Journal of Universal Computer Science, vol. 19, no. 15 (2013), 2198-2206
submitted: 29/1/13, accepted: 30/8/13, appeared: 1/9/13 © J.UCS

Behavioral and Temporal Rule Checking for Gaussian Random Process – a Kalman Filter Example[1]

Doron Drusinsky
(Naval Postgraduate School, Monterey, CA, USA, and Time Rover, Inc., USA
ddrusins@nps.edu)

Abstract: This paper describes a behavioral and temporal pattern detection technique for state-space systems whose state is a random variable such as the state estimated using a Kalman filter. Our novel behavioral and temporal pattern detection technique uses diagrammatic, intuitive, yet formal specifications based on a dialect of the UML of the kind used to monitor or formally verify the correctness of deterministic systems. Combining these formal specifications with a special code generator, extends the deterministic pattern detection technique to the domain of stochastic processes.

We demonstrate the technique using a Ballistic trajectory Kalman filter tracking example in which a pattern-rule of interest is *not* flagged when observing the sequence of mean track position values but is flagged with a reasonable probability using the proposed technique.

Keywords: Random process, Kalman Filter, UML, statecharts, monitoring, patterns
Categories: F.4.1, F.1.1, D.2.4

1 Introduction

Run-time Verification (RV) of formal specification assertions is a class of methods for monitoring the sequencing and temporal behavior of an underlying application and comparing it to the correct behavior as specified by a formal specification pattern. Some published RV tools and techniques are: the TemporalRover and DBRover [Drusinsky 2000], PaX [Haveland and Rosu 2004] and RT-Mac [Sammapun, Lee and Sokolsky 2005], all of which use extensions and variants of Propositional Linear-time Temporal Logic (PLTL) as the specification language of choice, and the StateRover [StateRover] that uses deterministic and non-deterministic statechart diagrams as its specification language. In [Drusinsky 2011], Drusinsky describes the application of RV using statechart assertions (based on Harel statecharts [Harel 1987]) to the verification of DoD and NASA applications, and to those of the Brazilian Space agency.

The Unified Modeling Language (UML) consists of a set of standardized (ISO/IEC 19501:2005), general-purpose modeling languages mostly used in the field of software engineering. It includes graphic notation techniques to display visual models of object-oriented software-intensive systems. Since its inception in the early

[1] This research was funded by a grant from the U.S. Defense Threat Reduction Agency (DTRA). The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

1990's, the UML has been used mostly for documentation and informal specification. Consequently, it has been widely viewed as an informal language. In spite of its mostly informal usage, some of the visual languages of the UML can be used in a rigorous and formal way. Specifically in this paper, we are interested in formal specifications using UML state diagrams, known in the literature as UML state machines (UML-SM). UML-SM's are based on Finite State Machines (FSM's), which have been used for several decades for formal specification and for design and synthesis of software and hardware. In this paper we use formal specifications based on UML-SM's.

In [Drusinsky, 2012] the author described a process for monitoring and pattern detection of sequences of hidden financial system states using HMM's connected to UML-SM formal specification assertions. This paper is a counter-part technique for the monitoring and pattern detection of continuous stochastic-process using UML-SM formal specification assertions.

The Kalman Filter (KF), named for Rudolf (Rudy) E. Kálmán, is an algorithm that uses a series of measurements observed over time, containing noise (random variations) and other inaccuracies, combined with an a-priori model estimate, and produces estimates of unknown (hidden) variables that tend to be more precise than those based on a measurement or a-priori estimation alone. The KF operates recursively on streams of noisy input data to produce a statistically optimal estimate of the underlying system state [Kalman,1960]. KF's are used in many consumer and military applications such as and GPS and missile tracking.

A straightforward approach for performing deterministic monitoring or pattern matching of the estimated KF state is to monitor its estimated mean state values. With this approach, an existing RV tool is used to monitor a stream of mean KF state values for a pattern of interest. The drawback of this approach is that for certain applications, such as defense and security applications, merely detecting whether a pattern is detected with a probability greater than 0.5 is insufficient. For example, a missile defense system would likely attempt to kill a threat that has a probability of 0.45 of hitting a target. In contrast, the technique demonstrated in this paper performs probabilistic monitoring of a stream of estimated KF mean and covariance values.

Hidden Markov Models (HMM's) are a well-developed technology for classification of multivariate data that have been used extensively in speech recognition, handwriting recognition and even sign recognition [Rabiner 1989]. They are usually used for classification, such as classifying a spoken word as "red", "blue", or "green". An HMM is defined by its parameters, chief of which are the number of states and transition probabilities, which are often computed using a learning data-set.

In contrast with HMM's, our suggested UML-SM formal specification assertions do not contain probabilities. In fact, as discussed in section 2, the same UML-SM specifications are used for monitoring of both deterministic and stochastic data streams. A special code generator is used to direct the monitor to a specific, deterministic or stochastic input stream.

The technique suggested in this paper is positioned as a hybrid pattern detection technique that combines patterns written by humans with statistical algorithms – such as KF's. In other words, it is positioned as a hybrid between formal specification and run-time verification techniques (e.g., [Drusinsky (2006)], [Drusinsky (2011)], [Drusinsky 2000]) and widely used statistical estimation algorithms.

The rest of the paper is organized as follows. Section 2 provides an overview of behavioral pattern detection using deterministic UML-SM specifications. Section 3 provides an overview of KF's. Finally, section 4 describes our proposed pattern detection architecture and a process that uses a combination of hidden and visible data, using a KF connected to a behavioral pattern detection monitor.

2 Behavioral Pattern Detection using Deterministic UML-SM Specifications– an Overview

Consider the following natural language (NL) patterns for a Ballistic trajectory tracking system; the NL pattern is specified as being flagged when a scenario conforms to the pattern:

R1. *Flag a ballistic object if once M meters from the an asset (e.g., the Pentagon), threatens the target and continues to do so for T additional seconds, where "threatens the asset" means its estimated hit location, as calculated using the a-priori state equations, is within a perimeter of less than ACCURACY meters around the assets absolute location.*

During the flight of the ballistic object, both its position and the eventual hit location are unknown. They are therefore estimated; the present position is estimated using a Kalman filter (i.e., using both a-priori and a-posteriori information), whereas the hit location is estimated using a-priori information. The reason for using only the a-priori state-equations for hit location estimation in requirement R1 is that a-posteriori information requires measurements of the ballistic object eventual hit location, measurements that do not exist when the object is M meters away from the asset.

[Fig. 1] depicts a UML-SM for R1. As described in [Drusinsky, 2006], a UML-SM is a classical state-diagram which is potentially augmented with hierarchy, flowcharting capabilities, a Java action language, and a built in Boolean flag named *bFlag* whose default value is *false*, with a *true* value indicating that the pattern has been flagged (e.g., per pattern R1, it flags when the input scenario conforms to R1); see [Drusinsky, 2006, Drusinsky, 2012] for further details. In [Fig. 1], whenever the missile detection system calculates a new distance for the ballistic object being tracked, a *newDistance* event is emitted; it triggers the UML-SM to possibly change states. It will do so if its present-state is *Init* or *DistM*, and if the corresponding transition guard evaluates to *true*. The transition guards are *getDist()<=M* and *Math.abs(getExtrapolatedHitLocation()-*$Target_{xpos}$*)>ACCURACY,* respectively. The method *getDist* returns the objects a-priori estimation of the distance from the defended asset. The method *getExtrapolatedHitLocation* returns the *x* position of the estimated hit position, and $Target_{xpos}$ is the *x* distance of the target from the origin. The *timer.restart* and *timeoutFire* methods relate to the initialization and expiration of a T (using T=5 in [Fig. 1]) second timer.

Deterministic pattern matching for requirement R1 is performed by comparing a sequence of ballistic trajectory estimates to the behavior of the pattern set. The StateRover tool does so using a two step process. First, the stream, or sequence, of ballistic trajectory estimates (x state mean values) is converted into an equivalent JUnit test [JUnit], and the pattern is code-generated into an equivalent Java class

(details about this code generator are available in [Drusinsky (2006)]). Next comes an RV step where a log file containing the system's behavior is automatically converted into a JUnit test case; the JUnit test is executed, thereby checking that the systems behavior, as captured by the log file, conforms to the pattern.

Section 4 describes the proposed technique for probabilistic pattern matching for R1.

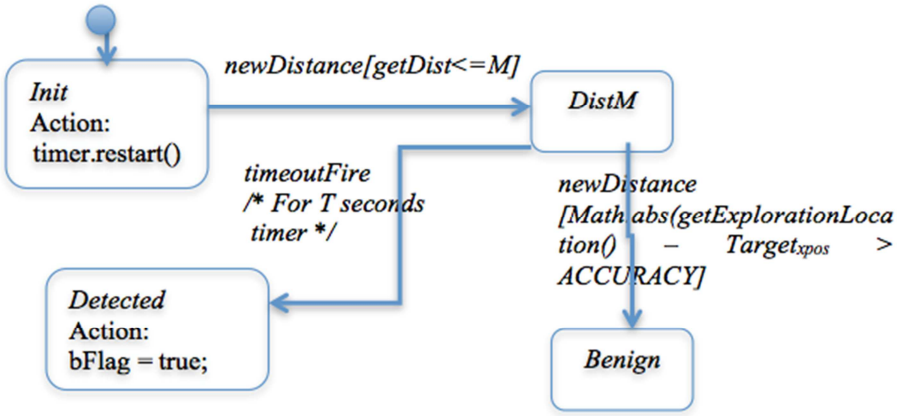


Figure 1: A UML-SM for requirement R1

3 Kalman Filters

A Kalman filter is an optimal estimator that infers parameters of interest from inaccurate and uncertain measurements (the a-posteriori knowledge) and a mathematical model of the system (the a-priori knowledge). When all noise is Gaussian, the Kalman filter minimizes the mean square error of the estimated state parameters (mean and covariance).

A typical state-space representation of a dynamic system is represented by a state equation and a real valued measurement equation:

$$x_k = A\,x_{k-1} + Bu_k + w_{k-1}$$
$$z_k = H\,x_k + v_k$$

where k is the time step, x_k is a n_k dimensional real valued state vector, u_k is a n_u dimensional known input vector, w_k is (unknown) zero mean, normally distributed white process noise with covariance $E[w_k\ {w_k}^T] = Q$, v_k is the zero mean, Normally distributed white process measurement noise with covariance R, and z_k is a linear combination, determined by matrix H, of the signal value and the measurement noise.

The square matrix A is the a-priori knowledge relating the state at step k-1 to the state at present step k in the absence of process or measurement noise. The matrix B relates the input u to the state x.

As its equations suggest, a Kalman filter is a recursive filter with a Markov property where the state at step k-1 depends on the state at step k but on no prior step. At every step, it emits estimation parameters in the form of a mean state and a covariance matrix P for the estimated state.

The Kalman Filter operates in two phases per step: the time update (prediction) phase and the measurement (correction) update phase.

The time update phase consists of two equations:

$$\overline{x_k} = A\,x_{k-1} + Bu_k$$
$$\overline{P_k} = A\,P_{k-1}\,A^T + Q$$

The measurement update consists of three equations:

$$K_k = \overline{P_k}\,H^T\,(H\,\overline{P_k}\,H^T + R)^{-1}$$
$$x_k = \overline{x_k} + K_k\,(z_k - H\,\overline{x_k})$$
$$P_k = (1 - K_k\,H)\,\overline{P_k}$$

The literature contains a wide variety of Kalman filter variants, such as the Extended Kalman Filter (EKF) [Anderson, 1979] and the Unscented Kalman filter (UKF) [Julier and Ullmann, 1997], which extend the original KF to non linear system estimation.

The example used in this paper is a ballistic trajectory tracking system, such as in a ballistic defense system. For simplicity, we used a two dimensional system representation where the state consists of x,y position and velocity (x being distance from origin and y being altitude). The well-known dynamic equations are:

$$x_{pos} = \overline{x_{pos}} + x_{speed}\,t$$
$$y_{pos} = \overline{y_{pos}} + y_{speed}\,t$$
$$y_{speed} = \overline{y_{speed}} - g\,t$$

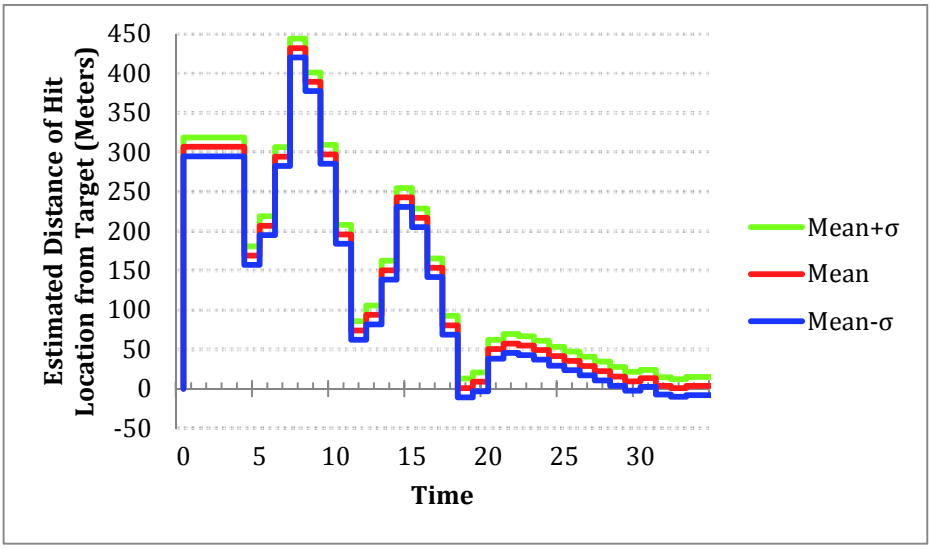
where t is the time increment between steps, $\overline{x_{pos}}$, $\overline{y_{pos}}$ and x_{pos}, y_{pos} are the position vectors before and after the time increment, respectively, and likewise for the speed vectors $\overline{y_{speed}}$, $\overline{y_{speed}}$, x_{speed}, and y_{speed}. The system initializes with the position vector (0,0), and an initial velocity V at angle θ; g is the gravitational constant.

The goal of the system is to give sufficient notice before the ballistic object hits a designated target, hence the UML-SM assertion of [Fig. 1] (which represents requirement R1), which informs the system that a ballistic object is persistently threatening an asset.

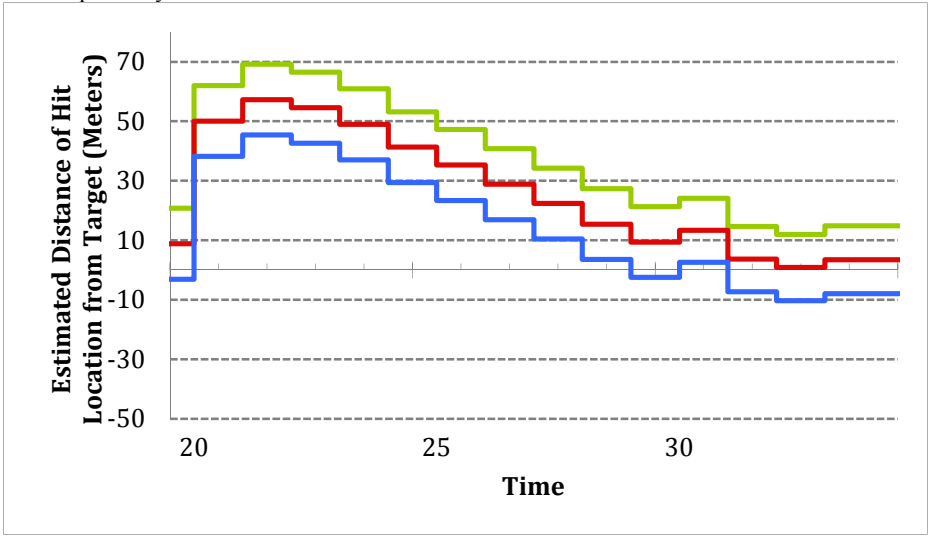
The purpose of the Kalman filter is to provide an optimal estimation of the track of the Ballistic object for the purpose of interception. A simple implementation of requirement R1 is to construct a deterministic monitor (as discussed in [Drusinsky, 2011]), for R1 that operates on the stream of mean state values generated by the KF. Assuming the state has a Gaussian distribution, deterministic monitor essentially flags objects whose probability of hitting an asset is greater than 0.5. However, depending on the sensitivity of an asset, it might be prudent to intercept objects with some other probability threshold.

The suggested monitoring technique enables the interception of a threat based on a safer probability threshold of, for example, 0.2. [Fig. 2a] depicts a plot of the estimated distance of the hit location from the target under a simulation that uses M=1707, T=5 seconds (M and T are defined in section 2, requirement R1), and a target location of $Target_{xpos}$=6707 meters. The middle line represents the mean distance, while the top and bottom lines represent the mean plus and minus standard deviation (σ), respectively. [Fig. 2b] is a portion of that plot starting at time t=20. It shows the ballistic object being less than M meters from the target at time t=22

seconds. The mean estimated distance of extrapolated hit location from the target is 48 meters at time *t*=23. Consequently, for ACCURACY=45 (ACCURACY is defined in section 2, requirement R1), the UML-SM of [Fig. 1] is *not* flagged for this ballistic object when using the above-mentioned deterministic approach. However, when using the probabilistic monitor discussed in section 4, the UML-SM is flagged with probability 0.3285, a probability high enough to justify interception.



a. Estimated distance of hit location from target. The middle line is the mean distance, while the top and bottom lines represent the mean plus and minus standard deviation, respectively.



b. Estimated distance of hit location from target during the last 15 seconds of flight.

Figure 2: Estimated distance of hit location from target

4 The Suggested Technique: Monitoring a Kalman Filter

The suggested pattern detection technique combines deterministic behavioral pattern detection techniques for the detection of behavioral patterns written by human experts, and recursively estimated KF parameters, namely the median and covariance of the estimated state. It does so using a loosely coupled approach, where two respective components (namely, the pattern recognizer and the KF) have little dependence, thereby easing their respective development and maintenance tasks.

To enable pattern detection of a KF with respect to R1 and its corresponding UML-SM of [Fig. 1], we apply a special code generator that generates a probabilistic implementation for the UML-SM, one that operates on the mean and covariance parameters of the KF; consequently, UML-SM pattern-matching consists of two modules: the KF module, and a module with the generated code for the UML-SM pattern - which uses visible information (such as the *newDistance* event of [Fig. 1] for example) as well as KF mean and covariance estimations on a cycle by cycle basis.

Standard UML notation labels UML-SM transitions with pairs: $event_t[condition_t]$, where $condition_t$ is optional. With this notation, a UML-SM transition from state S_1 to state S_2 that is labeled $event_t[condition_t]$ causes the UML-SM to change its state from state S_1 to state S_2 when event $event_t$ fires but only if $condition_t$ is *true*. A deterministic UML-SM implementation allows at most one transition to fire at any time *t*. In contrast, our probabilistic implementation allows more than one transition to fire at any time *t,* but with certain associated probabilities, as explained below. While in general we assume that given event $event_t$ and condition $condition_t$ either can be deterministic or estimated, for brevity reasons, we restrict our results to the cases where $event_t$ is deterministic; the case where it is estimated case is discussed in [Drusinsky, 2012].

KF estimations have an associated Gaussian probability distribution with mean vector μ, and covariance matrix *P*. Hence, the UML-SM pattern recognizer module operates on sequences of pairs of numbers provided by the KF module, in the form of $I = <\mu_1, P_1>, <\mu_2, P_2>,.., <\mu_T, P_T>$, where μ_i is the KF's estimated mean and P_i is the corresponding covariance matrix (with 0 variance if the KF output is deterministic).

The UML-SM implementation consists of a collection $\boldsymbol{C}$ of instances, or copies, of the UML-SM, called configurations. Each configuration executes as a standalone pattern and preserves its own present-state. Each configuration *Con* has a probability measure $Pr(Con)$, called the Configuration Probability Measure (CPM), that measures the probability the UML-SM is behaving as suggested by *Con*, i.e., that its present-state is *Con*'s present state.

Upon startup, $\boldsymbol{C}$ consists of a single configuration $Con_{default}$ whose present-state, denoted $PS(Con_{default})$, is the pattern's default state (e.g., state *Init* in [Fig. 1]), and having probability $Pr(Con_{default}) = 1$.

All configurations of $\boldsymbol{C}$ respond to a pair $< \mu_t, P_t>$ of *I*, by substituting the present configuration *Con* with two configurations: *Con*1 and *Con*2*,* whose present-state probabilities are calculated as follows. For each *Con*, all outgoing transitions are evaluated as follows. First, $Pr(condition_t)$, the probability of the transitions' condition, is calculated using the standard Gaussian cumulative density function (CDF), which is based on the KF estimated mean and covariance. For example, the condition $getDist() <= M$ of [Fig. 1] has, at time *t,* a probability,

$Pr(getDist() <= M)$, that is equal to CDF $F_x(M)$ of a Gaussian distribution with a mean of $Target_{xpos}$ - μ_t and covariance matrix P_t.

Subsequently:

- $Con1$ and $Con2$*'s* probabilities are calculated as: $Pr(Con1) = Pr(Con)Pr(condition_t)$, and $Pr(Con2) = Pr(Con)(1 - Pr(condition_t))$.
- Let $PS(Con)$ denote *Con*'s present-state. $PS(Con1)$ and $PS(Con2)$ are determined as in a deterministic UML-SM, assuming $condition_t = true$ and $condition_t = false$, respectively. In [Fig. 1] for example, when considering *Con=Init* and the transition *Init→DistM* fires, the two resulting *Con'*s are *DistM* and *Init*.

C configurations are routinely (i.e., every cycle *t*) managed as follows. All configurations *Con`* with the identical present-state values are merged into a single configuration $Con_{merged}$, using the sum of all $Pr(Con`)$ as $Pr(Con_{merged})$.

The UML-SM declares a Probability of Flagging *(POF)*, i.e., the probability its corresponding NL requirement has been flagged, on a cycle by cycle basis, being the sum of all $Pr(Con)$ for all configurations *Con* such that $PS(Con)$ is an state where *bFlag=true*.

5 Conclusion and Future Research

We have demonstrated a technique for performing probabilistic pattern detection of a Gaussian process. As in previous papers we demonstrated probabilistic pattern detection using UML-SM specifications combined with HMM's, this paper demonstrates the benefits of probabilistic monitoring of the distribution of a random variable (whether categorical, as in the HMM case [Drusinsky 2012], or continuous and Gaussian, and in this paper) over monitoring its mean, or mean plus/minus standard deviation values. This approach is particularly useful for safety-critical, or mission-critical systems in which the threshold for pattern violation is lower than many other types of systems.

We also plan on applying this technique to automatic pattern detection within large volumes of cyber data, in an effort to identify malicious or dangerous behavioral patterns.

We are currently building a special StateRover code-generator that generates weighted/probabilistic implementation code for UML-SM specifications.

Acknowledgements

This research was funded by a grant from the U.S. Defense Threat Reduction Agency (DTRA).

References

[Andersen, 1979] Anderson, B.D.O. and Moore, J.B. (1979). Optimal Filtering. Englewood Cliffs, New Jersey: Prentice–Hall.

[Drusinsky, 2000] Drusinsky, D.: "The Temporal Rover and the ATG Rover"; Proc. SPIN 2000 Workshop, LNCS 1885, Springer-Verlag, (2000), 323-329.

[Drusinsky, 2006] Drusinsky, D.: "Modeling and Verification Using UML Statecharts – A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking"; Elsevier (2006).

[Drusinsky, 2011] Drusinsky, D.: "Practical UML-based Specification, Validation, and Verification of Mission-critical Software"; DogEar Publishing (2011).

[Drusinsky, 2012] Drusinsky, D.: "Behavioral and Temporal Pattern Detection within Financial Data with Hidden Information", Journal of Universal Computer Science, Vol. 18, No. 14, pp. 1950-1966.

[Harel, 1987] Harel, D.: "Statecharts: A visual formalism for complex systems"; Science of Computer Programming, 8, 3 (1987), 231–274.

[JUnit] JUnit, http://www.junit.org

[Julier and Ullmann, 1997] Julier, S.J.; Uhlmann, J.K. (1997). "A new extension of the Kalman filter to nonlinear systems". Int. Symp. Aerospace/Defense Sensing, Simul. and Controls 3. Retrieved 2008-05-03.

[Kalman,1960] "A new approach to linear filtering and prediction problems". Journal of Basic Engineering 82 (1): 35–45. Retrieved 2008-05-03.

[Rabiner 1989] Rabiner, L.W.: "A Tutorial on Hidden Markov models and Selected Applications in Speach Recognition"; Proc. of the IEEE, **77**, 2 (1989).

[Sammapun, Lee and Sokolsky 2005] Sammapun, U., Lee, I., and Sokolsky, O.: "RT-MaC: Runtime Monitoring and Checking of Quantitative and Probabilistic Properties"; Proc. 11th IEEE Int'l Conf. Embedded and Real-Time Computing Systems and Applications, IEEE, (2005), 147-153.

[StateRover] The StateRover, http://www.time-rover.com