Runtime Fault Detection in Programmed Molecular Systems

SAMUEL J. ELLIS, Molecular Sciences Software Institute
TITUS H. KLINGE, Carleton College
JAMES I. LATHROP, JACK H. LUTZ, ROBYN R. LUTZ, ANDREW S. MINER, and HUGH D. POTTER, Iowa State University

Watchdog timers are devices that are commonly used to monitor the health of safety-critical hardware and software systems. Their primary function is to raise an alarm if the monitored systems fail to emit periodic "heartbeats" that signal their well-being. In this article, we design and verify a *molecular watchdog timer* for monitoring the health of programmed molecular nanosystems. This raises new challenges, because our molecular watchdog timer and the system that it monitors both operate in the probabilistic environment of chemical kinetics, where many failures are certain to occur and it is especially hard to detect the absence of a signal.

Our molecular watchdog timer is the result of an incremental design process that uses goal-oriented requirements engineering, simulation, stochastic analysis, and software verification tools. We demonstrate the molecular watchdog's functionality by having it monitor a molecular oscillator. Both the molecular watchdog timer and the oscillator are implemented as chemical reaction networks, which are the current programming language of choice for many molecular programming applications.

CCS Concepts: • **Software and its engineering** → *Designing software*; Formal software verification; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks*; Embedded and cyber-physical systems;

Additional Key Words and Phrases: Molecular programming, molecular system safety, chemical reaction networks, requirements engineering, probabilistic model checking

ACM Reference format:
Samuel J. Ellis, Titus H. Klinge, James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, Andrew S. Miner, and Hugh D. Potter. 2019. Runtime Fault Detection in Programmed Molecular Systems. *ACM Trans. Softw. Eng. Methodol.* 28, 2, Article 6 (March 2019), 20 pages.
https://doi.org/10.1145/3295740

1 INTRODUCTION

Molecular programming uses the information processing inherent in chemistry, especially the chemistry of DNA and other biomolecules, to engineer useful nanoscale systems. Molecular programming applications including diagnostic biosensors, medical therapeutics, molecular robots,

This research was supported in part by National Science Foundation grants 1247051 and 1545028.
Authors' addresses: S. J. Ellis, Molecular Sciences Software Institute, 1880 Pratt Dr., Suite 1100, Blacksburg, VA 24060; email: sjellis@vt.edu; T. Klinge, Dept. of Mathematics and Computer Science, Drake University, 2507 University Ave., Des Moines, IA 50311; email: titus.klinge@drake.edu; J. I. Lathrop, J. H. Lutz, R. R. Lutz, A. S. Miner, and H. D. Potter, Dept. of Computer Science, 226 Atanasoff Hall, Iowa State Univ., Ames, IA 50011; emails: {jil, lutz, rlutz, asminer, hdpotter}@iastate.edu.

1049-331X/2019/03-ART6 $15.00
https://doi.org/10.1145/3295740

and bio-compatible materials already exist in the laboratory and are poised to have a major impact on society.

This article proposes the design of verifiable safety mechanisms for molecular programming applications. This is an important and timely objective, because many of the planned future uses of molecular programmed systems are safety-critical, such as biosensors to detect pollutants in water or drug therapeutics to deliver customized medicine only to the locations in the body where disease has been detected [18, 39, 63, 65]. During the rapid expansion of the field of molecular programming, basic research has been rightly focused on normal operation (systems doing what they should). However, as the field progresses to development and deployment, it will also need to focus on avoiding hazards (systems not doing what they should not do). Designing and formally verifying safety mechanisms for programmable control of molecular systems now will help enable this future focus on safety [21].

In many safety-critical systems, the failure of a monitored system can be dangerous if it goes undetected. A standard remedy is to introduce a *software watchdog timer*, a safety mechanism whose responsibility is to monitor for the occurrence of the failure and to raise an alarm that can trigger recovery action if a failure occurs [29, 38]. For example, on the Voyager spacecraft, a heartbeat was sent every two seconds from the attitude control computer to the command computer responsible for monitoring its health. If the attitude control computer's self tests failed, then no heartbeat was generated. "After about 10 seconds passed with no heartbeats, the command computer would issue a switch-over command to the backup processor" [44]. Correct behavior of a watchdog timer in the context of the system it monitors is essential. For example, the faulty integration of a watchdog timer such that it did not actually monitor the throttle may have resulted in the failure of a major task in the Toyota unintended acceleration accidents [30].

A watchdog timer receives a periodic heartbeat from the system that it is monitoring. Receipt of the heartbeat resets the watchdog timer, indicating that the monitored system is still alive. When the heartbeat signal stops, the absence of the heartbeat causes the watchdog timer to time out and "bark." Outputting this alarm indicates that a fault has occurred in the monitored system that led to its experiencing a service failure [4]. Watchdog timers often serve both to detect the failures of monitored systems and to trigger their recovery, either through corrective interventions or automated recovery actions.

Guided by the successes of software watchdog timers, we have chosen a *Molecular Watchdog Timer* (MWT) as our first safety mechanism for molecular programming applications. This MWT should monitor a molecular system at runtime, detect when the heartbeat signal from the monitored system stops, and alarm to trigger its recovery.

The design of an MWT is an ambitious goal. Monitoring for the absence of an event is especially difficult in the molecular domain. Detecting the non-occurrence of an expected event is often not yet possible for components that execute outside the laboratory such as *in vivo* applications. For example, some of the most promising planned molecular systems involve drug delivery to tumors. For such systems, runtime monitoring and detection of faults must take place in the same molecular environment as the programmed system itself. Additional challenges to detecting faults at runtime in molecular programmed systems include the facts that the behavior of molecular systems is probabilistic; the components are nanoscale, so runtime observation is non-trivial; there are very many components, so scalability is a problem; and the components are fault-prone, so any fault that can occur probably will occur in a significant number of components.

The main contribution of this article is the requirements specification, design, and verification of an MWT for molecular programming applications. This contribution includes the following features:

(1) We develop a goal-based model and formally verify the requirements for an MWT using the Isabelle proof assistant [46].
(2) Our MWT is designed as a *chemical reaction network*, a well-understood mathematical model [1, 2] that is suitable for automatic compilation into DNA implementations [53, 54].
(3) Our MWT detects heartbeat failures at runtime and alarms accordingly.
(4) Our MWT's timing functions are carried out by *stochastic delay ladders*, introduced here, that are provably reliable, both for detecting failures and for avoiding false alarms.
(5) In addition to alarming, our MWT can trigger the recovery of a monitored system.
(6) Our MWT is automatically reusable rather than having to be discarded after a single failure.
(7) Our MWT is tested with a specific monitored system, a molecular oscillator widely used as a benchmark, modified to produce a heartbeat. Two very different types of oscillator failure can interrupt this heartbeat.
(8) Our MWT is embedded with oscillators at two widely separated ranges of size, with end-to-end behavior verified and validated at these scales using model checking [7, 33] and simulation, respectively.
(9) These verifications demonstrate that our MWT performs correctly with the oscillators, enabling them to recover correctly from both types of failure.

Earlier versions of our MWT were reported in References [20, 22]. The version reported here has a different architecture and improved functionality. Regarding the above list of features, the earlier versions had goal-based models as in feature (1), but these were hand-verified. The earlier versions had features (2) and (3) and rudimentary, inexplicit versions of feature (4). The present article's use of the Isabelle proof assistant in feature (1), its systematic treatment of the delay ladders in feature (4), and all aspects of features (5) through (9) are new work. This article describes an MWT re-designed to be reusable, embeddable with the system it monitors, and capable of triggering a monitored system's recovery at runtime.

A broader contribution of the article is to demonstrate the new use of software engineering techniques and tools to create and verify the requirements and design for a programmable safety mechanism, the MWT, that will be needed for future molecular systems such as biocompatible drug delivery nanodevices. We have sought to make the development approach that we use general enough to guide future molecular design work on additional safety mechanisms. We claim that software engineering helps achieve molecular programmed systems that are safe for use in a dynamic and only partially understood physical environment and show in our results the benefits of a software engineering approach to creating new molecular systems.

The rest of the article is organized as follows. Section 2 describes the goal-oriented requirements analysis and machine-checked refinement proofs for the MWT. Section 3 presents the chemical reaction network design models to achieve these capabilities. Section 4 shows how simulation, probabilistic model checking, and mathematical proofs are used to validate and verify the design. Section 5 describes related work, and Section 6 gives concluding remarks.

2 REQUIREMENTS

In this section, we describe a requirements engineering process for programmed molecular systems and use it to develop the requirements for a runtime fault detection device called a Molecular Watchdog Timer (MWT). We first describe informally the high-level requirements for the new system to be built. We then describe the iterative process by which the requirements were formally

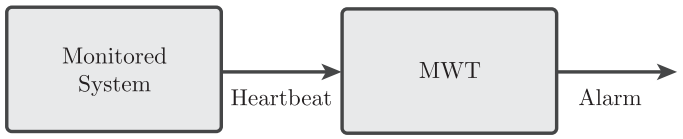

Fig. 1. Context diagram.

specified, analyzed, corrected, refined, and verified as our understanding of what was needed improved. Lastly, we discuss how the interplay between requirements and design contributed to the incremental improvement of the requirements. This ongoing process made the requirements significantly more complete, accurate, and realistic (i.e., feasible to implement in DNA).

Despite the difficulty of "getting the requirements right" for a new molecular device, our use of requirements engineering techniques in molecular programming is novel. We show how engaging in goal-oriented requirements engineering can benefit the design of programmed molecular systems, especially by finding and solving problems early in development.

Figure 1 shows a context diagram [60] for the MWT. The Monitored System sends a regular signal (labeled "Heartbeat") to the MWT. The MWT processes this signal and outputs an Alarm to an external observer or system if the signal has stopped.

2.1 Goal Modeling

We used a goal-oriented requirements engineering approach that we based on van Lamsweerde's KAOS [60] and introduced in References [40, 41] to specify and analyze the MWT system goals. Goal-oriented techniques support systematic and incremental refinement from informal descriptions to formal specification of properties for simulation and model checking.

We first performed goal modeling to understand what was needed to achieve an MWT. A goal model is an AND/OR graph in which the top-level node describes the high-level functional requirement of the system-to-be and the leaf nodes are the subgoals whose collective satisfaction implies satisfaction of the top-level goal. The goals are specified as goals to ACHIEVE, MAINTAIN, or AVOID various conditions in view of the domain properties (such as physical laws governing molecular interactions). An AND node is satisfied provided that its children nodes are satisfied. An OR node shows alternative refinements of a node. Our goal model's nodes are all AND nodes.

The AND refinement of high-level goals and the graphical presentation of the results is sufficiently intuitive to be useful in our discussions with molecular biologists. It also allows both top-down and bottom-up development of the hierarchical tree as understanding improves. The logical underpinnings of KAOS support formal analysis and clean traceability between the textual descriptions of the goals and the formal specification of the goal model.

Our high-level goal is that at runtime the MWT shall issue an alarm if and only if the monitored system does not provide a heartbeat within a specified time. Figure 2 shows the goal model for the MWT. The top-level goal, *Achieve [Alarm iff no Heartbeat provided within t time]*, describes the intent of the system-to-be. That goal is AND-refined into two subgoals, such that it can be satisfied if both of the two second-level goals are met. The subgoals are further refined until each leaf goal can be assigned to an agent (discussed in Section 3). An agent is a system component with responsibility for satisfying the goal(s) assigned to it [60].

The MWT's client can use the alarm signal output by the MWT either as an externally observable *alarm* that notifies the client when the heartbeat stops or as a *recovery trigger* prompting the initiation of some recovery action when the heartbeat stops. The first usage scenario is intended for scientific observations where the alarm signal might, for example, be a fluorescent molecule

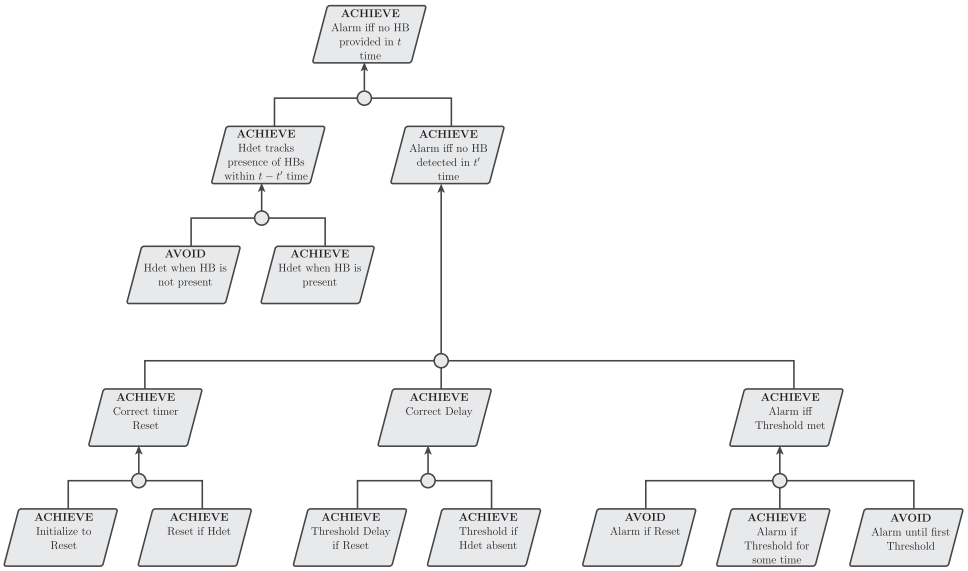

Fig. 2. Goal model.

visible to a human technician. The second scenario is useful when the MWT is monitoring an adjacent molecular system that is capable of autonomous recovery and is a focus of this work.

2.2 Environmental Assumptions

Environmental assumptions are statements about the system's operational context that are accepted as true by the developers [59]. The proposed MWT will be made of DNA strands and will operate, both literally and figuratively, in a fluid, molecular environment. For satisfaction of the subgoals to imply that the parent goal is satisfied, we must make certain assumptions about the operational environment. For example, we assume that the chemical solution in which the MWT operates is well mixed. Other environmental assumptions needed to prove the satisfiability of the top-level goal are that the heartbeat species, H, is intrinsically ephemeral, meaning that it will not persist and will decay over time; that the number of H molecules in the heartbeat pulse is in a certain range; and that no molecules in the environment other than the heartbeats interact with the MWT.

If these environmental assumptions are false or cease to be true, then the validity of the solution may be at risk. We will see in the following how the process of formalizing the leaf goals, proving that the leaf goals implied the top-level goal, and verifying the design forced us to revisit and revise several of our original environmental assumptions.

2.3 Goal Formalization

For every goal, we needed both a natural language description and a formal specification. We specified the goals in continuous stochastic logic (CSL) [5] because of its availability in the model checking tool that we use and because it handles the continuous time Markov chains (CTMCs) [1, 2] that form the semantics of our chemical reaction networks (described in Section 3). Formally specifying the goals and assumptions enabled us to prove that the satisfaction of the lower-level goals, together with the stated assumptions, implies the satisfaction of the higher-level goals.

The complete CSL specifications for the goal model appear in the table in the Supplemental Material. For each node in the goal model, the table there shows (1) the name of the node, (2) the formal specification of the node in CSL, and (3) the agent-assigned responsibility for each leaf goal. The goals are ordered breadth first following Figure 2. As an example, the leaf goal *Achieve [Alarm if Threshold for some time]* has the CSL specification $\mathcal{P}_{\geq 1}\Box[Th_H \Rightarrow \mathcal{P}_{\geq 1-\eta_4}\Diamond_{\leq w_{th}}(Alarm \vee \neg Th_H)]$ and is assigned to the Threshold Filter as described in Section 3.

We proved that the goals, together with the specified environmental assumptions and facts and parameter values in their specified ranges, satisfy the top-level goal. This proof was carried out manually and iteratively during the development of the goal model.

2.4 Obstacle Analysis

The MWT must work safely and with high reliability to detect and recover from faults in the molecular applications for which it is intended, such as bio-sensing and medical therapeutics. A major challenge in developing the MWT was determining accurately the requirements for a *feasible* system that could operate in the molecular environment. To do this, we needed to analyze whether the stated goals could be realistically satisfied in this stochastic and dynamic setting.

To investigate feasibility, we used KAOS's obstacle analysis extensively [11, 60, 61]. An obstacle to a goal is a precondition for that goal's non-satisfaction [60]. Given a goal model in which A and B are the AND subgoals of parent goal C, an obstacle for C is a state of affairs in which A and B are true and C is false.

Obstacle analysis *identifies* ways in which the goals might fail to be satisfied (i.e., obstacles to satisfaction), *assesses* the likelihoods and impacts of the obstacles, and investigates how to *resolve* them. We found in previous work on DNA nanopliers that the early analysis of obstacles to satisfying the goals worked well in helping find and remedy missing and unrealistic requirements in the programmed molecular system [41].

Most of the obstacles for the MWT were subtle and found manually while proving that subgoals satisfied their parent goal. The process of proving the satisfiability of the formal CSL goals repeatedly revealed both additional cases and uncertainties that could be introduced by the stochastic behavior. Gaps and errors in the goal model often were due to the non-deterministic, asynchronous nature of reactions, to the very large number of molecular agents operating in parallel, and to the physical environment in which the MWT operates. Model-based mathematical analysis, simulation with the MATLAB extension, SimBiology, and probabilistic model-checking with PRISM [32], as described in the following, helped us assess the likelihood and impact of candidate obstacles.

Representative examples of obstacles we found in the early version of the MWT were previously reported in Reference [22]. These include:

Incorrect agent. We initially assigned a binary counting device introduced in Reference [27] to be the clock agent responsible for detecting the absence of a heartbeat. However, simulation in SimBiology revealed that this device did not satisfy our specification. It was designed to work in a setting in which it is assumed that all reactions are "fast" or "slow,' and that all fast reactions occur before all slow reactions. The stochastic (and more realistic) model on which we work violates this assumption, allowing slow reactions to interfere with the clock's function. Over time, the accumulation of such violations leads to failure of the clock. To resolve this, we assigned responsibility for that goal to a new agent in which the delay is instead achieved by a programmed cascade of interactions.

Missing property. In refining a goal into two subgoals, we had to introduce the domain property that it takes a positive amount of time to detect a heartbeat. This is because the detection occurs via the chemical interaction of the heartbeat molecules with the molecular component that detects missing heartbeats. The subgoals did not satisfy the goal without this domain property.

This obstacle was resolved by introducing a "grace period" before heartbeat detection is required. When we initially failed to propagate the addition of the grace period backup to the parent goal, model checking with PRISM detected the omission, and it was corrected.

Incorrect initialization. Model checking revealed a failure mode that can occur just after the MWT begins execution. The initial intent was that operation of the MWT begin at time zero, i.e., when the MWT was "poured into the test tube." However, as specified, the MWT could violate this intent by alarming *before* the monitored system had a chance to send a heartbeat. To resolve this, we added a new CSL property to the high-level goal specifying that the alarm must remain off for a period of time after initialization. This new goal propagated through the goal diagram to create a new leaf-goal, *Achieve [Initialize to Reset]*, specifying that the timer must be considered to be in a reset stage at initialization. Together with the leaf goal *Achieve [Threshold Delay if Reset]*, this implies that the alarm will not be active upon initialization. We proved manually that the implication holds after the change and confirmed using PRISM that a model with an alarm in the initial state satisfies the goals.

Obstacle analysis helped identify missing requirements, explore design alternatives, and find idealized environmental assumptions that had to be weakened to conform to physical realities. The obstacle analysis process was on-going and a major contributor to de-idealizing the goals for the MWT into requirements that were feasible for implementation in a programmed molecular system operating in a chemical "soup." During the obstacle analysis, we experienced extensive back-and-forth interweaving between the requirements and the design. This iterative, incremental nature of the modeling and analysis effort is typical of complex systems [62] and often described in terms of "twin peaks" [47].

Related work on specifying goals in uncertain environments formalizes the required degree of goal satisfaction, as in Reference [37], or the required probability of goal satisfaction, as in Reference [10]. However, molecular systems such as the MWT may have more than 10^{10} individual components in solution, so failures with any significant probability probably will occur in many individual components. The system must nevertheless be robust enough to operate correctly (to alarm or not to alarm) with probability approaching 1, even in the presence of some component failures. The watchdog timer design must be one in which we have confidence that, if the system it is monitoring fails, the MWT will detect and notify us, and that if the MWT notifies us that the monitored system has failed, then we can trust its accuracy.

2.5 Verifying the Goal Model

After the goal model was stabilized, we re-proved its internal correctness (the fact that the leaf goals imply the top-level goal) with the aid of the Isabelle proof assistant [45, 46]. In order to maximize the use of Isabelle's automated features and the readability of the result, we used a hybrid of human mathematics and automated theorem proving. Specifically, we formulated a list of eight CSL lemmas that are relatively simple (especially, not burdened with specific parameters of our design) and capture the higher-order logical aspects of our goals that are not readily amenable to automation. After proving these lemmas succinctly and transparently in Section 3.2 of the Supplemental Material, we used Isabelle in Section 4 of the Supplemental Material to prove that, given these lemmas, the goal model is internally correct.

The verification of the goal model is reported completely in the Supplemental Material.

More specifically, every CSL operator defines a state formula, so we represent CSL operators in Isabelle as paramatrized functions from CTMC states to the set {*true*, *false*}. For example, we can supply the $\mathcal{P}\Diamond$ operator with probability and time parameters α and t and a component CSL formula ϕ; the result is the CSL formula $\mathcal{P}_{\geq\alpha}\Diamond_{\leq t}\phi$, which can take on the values *true* or *false* at different CTMC states.

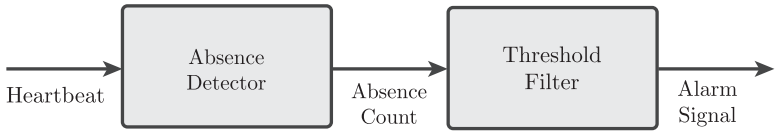

Fig. 3. High-level design.

We define rules for logical connectives in CSL using lambda expressions. For example, given two CSL formulae ϕ and ψ, we define conjunction as

$$\phi \wedge \psi = \lambda x\ \phi(x) \wedge \psi(x).$$

That is, $\phi \wedge \psi$ is a single function that returns true if both ϕ and ψ are true for the given argument. In combination with the operator definitions discussed previously, logical connectives like this are sufficient to encode arbitrary CSL statements and implications in Isabelle.

To construct a verified proof using this formalism in Isabelle, we provide a sequence of intermediate goals that link our assumptions and other known statements with our final goal. For each step in this proof, we provide Isabelle with assumptions and lemmas to reference and with proof methods to apply. Isabelle's powerful Sledgehammer tool [43, 48] automated significant parts of this process, making it easier to supply the correct facts and methods and take larger steps.

3 DESIGN

The goal-oriented requirements refinement and analysis described previously assigns responsibility for achieving the MWT's leaf goals to two system agents: the Absence Detector component and the Threshold Filter component, which outputs an Alarm signal. Figure 3 shows the high-level design with these two components. The connectors among the components reflect the intended flow of information from detection of individual heartbeats or their absences, to determining whether the incidence of faults exceeds a programmed threshold and issuing an alarm signal when that threshold is exceeded. We describe the mapping of leaf goals in the goal model to the components responsible for them and the detailed modeling of the components in the following.

3.1 Chemical Reaction Networks as a Programming Language

Our MWT design uses the language of chemical reaction networks (CRNs), which are abstract models of molecular processes in well-mixed solutions.[1] All CRNs in this article are *stochastic CRNs*, which model processes in which the presence or absence of very small numbers of certain types of molecules (e.g., a single copy of a viral genome in a living cell) may be significant. We henceforth omit "stochastic" from the terminology.

The CRN model, which goes back at least to 1940 [17], has three desirable features. First, it is mathematically simple. A CRN is a finite collection of *reactions*, each of which has a simple form, such as $A + C \xrightarrow{r} 2B + C$, where the *species* A, B, and C, are abstract types of molecules and the *rate constant* r is a positive real number representing the "propensity" of an A and a C that encounter one another to react, thereby being replaced by two Bs and a C in the solution. A species that, like the species C here, appears on both sides of a reaction is called a *catalyst* of the reaction. Catalysts are extremely important in biochemical processes, and they are extremely useful in our MWT construction. A *state* of a CRN is a vector specifying the number of each species present, and the dynamics of the CRN proceed as a continuous time Markov process with rates derived from the rate constants [1–3].

[1] This expository subsection is adapted from Reference [22].

A second feature of CRNs is that they are very general. Every algorithm can, in at least one sense, be efficiently simulated by a CRN [53].

A third desirable feature of CRNs, discovered recently, is that they can be implemented in a uniform way using DNA strand displacement reactions [54]. This is fortuitous, because dynamic systems in DNA nanotechnology, including DNA walkers and logic circuits, are typically implemented using DNA strand displacement reactions [52, 65]. The details of strand displacement reactions are not needed for this article, but it is relevant to note that they are relatively easy to implement in the laboratory and that they are easy to specify. There is a programming language, DSD, in which a large, expressive class of such reactions can be specified and compiled into abstract DNA sequences [36, 50]. CRNs have recently been used as a higher-level programming language that can be compiled into DSD [6, 14].

3.2 Stochastic Delay Ladders

The timing functions in our MWT are carried out by *stochastic delay ladders* (or simply *ladders*), which are CRNs introduced here having very predictable behaviors. The simplest type of ladder is a $k+1$-rung *unary ladder*, which consists of species $X_0, \ldots, X_k$ and the $2k$ reactions

$$\begin{aligned} X_i &\xrightarrow{u} X_{i+1} \quad (0 \le i < k), \\ X_i &\xrightarrow{r} X_0 \quad (0 < i \le k). \end{aligned} \tag{1}$$

We call $X_0, \ldots, X_k$ the *rungs*, u the *upward rate constant*, r the *reset rate constant*, and k the *height* of this ladder. The ladder is "unary," because each of its reactions has only one molecule on its left-hand side. The ladder is initialized with all its population on the bottom rung, i.e., a positive integer p instances of the species X_0 and no instances of X_i for $0 < i \le k$. Over time, members of this population "try to climb" the ladder, sometimes going up from one rung to the next and sometimes falling all the way back to the bottom rung. (This is a CRN implementation of the "frog in the well" Markov process [3].) The total population p of the ladder remains fixed throughout these climbing attempts.

Because the ladder Equation (1) is unary, its kinetic behavior is linear. This implies that a ladder with population p behaves exactly like an aggregate of p statistically independent ladders with population 1. (Most CRNs have nonlinear behavior and thus cannot be decomposed into independent, single-molecule nanodevices in this manner.) Since p is usually large (and often *very* large), this statistical independence enables us to predict with high confidence how long it will take (as a function of p, u, r, and k) for a given fraction of the population to simultaneously occupy the top rung of the ladder.

In practice, the rate at which reactions occur is governed more by catalysts than by rate constants. Thus, instead of the unary ladder Equation (1), we introduce *catalyzed ladders* of the form

$$\begin{aligned} X_i + U &\xrightarrow{1} X_{i+1} + U \quad (0 \le i < k), \\ X_i + R &\xrightarrow{1} X_0 + R \quad (0 < i \le k). \end{aligned} \tag{2}$$

The kinetics of stochastic CRNs make Equation (2) very similar to Equation (1). For example, if $\#X_i(t)$ is the number of X_i molecules at time t and $u(t)$ is the *concentration* of U molecules at time t (i.e., $u(t) = \#U(t)/V$, where V is the volume of the solution), then at time t the first reaction in Equation (1) takes place at rate $u\#X_i(t)$, while the first reaction in Equation (2) takes place at rate $u(t)\#X_i(t)$. In particular, if $u(t) = u$ and $r(t) = r$ are constant, then the unary ladder Equation (1) and the catalyzed ladder Equation (2) have identical statistical behaviors. Moreover, even if $u(t)$ and $r(t)$ fluctuate, but do so independently of the ladder's rung populations, the catalyzed ladder

Equation (2) enjoys the same decomposability into independent, population-1 ladders as the unary ladder Equation (1).

When controlling reaction rates by catalysts in the above manner, one often takes the rate constants to be 1, as we have done in Equation (2). In this case, the rate constants are omitted from the notation and assumed to be 1, as we do in the following.

It is now straightforward to specify the two main components of our MWT.

3.3 Absence Detector

This component detects when a heartbeat signal has not been present for a specified period of time. The heartbeat is a "pulse" of a specific molecular species H that is expected to be periodically output by the molecular application that is being monitored by the MWT. If the heartbeat is not detected by the MWT for an extended period of time, then we can conclude that the molecular application being monitored has failed. The Absence Detector is assigned responsibility for achieving the leaf goals *Avoid [Hdet when Heartbeat is not present]*, *Achieve [Hdet when Heartbeat is present]*, *Achieve [Initialize to Reset]*, *Achieve [Reset if Hdet]*, *Achieve [Threshold Delay if Reset]*, and *Achieve [Threshold if Hdet is absent]* as shown in Figure 2.

The Absence Detector component consists of the catalyzed ladder

$$L_i + U \rightarrow L_{i+1} + U \quad (0 \leq i < k),$$
$$L_i + H \rightarrow L_0 + H \quad (0 < i \leq k).$$

We also write Y for the top rung L_k of this ladder to emphasize its special role as the upward catalyst for the Threshold Filter described in the following.

3.4 Threshold Filter

This component detects when a target number of individual instances of the Absence Detector have reached the L_k state. The Threshold Filter trips an alarm if and only if enough Absence Detectors are in the L_k state to overcome the *constant* number of instances of the reset catalyst R of the Threshold Filter. The Threshold Filter is assigned responsibility for the leaf goals *Avoid [Alarm if Reset]*, *Achieve [Alarm if Threshold for some time]*, and *Avoid [Alarm until first Threshold]*.

The Threshold Filter consists of the catalyzed ladder

$$T_i + Y \rightarrow T_{i+1} + Y \quad (0 \leq i < k),$$
$$T_i + R \rightarrow T_0 + R \quad (0 < i \leq k).$$

We also write D for the top rung T_k of the Threshold Filter. This species D is the alarm species of the MWT. It is used by external modules to trigger a response. In previous work [22], we described a simple, one-time, Alarm response produced by amplifying the alarm molecular species. In this article, we will present a more powerful response, namely a Recovery component, after introducing a monitored system in Section 4.2.

4 DESIGN VERIFICATION

Formal design analysis provides some assurance that the behavior specified in the design matches the system's intended behavior. We must ensure that with very high probability when there is a heartbeat, the MWT does not alarm, and that with very high probability when there is no heartbeat, the MWT quickly alarms.

In this section, we first describe the analysis and verification techniques used to verify the design of the MWT and report the verification results. We check that the MWT works for a very long time in normal conditions (i.e., when the heartbeat from the monitored system is present) and show via

quantitative simulation that the MWT alarms as quickly as the client needs (indicated via the initial parameter values) when the heartbeat from the external system disappears.

We then demonstrate the functionality of the MWT by introducing a specific example of a system that needs to be monitored at runtime and verifying that its heartbeat behavior is correct. We connect the CRN model of the monitored system to that of the MWT and verify the correct functioning of the composed system. Finally, we describe how we extended the existing system to enable the MWT to not only detect the monitored system's failure at runtime, i.e., the absence of the expected heartbeat pulse, but also to trigger the monitored system's runtime recovery, and how we verified this additional capability.

4.1 Verifying the MWT Design

To check the correctness and robustness of the MWT design, we followed an incremental process of simulation of the CRN model for sanity checks and selection of likely parameter value ranges, followed by model checking of the CSL leaf goals on the CRN model across those parameter values. We also injected faults that had been previously discovered by analytical reasoning to confirm that the model checkers found them. To recap, the agents to which the leaf subgoals in the goal model are assigned—the Absence Detector and the Threshold Filter—are described in the CRN high-level programming language. The properties to be checked against the CRN model are the CSL formal specification of the leaf goals.

CRN input to verification tools. A strength of CRN as a language for molecular programming is that a CRN model can be readily imported into MATLAB's SimBiology package, allowing simulations to be run on it. We used SimBiology to understand the behavior and performance of our models and to debug them.

A CRN model can also be used as input into the probabilistic model checker PRISM [32], used previously to analyze molecular systems, e.g., in References [31, 33]. We used PRISM to verify that the MWT design satisfied the CSL properties derived from the goal model. The MWT model takes a number of parameters ranging from rate constants (of DNA reactions) to the length of the absence detector ladders (which are DNA strands). The parametrized design allowed us to automate testing across ranges of parameter values for optimization and verification.

Parameters. The values of the client parameters are specified by the client and depend on the system being monitored. For example, how quickly a heartbeat failure must be detected will vary among different applications. Additionally, there are modeling parameters that were needed to verify the realizability of the specified goals. An example is the number of each species of molecule represented in a particular model. The values for these parameters vary as the design space is explored. Their correct representations emerge from the formal analysis, mathematical proofs that satisfaction of the subgoals given the environmental assumptions satisfy the root goal, simulation of the composed system (monitored component and MWT), and the model checking results. Correct parametrization of the models was time-consuming and incremental. We used custom MATLAB scripts to generate models of arbitrary parameters that could integrate with the model checkers. These scripts along with some features provided by PRISM automated the exploration of the parameter space to discover models that provably satisfied our requirements.

Reusable MWT. Making the MWT reusable enables the composed system (monitored system and MWT) to recover from faults and then continue execution autonomously, without outside intervention. This is important if the MWT is to be used *in vivo*, for example. If the MWT is not reusable, then even if the monitored system recovers, it is no longer being monitored. The technical difficulty in moving from a throw-away MWT (one-time use) to a reusable MWT was the creation of a design for which the initial condition (the set of values for the parameters) could be restored autonomously. At initialization, we thus configured the MWT to begin with the majority

of rung molecules in the lower rungs of the ladder. When a heartbeat is detected, the absence detector enters this reset state again. This resets the MWT for reuse. We validated via simulation in SimBiology that the MWT is reusable rather than needing to be discarded after a single use. After having alarmed, it is reset when the monitored system begins issuing heartbeats again.

4.2 Verifying the Interaction of the MWT with a Monitored System

The question that underlies the verification of the interaction of the MWT with a monitored system is, "Could any such composed molecular system satisfy the assumptions we make on it and be used productively?" Given current model-checking limits, we verify the interaction at two very different molecular population scales, both of which are in realistic ranges of molecular counts.

First, we verify using *simulation and probabilistic model checking* that the interaction of an abstract monitored system with the MWT is correct with respect to the requirements and show that the assumptions we make on this hypothetical monitored system are realistic. This first version operates on an abstract signal received from a hypothetical monitored system. We simulate using SimBiology and model-check using PRISM that the MWT transforms the absence of a heartbeat signal into an alarm signal correctly. The size for which we can model-check it is small, with molecular counts up to 5. For example, 5 Absence Detectors produced a CTMC with over 150,000 states, while 10 Absence Detectors produced a CTMC with over 9 million states.

Second, we validate by *simulation* the correct interaction of an example monitored system with the MWT and show that the assumptions made on this specific monitored system are realistic. We first introduce an example of a monitored system and verify its correctness, using simulation and model checking. This entails verifying that the presence or absence of its heartbeat is correlated to the monitored system's health or failure. The example monitored system cannot be model-checked for all possible values of its parameters due to the size of the possible space; however, we simulated it with up to a total species population count of 100,000.

Introducing a system to be monitored. To demonstrate the capabilities of the MWT design, we used it to monitor the health of a standard molecular system, namely an oscillator. Chemical oscillators occur widely in nature, so are important, and synthetic molecular oscillators previously have been used as benchmarks in multiple projects, e.g., References [6, 15, 16, 23, 26, 35]. See Section 5 for more information. Another advantage of selecting the oscillator is that we could readily extend it to output a heartbeat (to test the normal case) and cause it to cease output of a heartbeat (to test the failure cases).

We used the Lotka-Volterra Three-phase Oscillator [12], which employs three species A, B, and C. Each of the three species corresponds to a single phase of the oscillator. The oscillator is initialized with the molecular count of one of the three species being high and the molecular counts of the other two species being low. After initialization, the oscillator will cycle between the phases following the order A to B to C and then back to A. As an example, consider the following case. If A is dominating and B and C have similar molecular counts, then reaction Equations (3) and (5) in the following equations are equally likely to occur. However, when reaction Equation (3) or reaction Equation (5) fires, the rates of all the reactions change, increasing the rate of reaction Equation (3) and decreasing the rate of reaction Equation (5). This continues until B is dominating, completing the transition to phase B. A similar sequence of events occurs for each phase transition.

We extended the CRN model for the stochastic three-phase oscillator with a heartbeat interface that produces a heartbeat (H) when the oscillator is healthy. A heartbeat interface is required to use the MWT to monitor the oscillator. The CRN for the oscillator plus its heartbeat interface is

$$A + B \xrightarrow{k} 2B + H, \tag{3}$$

$$B + C \xrightarrow{k} 2C, \tag{4}$$

$$C + A \xrightarrow{k} 2A, \tag{5}$$

$$H \xrightarrow{k_2} \emptyset. \tag{6}$$

To be useful, the heartbeat interface must cover all possible failure modes. In a stochastic setting, there are two possible faults in the oscillator that cause it to fail. First, if any of the three species A, B, or C has a molecular count of zero, the oscillations will stop and the oscillator will fail. Second, if the oscillator spends a large amount of time with all three species near a state of equilibrium, the oscillations will become negligible and desultory and the oscillator fail. While the oscillator is working correctly, a large number of H molecules will be created as reaction Equation (3) occurs. For this interface to be correct, it must produce fewer heartbeats in the case of a fault. For the first fault, if any of the three oscillator species has a count of zero, all oscillator reactions will quickly have a rate of 0, causing a stop in the production of heartbeat molecules. For the second fault, if all three oscillator species are in equilibrium, then the H species will fall to a roughly constant amount low enough to cause the Absence Detector to activate.

Checking correlation of heartbeat with oscillator's state. In "real-world" scenarios, it is the monitored system's responsibility to provide a correct heartbeat, meaning that the MWT assumes that the heartbeat accurately reflects the normal or failed state of the monitored system. However, to confirm that the MWT operates correctly and robustly, we had to first confirm that the oscillator outputs a correct heartbeat. We thus had to check (1) that the oscillator's health correlates with the presence (healthy) or absence (unhealthy) of heartbeat molecules at its interface, and (2) that the MWT's behavior correlates with the presence or absence of heartbeat molecules over a period of time at its interface with the monitored system.

A state of the monitored system can be healthy or unhealthy. Informally, in a healthy state, a heartbeat will be sent within a reasonable time or the state will quickly become unhealthy. In an unhealthy state, no heartbeat will be sent within a reasonable time or the state will quickly become healthy. We define a state to be *healthy* at time t as A, B, C > 0 AND $(A-B)^2 + (B-C)^2 + (C-A)^2 > \tau$, where τ is defined to ensure that the oscillator is deemed unhealthy if its species counts approach equilibrium.

The three properties to be verified are:

Achieve [Produce heartbeats while healthy]

$$\mathcal{P}_{\geq 1}[\Box(\text{healthy} \Rightarrow \mathcal{P}_{\geq 1-\delta_1}[\Diamond_{\leq t_1}((\text{hbHigh} \ \vee \ \neg\text{healthy})])]$$

Avoid [Produce heartbeats while unhealthy],

$$\mathcal{P}_{\geq 1}[\Box(\neg\text{healthy} \Rightarrow \mathcal{P}_{\geq 1-\delta_2}[\Diamond_{\leq t2}(\mathcal{P}_{\geq 1-\delta_3}[\text{hbLow } \mathcal{W} \ (\mathcal{P}_{\geq 1-\delta_4}[\Box_{\leq t_3}\text{healthy}])])])],$$

Heartbeat decays

$$\mathcal{P}_{\geq 1}[\Box(\text{hbHigh} \Rightarrow \mathcal{P}_{1-\delta_5}[\Diamond_{\leq t4}\neg\text{hbHigh}])].$$

We simulated in SimBiology the oscillator and heartbeat interface with a range of initial counts of A, B, and C up to 1000 (e.g., 80% in A, 10% in B and C) and an initial count of 0 for H, and checked that these three properties held in the simulations. The simulations demonstrated that the presence or absence of a heartbeat correlates with the health of the oscillator in both of the two failure modes. Using a CTMC model of the oscillator with total population of 200, we then verified in PRISM that the oscillator plus heartbeat interface satisfied the goals. The model checker verified true for the three CSL properties above.

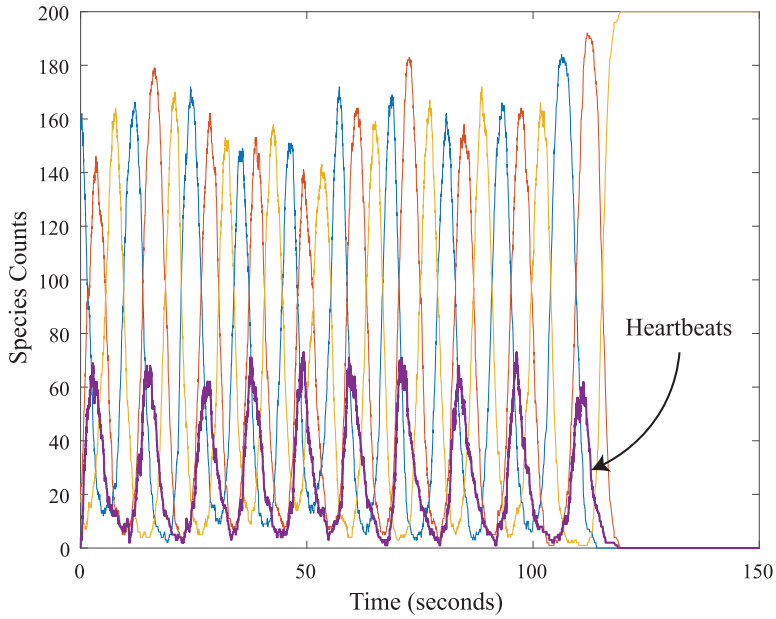

Fig. 4. A simulation of the oscillator with the heartbeat interface.

Figure 4 shows a simulation of the oscillator for the first failure mode, in which one of the species counts went to 0. Since no oscillations could occur, heartbeats ceased to be produced by the oscillator.

4.3 MWT Triggers Oscillator's Runtime Recovery

A system is more robust if it can autonomously recover from a failure. Beyond detecting and reporting the failure of the monitored system, we also sought to use the MWT's Alarm signal to trigger autonomous recovery in a monitored system after it has failed. To demonstrate this, we constructed a recovery component that, upon receiving the MWT's Alarm, will recover the oscillator from either of its failure modes.

The CRN for the oscillator's recovery module is

$$D + A \xrightarrow{k} D + B, \tag{7}$$

$$D + B \xrightarrow{k} D + C, \tag{8}$$

$$D + C \xrightarrow{k_2} D + A, \tag{9}$$

where the third reaction has a different rate from the other two.

These reactions are triggered by the presence of the MWT Alarm component's output signal (the D's) produced when the MWT detects that the oscillator has failed. In both the case where the oscillator fails due to running out of the species A, B, or C, and the case where it fails because it reached an equilibrium state (an equal number of A's, B's, and C's) such that heartbeats stop, these reactions will recover the oscillator. The recovery "jump-starts" the oscillator and, when the heartbeat starts up again, the Alarm signal (the D's) fade away. To check the correct behavior of the recovery capability, we ran simulations with populations of 100 to 1,000 molecules for the oscillator and varying percentages of D molecules. The MWT's Alarm signal correctly triggered the oscillator recovery.

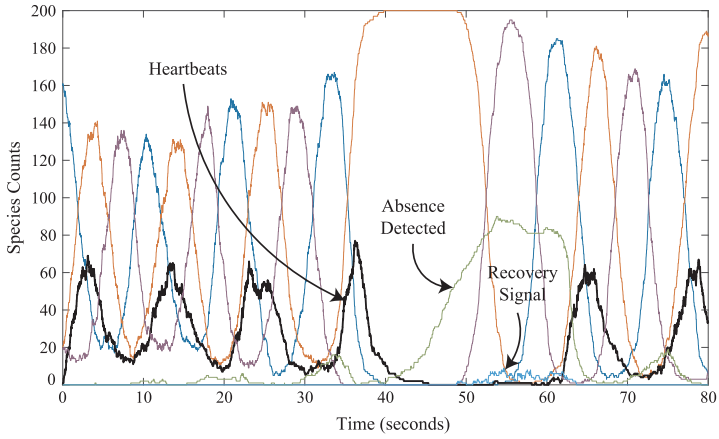

Fig. 5. A simulation of the MWT detecting the oscillator's failure and triggering its recovery.

Figure 5 shows a single stochastic simulation of the composed system. Initially, the oscillator works and produces heartbeats. During this time, the absence-detected signal remains low. However, once the oscillator fails and the heartbeats stop, the absence is quickly detected and the recovery signal is released to trigger the oscillator to recover. Shortly after the recovery signal is turned on, the oscillator begins normal operation again.

Generalizing to other applications. Usage of the MWT is intended to be broad. The MWT is designed to work with any molecular system that needs to be monitored for the absence of a heartbeat, meaning that it is independent of how the heartbeats are input to it. The oscillator provides an example of how, given a heartbeat interface from a client-monitored system specified as a rate at which heartbeats are produced and in what quantity, we can construct an MWT to monitor it.

To use our MWT to monitor a system, the client-monitored system (here, the oscillator) describes what it needs the MWT to do by specifying a *polytope*, that is a multi-dimensional space, defined by four parameters. The four client-provided parameters are: u, the minimum time between a heartbeat and an alarm; v, the maximum time between a heartbeat and an alarm; ε, the probability of error allowed by the u delay; and δ, the probability of error in achieving the v delay. For example, a client might specify that the MWT should allow a minimum of 10s and a maximum of 20s after a heartbeat before an alarm, and it must achieve these time bounds with probabilities of 95% for both. Thus, u is 10, v is 20, and ε and δ are 5%. The client also specifies the minimum and maximum size of the heartbeat pulse. If the client gives us parameter values from within this space, then we **will provide** a design model for an MWT that satisfies the goal diagram.

There are other, internally generated (rather than client-defined) parameters that formalize the goal diagram's constraints. For example, ε is broken into two internally generated parameters to enable the goal proofs. Thus, in addition to the four client-provided parameters, there are 22 internally generated parameters specifying probability and timing constraints. These are listed in the Supplemental Material.

4.4 Mapping to Molecules

We have designed and verified our MWT at the CRN level of abstraction, but it is useful to estimate the feasibility of actually implementing our design in DNA. For this purpose we used the compiler reported in a very recent paper by Badelt et al. [6]. Using this compiler and its encoding of the translation scheme of Chen et al. [14], we compiled the CRN for a small MWT and oscillator

into DNA strands. The MWT has a four-rung Absence Detector ladder and a five-rung Threshold Detector ladder and includes the three recovery reactions. As reported in the Supplemental Material, this MWT alone compiles to 122 distinct DNA strands. The combination of this MWT with the CRN for the three-phase oscillator (modified to include heartbeat production) compiles to 147 distinct DNA strands. DNA strand displacement systems of this size are already feasible for laboratory implementation. For example, Qian and Winfree [51] reported the successful implementation of a 130-strand DNA displacement device. Larger MWTs (ones with longer stochastic delay ladders) will be feasible in the near future.

4.5 Discussion

To summarize, the MWT is a programmable, molecular safety mechanism. In the creation of the design for the MWT, this article has proposed a new use of software engineering techniques and tools in a development process that can be applied more generally to create other molecular systems. The goal-oriented requirements engineering process described in Section 2 systematically develops the requirements for a molecular programmed system, using continuous stochastic logic (CSL) to specify the requirements (leaf goals) and to verify the goal refinement with machine-checkable proofs. The assignment of requirements to the components' designs responsible for achieving them is described in Section 3. The designs are formally specified as stochastic chemical reaction networks (CRNs). CRNs recently have become widely used to specify programmed molecular systems, since compilers now exist from CRNs into lower-level (DNA-strand-level) designs and from there into molecules. Modeling the designs as CRNs supports both simulation and verification, and both MATLAB's SimBiology and the PRISM probabilistic model checker accept CRN input. This section has described the simulation and verification of the MWT, first as a stand-alone device; second, when connected to a device, such as the oscillator, that needs to be monitored for a failure event; and third, when connected to a device that needs to be triggered by the MWT to also recover from its failure event at runtime.

5 RELATED WORK

In this section, we briefly describe additional related work in molecular design software, model checking, and molecular oscillators.

Molecular design software. Other design tools for DNA computational devices exist but operate at a very detailed design level. Two open-source software tools are CaDNAno [19] and CANDO [28]. They are widely used to design, debug, and optimize the stability and physical properties (torque, flexibility, energy wells) of two-dimensional and three-dimensional DNA origami structures and operate at a much lower level than our design considerations here.

Model checking. In related work, Kwiatkowska and Thachuk described the probabilistic verification of CRNs for biological systems using the probabilistic model checker PRISM [33]. Their work showed the benefits of probabilistic model checking for molecular systems and informed our work for the MWT. PRISM interfaces with Visual DSD, a design tool for DNA strand displacement [36].

For large systems, including molecular ones, there is a disconnect between the size of the model that can be automatically checked and the system. One of the problems we face is how to prune the model such that we can do meaningful model checking. Pavese, Barberman, and Uchitel described how to develop partial explorations of a system model automatically [49]. Their technique has promise for use in molecular programs that we hope to explore. Since many molecular programs deal with extremely large, if not infinite, state spaces, probabilistic model checking on partial system explorations might provide bounds on the reliability of a molecular system that is too large to model check.

Stochastic models for systems often have distributions that are empirically determined or only partially known. Moreover, small differences between stochastic models' parameter values and their real-world counterparts can change the results of verification. Meedeniya et al. have used Monte Carlo simulations to generate a reliability evaluation of a probabilistic model of an antilock brake system with uncertain parameters [42]. Su, Chen, Feng, and Rosenblum recently extended previous work by Su and Rosenblum [57] on perturbations in model-checking parameters in discrete-time Markov chains to allow model checking on time-bounded CTMCs with imprecise values for transition rates [56]. Since molecular systems have imprecise reaction rates, determining the effects of parameter variance on the models is important, and the applicability of their approaches to programmed molecular systems merits investigation.

More broadly, there has been significant recent progress in modeling biological or chemical systems. Yordanov et al. formalized and encoded DNA computing to allow use of Satisfiability Modulo Theories (SMT) [64]. Fisher, Harel, and Henzinger performed computational modeling of biological systems as reactive systems [24]. Hetherington et al. and Sumner et al. composed an advanced computational model of a biological system from sub-models describing its different aspects [25, 58]. David et al. created translators to convert SimBiology models for biological systems into CTMCs for stochastic model checking or into ODEs for simulation [16].

Molecular oscillators. Hori and Murray, in a recent paper on synthetic biochemical oscillators, stated that, "The reliable engineering of oscillators is an important milestone towards robust synthesis of more complex dynamical circuits in synthetic biology" [26]. Gene regulatory networks, for example, use oscillators, and Fern et al. recently reported the use of timer circuits to precisely coordinate chemical events *in vitro* [23]. Three-phase oscillators seem to have been first reported in Reference [34] and more recently in References [12, 13, 35]. The Two-phase Lotka-Volterra Oscillator also has been studied in the context of DNA strand displacement in References [35, 54]. Ballarini, Mardare, and Mura and Ballarini and Guerriero presented analyses of the three-phase oscillator using PRISM and described both of the failures modes that our MWT design successfully detects [8, 9]. Srinivas et al. recently implemented in DNA a new oscillator specified by CRNs using a compiler they created to translate the CRNs into DNA strands [55].

6 CONCLUSION

Monitoring the health of programmed molecular systems at runtime is critically important. Envisioned applications such as biocompatible diagnostic systems and smart-drug therapy systems will need such monitoring capabilities to operate safely. Using goal-oriented requirements engineering, machine-checked proofs, reaction network modeling, stochastic simulation, and probabilistic model checking, we have designed and verified an MWT that can monitor a molecular system at runtime, detect when the heartbeat signal from the monitored system stops, and alarm to trigger its recovery. The MWT is modular, designed to operate correctly in the probabilistic chemical environment, and robust to failure-prone components. Using chemical reaction networks as a programming language, we have implemented both the MWT and a monitored system (a molecular oscillator) as chemical reaction networks. We have demonstrated the MWT's capabilities by showing that the MWT reliably detects failures of the oscillator and triggers its recovery at runtime.

Many other programmed molecular systems will be needed and developed in the future. The MWT is an example of a cybermolecular system, a molecular programmed system that senses and controls its environment, including other molecular systems. Cybermolecular systems and bio-compatible computing devices are moving rapidly from the laboratory to widespread usage in daily life. We hope that our software-engineering-inspired approach to designing and verifying the molecular watchdog timer can assist in the future design of predictable and safe molecular systems.

ACKNOWLEDGMENTS

We thank Samik Basu, Gianfranco Ciardo, Anthony Finkelstein, Carlo Ghezzi, Marta Kwiatkowska, Axel van Lamsweerde, Paul Rothemund, and Erik Winfree for useful discussions. We thank Jeremy Avigad and Johannes Hölzl for useful suggestions on theorem provers. We thank the anonymous referees for remarks that improved the article.

REFERENCES

[1] David F. Anderson and Thomas G. Kurtz. 2011. Continuous time markov chain models for chemical reaction networks. In *Design and Analysis of Biomolecular Circuits*, Heinz Koeppl, Gianluca Setti, Mario di Bernardo, and Douglas Densmore (Eds.). Springer, 3–42.

[2] David F. Anderson and Thomas G. Kurtz. 2015. *Stochastic Analysis of Biochemical Systems*, Vol. 1.2. Springer International Publishing.

[3] Krishna B. Athreya and Soumendra N. Lahiri. 2006. *Measure Theory and Probability Theory*. Springer.

[4] Algirdas Avizienis, J.-C. Laprie, Brian Randell, and Carl Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Depend. Secure Comput.* 1, 1 (2004), 11–33.

[5] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. 2000. Model-checking continuous-time Markov chains. *ACM Trans. Comput. Logic* 1, 1 (2000), 162–170.

[6] Stefan Badelt, Seung Woo Shin, Robert F. Johnson, Qing Dong, Chris Thachuk, and Erik Winfree. 2017. A general-purpose CRN-to-DSD compiler with formal verification, optimization, and simulation capabilities. In *Proceedings of the 23rd International Conference on DNA Computing and Molecular Programming* (Lecture Notes in Computer Science). 232–248.

[7] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press.

[8] Paolo Ballarini and Maria Luisa Guerriero. 2010. Query-based verification of qualitative trends and oscillations in biochemical systems. *Theor. Comput. Sci.* 411, 20 (2010), 2019–2036.

[9] Paolo Ballarini, Radu Mardare, and Ivan Mura. 2009. Analysing biochemical oscillation through probabilistic model checking. *Electron. Notes Theor. Comput. Sci.* 229, 1 (2009), 3–19.

[10] Antoine Cailliau and Axel van Lamsweerde. 2013. Assessing requirements-related risks through probabilistic goals and obstacles. *Requir. Eng.* 18, 2 (2013), 129–146.

[11] Antoine Cailliau and Axel van Lamsweerde. 2014. Integrating exception handling in goal models. In *Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE'14)*. 43–52.

[12] Luca Cardelli. 2009. Artificial biochemistry. In *Algorithmic Bioprocesses*, Anne Condon, David Harel, Joost N. Kok, Arto Salomaa, and Erik Winfree (Eds.). Springer, Berlin, 429–462.

[13] Luca Cardelli. 2013. Two-domain DNA strand displacement. *Math. Struct. Comput. Sci.* 23, 2 (2013), 247–271.

[14] Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. 2013. Programmable chemical controllers made from DNA. *Nature Nanotechnol.* 8, 10 (2013), 755–762.

[15] Neil Dalchau, Georg Seelig, and Andrew Phillips. 2014. Computational design of reaction-diffusion patterns using DNA-based chemical reaction networks. In *Proceedings of the 20th International Conference on DNA Computing and Molecular Programming*. 84–99.

[16] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, and Sean Sedwards. 2015. Statistical model checking for biological systems. *Int. J. Softw. Tools Technol. Transfer* 17, 3 (2015), 351–367.

[17] Max Delbrück. 1940. Statistical fluctuations in autocatalytic reactions. *J. Chem. Phys.* 8, 1 (1940), 120–124.

[18] Shawn M. Douglas, Ido Bachelet, and George M. Church. 2012. A logic-gated nanorobot for targeted transport of molecular payloads. *Science* 335, 6070 (2012), 831–834.

[19] Shawn M. Douglas, Adam H. Marblestone, Surat Teerapittayanon, Alejandro Vazquez, George M. Church, and William M. Shih. 2009. Rapid prototyping of 3D DNA-origami shapes with caDNAno. *Nucl. Acids Res.* 58, 24 (2009), 1–6.

[20] Samuel Jay Ellis. 2014. *Designing a Molecular Watchdog Timer for Safety Critical Systems*. Master's Thesis. Iowa State University.

[21] Samuel Jay Ellis. 2017. *Devices for Safety-critical Molecular Programmed Systems*. Ph.D. Dissertation. Iowa State University.

[22] Samuel J. Ellis, Eric R. Henderson, Titus H. Klinge, James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, Divita Mathur, and Andrew S. Miner. 2014. Automated requirements analysis for a molecular watchdog timer. In *Proceedings of the 29th International Conference on Automated Software Engineering*. ACM, 767–778.

[23] Joshua Fern, Dominic Scalise, Angelo Cangialosi, Dylan Howie, Leo Potters, and Rebecca Schulman. 2017. DNA strand-displacement timer circuits. *ACS Synth. Biol.* 6, 2 (2017), 190–193.

[24] Jasmin Fisher, David Harel, and Thomas A. Henzinger. 2011. Biology as reactivity. *Commun. ACM* 54, 10 (2011), 72–82.
[25] J. Hetherington, T. Sumner, R. M. Seymour, L. Li, M. Varela Rey, S. Yamaji, P. Saffrey, O. Margoninski, I. D. L. Bogle, A. Finkelstein, and A. Warner. 2012. A composite computational model of liver glucose homeostasis. I. Building the composite model. *J. Roy. Soc. Interface* 9, 69 (2012), 689–700.
[26] Yutaka Hori and Richard M. Murray. 2015. Engineering principles of synthetic biochemical oscillators with negative cyclic feedback. In *Proceedings of the 54th IEEE Conference on Decision and Control (CDC'15)*. 584–589.
[27] Hua Jiang, Marc Riedel, and Keshab Parhi. 2011. Synchronous sequential computation with molecular reactions. In *Proceedings of the 48th Design Automation Conference*. ACM, IEEE, 836–841.
[28] Do-Nyun Kim, Fabian Kilchherr, Hendrik Dietz, and Mark Bathe. 2012. Quantitative prediction of 3D solution shape and flexibility of nucleic acid nanostructures. *Nucl. Acids Res.* 40, 7 (2012), 2862–2868.
[29] John Knight. 2012. *Fundamentals of Dependable Computing for Software Engineers*. CRC Press.
[30] Phil Koopman. 2014. A Case Study of Toyota Unintended Acceleration and Software Safety. Retrieved on February 23, 2019 from https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf.
[31] M. Kwiatkowska. 2014. Challenges in automated verification and synthesis for molecular programming. In *Essays for the Luca Cardelli Fest*. Microsoft Research, 155–170.
[32] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (Lecture Notes in Computer Science)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.), Vol. 6806. Springer, 585–591.
[33] Marta Kwiatkowska and Chris Thachuk. 2014. Probabilistic model checking for biology. *Softw. Syst. Saf.* 36 (2014), 165–189.
[34] Michael Lachmann and Guy Sella. 1995. *The Computationally Complete Ant Colony: Global Coordination in a System with No Hierarchy*. Springer, Berlin, 784–800.
[35] Matthew R. Lakin, Simon Youssef, Luca Cardelli, and Andrew Phillips. 2012. Abstractions for DNA circuit design. *J. Roy. Soc. Interface* 9, 68 (2012), 470–486.
[36] Matthew R. Lakin, Simon Youssef, Filippo Polo, Stephen Emmott, and Andrew Phillips. 2011. Visual DSD: A design and analysis tool for DNA strand displacement systems. *Bioinformatics* 27, 22 (2011), 3211–3213.
[37] Emmanuel Letier and Axel van Lamsweerde. 2004. Reasoning about partial goal satisfaction for requirements and design engineering. In *Proceedings of the 12th International Symposium on Foundations of Software Engineering*. ACM, 53–62.
[38] Nancy G. Leveson. 1995. *Safeware: System Safety and Computers*. Addison-Wesley Publishing Company.
[39] Xiaowei Liu, Yan Liu, and Hao Yan. 2013. Functionalized DNA nanostructures for nanomedicine. *Israel J. Chem.* 53, 8 (2013), 555–566.
[40] Robyn Lutz, Jack Lutz, James Lathrop, Titus Klinge, Eric Henderson, Divita Mathur, and Dalia Abo Sheasha. 2012. Engineering and verifying requirements for programmable self-assembling nanomachines. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE, 1361–1364.
[41] Robyn R. Lutz, Jack H. Lutz, James I. Lathrop, Titus H. Klinge, Divita Mathur, Donald M. Stull, Taylor G. Bergquist, and Eric R. Henderson. 2012. Requirements analysis for a product family of DNA nanodevices. In *Proceedings of the 20th International Conference on Requirements Engineering*. IEEE, 211–220.
[42] Indika Meedeniya, Irene Moser, Aldeida Aleti, and Lars Grunske. 2014. Evaluating probabilistic models with uncertain model parameters. *Softw. Syst. Model.* 13, 4 (2014), 1395–1415.
[43] Jia Meng, Claire Quigley, and Lawrence C. Paulson. 2006. Automation for interactive proof: First prototype. *Info. Comput.* 204, 10 (2006), 1575–1596.
[44] NASA. 1988. Computers in Spaceflight: The NASA Experience. Retrieved on February 23, 2019 from https://history.nasa.gov/computers/contents.html.
[45] Tobias Nipkow and Gerwin Klein. 2014. *Concrete Semantics* (1st ed.). Springer International Publishing.
[46] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL* (1st ed.). Lecture Notes in Computer Science, Vol. 2283. Springer-Verlag.
[47] Bashar Nuseibeh. 2001. Weaving together requirements and architectures. *IEEE Comput.* 34, 3 (2001), 115–117.
[48] Lawrence C. Paulson and Kong Woei Susanto. 2007. Source-level proof reconstruction for interactive theorem proving. In *Theorem Proving in Higher Order Logics*, Klaus Schneider and Jens Brandt (Eds.). Springer, Berlin, 232–245.
[49] Esteban Pavese, Víctor Braberman, and Sebastián Uchitel. 2016. Less is more: Estimating probabilistic rewards over partial system explorations. *ACM Trans. Softw. Eng. Methodol.* 25, 2, Article 16, 47 pages.
[50] Andrew Phillips and Luca Cardelli. 2009. A programming language for composable DNA circuits. *J. Roy. Soc. Interface* 6, 4 (2009), S419–S436.
[51] Lulu Qian and Erik Winfree. 2011. Scaling up digital circuit computation with DNA strand displacement cascades. *Science* 332, 6034 (2011), 1196–1201.

[52] David Soloveichik. 2008. *Molecules Computing: Self-assembled Nanostructures, Molecular Automata, and Chemical Reaction Networks*. Ph.D. Dissertation. California Institute of Technology.
[53] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. 2008. Computation with finite stochastic chemical reaction networks. *Natural Comput.* 7, 4 (2008), 615–633.
[54] David Soloveichik, Georg Seelig, and Erik Winfree. 2010. DNA as a universal substrate for chemical kinetics. *Proc. Natl. Acad. Sci.* 107, 12 (2010), 5393–5398.
[55] Niranjan Srinivas, James Parkin, Georg Seelig, Erik Winfree, and David Soloveichik. 2017. Enzyme-free nucleic acid dynamical systems. *Science* 358, 6369 (2017).
[56] Guoxin Su, Taolue Chen, Yuan Feng, and David S. Rosenblum. 2017. ProEva: Runtime proactive performance evaluation based on continuous-time Markov chains. In *Proceedings of the 39th International Conference on Software Engineering (ICSE'17)*. 484–495.
[57] Guoxin Su and David S. Rosenblum. 2013. Asymptotic bounds for quantitative verification of perturbed probabilistic systems. In *Formal Methods and Software Engineering*. Lecture Notes in Computer Science, Vol. 8144. Springer, 297–312.
[58] T. Sumner, J. Hetherington, R. M. Seymour, L. Li, M. Varela Rey, S. Yamaji, P. Saffrey, O. Margoninski, I. D. L. Bogle, A. Finkelstein, and A. Warner. 2012. A composite computational model of liver glucose homeostasis. II. Exploring system behaviour. *J. Roy. Soc. Interface* 9, 69 (2012), 701–706.
[59] T. Tun, R. Lutz, B. Nakayama, Y. Yu, D. Mathur, and B. Nuseibeh. 2015. The role of environmental assumptions in failures of DNA nanosystems. In *Proceedings of Workshop on Complex Faults and Failures in Large Software Systems*.
[60] Axel van Lamsweerde. 2009. *Requirements Engineering—From System Goals to UML Models to Software Specifications*. Wiley.
[61] Axel van Lamsweerde and Emmanuel Letier. 2000. Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Softw. Eng.* 26, 10 (2000), 978–1005.
[62] Michael W. Whalen, Andrew Gacek, Darren D. Cofer, Anitha Murugesan, Mats Per Erik Heimdahl, and Sanjai Rayadurgam. 2013. Your "What" is my "How": Iteration and hierarchy in system design. *IEEE Softw.* 30, 2 (2013), 54–60.
[63] John C. Wooley and Herbert S. Lin. 2005. *Catalyzing Inquiry at the Interface of Computing and Biology*. National Academies Press.
[64] Boyan Yordanov, Christoph M. Wintersteiger, Youssef Hamadi, and Hillel Kugler. 2013. SMT-based analysis of biological computation. *NASA Formal Methods* 7871 (2013), 78–92.
[65] David Yu Zhang and Georg Seelig. 2011. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature Chem.* 3, 2 (2011), 103–113.

Received September 2017; revised October 2018; accepted November 2018