Visual Timed Event Scenarios*

A. Alfonso, V. Braberman, N. Kicillof
Departamento de Computación – FCEN
Universidad de Buenos Aires, Argentina
{aalfonso, nicok, vbraber}@dc.uba.ar

A. Olivero†
Centro de Estudios Avanzados – CEAV
Universidad Argentina de la Empresa, Argentina
aolivero@uade.edu.ar

Abstract

Formal description of real-time requirements is a difficult and error prone task. Conceptual and tool support for this activity plays a central role in the agenda of technology transference from the formal verification engineering community to the Real Time Systems development practice. In this article we present VTS, a visual language to define complex event-based requirements such as freshness, bounded response, event correlation, etc. The underlying formalism is based on partial orders and supports real-time constraints. The problem of checking whether a timed automaton model of a system satisfies these sort of scenarios is shown to be decidable. Moreover, we have also developed a tool that translates visually specified scenarios into observer timed automata. The resulting automata can be composed with a model under analysis in order to check satisfaction of the stated scenarios. We show the benefits of applying these ideas to some case studies.

1. Introduction

Critical systems are found in an increasing variety of application fields and industries like electronics, control, aeronautics, health equipment, etc. Most of them are embedded systems, controlling devices that may risk lives or damage assets, hence termed safety-critical systems. These applications generally involve concurrency and control aspects, as well as real-time requirements that challenge the development process. Their construction benefits from advanced tools and techniques in order to support reliable and economically feasible development and verification processes.

Computer Aided Verification techniques, and particularly Modelchecking, constitute a promising approach that deals with these kinds of systems. These techniques have originally and mainly been applied to the automatic verification of circuits and protocols but their use is being extended to software intensive systems (e.g. [9, 5]). Moreover, formal models are being explored as a means to perform testing [21] and monitoring [19] of applications.

Providing user-friendly means for the description of formal requirements plays an important role in this technology transfer agenda (e.g. [15, 18]). A remarkable example is the pattern-based approach and catalogue of verification properties defined in [15], which bridges the gap between natural language and verification logics, and also suggests that a large number of formal properties stated in practice fall into a small set of categories.

When developing VTS, our scenario-based notation, we focused on expressing real-time requirements in a visual and friendly fashion, avoiding the use of formal logics such as TCTL [2], and the explicit use of observers, which may become in practice an overwhelming and error-prone task, even for practitioners trained in the use of formal methods.

The need for visual and simple formalisms (instead of powerful but often cumbersome logics) when dealing with event-based requirements has been pointed out in several works (e.g. [26, 18]). Specifying the whole expected behavior of a system, even in an abstract form, is usually a difficult task. We believe that the use of (partial) scenarios is a key strategy in dealing with the problem of expressing event-based properties. In particular, our approach consists in graphically describing those generic scenarios of the analyzed model which violate the requirements. A VTS scenario is basically an annotated partial order of relevant events which can be regarded as a pattern, in the sense that it denotes a (possibly infinite) set of matching executions.

VTS is meant to existentially predicate on executions. That is, it is used to state a simple but yet relevant family of questions of the form "Is there a potential run of the system that can match this generic scenario?". These questions are closely related to the concept of *anti-scenario* or *forbidden scenario* (e.g. [18, 28]) as we use them to express infringements of safety or progress requirements. These questions turn out to be decidable, and we have developed a tool to

* Research partially supported by ANPCyT project BID-PICT 11738 and Microsoft Research Embedded Systems Innovation Excellence Award
† Partially supported by UADE projects ISI03B and TSI04B

edit scenarios and model check them over timed automata models of the system under analysis. This article is structured as follows. We start by giving an intuitive approach to VTS by means of some basic examples. We then present the formal syntax and semantics of VTS, together with a real-life example, and we provide a mapping from VTS scenarios to timed automata in order to model check system models. In the next section we show the value and versatility of VTS and the software tool by applying it to a case study. We then compare our approach with other notations based on scenarios or event correlation. To close the article we provide a summary and a discussion on current and future lines of research.

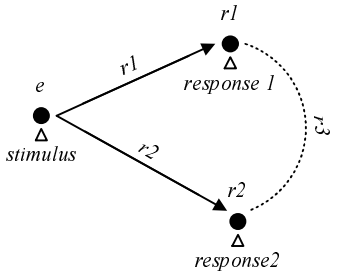
2. Introducing VTS

The basic elements of our graphical notation are points connected by lines and arrows. Points are labeled by (possibly empty) sets of events, meaning that the point stands for an occurrence of one of the events during execution. An arrow between two points indicates precedence of the source point with respect to the destination. Events labeling the arrow are interpreted as forbidden events between both points. The following examples serve as an introduction to VTS.

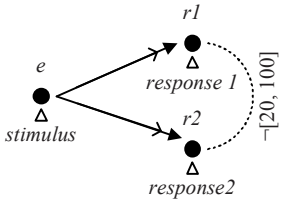
The scenario in Fig. 1(a) expresses a predicate about a run that is true if and only if it contains a stimulus e followed by two responses (the next $r1$- and $r2$-events) not separated by another response $r3$. Triangles bellow points are used to assign them an optional name. The arrows from $stimulus$ to $response1$ and 2 indicate that e occurs before $r1$ and $r2$. To state that the point $response1$ is the first occurrence of its label after e (i.e. there is not another $r1$ between e and $r1$), the arrow $stimulus$-$response1$ was labeled $r1$. In order to express the condition "... which are not separated by another response $r3$", a dashed line links $response1$ and $response2$, with label $r3$. We use a line instead of an arrow because there is no precedence between $response1$ and $response2$ (their relative order is unimportant). To illustrate when a given execution *matches* the VTS scenario of Fig. 1(a), suppose we have the following sequences of events:

$$s1 : ...a, e, b, c, r1, d, r3, r2, z...,$$
$$s2 : ...a, e, b, c, r1, d, r2, f, r3, z...,$$
$$s3 : ...a, e, b, r2, r1, c, r3, r2, z....$$

Then sequences $s2$ and $s3$ match the scenario, because the first $r1$-event and $r2$-event after the only e-event have no $r3$-event in between. If this should be interpreted as a *negative* scenario, we would discard systems producing $s2$ and $s3$ for violating our requirement. On the contrary, the sequence $s1$ does not match the scenario, and therefore would satisfy (agree with) the requirement.



(a) Separated responses



(b) Correlated responses

Figure 1. VTS scenarios

The trained reader should immediately appreciate the simplicity of this notation over non-graphical alternatives. For example the following TCTL [2] formula[1] expresses the same requirement as the scenario in Fig. 1(a).

$$\begin{aligned}&\texttt{after}(e) \Rightarrow ((\neg(\texttt{after}(r1) \vee \texttt{after}(r2)))\, \exists\mathcal{U} \\ &\qquad (\texttt{after}(r1) \wedge (\neg\texttt{after}(r3)\, \exists\mathcal{U}\, \texttt{after}(r2)))) \\ &\quad \vee((\neg(\texttt{after}(r1) \vee \texttt{after}(r2)))\, \exists\mathcal{U} \\ &\qquad (\texttt{after}(r2) \wedge (\neg\texttt{after}(r3)\, \exists\mathcal{U}\, \texttt{after}(r1))))\end{aligned}$$

VTS has the virtue, over this kind of notations, of being intuitive enough to be used by industry practitioners lacking academical training, while still possessing a formal semantics and serving as an input for formal verification methods such as modelchecking.

Let us describe a more complex requirement including temporal constraints: a correlated response. Fig. 1(b) shows a VTS scenario for the violation of the requirement that whenever two responses follow a given stimulus e (i.e., the next $r1$- and $r2$-events), they should be separated by at least 20 and at most 100 time units (t.u.). As in Fig. 1(a), the arrows indicate the relative ordering between e, $r1$ and $r2$. Here we introduce an abbreviation to represent a frequent sub-pattern: certain point represents the *next* occurrence of $r1$ after e. The abbreviation is a second (open) arrow near $r1$, and is equivalent to adding $r1$ as a forbidden event on

1 We are using a trick to predicate on states instead of events as is used in the Kronos tool [10]. Another possible technique are fluents [17].

the arrow (as we did in Fig. 1(a)). Conversely, in order to express that there is not another e-event after a particular e and before an $r1$-event (i.e. to signal the *previous* e-event before $r1$) there is a symmetrical notation: an open arrow near the e extreme. The restriction $\neg[20, 100]$ on the dashed line means that the temporal distance between the responses is smaller than 20 t.u. or greater than 100 t.u.

Scenarios including temporal restrictions must be matched to time-stamped sequences. Let us consider the previous sequences, with a timestamp added to each event (for the sake of simplicity, we use natural numbers as timestamps, but VTS admits any non-negative real number):

$s4$: ... $\overset{a}{12}\ \overset{e}{15}\ \overset{b}{39}\ \overset{c}{50}\ \overset{r1}{72}\ \overset{d}{123}\ \overset{r3}{140}\ \overset{r2}{148}\ \overset{z}{155}$ ...,

$s5$: ... $\overset{a}{3}\ \overset{e}{7}\ \overset{b}{12}\ \overset{c}{88}\ \overset{r1}{109}\ \overset{d}{111}\ \overset{r2}{114}\ \overset{f}{121}\ \overset{r3}{125}\ \overset{z}{152}$ ...,

$s6$: ... $\overset{a}{5}\ \overset{e}{7}\ \overset{b}{69}\ \overset{r2}{78}\ \overset{r1}{87}\ \overset{c}{100}\ \overset{r3}{146}\ \overset{r2}{152}\ \overset{z}{199}$ ....

It can be seen that $s4$ does not match the scenario in Fig. 1(b), because the temporal distance between $r1$ and $r2$ (148 - 72 = 76 t.u.) is not in the time span $\neg[20, 100]$. The opposite happens for $s5$ with inter-response distance of 114 - 109 = 6 t.u.; and $s6$, with 87 - 78 = 9 t.u.

Another useful idiom asserts that something happens (or not) since the beginning or until the end of a given execution. VTS has two special symbols: a big full circle for *begin*, and two concentric circles for *end*. The scenario in Fig. 2 states that the temporal distance since the first a to the last b (both *in the whole execution*) is at most 100 t.u.

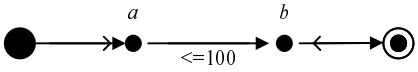


Figure 2. Begin and end symbols

VTS can also identify the *first* or the *last* event in a set. The graphical representation for the first event in a set is a point linked to each point in the set by dotted lines ending in small empty circles. The notation for the last event uses full circles. Nested combination of first and last representatives as in Fig. 3 are allowed. Part of this scenario will match situations where, given two sets of events, $\{a1, a2\}$ and $\{b1, b2, b3\}$, the temporal distance between the last event to occur in each set is less than 100 t.u. Fig. 3 also shows how VTS can convey very complex properties of a system. This scenario depicts the case where a watchdog is turned on ($wd\ on$) at least 50 t.u. before any event in both sets occurs, and not turned off until all monitored events have occurred. The watchdog is supposed to detect the case described above (certain time spread between the first and the last group to end). As this is a negative scenario, it matches (faulty) traces in which all these conditions are met and yet the watchdog fails to issue an alarm.

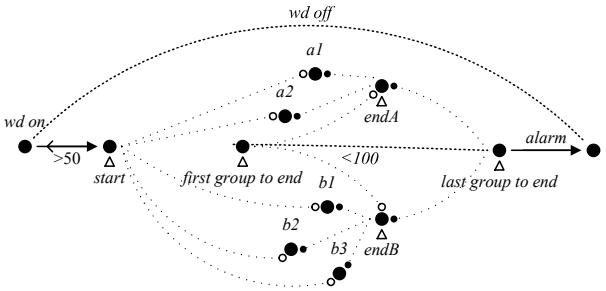


Figure 3. Representatives

Fig. 4 shows the complete graphical notation of VTS.

3. Formalizing VTS

We call $\mathfrak{I}_{\mathbf{N}}$ the set of integer-bounded intervals of positive real numbers. A *time restriction* is a formula of the form θ or $\neg\theta$, where θ is an interval in $\mathfrak{I}_{\mathbf{N}}$. Φ is the set of all time restrictions. Given a non-negative real number t and an interval θ, we say $t \vDash \theta$ iff $t \in \theta$ and $t \vDash \neg\theta$ iff $t \notin \theta$.

Definition 1 (Sequence). Given a set C, a *sequence over* C is a (possibly infinite) sequence of elements from C. Given a sequence s, $|s|$ is its length (we say that $|s| \overset{def}{=} \infty$ when s is infinite). $\Pi(s) \overset{def}{=} \{i \in \mathbf{N} \,/\, 0 \leq i < |s|\}$ is the set of positions of s. Given $i, j \in \Pi(s)$, s_i is the i^{th} element of s; $s_{i]}$ is the prefix ending at position i; $s_{[i}$ is the suffix starting at position i and $s_{[i,j]}$ is the subsequence from position i to position j (if $i > j$, $s_{[i,j]} \overset{def}{=} s_{[j,i]}$). Using '(' or ')' instead of '[' or ']' means the corresponding subsequence does not include its border(s). We call $first(s)$ the first element of s. If s is finite, $last(s)$ is its last element.

Definition 2 (Temporal Sequence). A *temporal sequence* is a weakly increasing sequence of timestamps (i.e. non negative real numbers). Given a finite temporal sequence τ we define $\Delta(\tau)$ as the time elapsed during τ. $\Delta(\tau) = last(\tau) - first(\tau)$ or 0 if $|\tau| = 0$.

Definition 3 (Trace). A *trace* over a set C is a pair $\langle s, \tau \rangle$ where s is a sequence over $C \cup \{\lambda\}$ and τ is a temporal sequence of the same length.

3.1. VTS Syntax

Definition 4 (Scenario). A *scenario* is a tuple $\langle \Sigma, P, \ell, <, <_{first}, <_{last}, \gamma, \delta \rangle$, where Σ is a finite set of events; P is a finite set of points; $\ell : P \rightarrow 2^{\Sigma}$ is a function that labels each point with a set of events;

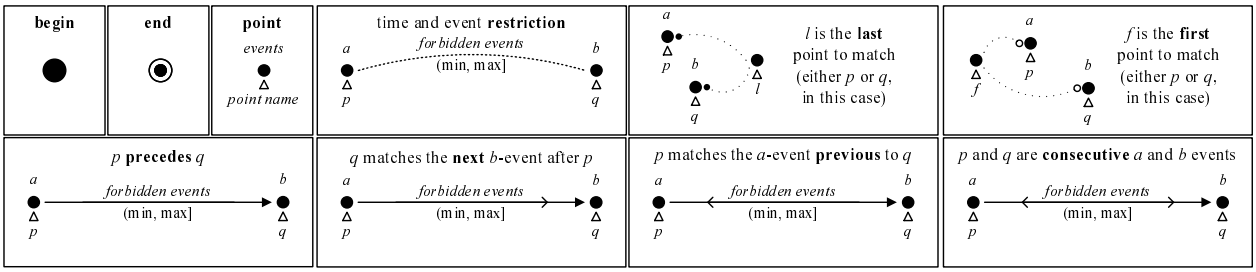


Figure 4. VTS graphical notation

$<\subseteq P \uplus \{\mathbf{0}\} \times P \uplus \{\infty\}$ is a precedence relation among points ($\mathbf{0}$ and ∞ represent the beginning and the end of execution, respectively); $<_{first}\subseteq\ P \times P$ ties a point to the first of a set; $<_{last}\subseteq\ P \times P$ ties a point to the last of a set; $\gamma : P \uplus \{\mathbf{0}, \infty\} \times P \uplus \{\mathbf{0}, \infty\} \rightarrow 2^{\Sigma}$ assigns to each pair of points the set of events forbidden between them; and $\delta : P \uplus \{\mathbf{0}\} \times P \uplus \{\mathbf{0}\} \rightarrow \Phi$ assigns to each pair of points a restriction for the time elapsed between them. The transitive closure $(< \cup <_{first} \cup <_{last})^{+}$ must be a partial order over $P \uplus \{\mathbf{0}, \infty\}$. For all points x and y; $\gamma(\mathsf{x}, \mathsf{y}) = \gamma(\mathsf{y}, \mathsf{x})$ and $\delta(\mathsf{x}, \mathsf{y}) = \delta(\mathsf{y}, \mathsf{x})$ must hold.

3.2. VTS Semantics

The semantics of VTS assigns to each scenario a set of traces satisfying it. Labelled points represent events in the traces. A point can be matched to a particular position in a trace if the event in that position is among the allowed events associated to the point by the labelling function ℓ. Points that are not labelled are called *instants*. They represent moments in the execution not necessarily associated with an event.

A point p in $Dom(<_{first})$ (resp. $Ran(<_{last})$) is called a *representative*, as it represents the first (resp. last) point in the set $R = \{\mathsf{p}'/\mathsf{p} <_{first} \mathsf{p}'\}$ (resp. $\{\mathsf{p}'/\mathsf{p}' <_{last} \mathsf{p}\}$) to be matched in a trace. The sets R for a (first- or last-) representative p, are denoted $FirstOf(\mathsf{p})$,$LastOf(\mathsf{p})$, resp. Non-representative points are called *concrete*. Intuitively, a *matching* is a mapping between points in a scenario and positions in a trace, intended to show how the trace satisfies the scenario. Two concrete points must match different positions in a trace[2].

Definition 5 (Matching). Given a scenario $\mathcal{S} = \langle \Sigma, P, \ell, <, <_{first}, <_{last}, \gamma, \delta \rangle$, a trace $\sigma = \langle s, \tau \rangle$ over Σ and a mapping $\hat{\cdot} : P \mapsto \Pi(\sigma)$; we say that $\hat{\cdot}$ is a *matching* between $\mathcal{S}$ and σ iff for all points $\mathsf{p}, \mathsf{q} \in P$: **M1** $\ell(\mathsf{p}) = \emptyset$ or $s_{\hat{\mathsf{p}}} \in \ell(\mathsf{p})$; **M2** $\hat{\cdot}$ is injective for concrete points; **M3** if $\mathsf{p} < \mathsf{q}$ then $\hat{\mathsf{p}} < \hat{\mathsf{q}}$; **M4** $s_{(\hat{\mathsf{p}},\ \hat{\mathsf{q}})} \cap \gamma(\mathsf{p}, \mathsf{q}) = \emptyset$; **M5** $s_{\hat{\mathsf{p}})} \cap \gamma(\mathbf{0}, \mathsf{p}) = \emptyset$ and $s_{(\hat{\mathsf{p}} \cap \gamma(\mathsf{p}, \infty)} = \emptyset$; **M6** $\Delta(\tau_{[\hat{\mathsf{p}},\ \hat{\mathsf{q}}]}) \vDash \delta(\mathsf{p}, \mathsf{q})$; **M7** $\Delta(\tau_{[\hat{\mathsf{p}}]}) \vDash \delta(\mathbf{0}, \mathsf{p})$; **M8** if p is a first-representative (resp. last-) then $\hat{\mathsf{p}} = min\{\hat{\mathsf{r}}/\mathsf{r} \in FirstOf(\mathsf{p})\}$ (resp. max and $LastOf$).

We say that a trace σ *satisfies* a scenario $\mathcal{S}$ (noted $\sigma \vDash \mathcal{S}$) iff there exists at least one matching between them.

3.3. Stock Exchange Example

In a re-engineering project for the electronic market at Buenos Aires Stock Exchange, part of our team was involved in suggesting ways to rigorously specify real-time requirements for the problem of brokers placing bids and receiving updated data. The goal was to evaluate the success or failure of competing protocols at meeting those requirements. For instance, in Fig. 5 we show a timing constraint for the "simultaneity" of re-multicasts of messages in a TRMP-based protocol ([24]) for three receivers.

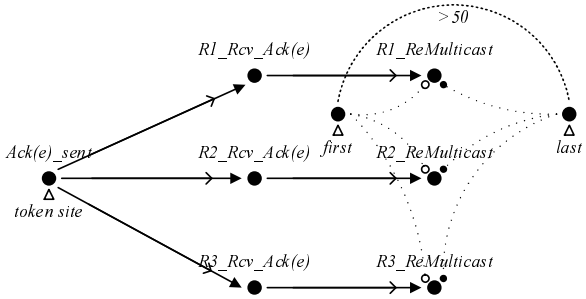
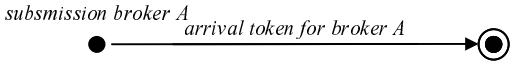


Figure 5. Stock Exchange: simultaneity

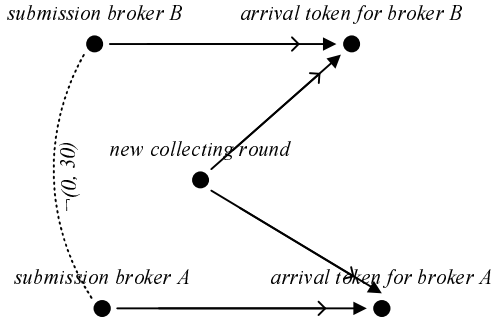
Another proposed protocol periodically broadcasts a token to collect the bids accumulated in brokers' machines. The broadcast acts as a logical tick. Following are some of the relevant requirements for this protocol expressed as anti-scenarios[3]. Fig. 6(a) expresses the requirement that

2 Weaker assumptions can still be expressed with several scenarios.

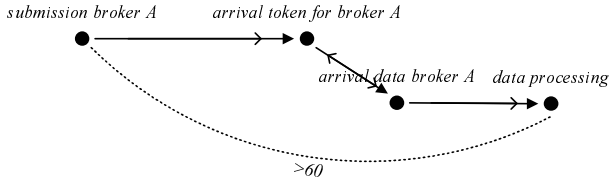
3 Other examples can be found at www.dc.uba.ar/people/proyinv/VTS.



(a) Bids are collected



(b) Bounded unfair time window



(c) Freshness of bids

Figure 6. More Stock Exchange scenarios

the token must eventually arrive. In this protocol, two bids from different brokers collected in the same token broadcast round belong to the same logical time tick. Such a pair of bids might thus be matched to the offers in an incorrect order (with respect to the wall clock time). Fig. 6(b) sets a time limit for this unfair behavior (that is, the anti-scenario detects two bids processed on the same round and submitted in two too-distant moments). Fig. 6(c) sets a bound on the age of the bid when the data is processed.

4. Model Checking Scenarios

In this section we provide a mapping from VTS scenarios to timed automata. Given a scenario, we describe a tableau that recognizes all traces matching the scenario.

Timed automata are a widely used formalism to model and analyze timed systems. They are supported by several tools (e.g. [10, 8]). Their semantics is based on labeled state-transition systems and time-divergent runs over them. A complete formal presentation can be found in [3, 10]. In a few words, a time automata is a finite state machine where clocks are introduced in order to predicate on maximum and minimum time that must or may elapse between transition occurrences.The parallel composition $\mathcal{A}_1 \parallel \mathcal{A}_2$ of TAs $\mathcal{A}_1$ and $\mathcal{A}_2$ is defined using synchronous product of automata ([10]).

Given a model of the system under analysis represented as a timed automaton $\mathcal{A}$ and a scenario $\mathcal{S}$, we define; $\mathcal{A} \vDash \mathcal{S}$ iff there exists a time divergent run r of $\mathcal{A}$, starting in the initial state of $\mathcal{A}$, such that $r \vDash \mathcal{S}$.

Definition 6 (Timed Automata). A *timed automaton* (TA) is a tuple $\mathcal{A} = \langle L, X, \Sigma, E, I, \mathsf{l}_0 \rangle$, where L (denoted $label(\mathcal{A})$) is a finite set of locations, X is a finite set of clocks (non-negative real variables), Σ (denoted $locs(\mathcal{A})$) is a set of labels, E is a finite set of edges, $I : L \overset{tot}{\rightarrow} \Psi_X$ is a total function associating to each location a clock constraint called the location's invariant, and $\mathsf{l}_0 \in L$ is the initial location (denoted $init(\mathcal{A})$). Each edge in E is a tuple $\langle \mathsf{l}, a, \psi, \alpha, \mathsf{l}' \rangle$, where $\mathsf{l} \in L$ is the source location, $\mathsf{l}' \in L$ is the target location, $a \in \Sigma$ is the label, $\psi \in \Psi_X$ is the guard, $\alpha \subseteq X$ is the set of clocks reset at the edge. The set of clock constraints Ψ_X for a set of clocks X is defined according to the following grammar: $\Psi_X \ni \psi ::= x \prec c | \psi \wedge \psi | \neg\psi$, where $x \in X$, $\prec \in \{<, \leq\}$ and $c \in \mathbf{N}$.

Usually, a TA $\mathcal{A}$ has an associated mapping $\mathcal{P} : L \mapsto 2^{Props}$ which assigns to each location a subset of a set of propositional variables ($Props$).

4.1. Tableau Construction

Given a scenario, a tableau (a non-deterministic timed automaton) can be built that recognizes all traces matching the scenario. Its construction is based on the notion of configurations –closed subsets of the PO that stand for possible sets of matched points in a run–, which are used to label automaton states. Transitions represent extensions of the matching from one configuration to another one.

For the rest of the section, when we refer to a scenario $\mathcal{S}$, we are referring to the tuple $\langle \Sigma, P, \ell, <, <_{first}, <_{last}, \gamma, \delta \rangle$.

A *configuration* Θ of a scenario $\mathcal{S}$ is a subset of P such that: **C1** Θ is left-closed under the relation $(< \cup <_{first} \cup <_{last})^+$; **C2** if a point p is a first-representative and $\mathsf{p} \in \Theta$ then $FirstOf(\mathsf{p}) \cap \Theta \neq \emptyset$; and **C3** if a point p is a last-representative, and $LastOf(\mathsf{p}) \subseteq \Theta$ then p is in Θ.

A set of points $F \subseteq P$ is a *single step extension* of a configuration Θ by label a (denoted $\Theta \overset{a}{\rightarrow} \Theta \cup F$) iff **E1** $\Theta \cup F$ is a configuration of $\mathcal{S}$; **E2** There exists at most one concrete point in F **E3** $(\nexists \mathsf{p}, \mathsf{q} \in F)(\mathsf{p} < \mathsf{q})$; **E4** $(\forall \mathsf{p}, \mathsf{q} \in F)(0 \vDash \delta(\mathsf{p}, \mathsf{q}))$ **E5** $(\forall \mathsf{p} \in F)(\ell(\mathsf{p}) = \emptyset \vee a \in \ell(\mathsf{p}))$; and **E6** if F' is a proper non-empty subset of F then $\Theta \cup F'$ is not a con-

figuration. We say that $\Theta \xrightarrow{\lambda} \Theta \cup F$ iff conditions **E1,E2, E3, E4** and **E6** hold, while $(\forall \mathsf{p} \in F)(\ell(\mathsf{p}) = \emptyset)$.

Given a configuration Θ, an event restriction between a pair of points $\langle \mathsf{p}, \mathsf{q} \rangle$ is said to be *active* in Θ iff p is in Θ and q is not in Θ (by definition, a restriction $\langle \mathbf{0}, \mathsf{p} \rangle$ is active if $\mathsf{p} \notin \Theta$, and $\langle \mathsf{p}, \infty \rangle$ is active only if $\mathsf{p} \in \Theta$). We call $\Gamma(\Theta) \stackrel{def}{=} \bigcup \gamma(\mathsf{p}, \mathsf{q})$ for all active restrictions $\langle \mathsf{p}, \mathsf{q} \rangle$ in Θ. Furthermore, an event restriction $\langle \mathsf{p}, \mathsf{q} \rangle$ is *strictly active* in Θ wrt a set of points F iff it is active and $\mathsf{q} \notin F$. We call $\Gamma_{\triangleright F}(\Theta) \stackrel{def}{=} \bigcup \gamma(\mathsf{p}, \mathsf{q})$ for all strictly active restrictions $\langle \mathsf{p}, \mathsf{q} \rangle$ in Θ, wrt the set of points F.

Given a time restriction φ and a clock x, we define $\psi_x(\varphi)$ as the clock constraint over x such that for every non-negative real number t, $\psi_x(\varphi)[x \backslash t]$ is true iff $t \vDash \varphi$. For example, $\psi_x((a, b])$ is the constraint $a < x \wedge x \leq b$.

Given a scenario $\mathcal{S}$ and $F \subseteq P$, we define the set $R_F \stackrel{def}{=} \{x_\mathsf{p} / \mathsf{p} \in F \wedge \exists \mathsf{q}.\delta(\mathsf{p}, \mathsf{q}) \neq [\mathbf{0}, \infty]\}$. Given a configuration Θ and an extension F, we define $\psi_\Theta^F \stackrel{def}{=} \bigwedge_{\mathsf{q} \in \Theta \uplus \{\mathbf{0}\},\ \mathsf{p} \in F} \psi_{x_\mathsf{q}}(\delta(\mathsf{q}, \mathsf{p}))$.

Definition 7 (Tableau construction). The *tableau* $\mathcal{T}_\mathcal{S}$ for VTS scenario $\mathcal{S}$ is a timed automaton $\langle L, X, \Sigma, E, I, \mathsf{l}_0 \rangle$ such that: **T1** $L = \{\Theta / \Theta$ is a configuration of $\mathcal{S}\} \uplus \{\mathsf{s}_{\mathsf{trap}}\}$, we call $\mathsf{s}_{\mathsf{accept}}$ the configuration P (i.e. the configuration with all the points in the scenario); **T2** $X = \{x_\mathsf{p} / \mathsf{p} \in P \uplus \{\mathbf{0}\}\}$;

T3 $E = \{\langle \Theta, a, \psi_\Theta^F, R_F, \Theta' \rangle / \Theta \xrightarrow{a} \Theta' \wedge a \notin \Gamma_{\triangleright F}(\Theta)\}$
$\cup \{\langle \Theta, \lambda, \psi_\Theta^F, R_F, \Theta' \rangle / \Theta \xrightarrow{\lambda} \Theta'\}$
$\cup \{\langle \Theta, a, true, \emptyset, \Theta \rangle / a \in \Sigma \wedge a \notin \Gamma(\Theta)\}$
$\cup \{\langle \Theta, a, true, \emptyset, \mathsf{s}_{\mathsf{trap}} \rangle / a \in \Gamma(\Theta)\}$
$\cup \{\langle \mathsf{s}_{\mathsf{trap}}, a, true, \emptyset, \mathsf{s}_{\mathsf{trap}} \rangle / a \in \Sigma\}$ where $\Theta' = \Theta \cup F$;
T4 $(\forall \mathsf{l} \in L)(I(\mathsf{l}) \equiv true)$; **T5** l_0 is the empty configuration.

Fig. 7(b) shows the resulting tableau for the VTS scenario in Fig. 7(a).

The central result presented in [**?**] states that a timed automaton (modeling the real-time system under analysis) comprises a run that matches a scenario iff that automaton can steer the tableau for the scenario to a location that stands for "all points matched". That verification can be performed by off-the-shelf timed modelchekers.

Theorem 4.1 (Model checking VTS). *Given a Timed Automata $\mathcal{A}$ and a scenario $\mathcal{S}$, with $\Sigma \subseteq label(\mathcal{A})$, $\mathcal{P}(accept) = locs(\mathcal{A}) \times \{\mathsf{s}_{\mathsf{accept}}\}$ and $\mathcal{P}(init) = \{(init(\mathcal{A}), init(\mathcal{T}_\mathcal{S}))\}$.*

$$\mathcal{A} \vDash \mathcal{S} \quad \textit{iff} \quad \mathcal{A} || \mathcal{T}_\mathcal{S} \vDash init \Rightarrow \exists\Diamond(\exists\Box\ accept)$$

To prove this result we first show that if we fix any non-trap node n as an acceptance node, then the tableau accepts all traces matching the configuration associated to n. Thus, since the $accept$ proposition is associated to P (the whole set of points), the satisfaction of the TCTL [2] property $init \Rightarrow \exists\Diamond\ accept$ ("$accept$ is reachable from $init$") would mean that we can match the scenario. However, we have to check whether we can remain in that acceptance node without receiving any forbidden events (i.e. events that should not occur until ∞), that is $init \Rightarrow \exists\Diamond(\exists\Box\ accept)$ ("It is possible to evolve from $init$ to $accept$ and remain there forever"). Verification engineers need not be aware of the existence of the temporal logic TCTL, since the verification goal is automatically generated by our tool when translating a scenario.

5. VTS Tool and Case Study

The VTS tool consists of a GUI and a translation component. The front-end is implemented as a Microsoft Visio® add-in to visually edit scenarios and save them as XML documents. These are processed by the translator, which performs the tableau procedure to obtain observers suitable for Kronos [10] and Uppaal [8] tools.

5.1. Remote Sensing

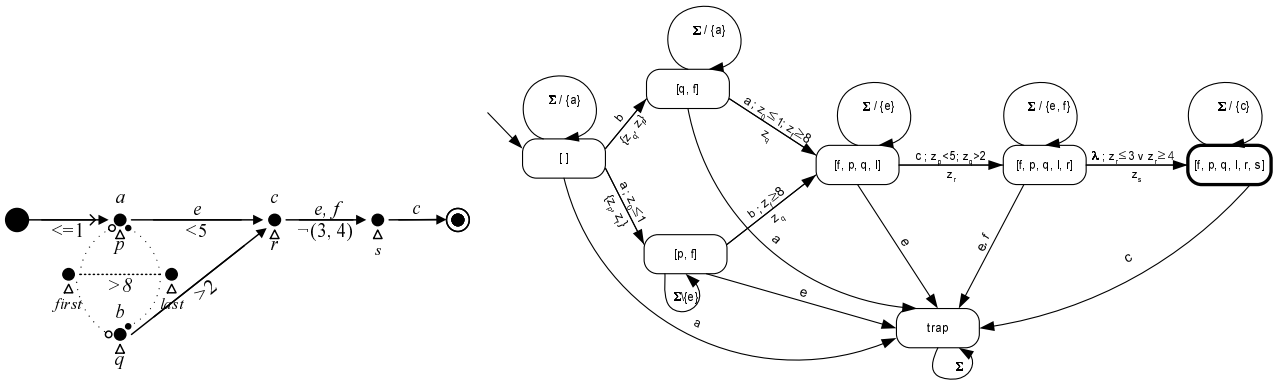
The Remote Sensing case study is a system basically consisting of a central component and two remote sensors. Sensors periodically sample a set of environmental variables and store the values in shared memory. When the central component needs them, it broadcasts a signal to the sensors with a minimal inter-arrival time ($RqstSent$ event). Each sensor runs a thread devoted to handle this message by reading the last stored value from shared memory and sending it back to the central component. The latter pairs the readings so that another process can use that piece of information to perform certain action on some actuators.

The model for this example was generated using the technique introduced in [11] to translate real-time system designs based on fixed-priority scheduling into timed automata. The idea underlying the translation is the use of fixed-priority scheduling theory to build a conservative (and practically analyzable) model of behavior which can be checked by model checkers such as Kronos and Uppaal.

In this case the model comprises 12 timed automata (one for each design component, featuring 12 clocks) interacting somewhat asynchronously.

For previous model checking techniques engineers needed to express requirements as a special timed automaton (the observer). The introduction of VTS greatly improves the usability of the approach through its user-friendly notation for expressing anti-scenarios.

A natural requirement for these system is synthesized in the anti-scenario of Fig. 8 where a request to collect a pair of data items from the central components is not fully answered in less than D time units.



(a) Scenario

(b) Tableau

Figure 7. VTS scenario and corresponding tableau

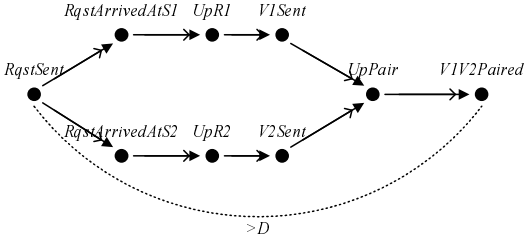


Figure 8. Remote Sensing bounded resp.

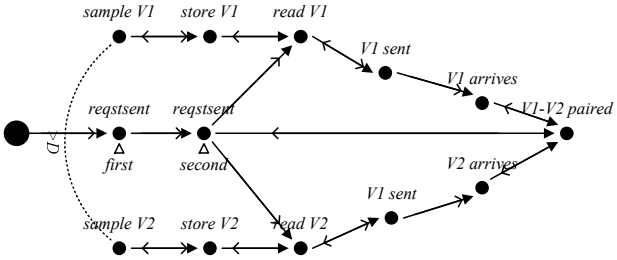


Figure 9. Remote Sensing correlation

A correlation requirement is informally stated as "paired values should have been sampled in not-too-distant time instants". Fig. 9 illustrates how to compactly formalize the idea of correlation in VTS, while also showing an advance use of scenarios available to verification engineers for avoiding state space explosion. A couple of $RqstSent$ events were added to the original scenario by the verification engineer in order to check correlation over the second request only, since she believes that observing the evolution of the second request should suffice to verify this property.

Table 1 summarizes the translation and verification results for some of the scenarios using a Windows-based version of Uppaal tool on a Pentium IV 1.6 Mhz PC with 512 MB. It includes the size of the observers, as well as verification options and optimization techniques used[4].

VTS scenario: Bounded Response Tableau size: #locs=29 and #trans=520				
Value for D	Transl. time	Verification		
		Match?	time	options
380	0.8 sec.	Yes	5.6 secs.	ObsSlice + -Was
390	0.8 sec.	No	5.8 secs.	ObsSlice + -Was
VTS scenario: Correlation Tableau size: #locs=57 and #trans=959				
Value for D	Transl. time	Verification		
		Match?	time	options
185	1.1 sec.	Yes	147 secs.	ObsSlice + -WasdC
195	1.1 sec.	No	6 secs.	-Was -A

Table 1. Verification results

6. Related work

Though other notations were proposed for negative scenarios (e.g., [28, 26, 18, 16]) VTS combines some existing and novel features in a way that makes it a powerful

4 We use $ObsSlice$, a slicer for timed models based on the topology of the observer (see [12]) that reduces the amount of work to be done by the model checker (often rendering the analysis feasible).

tool for the verification of real time applications: a simple formal syntax based on partial orders, an underlying descriptive semantics based on the idea of matching, an operative semantics based on Timed Automata –probably the most thoroughly studied and supported timed formalism for both dense and discrete time. Besides, events are neither restricted to be communication-events, nor to be consecutive; and the negation of event occurrence in intervals permits identifying next or previous events of a given sort. The concept of representative events and instants are novel features too, as are the begin and end points (specially suited for negating progress). Additionally, VTS deals with real-time constraints in a dense-time domain without using timers.

The TimeEdit tool ([26]) is also meant to simplify the task of expressing never claims (for the Spin or FeaVer modelchecking tools). Something similar could be said about an event-based variant of GIL (Graphical Interval Logic) presented in [6]. GIL is a logic featuring propositional operators, modalities and nesting. Both TimeEdit and GIL are based on timeline diagrams and do not feature partial ordering of events. While TimeEdit does not support timing constraints, GIL can constrain duration of intervals which are bounded by contiguous events. More precisely, in [6] it is remarked that they are able to construct and determinize Hybrid Automata oracles by hand when all intervals to be timed are specified using a bidirectional search operator. We believe that, while being more expressive (for example, by allowing nesting), GIL formulas are potentially more complex and difficult to understand than our event-based notation.

Timing Diagrams are a known notation in the context of hardware design. Regular Timing Diagrams ([4]) (and extensions) are a good exponent of this class of visual notation suited for asynchronous systems. They are basically a set of independent waveforms on signals where events (change of values) can be causally constrained –and time-constrained by a number of ticks of a given clock– or defined as concurrent. In this setting, RTDs induce deterministic matching and this implies sacrificing expressive power in favor of an efficient modelchecking.

Perhaps the most remarkable and widespread example of a visual formalism for scenario-based specifications are Message Sequence Charts and UML versions (ITU Z.120). Unlike our approach, MSCs are not used to describe negative scenarios but to describe some or all possible behaviors of a protocol in a systemic view. This usually results in more cumbersome and error-prone diagrams, as the complement of a very simple anti-scenario can consist of many possible and complex (positive) scenarios.

Uchitel et al. ([28]) introduced a language to describe negative scenarios based on MSCs to elicit requirements. Something similar was done, with a different purpose, in LSCs of Harel et. al. ([18]). Though we share the idea of working with a partial order to describe scenarios, our approach differs from both works in several aspects. On the one hand, we conceive our language as a means to express properties to be checked against a model or implementation under analysis; we neither focus on creating an executable specification language, nor on eliciting requirements. On the other hand, we are neither constrained to describe message interchange, nor to define the instances that perform events. Moreover, event visibility is treated quite differently in our case: two contiguous events in an scenario do not need to match contiguous events of the same kind in the run, except when explicitly noted. Also, we do not resort to after/until notation or triggering conditions to express when a matching is valid. And the first/last feature of VTS is not present in the cited approaches. Additionally, our tools deal with real-time constraints in a dense-time domain without using timers as is the case of [18]. Unlike LSCs, our semantic definitions are given in a declarative (denotational) way, and then the tableau procedure is presented, also declaratively, to show the algorithmic solution to the checking problem. Finally, our notation can be used to express some class of liveness constraints, as in the case of a pattern stating that a given stimulus is never answered.

Firley et.al. ([16]) mix UML sequence diagrams and modelchecking. Diagrams are translated into observers in the Uppaal formalism and Uppaal models are instrumented to be composed with the observer. Again, no semantics is given for the checking problem beyond the algorithmic translation itself and a set of proof obligations in branching logics. They follow the "contiguous semantics": the sequence specifies exactly the message exchange that may occur –thus featuring a sort of loop construct– except for the starting event in the optional interpretation. The presented construction only supports totally ordered sets of events. UML sequence diagrams do not support the concept of instants or liveness constraints.

To further compare VTS's expressive power to that of other languages, consider Fig. 3, which shows a case that is not (at least, naturally) expressible using any of the notations reported so far. The use of negative constraints on the occurrence of events (e.g. previous-match) together with first- and last-representatives for grouping activities (sets of events) is unique to VTS and can be used to compactly represent complex patterns such as this one.

As mentioned earlier, the VTS formalism can also serve as a means to analyze collected traces. It is worth mentioning that the need to describe event-scenarios is shared by at least three interconnected lines of research: monitoring and debugging of distributed systems (e.g. [7, 27, 13, 23]), event correlation detection (e.g. [29, 25]), and run-time verification (e.g. [19, 22, 14]). Works such as [7, 29, 23] use regular-expression-like syntax to denote event patterns.

While those approaches also feature the negation of events, precedence and timing constraints, we believe that visual formalisms like VTS are better suited for expressing requirements. Moreover, in the approaches based on regular expressions, some matching situations (such as constraining a pair of events which can be identified by matching other common related events) becomes at least a cumbersome task (see for instance Fig. 1(b)).

Some other works on debugging and visualization of distributed behavior (e.g. [27, 13]), are based on querying databases of events and messages. In most cases, instead of rooting the ideas on traces, causality graphs synthesized from the execution of a distributed system are queried. In these cases, precedence can only be stated on causality-related events and this limitation becomes a major obstacle when trying to express a real-time constraint between time-consecutive but causally-unrelated events. Again, we believe that relational queries are not really well suited to express requirements in an early stage of design. Event correlation notations are usually based on operational and expressive formalisms such as transducers (e.g. [25]). Their design goal is to build a scripting language for a middleware service in publish/subscribe architectures, and not necessarily to define a suitable language for expressing requirements. On the other hand, run-time verification literature usually bases the requirements on some sort of finite-trace linear-time temporal logic (e.g. [19, 22, 14]).

7. Conclusions and Ongoing Work

We presented VTS, a visual formalism to express and model check complex event-based requirements for Real-Time Systems. We can summarize the main achievements in terms of a set of design goals:

Simple Features: We base the notation on natural notions such as events and constraints. We do not resort to timers. Future expressive power extensions and syntactic sugar additions will be driven by pragmatic considerations on different application domains, always prioritizing not to obscure language semantics.

Simple and Precise Semantics: Formal semantics is a must for promising non-trivial automatic support. Moreover, the semantics of our formalism is based on a descriptive notion of matching, instead of an indirect and cumbersome translation into some other operational formalism.

Ease of Use: We identify two key factors for successful adoption of such a tool by the industry: notation should be visual and it should not require the entire description of behavior. Besides, the underlying notion of partial order allows engineers to compactly express concurrent scenarios.

Model Checking Tool Support: Scenarios are automatically translated into timed automata for model checking purposes. TA are probably the most thoroughly studied and supported timed formalism for both dense and discrete time.

Extensibility: We have already identified potential extensions of the language, like modularity and modalities, which seem to fit as natural generalizations.

Portability: Besides serving as a front-end for modelchecking, we believe that VTS can be tailored to other Verification Engineering applications, such as testing of control-intensive systems and run-time verification.

Our research group agenda includes the addition of syntactic sugar such as providing modularity, parametrization and extended event-name matching capabilities as well as enhancements of expressive power without sacrificing decidability and ease of use. In cases where the model has some built-in identification mechanism to allow traceability (for example, pairing a stimulus with its response), this extensions should allow engineers to describe a generic scenario without the need to explicitly depict chains of events, which would couple the requirement to a specific proposed solution.

In order to provide more evidence supporting the practical relevance of the approach we plan to study how some patterns ([15]) translate into scenarios to help the construction of requirements (some examples can be found in [1]). We also intend to compare the expressive power of VTS against finite-trace linear-time logics ([20]) identifying equally expressive fragments.

We are currently defining new modalities for scenarios to exploit branching nature of time. This will enable naturally expressing patterns such as: “when a given scenario is matched then one specific scenario continuation happens at least in one (all) future behaviors”. This kind of modality is not featured by any other current scenario-based notations grounded on linear-time semantics.

The fact that the notation is based on partial orders opens the door to the use of future modelchecking technology that could eventually avoid the explicit or symbolic traversal of the interleaved model. Moreover, scenarios may serve as a visual explanation of what may go wrong in a system execution generated by the model checker as a counterexample.

We also conjecture than VTS scenarios are amenable to serve as verification objectives in a run-time verification tool (which are usually expressed using fine-trace linear-time logics ([20])) or as test purpose to guide the extraction of test cases from formal models ([21]).

References

[1] A. Alfonso. Un lenguaje visual para la especificación y verificación automática de requerimientos de tiempo real complejos. Master's thesis, FCEyN. Universidad de Buenos Aires, 2003.

[2] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] N. Amla, E. A. Emerson, and K. S. Namjoshi. Efficient decompositional model checking for regular timing diagrams. In *Porc. of Intl. Conf. on Correct Hardware Design and Verification Methods*, volume 1703 of *LNCS*, pages 67–81. Springer Verlag, 1999.

[5] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. TIMES — A tool for modelling and implementation of embedded systems. In *Proc. of the 8th Conf. TACAS '02*, volume 2280 of *LNCS*, pages 460–464. Springer Verlag, 2002.

[6] G. S. Avrunin, J. C. Corbett, and L. K. Dillon. Analyzing partially-implemented real-time systems. In *Proc. of the 18th ACM/IEEE Conf. ICSE '97*, pages 228–238. IEEE, 1997.

[7] P. C. Bates. Debugging heterogeneous distributed systems using event-based models of behaviour. *ACM Transactions on Computer Systems*, 13(1):1–31, 1995.

[8] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *Proc. of the Intl. Conf. on Hybrid Systems*, pages 232–243. Springer Verlag, 1995.

[9] V. Bertin, E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venier, D. Weil, and S. Yovine. Taxys = esterel + kronos - a tool for verifying real-time properties of embedded systems. In *Proc. of Conf. CDC '01*. IEEE Control Systems Society, 2001.

[10] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Proc. of the 10th Intl. Conf. CAV '98*, volume 1427 of *LNCS*, pages 546–550. Springer Verlag, 1998.

[11] V. A. Braberman and M. Felder. Verification of real-time designs: combining scheduling theory with automatic formal verification. In *Proc. of the 7th ACM/SIGSOFT Intl. Conf. ESEC/FSE '99*, pages 494–510. Springer Verlag, 1999.

[12] V. A. Braberman, D. Garbervetsky, and A. Olivero. Improving the verification of timed systems using influence information. In *Proc. of the 8th Intl. Conf. TACAS '02*, volume 2280 of *LNCS*, pages 21–36. Springer Verlag, 2002.

[13] M. Consens and A. Mendelzon. Hy+: a Hygraph-based query and visualization system. In *Proc. of the ACM Intl. Conf. SIGMOD '93*, pages 511–516, 1993.

[14] D. Drusinsky. The temporal rover and the ATG rover. In *Proc. of the 7th Intl. Workshop SPIN*, volume 1885 of *LNCS*, pages 323–330. Springer Verlag, 2000.

[15] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. of the 21th ACM/IEEE ICSE '99*, pages 411–420. ACM Press, 1999.

[16] T. Firley, M. Huhn, K. Diethers, T. Gehrke, and U. Goltz. Timed sequence diagrams and tool-based analysis A case study. In *Proc. of the 2nd Intl. Conf. UML '99*, volume 1723 of *LNCS*, pages 645–660. Springer Verlag, October 1999.

[17] D. Giannakopoulou and J. Magee. Fluent model checking for event-based systems. In *Proc. of the ACM/SIGSOFT Intl. Conf. ESEC/FSE 2003*, pages 257–266. ACM, September 2003.

[18] D. Harel and R. Marelly. Playing with time: On the specification and execution of time-enriched lscs. In *Proc. of the 10th IEEE/ACM Intl. Symp. MASCOTS '02*, pages 193–202. IEEE Computer Society, 2002.

[19] K. Havelund and G. Rosu. Monitoring java programs with java pathexplorer. In *Proc. of the 1st Intl. Workshop RV '01*, volume 55. ENTCS, Elsevier, 2001.

[20] K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *Proc. of the 8th Intl. Conf. TACAS '02*, volume 2280 of *LNCS*, pages 342–356. Springer Verlag, 2002.

[21] J. C. Fernandez, C. Jard, T. Jeron, and G. Viho. Using on-the-fly verification techniques for the generation of test suites. In *Proc. of 8th Intl. Conf. CAV '96*, volume 1102 of *LNCS*, pages 348–359. Springer Verlag, 1996.

[22] I. Lee, S. Kannan, M. Kim, O. Sokolsky, and M. Viswanathan. Runtime assurance based on formal specifications. In *Proc. of the IEEE Intl. Conf. PDPTA '99*, 1999.

[23] M. Mansouri-Samani and M. Sloman. Gem: A generalized event monitoring language for distributed systems. *Distributed Systems Engineering Journal*, 4(2):96–108, 1997.

[24] N. F. Maxemchuk and D. H. Shur. An Internet multicast system for the stock market. *ACM Transactions on Computer Systems*, 19(3):384–412, 2001.

[25] C. Sanchez, S. Sankaranarayanan, H. Sipma, T. Zhang, D. Dill, and Z. Manna. Event correlation: Language and semantics. In *Proc. of the 3rd Intl. Conf. EMSOFT '03*, volume 2855 of *LNCS*. Springer Verlag, 2003.

[26] M. H. Smith, G. J. Holzmann, and K. Etessami. Events and constraints: A graphical editor for capturing logic requirements of programs. In *Proc. of the 5th IEEE Intl. Symp. RE '01*, pages 14–22, 2001.

[27] R. Snodgrass. A relational approach to monitoring complex system. *ACM Transactions on Computer Systems*, 6(2):157–196, 1988.

[28] S. Uchitel, J. Kramer, and J. Magee. Negative scenarios for implied scenario elicitation. In *Proc. of the 10th ACM/SIGSOFT Intl. Conf. FSE '02*, pages 109–118. ACM Press, 2002.

[29] D. Zhu and A. S. Sethi. Sel, a new event pattern specification language for event correlation. In *Proc. of the IEEE Intl. Conf. ICCCN '01*, pages 586–589, 2001.