state invariant is used to represent the system activation states. Moreover the sequence diagrams already have the consider/ignore in the form of combined fragments which were introduced earlier in the state machines using the *Observation* profile. Based on the use, we identify four state-event relations.

The **excludes** relation *state A excludes event e* is a bijective relation. Mathematically, given a partial ordering $\mathbb{S}$ having the state A ($[a_s, a_f]$) and a clock e, it can be expressed as $A \# e$ where $e \notin [a_s, a_f]$. It implies either $e < a_s \quad or \quad a_f < e$. Graphically it is modeled using a negative scenario, as shown in Figure 7. Here the SM1 while in state A is causes error on event e.

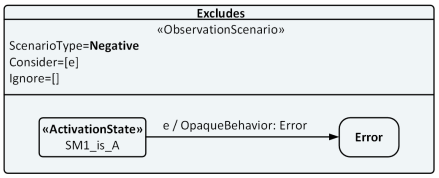


Fig. 7: A excludes e

The **triggers** relation is similar to triggers and starts relations for states. The relation *event e triggers state A* can be expressed mathematically, given a partial ordering $\mathbb{S}$ having the state A ($[a_s, a_f]$), an event e, a constant n and a clock clk, as

$$e \models A \quad after\ [m, n]\ on\ clk$$

It means a_s occurs within the duration $e + \Delta$, where Δ is between m and n ticks of event clk. Graphically, sequence diagram is used to model such relations as shown in Figure 8. The two lifelines represent the system under test and the observer. The *consider* combined fragment maintains the list of participating events for the temporal pattern just like ObservationScenario did for the state machines. *StateInvariants* are used in sequence diagrams to represent activation states. The state invariant 'SM1_is_A'represents the conditions when the state machine SM1 is in state A. *Duration constraint* element is used to specify the required delay. MARTE stereotype

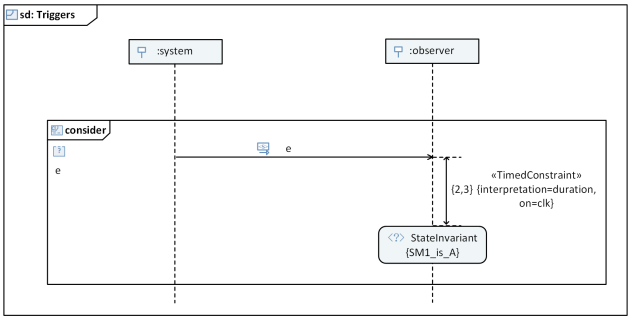


Fig. 8: e triggers A after [2,3] on clk

TimedConstraint is used to further specify the unit of interval and the associated clock.

The **forbids** relation is similar to forbids relation for states which is implemented using state machines but this forbids relation for events is implemented using sequence diagrams (it only has one state to consider). Event e forbids state A implies A must not occur after the event e triggers. Hence given a partial ordering $\mathbb{S}$ having the state A ($[a_s, a_f]$), an event e, the relation is expressed mathematically as $e \neg A$ which means $e \neq a_s$. As this relation involves an event, the graphical temporal pattern is best expressed using sequence diagram, as shown in Figure 9. Here in the forbids relation, the state invariant 'SM1_is_not_A'represents the conditions when the state machine SM1 is not (either not entered or already left) in state A .

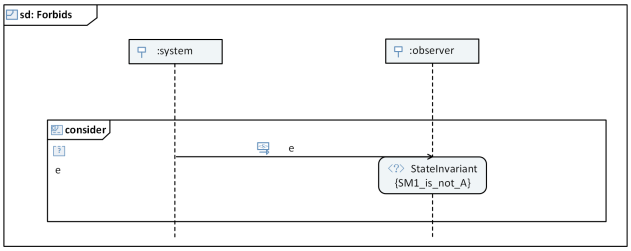


Fig. 9: e forbids A

The **terminates** relation is similar to finishes relation for states. The relation *event e terminates state A* can be expressed mathematically, given a partial ordering $\mathbb{S}$ having the state A ($[a_s, a_f]$), an event e, a constant n and a clock clk, as

$$e \dashv A \quad after\ [m, n]\ on\ clk$$

It means a_f occurs within the duration $e + \Delta$, where Δ is between m and n ticks of event clk. Graphically, it is implemented using the positive scenario state machine, as shown in Figure 10. Here it is important to mention that why a state machine is used for an event-based scenario. Here the important point to note down is that our event e will only trigger when the state machine SM1 is in state A (only then it can terminate). So obviously and activation state is required which will then lead to trace desired event.

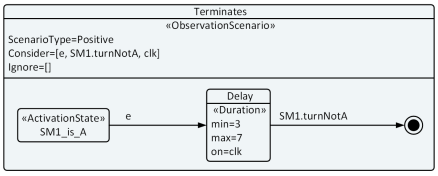


Fig. 10: e terminates A after [3,7] on clk

VI. Application of Temporal Patterns

The case study considered to demonstrate the approach is of the traffic light controller taken from the SystemVerilog Assertions Handbook [18]. It consists of a cross-road over North-South highway and the East-West farm road. There are sensors installed for the emergency vehicles and for the farm road traffic. Highway traffic is only interrupted if there is a vehicle detected by the farm road sensor. The architecture for the traffic light controller consists of two state machines, interface signals of the module and the timers. A few temporal verification properties of the design are discussed next.

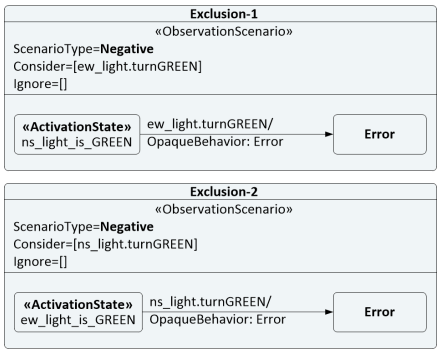


Fig. 11: ns_light=GREEN excludes ew_light=GREEN

Safety property, Never NS/EW lights both GREEN simultaneously. This property is the exclusion of two states ns_light.GREEN and ew_light.GREEN. From our library of graphical temporal patterns, we consider two state-event excludes temporal patterns, as shown in Figure 11. Here if ns_light_is_GREEN is the activation state from SM1, then in the generic pattern we replace the event e with the start event of the opposite light (turnGREEN of ew_light).

State of lights at reset. This constraint requires that whenever reset occurs, the ns_light turns off. This property shows that ns_light.OFF is the consequence of reset. From our library of graphical properties, we use the implies operator for the relation, shown in Figure 12. The implies relation is like the excludes state-state relation as it is further composed of two state-state relations: *starts* and *finishes*. The starts relation guards the beginning of implies relation while the finishes guards the end of the implication relation. Both the relations are implemented using positive scenarios.

State of lights during emergency. This constraint requires that whenever the emergency sensor triggers, ns_light switches from GREEN to YELLOW to RED. The book uses the timing diagram to explain the intended timing relation of the constraint. Text further specifies at another place (chapter 7, SystemVerilog assertions handbook [18]);

The design also takes into account emergency vehicles that can activate an emergency sensor. When the emergency sensor is activated, then the North-South and East-West lights will turn RED*, and will stay* RED *for a minimum period of 3 cycles.*

Yet the SystemVerilog assertion for the constraint tests the ns_light equals RED after two cycles of the emergency. The YELLOW state is never tested. This vindicates our statement that the textual requirement specifications are usually ambiguous, not precise, bulky and the information is scattered. We can implement this property using the triggers relation for the event emgcy_sensor and the state ns_light_is_RED, as shown in Figure 13. A delay of exactly one clock cycle is shown using the duration constraint and the MARTE stereotype.

Safety, GREEN to RED is illegal. Need YELLOW. This constraint is another example of the difference between the textual specification and the constraint implemented as assertion. Though the YELLOW state is specified in the text but it is never tested in the assertion. Here the graphical approach is clear and precise in using the *precedes* relation for states without defining any delay, shown in Figure 14. Here the name of the state 'ns_light_is_not_YELLOW' is not required (as it is not an activation state), and any desired name can be used. This property here ensures YELLOW comes before RED. How much before that is not specified. To avoid cases like YELLOW $\Rightarrow$ GREEN $\Rightarrow$ RED, we can add another constraint *ns_light*=GREEN *precedes ns_light*=YELLOW. A little varying intent can also be implemented using the forbids relation *ns_light*=GREEN *forbids ns_light*=RED which seems to be the desired one for the text 'green to red is illegal'

A. Observers

Graphical interpretation of properties are implemented in the form of *observers*. Verification by observers is a technique widely used in property checking [46], [47], [48]. They check the programs for the property to hold or to detect anomalies. In the presented framework, each temporal pattern is finally transformed into a unique observer code for a specific abstraction level (like TLM, RTL). It proposes to create a library of verification components for each graphical temporal pattern. An observer provides implementation to the semantically

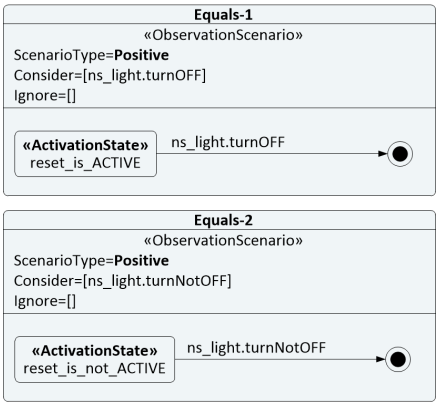


Fig. 12: reset=ACTIVE implies ns_light=OFF

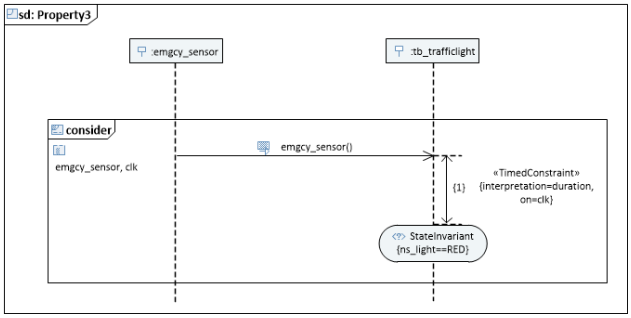


Fig. 13: emgcy_sensor triggers ns_light=OFF after [1,1] on clk

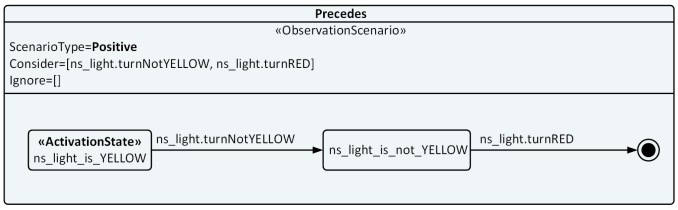


Fig. 14: ns_light=YELLOW precedes ns_light=RED

sound graphical patterns. An observer consists of a set of input parameters, one for each activation state and event. A special violation output is there to flag any anomaly in the behavior.

One important thing to note here is that there is a gap between the way property is captured in Verilog or other low-level HDLs and what the system specification actually requires. So to build these graphical patterns we assure that everything is explicit. When these patterns speak about state, we have state

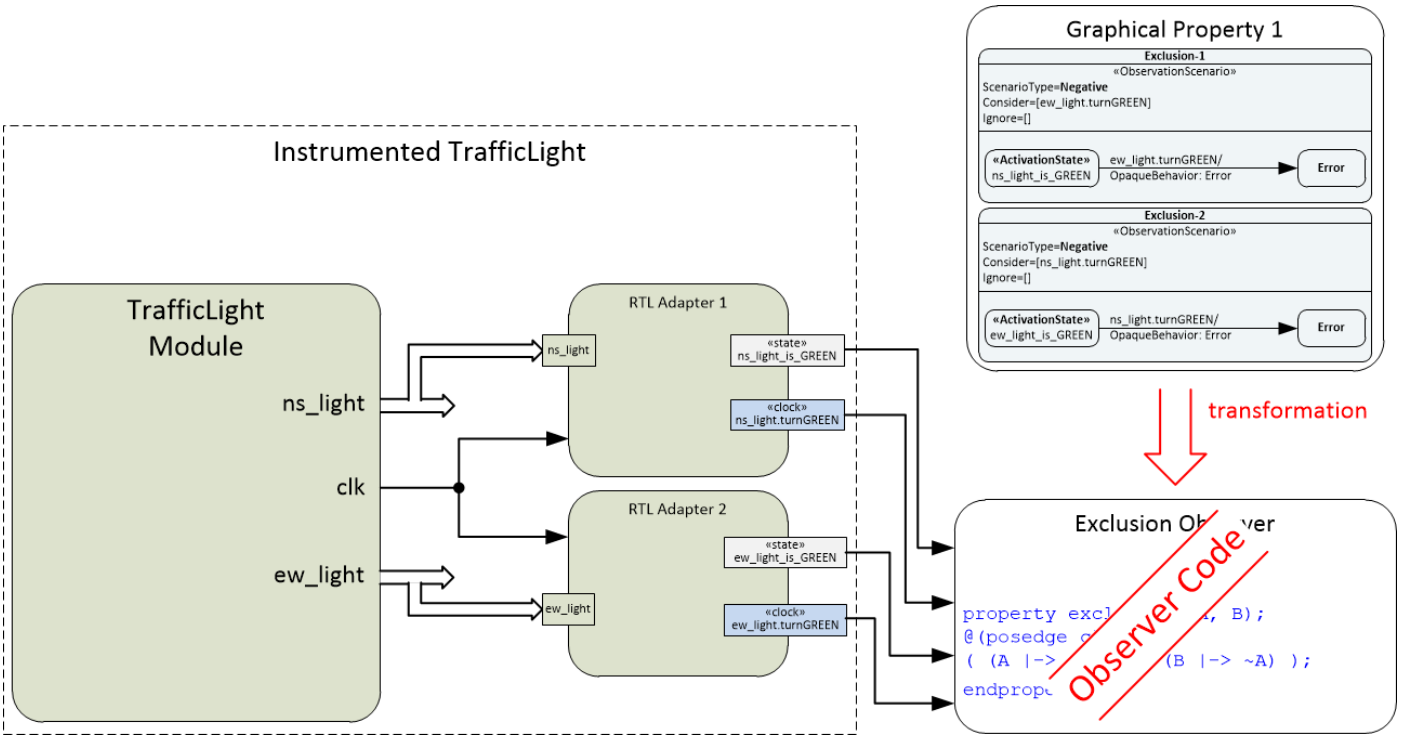


Fig. 15: Integration of Observer in the Verification Environment

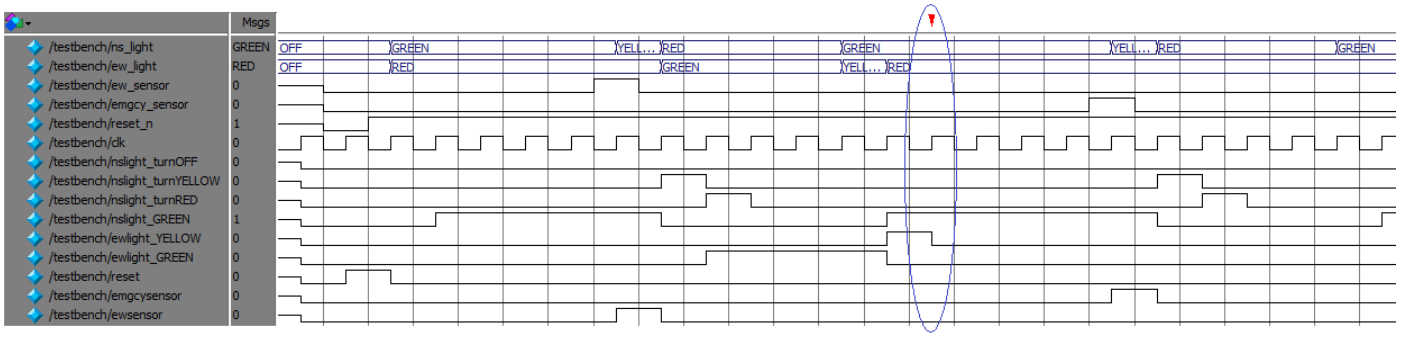


Fig. 16: Simulation Result of Observers for Traffic Light Controller

information to model and when they speak about events then we have event information. The way these patterns work is by relying on *adapters* as a glue logic. These adapters convert the signal or group of signals from the system to states and events. The property patterns implemented in the framework use these events and states. So adapters come in-between the module under verification and the observers. They receive inputs in the form of design module interface signals and state values. From this they generate the appropriate logical clock outputs and state identifiers to be consumed by the observers. Figure 15 shows the integration of observer code in the verification environment. Every design language should have its own set of adapters e.g., if the design module is in VHDL, adapters written in VHDL should be used that will interact with the design and provide the appropriate inputs to the observer. For example, *ns_light_is_*GREEN is a signal that is true whenever the traffic light output ns_light is in GREEN state. In Verilog, the adapter code for the state is given next.

```
always @ (posedge clk) begin
    nslightGREEN = (ns_light == GREEN);
end
assign ns_light_is_GREEN = nslightGREEN;
```

The clock *ns_light.turn*GREEN is the rising edge of this state output and represents the change in the state. In SystemVerilog, we can implement this logic using the *$rose* or the *$past* operators.

```
ns_light_turnGREEN =
$past(ns_light!=GREEN) && ns_light==GREEN;
```

RTL observers are cycle-accurate and need system clocks to operate. Observers in other abstraction levels may have different requirements.

B. Results

For the verification of the traffic light controller, four observers (implementing temporal patterns) from a predefined library were instantiated. Simulation trace in Figure 16 shows the design signals in the upper half and state/event outputs from the adapter in the lower half. Though the first three constraints satisfy this particular execution scenario, the last exclusion relation between the ns_light. GREEN and ew_light.YELLOW fails, as shown by the red marker in the execution trace (circled in the figure). This is exactly the case with this faulty FSM as presented in the book (chapter 7 case study) [18]. To summarize, some of the observations made from this work are:

- This framework makes explicit all the steps between the natural language specification, expressed as a UML diagram, and resulting design verification.
- Notion of adapters is introduced to remove ambiguities between the concepts of states and events.
- States are encoded to represent temporal patterns in behavior. These state-based relations are then transformed into CCSL events. These events are finally encoded as a property used for verification.

VII. Conclusion and Future Work

This paper presents a natural way to interpret UML diagrams annotated with features from MARTE to specify system requirements. It proposes a graphical approach to capture properties in a bid to replace temporal logic properties like in LTL. It also proposes a way to extend the existing capabilities of CCSL which can though represent state relations but is not practically meant for that task. The proposed approach identifies two major categories of temporal patterns, state-based and mixed state/event relations. Semantics of the states in both types of properties have been expressed as *state-start* and *state-end events*. An exhaustive set of state relations, based on Allen's work, have been proposed. Later these relations are implemented using a subset of state machine diagrams and sequence diagrams coupled with features from MARTE time model and Observation profile.

In the future, we wish to extend this work as a complete framework providing a comparative analysis of the presented operators with that of existing CCSL operators and code generation from the graphical properties. In regards to this continued effort, a tool plugin has been developed [49] for model transformation of such graphical patterns directly into CCSL and Verilog HDL based observers.

Acknowledgment

This project is partially funded by NSTIP (National Science Technology and Innovative Plan), Saudi Arabia under the Track "Software Engineering and Innovated Systems" bearing the project code "13-INF761-10".

References

[1] R. Drechsler, M. Soeken, and R. Wille, "Formal specification level: Towards verification-driven design based on natural language processing," in *Specification and Design Languages (FDL), 2012 Forum on*, Sept 2012, pp. 53–58.

[2] M. Soeken and R. Drechsler, *Formal Specification Level - Concepts, Methods, and Algorithms*. Springer, 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-08699-6

[3] I. Harris, "Extracting design information from natural language specifications," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, June 2012, pp. 1252–1253.

[4] J. Rumbaugh, I. Jacobson, and G. Booch, Eds., *The Unified Modeling Language Reference Manual*. Essex, UK, UK: Addison-Wesley Longman Ltd., 1999.

[5] Object Management Group, *Time Modeling in UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems*, Object Management Group (OMG) Std. version 1.1, 2011. [Online]. Available: http://www.omg.org/spec/MARTE/

[6] B. Selić and S. Gérard, *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[7] C. André, "Syntax and Semantics of the Clock Constraint Specification Language (CCSL)," INRIA, Research Report RR-6925, 2009. [Online]. Available: https://hal.inria.fr/inria-00384077

[8] C. André, J. Deantoni, F. Mallet, and R. De Simone, "The Time Model of Logical Clocks available in the OMG MARTE profile," in *Synthesis of Embedded Software: Frameworks and Methodologies for Correctness by Construction*, S. K. Shukla and J.-P. Talpin, Eds. Springer Science+Business Media, LLC 2010, Jul. 2010, p. 28, chapter 7. [Online]. Available: https://hal.inria.fr/inria-00495664

[9] D. M. Gabbay, I. Hodkinson, and M. Reynolds, *Temporal Logic (Vol. 1): Mathematical Foundations and Computational Aspects*. New York, NY, USA: Oxford University Press, Inc., 1994.

[10] A. Pnueli, "The Temporal Logic of Programs," in *18th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1977, pp. 46–57. [Online]. Available: http://dx.doi.org/10.1109/SFCS.1977.32

[11] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999. [Online]. Available: https://books.google.com.om/books?id=Nmc4wEaLXFEC

[12] B. Wile, J. Goss, and W. Roesner, *Comprehensive Functional Verification: The Complete Industry Cycle (Systems on Silicon)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[13] M. Chai and B.-H. Schlingloff, *Runtime Verification: 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*. Cham: Springer International Publishing, 2014, ch. Monitoring Systems with Extended Live Sequence Charts, pp. 48–63. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11164-3_5

[14] IEEE, "Property Specification Language (PSL)," IEEE, 2010. [Online]. Available: http://standards.ieee.org/findstds/standard/1850-2010.html

[15] J.-R. Abrial, E. Börger, and H. Langmaack, "The stream boiler case study: Competition of formal program specification and development methods," in *Formal Methods for Industrial Applications, Specifying and Programming the Steam Boiler Control (the Book Grow out of a Dagstuhl Seminar, June 1995)*. London, UK, UK: Springer-Verlag, 1996, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=647370.723887

[16] M. Al-Lail, W. Sun, and R. B. France, "Analyzing Behavioral Aspects of UML Design Class Models against Temporal Properties," in *Quality Software (QSIC), 2014 14th International Conference on*, Oct 2014, pp. 196–201.

[17] A. E. Haxthausen, "Automated generation of formal safety conditions from railway interlocking tables," *Int. J. Softw. Tools Technol. Transf.*, vol. 16, no. 6, pp. 713–726, Nov. 2014. [Online]. Available: http://dx.doi.org/10.1007/s10009-013-0295-9

[18] B. Cohen, S. Venkataramanan, A. Kumari, and L. Piper, *SystemVerilog Assertions Handbook: for Dynamic and Formal Verification*, 2nd ed. Palos Verdes Peninsula, CA, USA: VhdlCohen Publishing, 2010.

[19] P. L. Guernic, T. Gautier, J. Talpin, and L. Besnard, "Polychronous automata," in *2015 International Symposium on Theoretical Aspects of Software Engineering, TASE 2015*. IEEE Computer Society, 2015, pp. 95–102. [Online]. Available: http://dx.doi.org/10.1109/TASE.2015.21

[20] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *Proceedings of the 21st International Conference on Software Engineering*, ser. ICSE '99. New York, NY, USA: ACM, 1999, pp. 411–420. [Online]. Available: http://doi.acm.org/10.1145/302405.302672

[21] R. L. Smith, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil, "PROPEL: An Approach Supporting Property Elucidation," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: ACM, 2002, pp. 11–21. [Online]. Available: http://doi.acm.org/10.1145/581339.581345

[22] L. Baresi, C. Ghezzi, and L. Zanolin, *Testing Commercial-off-the-Shelf Components and Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ch. Modeling and Validation of Publish/Subscribe Architectures, pp. 273–291. [Online]. Available: http://dx.doi.org/10.1007/3-540-27071-X_13

[23] A. Alfonso, V. Braberman, N. Kicillof, and A. Olivero, "Visual timed event scenarios," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 168–177. [Online]. Available: http://dl.acm.org/citation.cfm?id=998675.999423

[24] H. Kugler, D. Harel, A. Pnueli, Y. Lu, and Y. Bontemps, "Temporal logic for scenario-based specifications," in *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 445–460. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-31980-1_29

[25] P. Zhang, L. Grunske, A. Tang, and B. Li, "A Formal Syntax for Probabilistic Timed Property Sequence Charts," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 500–504. [Online]. Available: http://dx.doi.org/10.1109/ASE.2009.56

[26] P. Zhang, B. Li, and L. Grunske, "Timed Property Sequence Chart," *J. Syst. Softw.*, vol. 83, no. 3, pp. 371–390, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2009.09.013

[27] M. Autili, P. Inverardi, and P. Pelliccione, "Graphical scenarios for specifying temporal properties: An automated approach," *Automated Software Engg.*, vol. 14, no. 3, pp. 293–340, Sep. 2007. [Online]. Available: http://dx.doi.org/10.1007/s10515-007-0012-6

[28] M. Autili and P. Pelliccione, "Towards a graphical tool for refining user to system requirements," *Electron. Notes Theor. Comput. Sci.*, vol. 211, pp. 147–157, Apr. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2008.04.037

[29] R. Gascon, F. Mallet, and J. Deantoni, "Logical Time and Temporal Logics: Comparing UML MARTE/CCSL and PSL," in *Proceedings of the 2011 Eighteenth International Symposium on Temporal Representation and Reasoning*, ser. TIME '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 141–148. [Online]. Available: http://dx.doi.org/10.1109/TIME.2011.10

[30] S. Konrad and B. H. C. Cheng, "Real-time Specification Patterns," in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 372–381. [Online]. Available: http://doi.acm.org/10.1145/1062455.1062526

[31] L. Di Guglielmo, F. Fummi, N. Orlandi, and G. Pravadelli, "DDPSL: An easy way of defining properties," in *Computer Design (ICCD), 2010 IEEE International Conference on*, Oct 2010, pp. 468–473.

[32] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293 – 303, 2009, the 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS07). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1567832608000775

[33] Runtime Verification Community. (2016) Runtime Verification Events 2001-16. [Online]. Available: http://runtime-verification.org/

[34] A. Bauer, M. Leucker, and C. Schallhart, "Runtime Verification for LTL and TLTL," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, pp. 14:1–14:64, Sep. 2011. [Online]. Available: http://doi.acm.org/10.1145/2000799.2000800

[35] C. Watterson and D. Heffernan, "Runtime verification and monitoring of embedded systems," *IET Software*, vol. 1, no. 5, pp. 172–179, October 2007.

[36] H. Yu, J. P. Talpin, L. Besnard, T. Gautier, H. Marchand, and P. L. Guernic, "Polychronous controller synthesis from MARTE CCSL timing specifications," in *Formal Methods and Models for Codesign (MEMOCODE), 2011 9th IEEE/ACM International Conference on*, July 2011, pp. 21–30.

[37] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, and A. Tang, "Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar," *IEEE Transactions on Software Engineering (TSE)*, vol. 41, no. 7, pp. 620–638, Jul. 2015.

[38] Object Management Group (OMG), "Unified Modeling Language (UML), Superstructure Specification, Version 2.4," 2011.

[39] C. André, F. Mallet, and R. de Simone, "Modeling time(s)," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, G. Engels, B. Opdyke, D. Schmidt, and F. Weil, Eds. Springer Berlin Heidelberg, 2007, vol. 4735, pp. 559–573. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-75209-7_38

[40] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, Nov. 1983. [Online]. Available: http://doi.acm.org/10.1145/182.358434

[41] J. Suryadevara, "Model Based Development of Embedded Systems using Logical Clock Constraints and Timed Automata," Ph.D. dissertation, Malardalen University, Sweden, 2013.

[42] F. Mallet, *Formal Modeling and Verification of Cyber-Physical Systems: 1st International Summer School on Methods and Tools for the Design of Digital Systems, Bremen, Germany, September 2015*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, ch. MARTE/CCSL for Modeling Cyber-Physical Systems, pp. 26–49. [Online]. Available: http://dx.doi.org/10.1007/978-3-658-09994-7_2

[43] ——, "Clock constraint specification language: specifying clock constraints with UML/MARTE," *Innovations in Systems and Software*

Engineering, vol. 4, no. 3, pp. 309–314, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11334-008-0055-2

[44] F. Mallet and C. André, "UML/MARTE CCSL, Signal and Petri nets," INRIA, Research Report RR-6545, 2008. [Online]. Available: https://hal.inria.fr/inria-00283077v4

[45] A. M. Khan, "Semantics of Graphical Temporal Patterns using UML/MARTE," NSTIP, Research Report. [Online]. Available: http://www.modeves.com/patterns.html

[46] N. Halbwachs, F. Lagnier, and P. Raymond, "Synchronous observers and the verification of reactive systems," in *Algebraic Methodology and Software Technology (AMAST93)*, ser. Workshops in Computing, M. Nivat, C. Rattray, T. Rus, and G. Scollo, Eds. Springer London, 1994, pp. 83–96. [Online]. Available: http://dx.doi.org/10.1007/978-1-4471-3227-1_8

[47] L. Aceto, A. Burgueño, and K. Larsen, "Model checking via reachability testing for timed automata," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, B. Steffen, Ed. Springer Berlin Heidelberg, 1998, vol. 1384, pp. 263–280. [Online]. Available: http://dx.doi.org/10.1007/BFb0054177

[48] S. Bensalem, M. Bozga, M. Krichen, and S. Tripakis, "Testing conformance of real-time applications by automatic generation of observers," *Electronic Notes in Theoretical Computer Science*, vol. 113, pp. 23 – 43, 2005, proceedings of the Fourth Workshop on Runtime Verification (RV 2004)Fourth Workshop on Runtime Verification 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157106610405251X

[49] A. M. Khan, "TemPAC: Temporal Pattern Analyzer and Code-generator EMF plugin." [Online]. Available: http://www.modeves.com/tempac.html