Towards Formal Specification Visualization for Testing and Monitoring of Cyber-Physical Systems

Bardh Hoxha, Hoang Bach, Houssam Abbas, Adel Dokhanchi,
Yoshihiro Kobayashi, and Georgios Fainekos
Arizona State University, Tempe, AZ, U.S.A.
Email: {bhoxha,hbach,hyabbas,adokhanc,ykobaya,fainekos}@asu.edu

***Abstract*—One of the main challenges in software development for safety-critical Cyber-Physical Systems (CPS) is in achieving a certain level of confidence in the system correctness and robustness. In order to perform formal monitoring, testing and verification of CPS, the fully modular tool S-TaLiRo is presented. The tool is designed for seamless integration with the Model Based Design (MBD) process in Matlab/Simulink™. S-TaLiRo performs robustness guided Metric Temporal Logic (MTL) testing and monitoring. Since writing specifications in MTL is an error prone task that requires expert temporal logic users, a graphical formalism for the development and visualization of specifications is presented. The article provides an up-to-date overview of S-TaLiRo. It includes a discussion on the benefits of the fully modular architecture and the challenges encountered in its development.**

I. Introduction

The need for testing, verification, and validation of CPS has been reinforced by multiple accidents [25], [20] over the years. One of the reasons software development for these systems is very challenging is due to non-trivial system interactions with the physical environment and challenging execution requirements which are directly related to the system platform. As a result, in recent years, there has been a trend to develop software for safety-critical CPS using the Model Based Design (MBD) paradigm. This approach allows testing and verification at earlier design stages, when only models of the system are available. To conduct system testing and verification, automatic tools such as HyTech [19], SpaceEx [17], CheckMate [31], Flow [9], Breach [14], C2E2 [34] and Strong [12] have been developed.

S-TaLiRo [5] is a tool for verification and testing of CPS (Fig. 1). It is a modular software tool that is built on the Matlab platform. S-TaLiRo can analyze hybrid automata, user defined functions (blackbox), arbitrary Simulink models, hardware-in-the-loop, and processor-in-the-loop models. S-TaLiRo performs automated randomized testing based on stochastic optimization techniques. The requirements on the system are defined in Metric Temporal Logic (MTL) [22].

MTL is a formalism that enables system engineers to express complex design requirements in a formal logic. One of the advantages of MTL is that it removes ambiguities that are generally inherent in requirements expressed in natural language. However, developing Metric Temporal Logic requirements necessitates formal mathematical training that many users may not have time or willingness to develop due to the steep learning curve. Therefore, a more accessible graphical formalism is needed that enables non-expert users to define such requirements.

The development of formal specifications through graphical formalisms has been studied in the past. In [7], the authors extend Message Sequence Charts and UML 2.0 Interaction Sequence Diagrams to propose a scenario based formalism called Property Sequence Chart (PSC). The formalism is mainly developed for specifications on concurrent systems. In, [37], PSC is extended to Timed PSC which enables the addition of timing constructs to specifications.

In this paper, we present a graphical formalism for the development of formal specifications specifically geared towards CPS. The formalism enables the visualization of a wide array of MTL specifications. It is designed for use with systems and signals and enables both event and time based specifications. This is the first time that a visual formal language representation is attempted for specifications over signals. A specification visualization tool is in development based on the graphical formalism presented in this work. The tool will be part of the modular framework of S-TaLiRo. The paper also provides an up-to-date overview of S-TaLiRo and its new functionalities. Finally, the paper discusses the modular architecture of S-TaLiRo, its benefits, and challenges faced in the development of the framework.

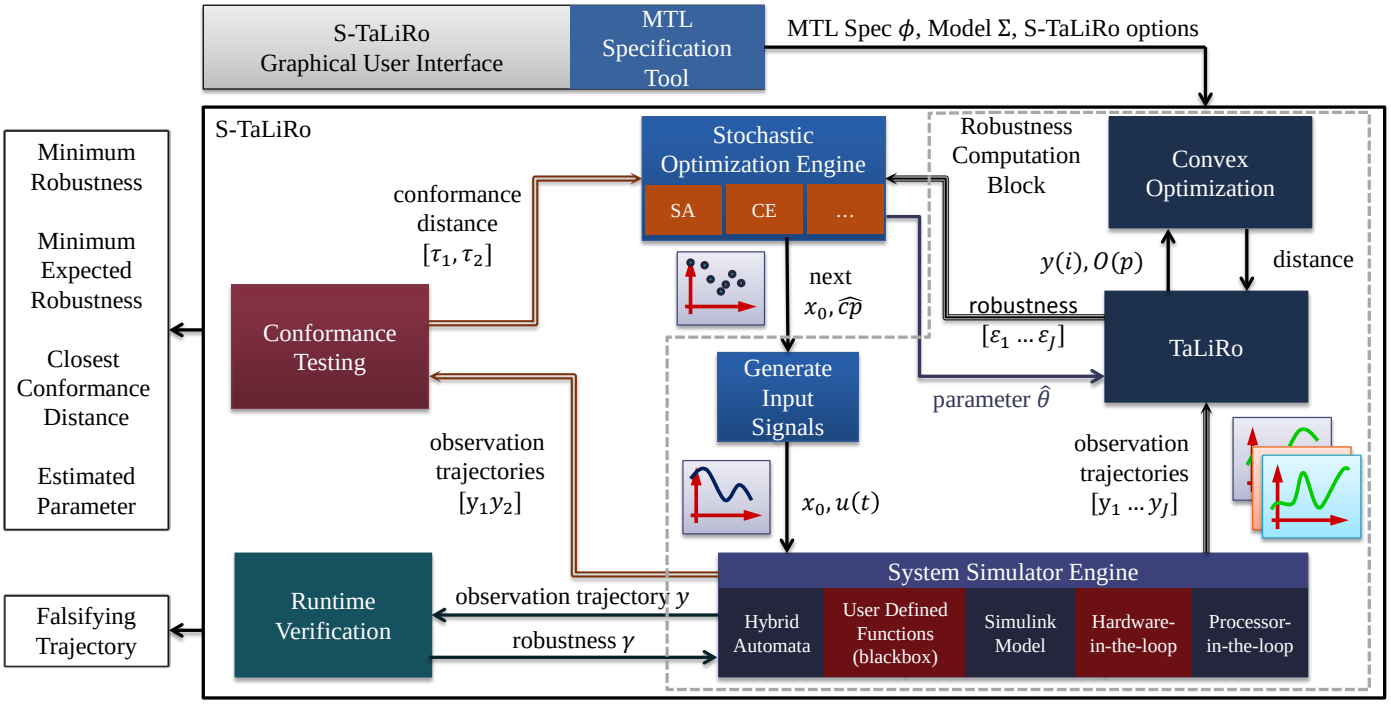

Fig. 1: The modular architecture of S-TaLiRo. The major functionalities of the framework: specification falsification, parameter estimation, conformance testing, and runtime verification. In the Stochastic Optimization Engine block, SA (Simulated Annealing) and CE (Cross-Entropy) are stochastic optimization functions.

II. Background

S-TaLiRo is designed for seamless integration in the model based design process (see Fig. 2). Once a model has been developed, S-TaLiRo will enable formalization of the requirements and their analyses on the system. In the following, we provide an overview of the fully modular architecture and functionalities (presented in Fig. 1). Table I provides an overview of the releases of S-TaLiRo and the main features added in each version.

A. Falsification

In general, the verification problem for MTL is undecidable [18]. Randomized testing methods can provide an effective solution for checking properties of CPS. S-TaLiRo uses the robustness estimate, as presented in [15], to cast the falsification problem of MTL formulas as an optimization problem [1]. In brief, the falsification method searches for *counterexamples* that prove that the system does not satisfy the specification. Different from boolean satisfaction notions, the robustness metric represents the satisfaction of a system trajectory over an MTL formula through a real number. While positive values indicate satisfaction, negative values indicate that the trajectory falsifies the MTL specification.

In general, the optimization problem cannot be presented in a closed functional form. Therefore, S-TaLiRo utilizes stochastic optimization techniques to search for system inputs and initial conditions which result in the global minimum robustness value. Although, it cannot guarantee that the global minimum is found, it has been shown in previous work [1] that the stochastic optimization methods perform exceptionally well in practical applications. Due to the modular architecture of S-TaLiRo, users can easily incorporate their preferred stochastic optimization functions. Once a falsifying trajectory is found, it is presented to the user for further analysis. For a more detailed presentation on the robustness guided falsification problem, experimental results, and applications see [16], [1], [29], [30].

B. Parameter Estimation

In Model Based Development (MBD) of CPS, often times it is desirable to automatically infer specifications that the system satisfies. Specifically, given a parametric specification, system engineers would like to infer the ranges of parameters for which the property holds on the system. Such a property exploration framework can be of great help to the practitioner. Not only will this framework help system developers explore system prop-

erties, but in the initial design stages, also make sure that the properties are well formalized and understood.

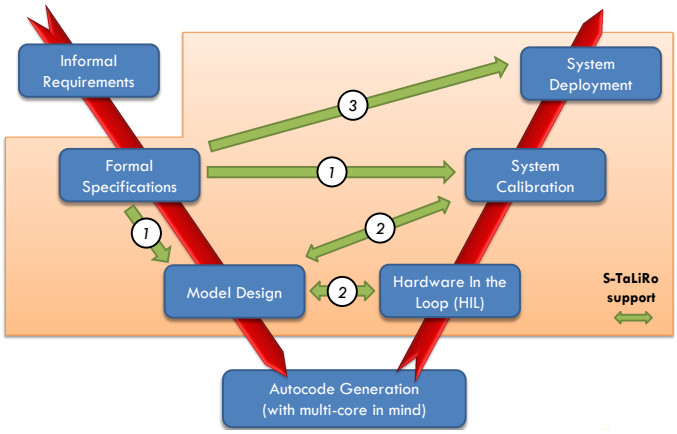

Fig. 2: S-TALIRO's role in MBD: (1) Iterative development and testing/verification of model [3]; (2) Conformance testing between model and HIL/PIL or tuned/calibrated model [2]; (3) Runtime monitoring of formal requirements [13]

Specifications are presented in parametric MTL (PMTL) formulas, which are MTL formulas where one or more parameters are present in the temporal operators or predicate parameters. An example of such a formula is $\phi_{par} = \neg(\Diamond_{[0,\lambda_1]}(speed \geq 100) \wedge \Box(rpm \leq \lambda_2))$.

In regards to the robustness metric, it has been noted [6], [21] that some PMTL formulas are monotonically non-increasing or non-decreasing. An example of such a formula is ϕ_{par}. As $[\lambda_1, \lambda_2]$ increases, the robustness value of the system cannot increase. For this class of formulas, using robust semantics for MTL, the parameter estimation problem can be converted into an optimization problem which can be solved by utilizing stochastic search methods.

The solution to the optimization problem provides a range of values for the parameter such that the specification is guaranteed not to hold on the system. In [36], the theory of parameter estimation was presented. It was shown that the framework can be used on the challenge problem published by Ford in 2002 [10].

C. Expected Robustness for Stochastic Systems

Due to the inherent stochasticity in many Cyber-Physical Systems (SCPS), there is a need for probabilistic methods that enable system engineers to verify that systems are robust and operate within set specifications. Previously, Statistical Model Checking (SMC) for SCPS was proposed [38], [11], where given a probability distribution on the parameters of the SCPS and a specification, SMC returns the probability that the specification holds on the system. However, the probability of success/failure is not always the most important factor in the analysis of these systems. In some cases, if the system fails, not only would system engineers like to know the probability, but also the severity of the violation of the specification. Furthermore, knowing the probability distribution of the input parameter is not a trivial matter.

The current version of S-TALIRO, includes the Expected Robustness Guided Monte Carlo (ERGMC) algorithm [3]. The method searches for a global minimizer for the expected temporal logic robustness of SCPS. The method utilizes recent results in stochastic optimization [24] that, under some conditions, provide finite time guarantees. Otherwise, the framework reduces to a best effort automatic test generation scheme. The stochastic optimization algorithm is guided by the MTL robustness metric.

In [3], the performance of the framework is demonstrated on a high fidelity SimuQuest [32] engine model. Both ERGMC and the Bayesian SMC [38] methods are included in the current version of S-TALIRO.

D. Runtime Verification

On-line monitoring enables users to observe the CPS behavior in real-time [27], [8], [35], whereas in off-line testing we need to stop the execution to check the system's behavior [26], [28]. In on-line monitoring, an independent monitor can observe the system execution/simulation without intruding on its functionality and it may report potential violations to a supervisor for further control actions. Similarly, in on-line monitoring of MTL robustness, the supervisor can also be informed about how much the requirements are satisfied or violated during simulation/execution.

Another application of on-line monitoring is in system testing. For very long system executions it may be problematic to store the whole execution trace for off-line testing. In contrast, on-line monitoring does not need the whole execution trace to verify the system. In addition, for these cases, system testing is facilitated since the on-line monitor can stop the simulation as soon as the specification is falsified.

S-TALIRO provides an on-line monitoring tool as a Simulink block that can run as an integrated module in the simulation process [13]. The user provides the

S-TaLiRo ver. 1.1 [5]	S-TaLiRo ver. 1.2	S-TaLiRo ver. 1.3	S-TaLiRo ver. 1.4	S-TaLiRo ver. 1.5	S-TaLiRo ver. 1.6	S-TaLiRo ver. 1.7 (under development)
Falsification functionality	Cross-entropy stochastic optimization function	Dynamic programming algorithm for robustness computation	Parameter estimation for specfications with one parameter	Signal control point timing distribution can be included in the search space	Parameter estimation for specifications with multiple parameters	On-line Monitoring
			ERGMC algorithm for SCPS	Support for the Parallel Computing toolbox	Local Descent Method	Specification Visualization Tool
			Time robustness computation algorithm	Random number generator seed can be added for replication of simulation results.		
			Interactive scripts for generating m-scripts for calling S-Taliro	Conformance testing functionality		

TABLE I: Release history of S-TaLiRo with corresponding features.

required specification as a bounded future and/or unbounded past MTL formula. The monitor block checks the Simulink generated traces with respect to the required MTL specification. The monitor block then computes the instant robustness estimate at each simulation step. The output of the monitor block can be used on a feedback loop in control applications.

E. Conformance testing

In model-based design, it is common for the design and verification teams to develop several models of the system, at different levels of abstraction, and for different purposes. For example, an RTL description of a circuit is used for functional verification, while a transistor-level netlist is needed for accurate estimation of power consumption. Some of these models may be derived in an automatic manner, with associated guarantees. Other models, however, are derived manually, and do not have a clear guaranteed, relation to the source model. For example, starting with a transistor schematic of an analog circuit (which is the *nominal* model), the designer creates a behavioral model of that circuit in a language like Verilog (which we call the *derived* model). This behavioral model is used in RTL simulation, and can be used in formal property verification. The correspondence between the two models exists only in the designer's mind, and in fact, might not be checkable formally because of the floating-point values generated by the analog circuit. Moreover, the analog designer may not be an expert in, say, Verilog, or indeed in RTL simulation, which may lead to issues in the Verilog behavioral model.

In such cases, it is important to get a quantitative estimate of the closeness between the two models' behaviors (i.e. their output waveforms), to understand how verification results on the derived model (e.g. the Verilog behavioral model) port over to the nominal model (e.g. the transistor schematic), which is fed to the next step in the design tool chain. In this manner, verification results, including formal verification, may be obtained on the simpler model, and ported over rigorously to the nominal model (for which it may not have even been possible to perform such verification). This closeness should be measured in 'space', i.e. it should capture the distance between the outputs' signal values, and in time, i.e. it must capture delays, dropped samples, and other differences in timing characteristics between the two models' outputs. Finally, from the perspective of the design or verification manager, it is important that the existing design flow is perturbed as little as possible by the measurement of this closeness. We call the process of computing this closeness degree from output traces of the systems conformance testing.

S-TaLiRo defines and implements a rigorous closeness measure, satisfying the above criteria, between the outputs of two systems [2]. Basing the closeness measure only on the outputs of the systems means that we only need the ability to simulate them, which is true of most industrial settings, considering that simulation-based verification remains a major component of industrial verification flows. Thus conformance testing can be integrated easily into most design and verification flows.

III. Visual Specification Tool

S-TaLiRo enables on-line monitoring, testing, and verification of CPS over MTL specifications. Developing MTL specifications requires a level of mathematical training that many users may not have. Furthermore, the training required takes a certain amount of time and

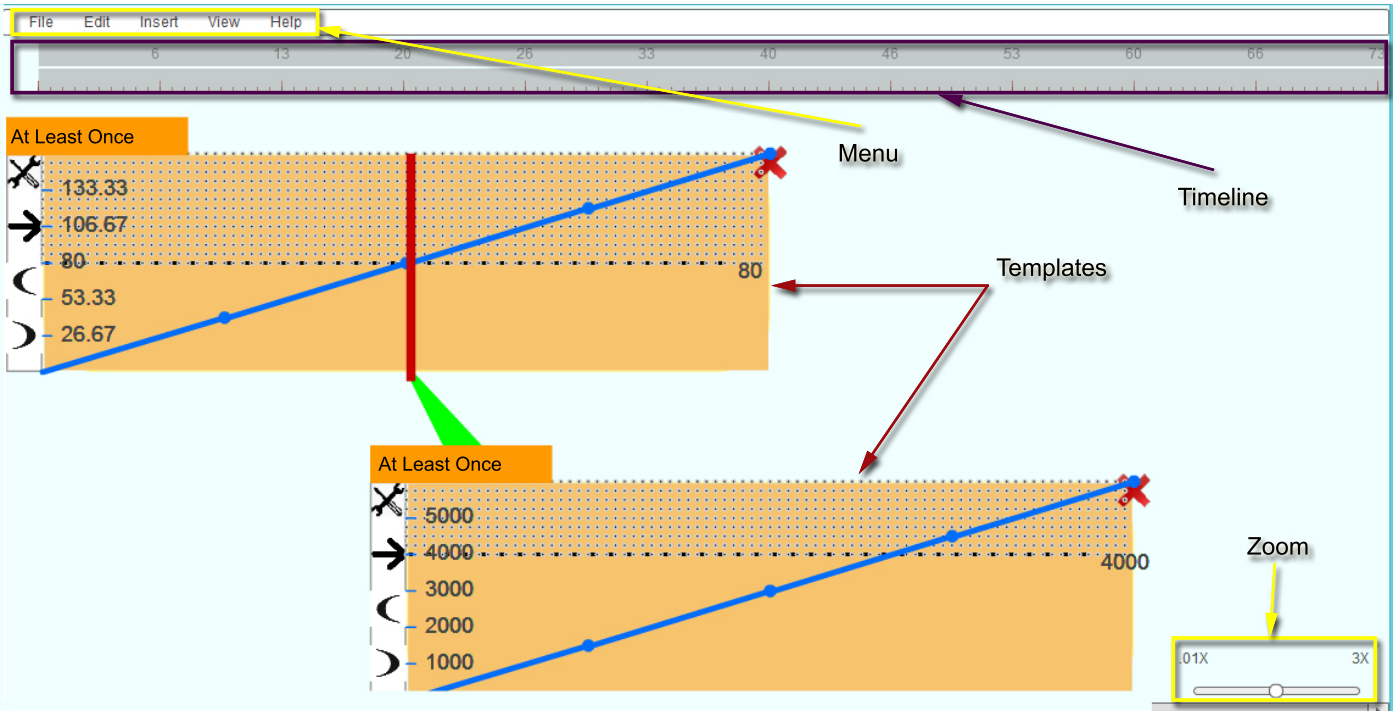

Fig. 3: Overview of the graphical user interface of the MTL specification tool. The example shown represents the MTL specification $\phi = \Diamond_{[0,40]}((speed > 80) \rightarrow \Diamond_{[0,40]}(rpm > 4000))$.

effort. This, coupled with the fact that writing formal specifications is an error prone task has decreased the willingness of the industry to utilize formal specifications. Therefore, making MTL accessible for widespread use is an important problem.

The topic of capturing requirements through graphical formalisms has been studied in the past [33], [4], [23], [7], [37]. However, to the best of the authors' knowledge, the work presented here is the first attempt for CPS to solve the accessibility problem of MTL specifications. The problem is approached from both an event and time based perspective. Both of these are necessary for reasoning over systems and signals. Consider the specification $\Diamond_{[0,5]}((speed > 100) \rightarrow \Box_{[0,5]}(rpm > 4000))$. It states that if within the first 5 seconds, *vehicle speed* goes over 100, then from that moment on, the *engine speed (rpm)*, for the next 5 seconds, should always be over 4000. Here both the sequence and timing of the events are of critical importance.

One of the challenges faced in the development of the graphical formalism was in maintaining the balance of the expressiveness of the tool and its usability. To achieve the latter, we placed several constraints on the types of signals used. Specifically, the signals and requirements are one dimensional which enables clear and structured visualization on a two dimensional user interface.

In Fig. 3, the user interface of the tool is presented along with its most critical components. The user interface is composed of a menu, horizontal timeline, rectangular blocks called templates, and a zoom scroll. While the passage of time is represented horizontally, the sequence of events is presented vertically. The formulas are generated from templates as well as the connections between them.

The main building blocks of the formalism are templates. These are used for defining temporal logic operators, their timing intervals, and the expected signal shape. The user starts with an empty template and a setup assistant presents the user with a sequence of dialog boxes that aid in the development of the template. The process is context dependent where each option selection leads to a potentially different set of options for the next step.

The first step in the template definition process is to define the temporal operator. Among the choices (and their corresponding MTL symbols) are: *Always* ($\Box$), *At Least Once* ($\Diamond$), *Eventually Always* ($\Diamond\Box$), *Repeatedly Often and Finally* ($\Box\Diamond$), and *now*. The options available enable users to define a wide range of specifications. The following sections will present examples of the set

of formulas that can be generated using this graphical formalism.

After the temporal operator is selected, the user will set the timing bounds for it. Many users might have difficulty defining timing bounds, especially for specifications with temporal operators such as *Eventually Always* ($\Diamond\Box$) and *Repeatedly Often and Finally* ($\Box\Diamond$). To clarify the issue, the tool provides a fill-in-the-blanks sentence format to the user. For example, if the operator *Eventually Always* is selected, the user will have to complete the following sentence with the timing bounds: "Eventually, between ___ and ___ seconds, the signal will become true, and from that point on, will stay true in the next ___ to ___ seconds". The set timing intervals are visualized with color shaded regions in the template.

The next step in the process is in defining whether the predicate will evaluate to true when the signal is above or below a set threshold. For example, for the *Always* ($\Box$) operator, a signal is selected that is either always above or below a specified threshold. Once either option is selected, various signals that fit the requirement are automatically generated and presented visually. Instead of drawing the signal, the user will select from one of the generated options. Consider the following example:

Example 1: A specification from the fragment of MTL formulas called *Safety* MTL specifications is presented. Specifically, the specification $\phi_1 = \Box_{[0,36]}(rpm < 4000)$. The formula states that in the next 36 seconds, *engine speed* should always be less than 4000. The corresponding graphical formalism for this formula is presented in Fig. 4. Note that, in regards to the specification, the signal can be of any shape as long as it is always below the 4000 threshold.

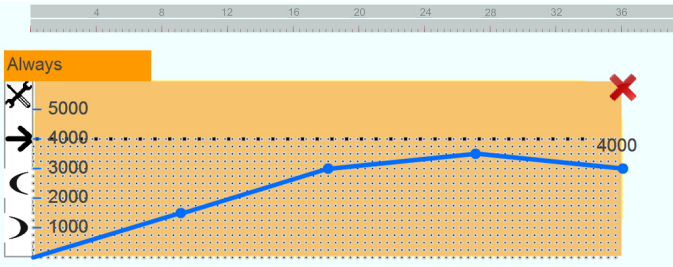

Fig. 4: Example 1: The graphical formalism for the *Safety* MTL specification $\phi_1 = \Box_{[0,36]}(rpm < 4000)$.

Consider the following example for the *At Least Once* ($\Diamond$) operator:

Example 2: A specification from the fragment of MTL formulas called *Reachability* MTL specifications is presented. Specifically, the specification $\phi_2 = \Diamond_{[0,40]}(speed > 100)$. The formula states that eventually, within the next 40 seconds, the vehicle speed will go over 100. The corresponding graphical formalism for this formula is presented in Fig. 5. Again, in regards to the specification, the signal can be of any shape as long as at one point, within the timing bounds of the temporal operator, it is above the 100 threshold.

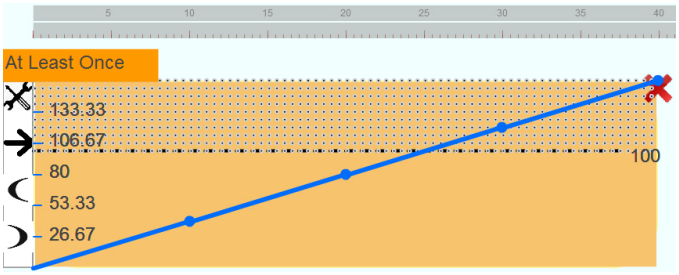

Fig. 5: Example 2: The graphical formalism for the *Reachability* MTL specification $\phi_2 = \Diamond_{[0,40]}(speed > 100)$.

For the *Eventually Always* ($\Diamond\Box$) operator, at least once in the timing interval of the eventually operator, the signal should go above the threshold and stay there for the entire timing interval of the always operator. Two types of shading will indicate the timing bounds of the MTL operators.

Example 3: Consider the specification $\phi_3 = \Diamond_{[0,30]}\Box_{[0,10]}(speed > 100)$. The formula states that at some point in the first 30 seconds, the vehicle speed will go over 100 and stay above for 20 seconds. The corresponding graphical formalism for this formula is presented in Fig. 6.

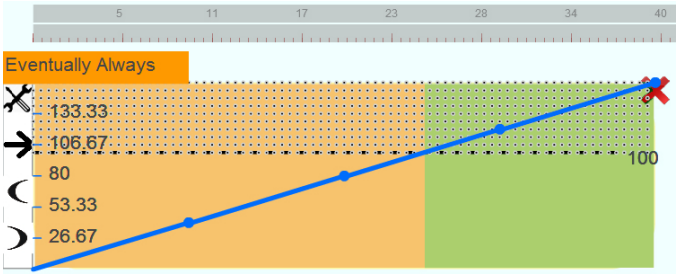

Fig. 6: Example 3: The graphical formalism for the MTL specification $\phi_3 = \Diamond_{[0,30]}\Box_{[0,10]}(speed > 100)$.

For the *Repeatedly Often and Finally* ($\Box\Diamond$) operator, an oscillating signal is presented where two types of shading indicate the timing intervals for each MTL operator. Consider the following example:

Example 4: The specification $\phi_4 = \Box_{[0,40]}\Diamond_{[0,13]}(speed > 100)$ is presented. The formula states that at every timestep of the simulation in the first 40 seconds, the speed will go over 100 within the next 13 seconds. The corresponding graphical formalism for this formula is presented in Fig. 7. No matter how far to the left or right the green shaded region is moved, contained within the orange region, there is always a point where the signal is above the threshold. Recall that the signal is automatically generated so that it satisfies the options previously selected.

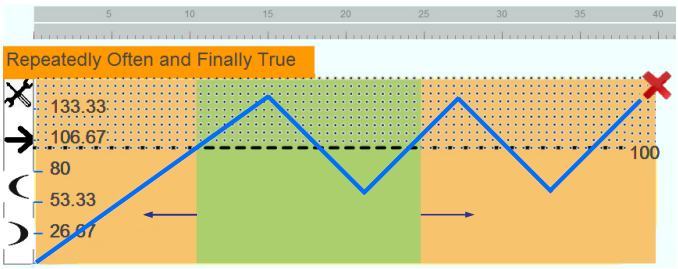

Fig. 7: Example 4: The graphical formalism for the MTL specification $\phi_4 = \Box_{[0,40]}\Diamond_{[0,13]}(speed > 100)$.

The next important concept in this graphical formalism is the relationship between templates.

First, the sequence relationship between two templates is presented. Assume that the first template is already created. If another template is added below it, then an order in the execution of the events is defined. The second template is only considered if the first template is evaluated to true. Formally, there is an implication relationship from the first template to the second. Consider the following example:

Example 5: The specification $\phi_5 = (\Diamond_{[0,40]}(speed > 100)) \rightarrow (\Diamond_{[0,30]}(rpm > 3000))$ is presented. The formula states that if, within 40 seconds, the vehicle speed is above 100 then within 30 seconds from time 0, the engine speed should be over 3000. The corresponding graphical formalism for this formula is presented in Fig. 8.

A second type of relationship enables the user to establish conjunction between two events. To achieve this, templates can be grouped. This is indicated by a bold black box. Doing so requires that both templates evaluate to true. Consider the following example:

Example 6: Specification $\phi_6 = (\Box_{[0,40]}(speed < 100)) \wedge (\Box_{[0,40]}(rpm < 4000))$. The formula states that, within 40 seconds, the vehicle speed should be less than

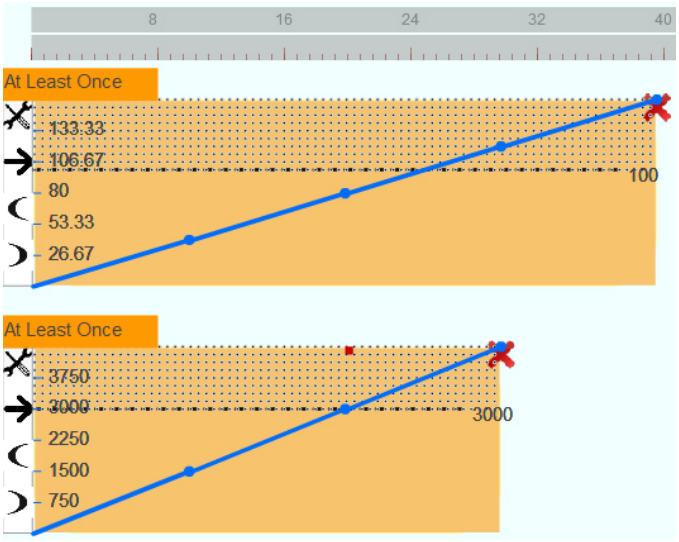

Fig. 8: Example 5: The graphical formalism for the MTL specification $\phi_5 = (\Diamond_{[0,40]}(speed > 100)) \rightarrow (\Diamond_{[0,30]}(rpm > 3000))$.

100 and the engine speed should be under 4000. The corresponding graphical formalism for this formula is presented in Fig. 9.

The third type of template relationship enables the user to establish relative timing between two templates. Consider the following example:

Example 7: Specification $\phi_7 = \Diamond_{[0,40]}((speed > 80) \rightarrow \Box_{[0,40]}(rpm > 4000))$. Here, the nested specification $\Box_{[0,40]}(rpm > 4000)$ is evaluated every time $(speed > 80)$ is true. If at any point in time within 0 and 40 seconds there is a case where $(speed > 80) \rightarrow \Box_{[0,40]}(rpm > 4000)$ then the formula evaluates to true. This formula is represented in the formalism with nested templates, otherwise referred to as parent and child templates. The second template is tabbed and connected to the first template using a green indicator. In the GUI, such a nested template is initiated by clicking on the signal of the parent template. The corresponding graphical formalism is presented in Fig. 10.

The variety of templates and the connections between them allow users to express a wide variety of specifications. The set of specifications that can be generated from this graphical formalism is a proper subset of the set of MTL specifications. Formally, the following grammar produces the set of formulas that can be expressed by the proposed graphical formalism:

$$
\begin{aligned}
\text{S} &::= \neg\text{T} \mid \text{T} \\
\text{T} &::= \text{A} \mid \text{B} \mid \text{C} \\
\text{A} &::= \text{P} \mid (\text{P} \wedge \text{A}) \mid (\text{P} \Rightarrow \text{A}) \\
\text{B} &::= \Box_{\mathcal{I}}\text{D} \mid \Diamond_{\mathcal{I}}\text{D} \\
\text{C} &::= \Box_{\mathcal{I}}\Diamond_{\mathcal{I}}\text{D} \mid \Diamond_{\mathcal{I}}\Box_{\mathcal{I}}\text{D} \\
\text{D} &::= (p \Rightarrow \text{A}) \mid (p \wedge \text{A}) \mid (p \Rightarrow \text{B}) \mid (p \wedge \text{B}) \\
\text{P} &::= p \mid \Box_{\mathcal{I}} p \mid \Diamond_{\mathcal{I}} p
\end{aligned}
$$

where p is an atomic proposition. In practice, the atomic propositions are automatically derived from the templates.

IV. Modularity in S-TaLiRo

Significant emphasis on the development of S-TaLiRo is placed in preserving the modularity of its many independent functions. In the following, we will present some of the benefits attained from its modularity.

Among the benefits is the ability to interchange parts or modules of the tool (see Fig. 1). For instance, the stochastic optimizer functions are isolated from the trajectory robustness computation functions. This allows for flexibility in the choice of stochastic optimizers. In fact, the user can utilize any other stochastic optimizer with user defined cost function.

The modular architecture allows for a wider usability of the tool's independent functions. The trajectory robustness computation functions can be used to analyze any trajectory (timed state sequence) over MTL specifications. This allows for conducting complex analysis

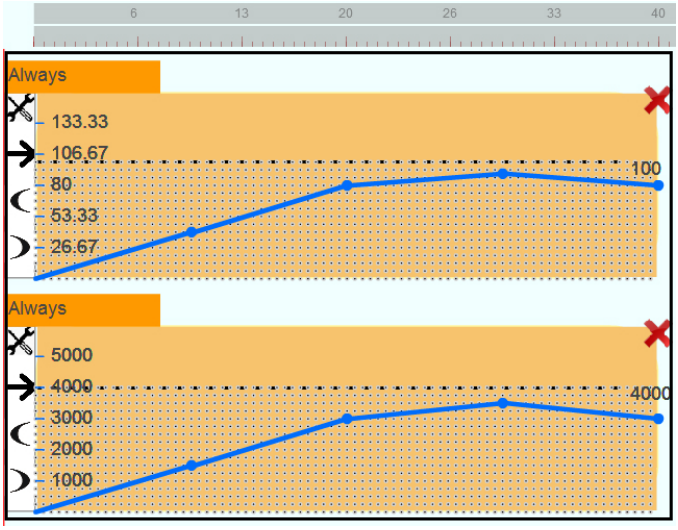

Fig. 9: Example 6: The graphical formalism for the MTL specification $\phi_6 = (\Box_{[0,40]}(speed < 100)) \wedge (\Box_{[0,40]}(rpm < 4000))$.

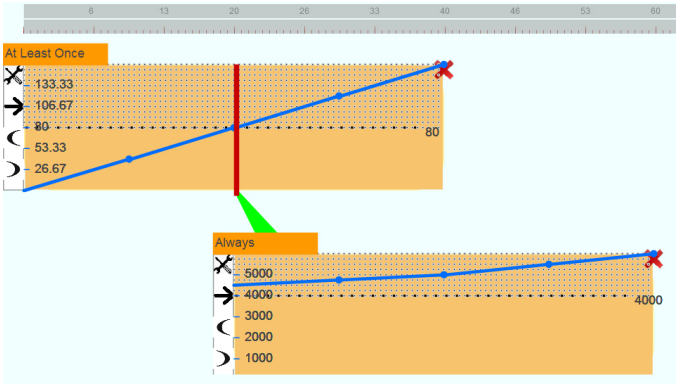

Fig. 10: Example 7: The graphical formalism for the MTL specification $\phi_7 = \Diamond_{[0,40]}((speed > 80) \rightarrow \Box_{[0,40]}(rpm > 4000))$.

of not only CPS system trajectories but over any other time series. For example, in order to test complex MTL specifications over the Dow Jones Industrial Average, or temperature levels in Tempe, Arizona.

Another useful module in the S-TaLiRo architecture is the system simulator function. This function allows for seamless simulation of arbitrary Simulink models, user-defined functions and blackbox models. In particular, testing can be performed over processor-in-the-loop and hardware-in-the-loop systems. Given a vector of input control points, initial conditions and parameters, which is the search space for the stochastic optimizer, its sub-modules can generate various input signal interpolations and simulate the system and output a trajectory. This enables the automatic testing of the model.

Other benefits of the modular architecture are improved maintainability and scalability. In both academic and industrial environments, modularity facilitates tool functionality development through several smaller projects focused on a particular functionality more than on developing the whole system.

For example, the code that implements conformance testing does not fall under the main S-TaLiRo umbrella. That is, it is separate from the MTL falsification core of S-TaLiRo. However, it does re-use several modules from the main S-TaLiRo code as-is, such as the system simulator mentioned above, and various system classes used to represent signals and the like.

V. Development Challenges

S-TaLiRo is built on the Matlab platform. The collaborative development process of S-TaLiRo is facilitated by revision and version control systems such as

Apache Subversion (SVN) and git. These, in conjunction with clients such as TortoiseSVN and TortoiseGit enable a convenient revision and version control in the Windows OS environment.

S-TaLiRo is developed in an academic environment with twelve developers through the years. In that period, the tool has been released several times (see Table I). To maintain high quality code with high performance, techniques such as peer review are utilized. Not only does this help with the quality of the code, but also helps the developers obtain knowledge on other modules of the framework. It helps maintain proper documentation and helps increase developer skills.

Since the tool has been in development over several years, a number of Matlab commands have been deprecated and removed from use in newer versions. Since most companies are hesitant to change their development process as soon as a newer version is out, it is necessary to maintain backwards compatibility. For example, our industrial partners use Matlab 2010b while most of our development is conducted in Matlab 2013b. To maintain backwards compatibility, when using a Matlab command, developers check when the command was introduced to the Matlab environment. An if statement on the version of Matlab is utilized to ensure that the commands are compatible. For example, S-TaLiRo includes an option to set the seed for the random number generator to enable users to reproduce testing and verification results. A Matlab command that can be used to set the seed for the random number generator is *rng*. However, this command was introduced in Matlab 2011a (version 7.12). As a result, any users with older versions of Matlab would not be able to use the tool. To fix the issue, for any earlier versions of Matlab, developers use the *RandStream* command.

Another challenge was faced when adding support for the Matlab Parallel Computing toolbox in S-TaLiRo. The toolbox enables users to conduct simulations and robustness computations in parallel. Initially, developers only added a check for whether the toolbox is installed. However, not only should the toolbox be installed, but also licensed. A company might only have a limited number of licenses. Once Matlab starts for a user, a license key is checked out and is not checked back in until the Matlab session is ended. A situation can easily arise where there are not a sufficient number of licenses. Therefore, a check on the license had to be added before utilizing the parallel toolbox.

VI. Conclusion and Future Work

This article has presented an up-to-date overview of the semi-formal monitoring, testing and verification tool S-TaLiRo. The main functionalities of the tool are presented. The article follows with a discussion on challenges faced in incorporating MTL specifications in the industry and proposes a graphical formalism to facilitate its use. The formalism enables users to visualize the event and time based components of specifications. Also, it enables non-expert users to develop MTL specifications. The article continues with a discussion on the modularity of S-TaLiRo and the challenges faced in the development of the tool.

As future work, the authors will finalize the development of the visual specification tool and conduct user studies to measure the effects of the proposed approach. Also, the authors will work on including the *Until* operator in the formalism without deteriorating the usability of the tool. This would extend the set of useful formal specifications that can be developed using this graphical formalism.

Acknowledgment

The authors would like to thank Kangjin Kim for the useful discussions. This work was partially supported under NSF awards CNS 1116136, IIP-0856090 and the NSF I/UCRC Center for Embedded Systems.

References

[1] H. Abbas, G. E. Fainekos, S. Sankaranarayanan, F. Ivancic, and A. Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems*, 12(s2), May 2013.

[2] H. Abbas, B. Hoxha, G. Fainekos, J. V. Deshmukh, J. Kapinski, and K. Ueda. Conformance testing as falsification for cyber-physical systems. Technical Report arXiv:1401.5200, January 2014.

[3] H. Abbas, B. Hoxha, G. Fainekos, and K. Ueda. Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In *The 4th Annual IEEE International Conference on CYBER Technology in Automation, Control, and Intelligent Systems*, 2014.

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

[5] Y. S. R. Annapureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and algorithms for the construction and analysis of systems*, volume 6605 of *LNCS*, pages 254–257. Springer, 2011.

[6] E. Asarin, A. Donzé, O. Maler, and D. Nickovic. Parametric identification of temporal properties. In *Runtime Verification*, volume 7186 of *LNCS*, pages 147–160. Springer, 2012.

[7] M. Autili, P. Inverardi, and P. Pelliccione. Graphical scenarios for specifying temporal properties: an automated approach. *Automated Software Engineering*, 14(3):293–340, 2007.

[8] D. A. Basin, F. Klaedtke, and E. Zalinescu. Algorithms for monitoring real-time properties. In *Runtime Verification*, volume 7186 of *LNCS*, pages 260–275. Springer, 2011.

[9] X. Chen, E. Abraham, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer-Aided Verification (CAV)*, volume 8044 of *Lecture Notes in Computer Science*, pages 258–263. Springer-Verlag, 2013.

[10] A. Chutinan and K. R. Butts. Dynamic analysis of hybrid system models for design validation. Technical report, Ford Motor Company, 2002.

[11] A. David, D. Du, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, and S. Sedwards. Statistical model checking for stochastic hybrid systems. In *Proceedings First International Workshop on Hybrid Systems and Biology*, number 92 in EPTCS, pages 122–136, 2012.

[12] Y. Deng, A. Rajhans, and A. A. Julius. Strong: A trajectory-based verification toolbox for hybrid systems. In *Quantitative Evaluation of Systems*, pages 165–168. Springer, 2013.

[13] A. Dokhanchi, B. Hoxha, and G. Fainekos. On-line monitoring for temporal logic robustness. In *Runtime Verification*, volume 8734 of *LNCS*, pages 231–246. Springer, 2014.

[14] A. Donze. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*, volume 6174 of *LNCS*, pages 167–170. Springer, 2010.

[15] G. Fainekos and G. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, September 2009.

[16] G. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel. Verification of automotive control applications using s-taliro. In *Proceedings of the American Control Conference*, 2012.

[17] G. Frehse, C. L. Guernic, A. Donz, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *Proceedings of the 23d CAV*, 2011.

[18] T. A. Henzinger. Temporal specification and verification of real-time systems. Technical report, DTIC Document, 1991.

[19] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. In *Proceedings of the 9th International Conference on Computer Aided Verification*, volume 1254 of *LNCS*, pages 460–463. Springer, 1997.

[20] E. J. Hoffman, W. L. Ebert, M. D. Femiano, H. R. Freeman, C. J. Gay, C. P. Jones, P. J. Luers, and J. G. Palmer. The near rendezvous burn anomaly of december 1998. Technical report, Applied Physics Laboratory, Johns Hopkins University, Nov. 1999.

[21] X. Jin, A. Donzé, J. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control (HSCC)*, April 2013.

[22] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[23] H. Kugler, D. Harel, A. Pnueli, Y. Lu, and Y. Bontemps. Temporal logic for scenario-based specifications. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 445–460. Springer, 2005.

[24] A. Lecchini-Visintini, J. Lygeros, and J. Maciejowski. Stochastic optimization on continuous domains with finite-time guarantees by markov chain monte carlo methods. *Automatic Control, IEEE Transactions on*, 55(12):2858 –2863, dec. 2010.

[25] J.-L. Lions, L. Lbeck, J.-L. Fauquembergue, G. Kahn, W. Kubbat, S. Levedag, L. Mazzini, D. Merle, and C. O'Halloran. Ariane 5, flight 501 failure, report by the inquiry board. Technical report, CNES, July 1996.

[26] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proceedings of FORMATS-FTRTFT*, volume 3253 of *LNCS*, pages 152–166, 2004.

[27] T. Reinbacher, K. Y. Rozier, and J. Schumann. Temporal-logic based runtime observer pairs for system health management of real-time systems. In *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *LNCS*, pages 357–372. Springer, 2014.

[28] G. Rosu and K. Havelund. Synthesizing dynamic programming algorithms from linear temporal logic formulae. Technical report, Research Institute for Advanced Computer Science (RIACS), 2001.

[29] S. Sankaranarayanan and G. Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In *ACM International Conference on Hybrid Systems: Computation and Control*, 2012.

[30] S. Sankaranarayanan and G. Fainekos. Simulating insulin infusion pump risks by in-silico modeling of the insulin-glucose regulatory system. In *International Conference on Computational Methods in Systems Biology*, 2012. [To Appear].

[31] B. I. Silva and B. H. Krogh. Formal verification of hybrid systems using CheckMate: a case study. In *Proceedings of the American Control Conference*, volume 3, pages 1679 – 1683, June 2000.

[32] Simuquest. Enginuity. http://www.simuquest.com/products/enginuity. Accessed: 2013-10-14.

[33] M. H. Smith, G. J. Holzmann, and K. Etessami. Events and constraints: A graphical editor for capturing logic requirements of programs. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 14–22. IEEE, 2001.

[34] P. Sridhar, S. Mitra, and M. Viswanathan. Verification of annotated models from executions.

[35] P. Thati and G. Rosu. Monitoring algorithms for metric temporal logic specifications. In *Runtime Verification*, volume 113 of *ENTCS*, pages 145–162. Elsevier, 2005.

[36] H. Yang, B. Hoxha, and G. Fainekos. Querying parametric temporal logic properties on embedded systems. In *Testing Software and Systems*, pages 136–151. Springer, 2012.

[37] P. Zhang, B. Li, and L. Grunske. Timed property sequence chart. *Journal of Systems and Software*, 83(3):371–390, 2010.

[38] P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 243–252, 2010.