Requirements Analysis for a Product Family of DNA Nanodevices

Robyn R. Lutz*† , Jack H. Lutz*, James I. Lathrop*, Titus H. Klinge*, Divita Mathur‡,
D. M. Stull*, Taylor G. Bergquist*, and Eric R. Henderson‡
*Department of Computer Science
Iowa State University, Ames, IA 50011 USA
{rlutz, lutz, jil, tklinge, dstull, knexer}@iastate.edu
†Jet Propulsion Laboratory
California Institute of Technology, Pasadena, CA 91104 USA
‡Department of Genetics, Development, and Cell Biology
Iowa State University, Ames, IA 50011 USA
{divita, telomere}@iastate.edu

***Abstract*—DNA nanotechnology uses the information processing capabilities of nucleic acids to design self-assembling, programmable structures and devices at the nanoscale. Devices developed to date have been programmed to implement logic circuits and neural networks, capture or release specific molecules, and traverse molecular tracks and mazes.**

Here we investigate the use of requirements engineering methods to make DNA nanotechnology more productive, predictable, and safe. We use goal-oriented requirements modeling to identify, specify, and analyze a product family of DNA nanodevices, and we use PRISM model checking to verify both common properties across the family and properties that are specific to individual products. Challenges to doing requirements engineering in this domain include the error-prone nature of nanodevices carrying out their tasks in the probabilistic world of chemical kinetics, the fact that roughly a nanomole (a 1 followed by 14 0s) of devices are typically deployed at once, and the difficulty of specifying and achieving modularity in a realm where devices have many opportunities to interfere with each other. Nevertheless, our results show that requirements engineering is useful in DNA nanotechnology and that leveraging the similarities among nanodevices in the product family improves the modeling and analysis by supporting reuse.

***Keywords*-requirements modeling and analysis, DNA nanotechnology, goal-oriented, product families, model checking.**

I. Introduction

DNA nanotechnology, pioneered by Seeman in the 1980s [1] and now growing explosively, undertakes to *program matter* to do our bidding at molecular and atomic scales. Exploiting the information processing capabilities of nucleic acids enables researchers to design complex structures and devices that assemble themselves from molecular components. Research in recent years has shown that DNA tile self-assembly can implement algorithms [2] and enjoys a very strong form of Turing universality [3]; that DNA strand displacement reactions can implement Boolean circuits [4], [5] neural networks [6], and molecular robots [7], [8], [9]; and that DNA origami can create two- and three-dimensional structures that can serve as targeted drug-delivery devices or "nano-breadboards" with hundreds of uniquely addressable sites, separated by just 6 nanometers, to which sensors and other features may be attached [10], [11]. This programming of matter is thus programming in the literal sense. As such it presents new opportunities and challenges for computer science and software engineering.

As DNA nanotechnology progresses (with doubling times of number and complexity of published nanodevices reminiscent of Moore's law) and extends from basic science to applications (with early work already underway in medicine [12], [9] and computer chip design [13]), increasingly sophisticated methods are required for managing and reasoning about complex nanosystems and their behaviors. DNA nanodevices carry out their tasks in the probabilistic, error-prone world of chemical kinetics. They are inherently distributed. Each *instance* of a nanodevice consists of a few (up to a few hundred to date, a few thousand in the near future) carefully designed molecular types, but the number of instances deployed, all in the same solution, is typically on the order of a nanomole (6×10^{14}). Hierarchical construction of complex nanosystems from modular nanodevices must take account of the fact that these components are interacting at distance scales in which the speed of diffusion is supersonic, so that components encounter one another randomly and frequently. Most DNA nanohnology experiments are already so complex that their initial *designs* require nontrivial computations with software tools such as caDNAno [14] or the DSD programming language [15]. Probabilistic model checking has also been used to make preliminary analyses of experimental designs [16], [17]. These design-stage computations, much faster and cheaper than the experiments themselves, are essential to the predictability, and hence the productivity, of DNA nanotechnology. As applications emerge in safety-critical areas, they will also be essential to the safety of DNA nanotechnology.

This paper investigates the use of requirements engineering methods to make DNA nanotechnology more productive, predictable, and safe. Our running example is a particular product family of DNA nanodevices, i.e., a set of nanodevices that have a high degree of commonality and some key variations among them [18], [19], [20]. This product family is small, consisting initially of just three kinds of DNA origami pliers, nanomechanical devices developed by Kuzuya, et al. [21], to sense (detect) the presence of target molecules in a solution. While we do not expect an off-the-shelf framework to suffice completely in this domain, we use van Lamsweerde's goal-oriented requirements engineering [22] as our framework for identifying the requirements that are common across this sensing product family and the requirements that are specific to individual nanodevices. The goal modeling of our product family shows that it is useful in this setting to augment the AND/OR goal diagrams of [22] to allow goal refinements that are mediated by threshold functions rather than ANDs or ORs. The obstacle analysis of our product family uncovers a new failure mode in which one of the nanodevices fails to detect target molecules when they are present at concentrations so high that they overwhelm the nanodevice.

Following the requirements analysis of the sensing product family, we add a new DNA nanodevice to the family. This new device, a zippered DNA tweezers developed by Landon, et al. [23], is a very recent variant of the first DNA strand-displacement nanomechanical device, the DNA tweezers of Yurke, et al. [24]. The requirements evolution brought on by this extension of the sensor product family seems to require little change to the goal model, until obstacle analysis reveals a new requirement for modularity of the zippered DNA tweezers in order that they can be integrated into complex hierarchical DNA nanomachines. This requirement turns out to also apply to one of the kinds of DNA origami pliers already in the product family. The extension of the sensing product family thus leads to a more complete set of requirements, even for the original product family.

Having thus identified the requirements, we construct a behaviorall model of the sensing product family and use the PRISM model checker [25] to verify that it satisfies both requirements that are common across the entire product family and requirements that are specific to one or more DNA nanodevices in the family. Our results indicate that requirements engineering improves the design of DNA nanodevices and that leveraging the commonalities among nanodevices increases both the efficiency and the quality of the requirements analysis.

II. Requirements Modeling and Analysis

We investigate the requirements for a product family of sensing DNA nanodevices where each member of the family is customized to sense, or detect, a different molecular or RNA target in a solution. Such nanodevices are of interest as future candidates for diagnosing diseases in the human body. The nanodevices are composed of DNA origami structures. Being bio-compatible, such nanodevices might be able to be deployed in the human body for in situ diagnosis, based on their ability to sense the presence of specific molecules in their operational environment. For example, an envisioned application is sensing the presence or absence of harmful molecules in human cells.

Each member of the product family is a nanodevice, and each nanodevice is itself composed of a very large number of self-assembled instances of that nanodevice. They must *autonomously detect and respond* to the presence or absence of a target by a shape change (e.g., from open jaws to closed jaws) and/or by fluoresence. The presence or absence of the target thus becomes observable and measurable to a technician in the laboratory using atomic force microscopy (AFM) or fluorescence spectroscopy.

A. A product family of DNA nanodevices

The product family of DNA nanodevices that we initially studied had three members in it: simple pliers, G-zipper pliers, and unzipper pliers. In early work we described the first member of the sensing product family, simple DNA pliers that close to grip a target molecule of SA (streptavidin tetramer) [26]. We show the figure from that paper again here as Fig. 1 because it so clearly illustrates some shared behavior that we will describe below.

The nanodevice instances are DNA origami structures programmed to self-assemble in the shape of *pliers* in the open position. Fig. 1(a) shows the open pliers as yellow crosses with arrows pointing to them. The pliers are programmed to, once assembled, capture a particular molecule of interest by closing it in their jaws, as shown in Fig. 1(b). The pliers should then release the molecule when unset strands of DNA are added to the solution, as shown in Fig. 1(c).

We later studied two additional sensing nanodevices and realized that the three products shared many common features, with a few key variations among them [18], [19], [20]. The G-zipper pliers (which use a G-quadriplex that acts as a zipper) close to grip a metal ion such as sodium (NA^+). The unzipper pliers begin in the closed position and "unzip" when they encounter human micro RNA (mRNA). We describe below the results from goal modeling and obstacle analysis on this product family, and the discovery of a significant, common robustness requirement.

We initially used the FAST (Family-Oriented Abstraction, Specification and Translation) process [18] to perform an informal commonality and variability analysis (CVA) of three DNA sensing nanodevices described in the literature. These systems were experimentally created by researchers in laboratories. This artifact-based CVA helped identify, distinguish and document the features that stay the same across systems and those that are expected to differ among products. An example of a commonality is "the nanodevices shall detect whether a target is present." An example of a

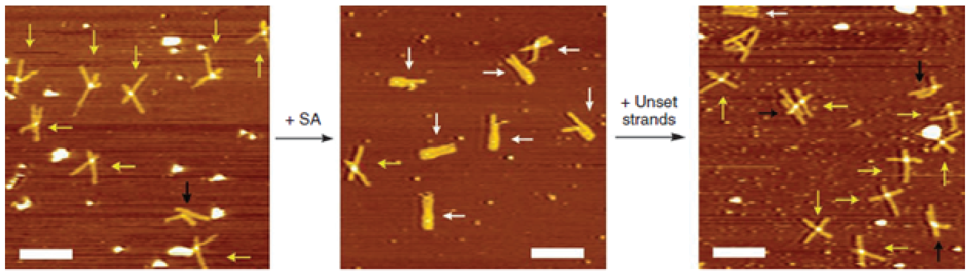

Figure 1. Atomic Force Microscope images of DNA origami pliers: (a) at initiation, DNA pliers are open in the *cross form*, (b) upon adding SA molecules to the solution, DNA pliers close to the *parallel form*, capturing the SA molecules in the pliers' jaws; (c)at reset via addition of *unset strands* DNA pliers re-open, releasing SA molecules. Adapted by permission from Macmillan Publishers Ltd: ref. [21], copyright 2011.

variation is "the simple pliers shall capture the target SA." Dependencies that exist among the alternatives, whereby choosing one variation requires choosing (or excluding) another variation, were represented as commonalities. An example is "if the nanodevice must unzip, then it must begin in the closed initial state."

In contrast to consideration of each nanodevice separately, we describe in the next subsections how these DNA sensing nanodevices are usefully considered as a product family. We describe the challenges that arose in creating the goal model for this family of asynchronous, complex and dynamic nanodevices; the results of the obstacle analysis in terms of making the product family requirements more complete; the advantages we found in a goal-oriented approach when we extended the product family with a new product; and how obstacle analysis of this new product identified a missing requirement for another product.

B. *Goal modeling*

Using the CVA described above as a starting point, we then developed a detailed goal diagram for the product family. We used product-line goal diagrams rather than feature models since the former readily capture satisfaction-based goal refinements in advance of formal modeling [22].

Fig. 2 shows the high levels of the hierarchical goal diagram. The top four tiers are identical for the products, confirming that the products have shared goals and subgoals at the first several levels of refinement. For example, the leftmost refinement into a subgoal, "Achieve[Many accurate signals of presence/absence of target]" is a commonality. At the lower levels, different products displayed alternative refinements of some subgoals as well as alternative assignments to agents [22]. For example, at the bottom left of the figure, there are three alternative refinements, or variations, depending on whether it is a simple pliers, G-zipper pliers, or unzipped pliers product. Domain experts among us increased our understanding of the required behavior of the systems and led to several rounds of corrections and improvements in the goal diagrams. (The dotted box at the top right is discussed below in the context of requirements evolution.)

A key difference between modeling for this domain and for others is that we need to distinguish the goals for the nanodevice (detecting the target) from the goals for the individual instances of the nanodevice (e.g., gripping a molecule). The instances are self-assembled DNA structures or strands. There are a very large number of them, roughly on the order of a nanomole. Each has a significant probability of failing to achieve its goal.

The desired behavior for the system is achieved if a sufficient number of the nanodevice instances achieve their assigned, nearly identical subgoals. As described in [26], from which the rest of this paragraph is taken, we found it useful to augment the AND/OR goal diagrams to allow goal refinements that are mediated by threshold functions, rather than ANDs or ORs. (While a threshold function can logically be realized as an OR of ANDs, this is exponentially larger and unintuitive.) We added a new kind of node, the THRESHOLD node, with an associated threshold ratio, TR. The meaning is that the goal is achieved if the number of subgoals that are achieved exceeds the fraction TR of the total number of subgoals of the THRESHOLD node. Note that threshold nodes are not probabilistic goals [27]. They depend not on the probability that something might happen, but on the multiplicity of times that it does happen.

For ease of presentation and review, we created separate goal diagrams for each product (with the top levels being identical). Fig. 3 shows the goal diagram for one of the products, the simple pliers. Each simple pliers has three primary states: open, closed, and anti-parallel. Open simple pliers can loosely capture an SA molecule in the left or the right side of its jaw and fully capture it by closing into a parallel form. The closed simple pliers fluoresce so that the sensing of the target molecule is visible with AFM. The Avoid subgoals at the bottom left of Fig. 3 were later identified during obstacle analysis, as described below.

C. *Obstacle Analysis*

To identify missing requirements, we performed goal-oriented obstacle analysis on leaf sub-goals of the individual members of the product family. To do this, we used the three-step approach described in [22], [28]: (1) identify

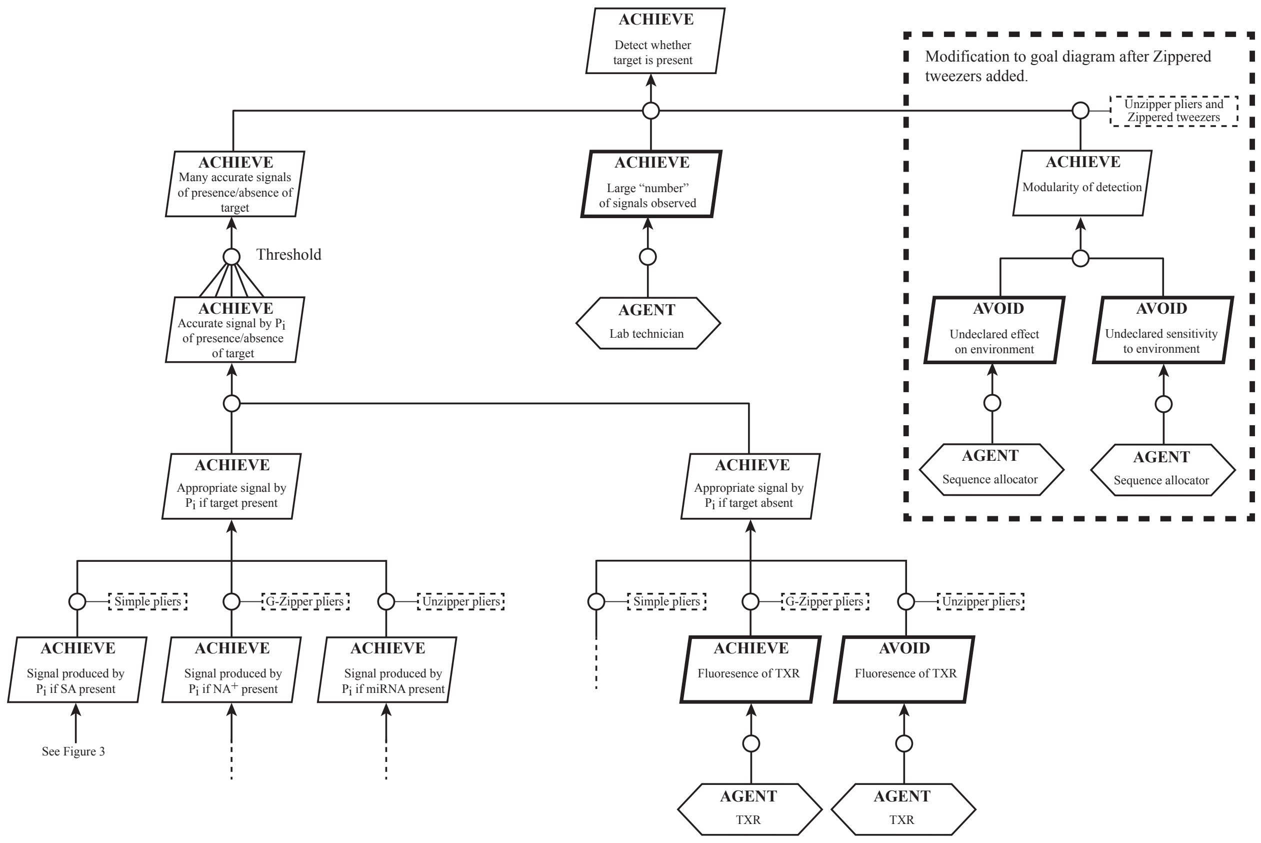
ACHIEVE
Detect whether target is present
Modification to goal diagram after Zippered tweezers added.
Unzipper pliers and Zippered tweezers
ACHIEVE
Many accurate signals of presence/absence of target
ACHIEVE
Large "number" of signals observed
ACHIEVE
Modularity of detection
Threshold
ACHIEVE
Accurate signal by P_i of presence/absence of target
AGENT
Lab technician
AVOID
Undeclared effect on environment
AVOID
Undeclared sensitivity to environment
AGENT
Sequence allocator
AGENT
Sequence allocator
ACHIEVE
Appropriate signal by P_i if target present
ACHIEVE
Appropriate signal by P_i if target absent
Simple pliers
G-Zipper pliers
Unzipper pliers
Simple pliers
G-Zipper pliers
Unzipper pliers
ACHIEVE
Signal produced by P_i if SA present
ACHIEVE
Signal produced by P_i if NA^+ present
ACHIEVE
Signal produced by P_i if miRNA present
ACHIEVE
Fluoresence of TXR
AVOID
Fluoresence of TXR
See Figure 3
AGENT
TXR
AGENT
TXR

Figure 2. High-level goal graph (commonalities)

candidate obstacles, (2) assess their likelihood and criticality, and (3) control or resolve them, e.g., by adding or modifying goals. Our application of obstacle analysis to the problem was informal and manual, in accord with the criteria for lightweight applications of formal methods to requirements modeling described in [29]. Because we were interested in the robustness of the system, we focused on failure modes.

The analysis identified a latent failure mode that could obstruct satisfaction of a sub-goal. Specifically, we had assumed that only one target molecule could bind to one nanodevice (i.e., by being gripped between its jaws). However, a possible failure mode is for one target molecule to bind to the left jaw of a nanodevice and another target molecule to bind to its right jaw. Having two target molecules between its jaws would prevent the jaws from closing and, thus, prevent target detection. This obstacle was confirmed as feasible by the domain experts among us. If the target molecules are present at a high enough concentration, they thus overwhelm the nanodevice and it fails to detect the target molecule's presence. This type of obstacle is called a *non-satisfaction obstacle* in [22]. We thus revised the goal graph to include the relevant Avoid subgoals (if the left ligand, or lining, or a simple pliers binds to a SA molecule, its right ligand should not bind to a different SA molecule, and similarly for the right ligand).

D. Extending the product family

We subsequently extended the product family with an additional, new product recently reported in a paper by Landon et al. [23]. This product is DNA zippered tweezers, as shown in Fig. 4. The tweezers are smaller than the other products and less susceptible to unintended interactions (i.e., nonspecific binding). Like the unzipper pliers, they open in the presence of the target. Upon addition of a closing strand, they close, thus cycling back to their original position. We describe how this affected the goal modeling and, in the following subsection, how this prompted additional obstacle analysis and the identification of a new goal for the product family to make members modular for reliable composition into larger systems.

The extension to the goal diagram was relatively straightforward. The commonalities among the four products were high, and the new zippered tweezers product shared the same high-level, product family goal diagram. In terms of variations, like the unzipper pliers already in the product family, the tweezers began in the closed state and opened upon detection of the target. Thus, the target molecule was not physically gripped during sensing for these two products. The zippered tweezers and the unzipped pliers thus shared

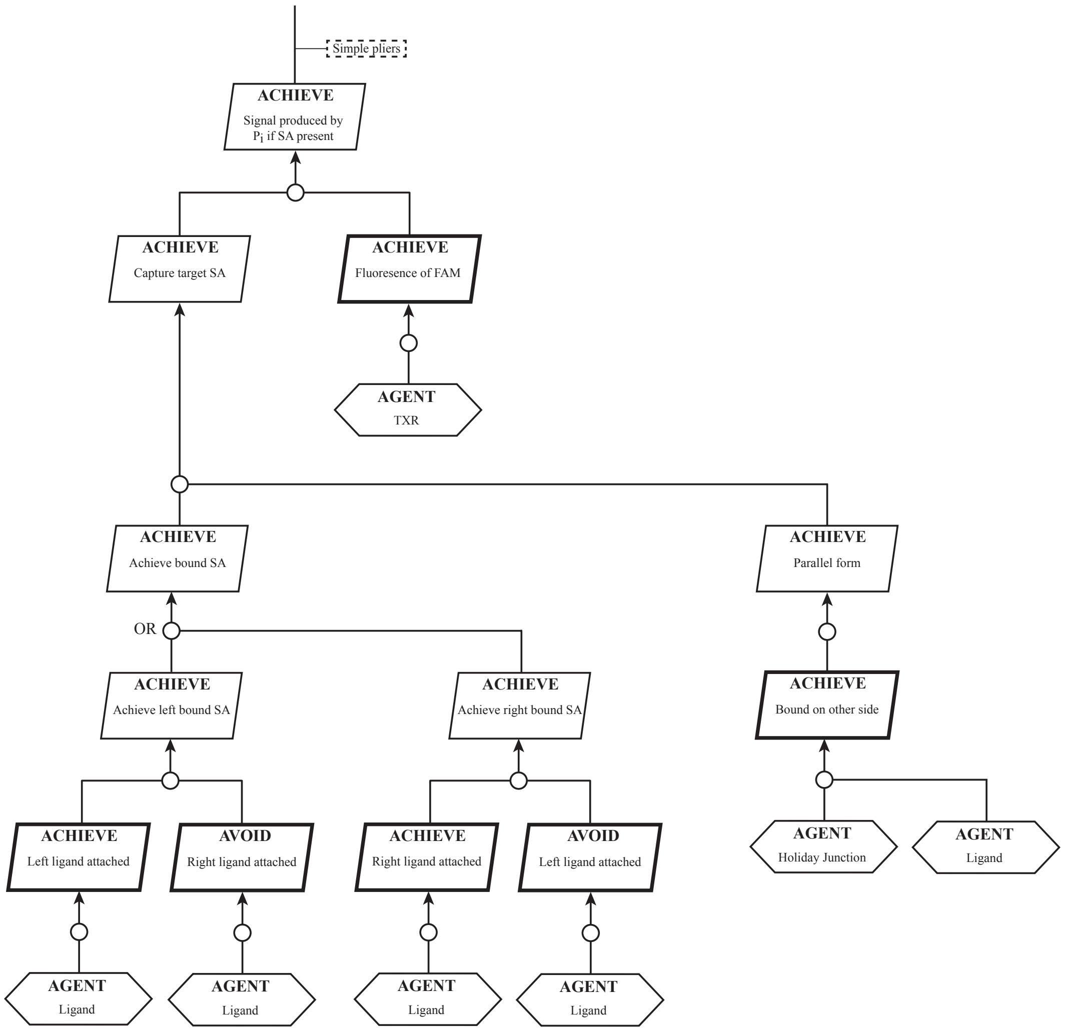

Figure 3. Low-level goal graph for simple pliers

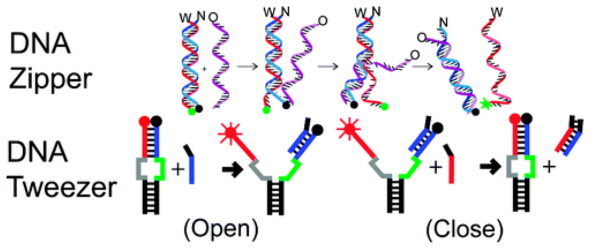

Figure 4. Extending the product family to include zippered tweezers. Reprinted with permission from [23]. Copyright 2012. American Chemical Society.

a further portion of the goal graph. However, at the lowest levels the zippered tweezers employed a somewhat different mechanism than the unzipper pliers to achieve sensing. This required the addition of a new agent assignment at the bottom level of its goal diagram (not shown here). We anticipate that the product family will continue to expand as additional DNA nanodevices are invented and found that the goal-oriented requirements modeling supports this type of product family extension.

E. *Requirements evolution*

More interesting were the new, high-level goals uncovered by the obstacle analysis of the newly added zippered tweezers. The dashed box in Fig. 1 shows how the goal graph changed.

The tweezers were created to be “compact and modular so that they could be easily integrated with complex hierarchical DNA nanomachines” [23]. The obstacle analysis led us to more thoroughly examine the possibility that unintended interactions among the tweezers could obstruct satisfaction of goals. A significant obstacle that can prevent satisfaction of the zippered tweezers’ requirement, “Achieve strand displacement by DNA strand” is crosstalk between tweezers, because it can prevent strand displacement.

The feasibility of this sort of unintended interaction has been shown experimentally and in formal verification of composed DNA gates [17]. Crosstalk is recognized as a potentially significant threat to successful self-assembly. Seeman described this in 2007, as "the key to any successful molecular engineering approach is to design the components of a construction not just so that they are capable of yielding the product, but also to ensure, insofar as possible, that no other product will be competitive with the target" [30].

The test tube solution is, in essence, a global memory. Each tweezers releases (writes) DNA strands into this shared environment and each tweezers uses (reads) DNA strands from this shared environment. The problem is that although we want opening of the tweezers (via DNA strand displacement) to be done by a specific *local* strand, the displacement instead might be done by a DNA strand that is floating freely in the test tube solution. In this case, the tweezers will have an unwanted interaction with other tweezers.

The obstacle analysis performed when the tweezers was added to the product family thus identified a significant missing requirement, namely *modularity of detection*. This is a requirement for both the tweezers and for another member of the product family that also uses strand displacement, the unzipped pliers. The members of the product family must be modular so that nanodevice products can be assembled into larger nanodevices with more complex behavior, while each of the component nanodevices continues to carry out its intended sensing function [21]. (Elbaz, Moshe, and Willner, for example, report the cyclic operation of a two-tweezer assembly [31].) This goal was added to the goal diagram (see dashed box in Fig. 1).

The modularity of detection goal was then refined into two subgoals: Avoid[Undeclared effect on environment] and Avoid[Undeclared sensitivity to environment]. These subgoals are similar to the attribute of *specificity* in chemistry, roughly that a molecule should interact with the intended target and with nothing else. They also fit nicely into Parnas' 4-variable model as constraints on SoftReq, the relation between input variables and output variables [32]. These subgoals were then assigned to the Sequence Allocator agent responsible for assigning DNA sequences to the strand components. (In actuality there are several agents used to achieve this allocation, ranging from the DSD programming language to the lab technician, but we do not describe these separately here.)

To summarize, requirements evolution prompted by the addition of a fourth product and the associated obstacle analysis added a modularity goal for the product family. This made the requirements for the product family more fault tolerant to the likely and problematic scenario of crosstalk when products are later composed. As described below, formal modeling and verification of common and product-specific properties provided some additional assurance that the goal diagrams were accurate.

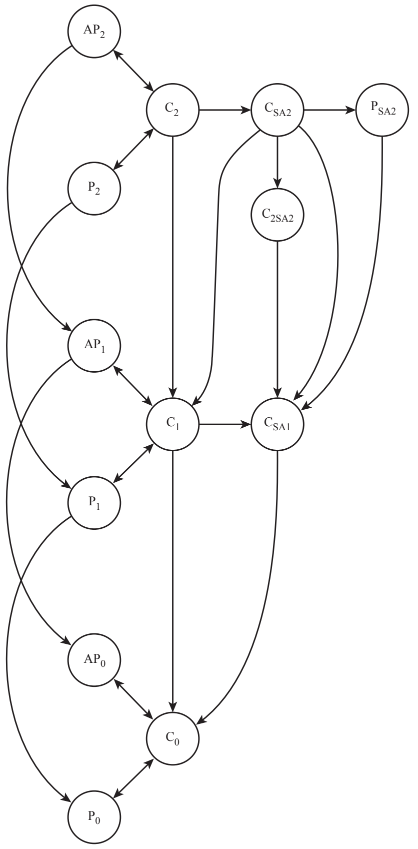

Figure 5. The PRISM model for the simple pliers. State labels are of the form X_{YZ}, where: $X \in \{C, P, AP\}$ indicates whether the pliers are in cross, parallel or antiparallel form, respectively, $Y \in \{\epsilon, SA, 2SA\}$ indicates the number of SA molecules caught by the pliers, and $Z \in \{0, 1, 2\}$ indicates the number of ligands attached to the pliers. For example, C_{2SA2} indicates that the pliers are in cross form, have captured two SA molecules, and still have both ligands attached.

III. Requirements verification

A. Probabilistic model checking

With the requirements identified, we then used formal verification techniques to reason about the correctness of the system. Since the dynamic behavior of these nanodevices can be described as a memoryless, stochastic, asynchronous process, we modeled this behavior with a continuous time Markov chain (CTMC). Because of the memoryless property of the process described by the CTMC, the amount of time between state transitions was distributed according to an exponential distribution $e^{-\lambda t}$ where the rate constant λ was in this context chosen based on the concentration of the reactants and the properties of the chemical reaction.

We performed analysis and verification of our derived models using PRISM, an open source program which can verify properties of probabilistic models [25]. We communicated requirements to PRISM in the form of properties, specified as formulae in a probabilistic temporal logic. For example, we verified requirements by asking PRISM

Boolean questions such as "Is the probability of failure less than 0.01?" and performed analysis by asking questions such as "What is the expected time of the first detection?"

B. Model implementation

We used PRISM's modeling language to define a model that described the behavior of a single nanodevice. We relied on mechanisms in PRISM's modeling language to expand this model to one that described the behavior of a population of nanodevices. Figure 5 shows the graphical representation of the model for a single simple pliers.

Recall that a CTMC is composed of a set of states and set of transitions with corresponding rates. Each transition corresponded to a first or second order chemical reaction. As such, the rate at which these reactions occured were defined by the product of the concentrations of the reactants with the rate constant K. We derived the concentrations of nanodevices in each configuration from the state of the expanded model. The rate constants were derived using a combination of known experimental results and theoretical analysis.

The state-structure, the transitions and corresponding rates were all common features of the models of the products in the product family. This was a direct result of common features of the physical systems being modeled. In addition, the pliers products shared a particular subgraph of the graphical representation of the single-pliers model. Specifically, the pliers in each of the pliers products could be in cross, parallel, or antiparallel form, such as the C_2, P_2, and AP_2 states in Figure 5.

The systems being modeled varied in the modes by which the nanodevices interacted with their respective target molecules. These differences manifested themselves as variations in the states which occurred in the respective Markov models. For example, the simple pliers could detect streptavidin (SA) molecules. The detection interactions were modeled by the addition of configurations $\{C_{SA2}$, C_{2SA2}, P_{SA2}, etc.$\}$ along with associated transitions and rates as shown in Figure 5.

Each simple pliers could be in 13 different configurations. This meant that the expanded model had approximately 13^n states where n is the number of pliers modeled. Similarly, the other expanded models grew exponentially in the number of nanodevices. Consequently, it was not feasible to precisely model large populations of nanodevices.

However, this was not necessary for the verification. The rate at which a chemical reaction occurred decreased exponentially with the number of reactants. Therefore, for reactions of order greater than some small finite constant, we assumed their effects were negligible.

C. Property specification

Having identified the requirements for the sensing product family, we used the PRISM probabilistic model checker to specify and verify them. The requirements were specified as properties in PRISM's continuous stochastic logic (CSL) language [33] . The reward-based paradigm that PRISM uses enabled us to specify many of these properties as queries that return useful non-Boolean quantities such as probabilities and expected waiting times [34].

For example, the following five properties specify requirements that Figure 2 shows are common to all nanodevices in the sensing product family. (Here the variable deadlock is the number of nanodevice instances that have reached a permanent configuration; NP is the number of nanodevice instances being modeled; and the Boolean variable target_detected holds in states in which the target molecule has been detected.)

1) `R{"T"}=? [ F deadlock = NP ]`
   "What is the expected time for the system to reach equilibrium? (all deadlock states)"
2) `R{"T"}=? [ F "target_detected" ]`
   "What is the expected time until the target is detected?"
3) `P=? [ F "target_detected" ]`
   "What is the probability that the target will eventually be detected?"
4) `P=? [ F<=t "target_detected" ]`
   "What is the probability of target being detected within time t?"
5) `P>0.94 [ F<=0.072 "target_detected" ]`
   "Is the probability of the target being detected within 2 hours greater than 0.94."

For an example of a requirement that is specific to one nanodevice in the product family, consider the failure mode for simple pliers discussed at the end of section II.C. Figure 3 enables us to specify a variable, errors, that is the number of modeled instances of the simple pliers for which this failure mode occurs. The following two properties were then specified.

1) `R{"errors"}=? [ S ]`
   "What is the expected number of errors at equilibrium?"
2) `P=>0.999 [ G errors < 0.01 ]`
   "Is the probability that more than 1% of the nanodevices in solution deadlock in an error state greater than 0.01?"

D. Verification results

After the requirements were specified as CSL properties, they were verified using PRISM. Properties that are common across the entire sensing product family were reused intact for each of the nanodevices. For example, consider properties 1-5 in section III-C. The reuse of these properties was achieved by using the Boolean label target_detected as an interface to each of the nanodevices. For example, in the simple pliers, this condition measures the intensity of

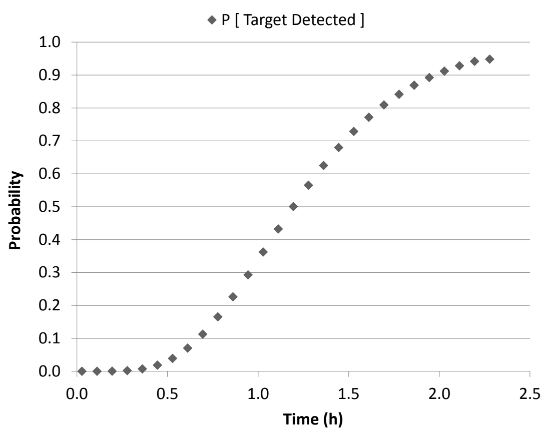

Figure 6. Plot of Property 4 in section III-C

TXR fluorescence, and becomes true when TXR becomes sufficiently low (in this case, below 20%). Note that this interface is shared among some of the products; specifically the simple pliers and the G-zipper pliers utilize this same condition. However, the unzipped pliers achieves detection when the intensity of TXR becomes sufficiently high, specifically over 80%.

The last two properties listed in section III-C were specific to potential errors that could occur in the simple pliers product. These properties were also verified using PRISM, and were of particular interest because the potential error state of the simple pliers resulted in a deadlock state. Therefore, if a pliers ever entered this state, that pliers became permanently disabled. Therefore it was important to verify that this was a rare event. For properties required in only one nanodevice, there was no reuse. However, if the product family grows over time (as, in fact, this one did), unique variations and their associated properties may recur in future products. The opportunity for reuse of previously formally specified properties then may help ease the burden of identifying and specifying model checking properties for new members.

Some of the properties were dependent on a variable such as time. Using those properties, we were able to reason about the probability of detection with varying time. Figure 6 shows property 4 for the simple pliers product plotted over time. It is important to note that each point on this graph was a verified value for that specific time.

Other properties calculated exact expected values or probabilities, such as properties 1, 2, and 3 in section III-C above. These expected values were also computed precisely using the PRISM verification engine and gave added assurance that the requirements accurately described the intended behavior.

IV. Related work

The primary area of related work is the formal verification and more specifically, model checking, of software product lines (SPLs), an active research area with many contributions. Fischbein, Uchitel, and Braberman proposed using Modal Transition Systems (MTS) to model SPLs [35]. MTS supports modeling and reasoning in the presence of partial behavioral information, allowing description of optional and variant behaviour of the different systems that compose the product line. Fantechi and Gnesi subsequently extended Modal Transition Systems (MTS) with variability operators to model SPLs [36]. They used ACTL temporal logic, an action based version of CTL, to express and verify properties of the system. Asirelli, Beek, Fantechi and Gnesi have since combined MTS with a branching-time temporal logic to allow greater flexibility in defining product families [37]. Lauenroth, Toehning, and Pohl modeled the behavior of SPLs with labeled automata, where the labels are the features of a particular product in the family, and used Computational Tree Logic (CTL) to describe properties to be model checked [38].

In a 2012 paper Cordy, Classen, Schobbens, Heymans and Legay built on their previous work [39] to symbolically model check SPLs efficiently [40]. They extended their Featured Transition Systems to handle SPLs that evolve over time. The goal is to reduce the complexity of formal verification on new features by only verifying the differences.

Compositional model checking similarly saves effort by reusing results from the model checking of common features to reduce the amount of re-verificaton needed. Krishnamurthi and Fisler have described incremental, compositional model checking of systems using aspects to verify the behavioral properties [41]. Liu, Basu and Lutz subsequently showed how every new, sequentially composed product in the product line could be compositionally model checked by generating and reusing variation-point obligations [42].

The major difference between our work and previous research is our use of probabilistic model checking. We need this capability because the nanodevices in our product family are molecular and therefore inherently probabilistic. The most relevant work is that of Lakin et al. They have recently used the DSD strand displacement specification language and the probabilistic model checker PRISM to verify properties on simple DNA reaction gates and then composed them into a larger DNA system that implements the approximate majority protocol [17]. Our work differs from theirs in focusing on the identification, analysis and verification of requirements rather than, as they do, on design. Filieri, Ghezzi and Tamburrelli have also recently developed efficient runtime model checking to verify reliability requirements described in a Discrete Time Markov Chain model. Their approach out-performed PRISM in experiments [43].

There has been little work at the interface of requirements engineering and biological or chemical systems. Fisher, Harel and Henzinger have described the computational modeling of biological systems as reactive systems [44].

Hetherington et al. have assembled a complex, computational model of liver function from seven sub-models of different aspects of the system, each previously validated with experimental data [45], [46]. This work addressed the challenges of composing models of complex biological systems and demonstrated the predictive advances and efficiencies possible via the principled reuse of model building.

V. Conclusion

We found that use of goal-oriented requirements engineering methods improved the modeling and analysis of requirements for a product family of sensing nanodevices by revealing new failure modes and facilitating reuse. This in turn made the development process more productive, predictable and safe.

The goal diagrams for the product family showed a high degree of common goals and subgoals among the nanodevices. These commonalities simplified reasoning about their behavior as well as the process of goal modeling. We used the threshold gates that we defined in [26] to describe goal refinements involving the action of a very large number of instances of the nanodevice mediated by thresholds.

The obstacle analysis that we performed resulted in identification of a latent failure mode in which too high a concentration of target molecules could prevent detection of the target in one nanodevice. We added requirements to avoid this failure mode, making the nanodevice more robust. Since an envisioned use of the sensing nanodevices is for diagnostics in human cells, this may also make the nanodevices safer for use.

When we extended the product family with a new nanodevice, we saw a high degree of reuse in the goal diagram. This reuse made the evolution process more efficient. The obstacle analysis on the new nanodevice found a potential for interference between instances of nanodevices that could prevent successful detection, not just in this nanodevice but also in another one. The goal graphs were updated in response, once again improving the safety of the product family requirements.

The requirements for the nanodevice product family described in the goal diagrams were formally specified in CSL and model checked using the probabilistic symbolic model checker PRISM. The verification process benefited from high reuse among the product family PRISM models and from the common CSL properties that were verified. The model checking capability of PRISM proved useful in exploring and understanding the probabilistic behavior of the self-assembling nanodevices in a world of chemical kinetics.

Acknowledgments

We thank Samik Basu, Anthony Finkelstein, and Axel van Lamsweerde for useful discussions. This research was supported by NSF grants 1143830, 0916275, and 0652569. Part of the first author's work was performed while on sabbatical at Caltech and at the Open University, UK. Part of the second author's work was performed while on sabbatical at Caltech and at the Isaac Newton Institute for Mathematical Sciences.

References

[1] N. Seeman, "Nucleic acid junctions and lattices," *Journal of Theoretical Biology*, vol. 99, pp. 237–247, 1982.

[2] E. Winfree, "On the computational power of DNA annealing and ligation," in *DNA Based Computers II: DIMACS Workshop, June 10-12, 1996*. AMS, 1998, pp. 191–213.

[3] D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, and D. Woods, "The tile assembly model is intrinsically universal," *ArXiv e-prints*, Nov. 2011.

[4] L. Qian and E. Winfree, "A simple DNA gate motif for synthesizing large-scale circuits," *J Royal Soc Interface*, vol. 8, pp. 1281–1297, 2011.

[5] ——, "Scaling up digital circuit computation with DNA strand displacemen cascades," *Science*, pp. 1196–1201, 2011.

[6] L. Qian, E. Winfree, and J. Bruck, "Neural network computation with DNA strand displacement cascades," *Nature*, vol. 45, pp. 368–372, Oct. 2011.

[7] L. M. Smith, "Nanotechnology: Molecular robots on the move," *Nature*, vol. 465, no. 7295, pp. 167–168, 2010.

[8] D. Y. Zhang and G. Seelig, "Dynamic DNA nanotechnology using strand-displacement reactions," *Nature Chemistry*, vol. 3, pp. 103–113, 2011.

[9] S. M. Douglas, I. Bachelet, and G. M. Church, "A logic-gated nanorobot for targeted transport of molecular payloads," *Science*, vol. 335, no. 6070, pp. 831–834, 2012.

[10] P. W. K. Rothemund, "Folding DNA to create nanoscale shapes and patterns," *Nature*, vol. 440, pp. 297–302, 2006.

[11] J. Nangreave, D. Han, Y. Liu, and H. Yan, "DNA origami: A history and current perspective," *Current Opinion in Chemical Biology*, vol. 14, pp. 608–615, 2010.

[12] S. Venkataraman, R. Dirks, C. Ueda, and N. Pierce, "Selective cell death mediated by small conditional RNAs," in *Proc Natl Acad Sci USA*, vol. 107, no. 39, 2010, pp. 16 777–16 782.

[13] R. Kershner *et al.*, "Placement and orientation of individual DNA shapes on lithographically patterned surfaces," *Nature Nanotechnology*, vol. 3, pp. 557–561, 2009.

[14] S. M. Douglas, A. H. Marblestone, S. Teerapittayanon, A. Vazquez, G. M. Church, and W. M. Shih, "Rapid prototyping of 3D DNA-origami shapes with caDNAno," *Nucleic Acids Research*, vol. 37, no. 15, pp. 5001–5006, 2009.

[15] A. Phillips and L. Cardelli, "A programming language for composable DNA circuits," *Journal of the Royal Society Interface*, vol. 6, pp. S419–S436, 2009.

[16] M. Lakin and A. Phillips, “Modelling, simulating and verifying Turing-powerful strand displacement systems,” in *DNA 17*, 2011.

[17] M. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips, “Design and analysis of DNA strand displacement devices using probabilistic model checking,” *J Royal Soc Interface*, 2012.

[18] D. Weiss and C. Lai, *Software product-line engineering: a family-based software development process*. Addison-Wesley, 1999.

[19] P. C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, ser. SEI Series in Software Engineering. Addison-Wesley, 2001.

[20] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[21] A. Kuzuya, Y. Sakai, T. Yamazaki, Y. Xu, and M. Komiyama, “Nanomechanical DNA origami ‘single-molecule beacons’ directly imaged by atomic force microscopy,” *Nat Commun*, vol. 2, 2011/08/23/online.

[22] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Chicester, England: Wiley, 2009.

[23] P. B. Landon, S. Ramachandran, A. Gillman, T. Gidron, D. Yoon, and R. Lal, “DNA zipper-based tweezers,” *Langmuir*, vol. 28, no. 1, pp. 534–540, 2012.

[24] B. Yurke, A. J. Turberfield, J. Allen P. Mills, F. C. Simmel, and J. L. Neumann, “A DNA-fuelled molecular machine made of DNA,” *Nature*, vol. 406, pp. 605–608, 2000.

[25] M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of probabilistic real-time systems,” in *CAV11*, 2011, pp. 585–591.

[26] R. Lutz, J. Lutz, T. Klinge, E. Henderson, D. Mathur, and A. Sheasha, “Engineering and verifying requirements for programmable self-assembling nanomachines (NIER track),” in *Int’l Conf on Software Eng*. IEEE, 2012, to appear.

[27] E. Letier and A. van Lamsweerde, “Reasoning about partial goal satisfaction for requirements and design engineering,” in *SIGSOFT FSE*, 2004, pp. 53–62.

[28] A. van Lamsweerde and E. Letier, “Handling obstacles in goal-oriented requirements engineering,” *IEEE Trans. Software Eng.*, vol. 26, no. 10, pp. 978–1005, 2000.

[29] S. M. Easterbrook, R. R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton, “Experiences using lightweight formal methods for requirements modeling,” *IEEE Trans. Software Eng.*, vol. 24, no. 1, pp. 4–14, 1998.

[30] N. Seeman, “An Overview of Structural DNA Nanotechnology,” *Molecular Biotechnology*, vol. 37, no. 3, pp. 246–257, Nov. 2007.

[31] J. Elbaz, M. Moshe, and I. Willner, “Coherent activation of DNA Tweezers: A SET/RESET Logic System,” *Angewandte Chemie*, vol. 121, no. 21, pp. 3892–3895, 2009.

[32] D. L. Parnas and J. Madey, “Functional documents for computer systems,” *Sci. Comput. Program.*, vol. 25, pp. 41–61, October 1995.

[33] A. Aziz, K. Sanwal, V. Singhal, R. K. Brayton, and R. K. Brayton, “Model-checking continous-time markov chains.” *ACM Trans. Comput. Log.*, pp. 162–170, 2000.

[34] M. Kwiatkowska, G. Norman, and D. Parker, “Stochastic model checking,” in *SFM’07*, 2007, pp. 220–270.

[35] D. Fischbein, S. Uchitel, and V. A. Braberman, “A foundation for behavioural conformance in software product line architectures,” in *ROSATEA*, 2006, pp. 39–48.

[36] A. Fantechi and S. Gnesi, “Formal modeling for product families engineering,” in *SPLC*. IEEE Computer Society, 2008, pp. 193–202.

[37] P. Asirelli, M. H. ter Beek, S. Gnesi, and A. Fantechi, “Formal description of variability in product families,” in *SPLC*, E. S. de Almeida, T. Kishi, C. Schwanninger, I. John, and K. Schmid, Eds. IEEE, 2011, pp. 130–139.

[38] K. Lauenroth, K. Pohl, and S. Toehning, “Model checking of domain artifacts in product line engineering,” in *ASE*. IEEE Computer Society, 2009, pp. 269–280.

[39] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay, “Symbolic model checking of software product lines,” in *ICSE ’11*, 2011, pp. 321–330.

[40] M. Cordy, A. Classen, P.-Y. Schobbens, P. Heymans, and A. Legay, “Managing evolution in software product lines: a model-checking perspective,” in *VaMoS*, U. W. Eisenecker, S. Apel, and S. Gnesi, Eds. ACM, 2012, pp. 183–191.

[41] S. Krishnamurthi and K. Fisler, “Foundations of incremental aspect model-checking,” *ACM Trans. Softw. Eng. Methodol.*, vol. 16, April 2007.

[42] J. Liu, S. Basu, and R. R. Lutz, “Compositional model checking of software product lines using variation point obligations.” in *Autom. Softw. Eng.*, 2011, pp. 39–76.

[43] A. Filieri, C. Ghezzi, and G. Tamburrelli, “Run-time efficient probabilistic model checking,” in *ICSE ’11*, 2011, pp. 341–350.

[44] J. Fisher, D. Harel, and T. A. Henzinger, “Biology as reactivity,” *Commun. ACM*, vol. 54, pp. 72–82, Oct. 2011.

[45] J. Hetherington, T. Sumner, R. M. Seymour, L. Li, M. V. Rey, S. Yamaji, P. Saffrey, O. Margoninski, I. D. L. Bogle, A. Finkelstein, and A. Warner, “A composite computational model of liver glucose homeostasis. I. building the composite model,” *Journal of The Royal Society Interface*, 2011.

[46] 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, “A composite computational model of liver glucose homeostasis. II. exploring system behaviour,” *Journal of The Royal Society Interface*, vol. 9, no. 69, pp. 701–706, 2012.