# Digital Evolution of Behavioral Models for Autonomic Systems [*]

Heather J. Goldsby, Betty H.C. Cheng,[†] Philip K. McKinley, David B. Knoester, and Charles A. Ofria
{hjg, chengb, mckinley, dk, ofria}@cse.msu.edu
Department of Computer Science and Engineering
Michigan State University
3115 Engineering Building
East Lansing, Michigan 48824 USA

## Abstract

*We describe an automated method to generating models of an autonomic system. Specifically, we generate UML state diagrams for a set of interacting objects, including the extension of existing state diagrams to support new behavior. The approach is based on* digital evolution, *a form of evolutionary computation that enables a designer to explore an enormous solution space for complex problems. In our application of this technology, an evolving population of* digital organisms *is subjected to natural selection, where organisms are rewarded for generating state diagrams that support key scenarios and satisfy critical properties as specified by the developer. To achieve this capability, we extended the* AVIDA *digital evolution platform to enable state diagram generation, and integrated* AVIDA *with third-party software engineering tools, e.g., the Spin model checker, to assess the generated state diagrams. To illustrate this approach, we successfully applied it to the generation of state diagrams describing the autonomous navigation behavior of a humanoid robot.*

## 1. Introduction

Increasingly, high-assurance applications rely on autonomic systems to react and respond to changing environmental concerns; examples include critical infrastructure protection and transportation systems. As such, it is essential that the corresponding autonomic reactions do not put the system into an inconsistent state or deliver unacceptable behavior. In an effort to promote separation of concerns,

we consider an *autonomic system* to comprise a collection of (non-adaptive) *target systems* and a set of adaptations that realize transitions among target systems in response to environmental changes. A key challenge with developing autonomic systems is to identify robust and resilient target systems that handle the various, sometimes adverse, environmental conditions [1]. A second challenge is to develop more abstract representations of these systems as a means to manage the complex functional and adaptation requirements and the corresponding implementations [1]. Model-driven development (MDD) technology [2] supports the systematic transformation of such abstract representations (graphical models) into more detailed models or formal specifications, and eventual generation of the corresponding code. While MDD offers an attractive development approach, the efforts required to create or revise models in order to start the MDD process (i.e., initial requirements-level or early design models) can be error prone and difficult to automate. To address both challenges, we propose a biologically-inspired approach to automatically generate requirements-level behavioral models for target systems, where the models support user-defined scenarios and satisfy formally specified safety-critical properties.

Several approaches have been developed to model target systems, including: architecture description languages [3, 4], goal models [5, 6], and state machines [6, 7]. Additionally, significant progress has been achieved in synthesizing behavioral models from scenarios [8–13] and from formally specified properties [13–15]. Scenarios and properties form the lower and upper bounds on the possible behavior of the system [13], respectively; that is, a scenario depicts what at least one path of behavior through a model must satisfy, and properties indicate what all paths through a model must satisfy. However, there are many behavioral models that exist in the solution space between these boundaries. One drawback of existing approaches is that they limit the exploration of the solution space for target systems to those envisioned by the developer who hand-crafted the model or

designed the synthesis algorithm. Since autonomic systems are reacting to a large number of environmental and internal conditions, whose combined effect may not be fully understood at development time, we need target systems that can deliver the essential services in a more robust and resilient fashion than human developers might envision [1].

In this paper, we describe an approach to generating behavioral models for target systems that leverages a technique used by living organisms to adapt to changing environmental conditions: *evolution*. Evolutionary computation methods such as the genetic algorithm (GA) and genetic programming (GP) have achieved considerable success in the world of computing, in some cases producing human-competitive designs [16]. Additionally, GAs and GPs have been used by search-based software engineering approaches [17]. Our approach is based on *digital evolution* [18], a branch of evolutionary computation in which a population of self-replicating computer programs exists in a user-defined computational environment and is subject to mutations and natural selection. These "digital organisms" compete for available resources (e.g., virtual CPU cycles) that enable the organism to survive and thrive; in addition, these organisms are subject to instruction-level mutations during replication. Whereas GAs and GPs evaluate each individual in the population and explicitly select individuals to move to the next generation, the evolution of digital organisms is more open-ended and thus more likely to discover novel and previously unknown solutions. Until recently, digital evolution has been used primarily by biologists to address questions that are difficult or impossible to study with organic life forms [19, 20]. However, digital evolution also provides a means to *harness* the power of evolution and apply it to problems in science and engineering [20–23], sometimes discovering strikingly clever and innovative solutions to complex problems.

Our approach uses digital evolution to generate behavioral models of target systems. Each digital organism is treated as a generator of UML state diagrams that depict system behavior: when the organism executes, it constructs an in-memory representation of a *behavioral model* that comprises one or more state diagrams. An organism's ability to compete for resources is directly related to whether the generated behavioral model meets the criteria specified by the developer. Mutations introduced during replication produce organisms with varying abilities to compete, while competition for resources gives rise to a population of organisms that produce increasingly better solutions. To implement this method, we extended the AVIDA digital evolution platform [18] in three key ways. First, we enabled each organism to have *instinctual knowledge*, namely, information embedded in the organism at birth. In this study, the instinctual knowledge comprises information about the UML class diagram elements (that describe the structural

elements) and an optional set of *seed state diagrams* (that is, UML state diagrams specifying the behavior of the existing system). Second, we enhanced the AVIDA instruction set to include instructions that enable an organism to use its instinctual knowledge to construct transitions in one or more state diagrams. Third, we enabled AVIDA to use third-party software to assess the behavioral models generated by organisms. For example, we make use of the Spin model checker [24] to check the AVIDA-generated state diagrams for adherence to safety-critical properties.

The primary contribution of this paper is a digital evolution technique for automatically generating behavioral models for an autonomic system. In contrast to other manual and synthesis techniques, this technique generates an array of possible solutions that are unbiased by human preconceptions. We illustrate this technique on the evolution of target systems for an autonomic robot navigation system [25]. The remainder of this paper is organized as follows. Section 2 provides background on AVIDA. Section 3 describes how we use AVIDA to generate behavioral models and describes some generation guidelines for autonomic systems. Section 4 applies this approach to generate a target system for an autonomous robot navigation system. Finally, in Section 5, we conclude with a discussion of the ramifications of this approach and future work.

## 2. AVIDA Background

While evolutionary computation has been studied since the 1960's, the subfield of digital evolution is much younger. The first experiments with populations of self-replicating computer programs were performed in 1990 in a system called Coreworld [26], and later improved upon in Tierra [27]. In 1993, Ofria and colleagues began development of AVIDA [18], in which self-replicating digital organisms evolve in an open-ended fashion with more parallels to natural evolution than other forms of evolutionary computation. Indeed, until recently AVIDA has been used primarily by biologists: observing evolution in digital organisms enables researchers to address questions that are difficult or impossible to study with organic life forms.

**AVIDA Operation.** Figure 1 depicts an AVIDA population and the structure of an individual organism. Each digital organism consists of a circular list of instructions (its *genome*) and a virtual CPU. Instructions are executed by each organism's virtual CPU. The instruction set is designed so that random mutations will always yield a syntactically correct program, albeit one that may not perform any meaningful computation [28].

An AVIDA environment comprises a number of *cells*, where a cell is a compartment in which an organism can live. Each cell can contain at most one organism, and the size of an AVIDA population is bounded by the number of cells in the environment. Organisms are *self-replicating*,

that is, the genome itself must contain the instruction to create an offspring. When an organism replicates, a cell to contain the offspring is selected from the environment, and any previous inhabitant of the target cell is replaced (killed and overwritten) by the offspring. Each population starts with a single organism that is capable only of replication, and different genomes are produced through random mutations introduced during replication. Mutation types include: replacing the instruction with a different one, inserting an additional, random instruction into the offspring's genome, and removing an instruction from the offspring's genome.
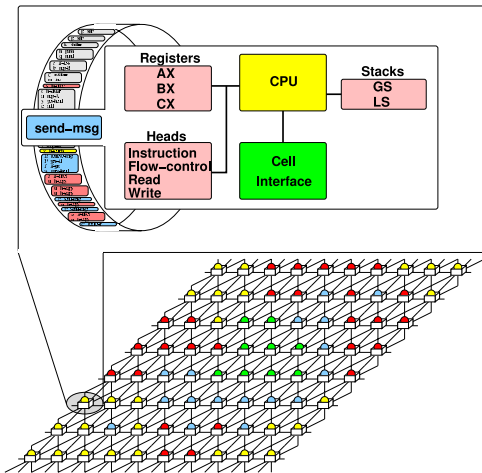


**Figure 1. Elements of AVIDA platform**

Developers use *tasks* to describe desirable organism behavior. Generally, tasks are defined in terms of the externally visible behaviors of the organisms (their phenotype). For example, a developer might define a task that rewards an organism for outputting the correct result of a particular computation (e.g., bitwise AND of two input values). In our study, tasks are defined in terms of application execution scenarios, functional properties, and software engineering metrics associated with the state diagrams constructed by an organism; see Section 3. Performing a task increases an organism's *merit* that determines how many instructions its virtual CPU is allowed to execute relative to the other organisms in the population. For example, an organism with a merit of 2 will, on average, execute twice as many instructions as an organism with a merit of 1. Since digital organisms are self-replicating and compete for space, a higher merit (all else being equal) results in an organism that replicates more frequently, spreading throughout and eventually dominating the population. Hence, AVIDA satisfies the three conditions necessary for evolution to occur [29]: replication, variation (mutation), and differential fitness (competition). AVIDA does not simulate evolution, it is an *instance* of evolution.

**Harnessing Digital Evolution.** In addition to its use by the biology community, AVIDA has also enabled researchers to *harness* the power of evolution and use it to solve problems in science and engineering [21]. Effectively, AVIDA provides a *digital Petri dish* for creating new computational behavior. In some studies, the behavior is intended to be deployed directly in computing systems that interact with the physical world, such as sensor networks and robot swarms [22, 30]. In this case, evolved AVIDA genomes are translated directly into code that executes on such devices.

## 3. Generating Behavioral Models in AVIDA

Our digital evolution-based method enables developers to generate and explore innovative behavioral models for target systems. Specifically, mutations produce state diagrams that developers might not otherwise discover, while natural selection pressures organisms to generate models that meet developer requirements. This blend of innovation and requirements satisfaction is especially pertinent for generating target systems that must respond to varying environmental conditions in a resilient and robust fashion.

### 3.1. Experimental Process Overview

To provide intuition regarding how AVIDA organisms generate a behavioral model, consider a typical experiment. As in other applications of AVIDA, a population starts with a single organism that is only capable of replication and a maximum population size of 3,600 organisms. As the organism and its offspring replicate, different genomes are produced through random mutations. Organisms that generate state diagrams exhibiting desired characteristics receive more merit and thus replicate faster. Therefore, over time, the population is composed of organisms that generate state diagrams that exhibit progressively more of the desired characteristics. If an organism generates state diagrams that support all the key scenarios and satisfy all of the properties, then it has successfully, and automatically, generated a UML behavioral model for a target system. We refer to the state diagrams that meet these requirements as *compliant state diagrams*. At this point, the experiment is successful and can be halted, or it may be allowed to proceed to find other sets of compliant state diagrams.

In general, we run 100 AVIDA experiments in parallel. Multiple experiments are performed to account for the stochastic nature of the evolutionary process. Each experiment is run for over 100,000 updates, where an update, on average, executes 30 instructions per organism (updates are the standard unit of time in AVIDA experiments).

Now, let us consider the normal life cycle of an organism. When the organism is created through the replication process, it is provided with instinctual knowledge of class diagram elements and optionally with seed state diagrams. As the organism executes the instructions in its genome, dif-

ferent pieces of instinctual knowledge are selected and used to create transitions. When the organism replicates, the state diagrams it generates are evaluated according to the criteria provided by the developer, and its merit is calculated. As the parent copies its genome to its offspring, mutations may be introduced. These mutations may cause the offspring to generate state diagrams that differ from those generated by its parent. The offspring is placed in a different cell, possibly replacing another organism. Then both the parent and the child begin execution at the start of their genome, each with the same instinctual knowledge. An organism may die of either old age or being overwritten by another organism.

## 3.2. AVIDA Extensions

To enable AVIDA organisms to generate compliant state diagrams, we extended the AVIDA platform in three ways.

**Instinctual Knowledge.** When a new organism is created through replication, it is provided with a seed file containing details of class diagram elements and the seed state diagrams. For each class, the seed file contains a list of triggers (operations), a list of guards (expressions built using attributes), a list of actions (the operations of classes related to it via associations), a list of states that can be used to generate a state diagram, and an optional seed state diagram. For example, Figure 2(a) depicts some of the lists that comprise the instinctual knowledge of an organism generating a robot navigation system. Specifically, the list of state diagrams, and the lists of states, states, triggers, guards, and actions for the ObstacleAvoidanceTimer are depicted. Each organism is provided with the same instinctual knowledge.

**New Instructions.** To enable organisms to manipulate state diagrams, we developed a new set of AVIDA instructions. Figure 2(b) depicts an organism's genome that includes some of these instructions and generates a behavioral model. These instructions are used to (1) *select* model elements and (2) *construct* new transitions. The selection instructions are used to index the lists that make up the organism's instinctual knowledge. Specifically, each type of list has its own set of instructions (one instruction per list item). The name of an instruction includes the type of element it is selecting (e.g., state diagram (sd), origin state (s-orig), destination state (s-dest), trigger (trig), guard(guard), or action (action)) and an integer representing a position in the list of that type. Third, the addTrans instruction constructs the transition described by the selection instructions. These application-independent instructions can be reused to generate behavioral models for different autonomic systems.

For example, Figure 2(b) depicts a portion of an organism's genome that constructs a transition and the generated behavioral model, where shaded states denote seed state diagram elements and white states denote generated elements. Instruction sd-2 selects the ObstacleAvoidanceTimer state diagram. The other selection instructions select the origin state (State1), destination state (State4), trigger (null), guard (obstacle=1), and action (ˆNavigationControl.restart). Instruction addTrans then constructs the actual transition, which is denoted as a dotted line in the ObstacleAvoidanceTimer state diagram depicted in Figure 2(c).

**New Tasks.** We defined a set of tasks to reward AVIDA organisms for generating state diagrams that meet the developer-specified behavioral constraints. Organisms that perform these tasks will have higher merit, execute their instructions faster, and have a better chance of replicating more frequently, thus spreading throughout the population. A given AVIDA experiment may use multiple scenario tasks or property tasks, or a combination thereof.

Scenario tasks (checkScenario) reward organisms that generate state diagrams that include an execution path. For each scenario, the developer must specify the messages between objects and may optionally include a start state for each object and specify whether the scenario should iterate. The reward for a checkScenario task is based upon the percentage of the execution path included in the state diagrams. Thus, to receive the maximum reward for a scenario, an organism must generate a state diagram for each object involved in the scenario that includes a path specifying the messages sent and received by the object for the scenario. We describe examples of checkScenario tasks in Section 4.

A property task (checkProperty) rewards organisms that generate state diagrams that adhere to a formally specified property. These tasks constrain the behavior of the interacting state diagrams. To enable AVIDA to determine if the generated state diagrams satisfy a stated property, we extended AVIDA to use external tools. Specifically, the checkSyntax task uses Hydra, an existing UML formalization engine [31], to translate a UML model into Promela, the specification language for the model checker Spin [24]. Next, the checkWitness task uses Spin to verify that at least one execution path (i.e., a witness trace) through the Promela model satisfies the functional property specified by the developer in Linear Temporal Logic (LTL). We accomplish this by negating the property and using Spin to search for a counter-example [32]. Lastly, if the checkWitness for a given property passes, then the checkProperty task uses Spin to verify that the Promela specification satisfies the same functional property. Additional details on the external and previously developed analysis process, and the underlying formalization framework can be found in [31].

Finally, software engineering metric tasks are used to reward organisms for generating state diagrams that meet commonly advocated software engineering metrics. Specifically, we defined the min-trans and determinism tasks. The min-trans task rewards an organism for generating state diagrams with fewer transitions, relative to the state diagrams generated by the rest of the population. The determinism task rewards an organism for generating state diagrams that
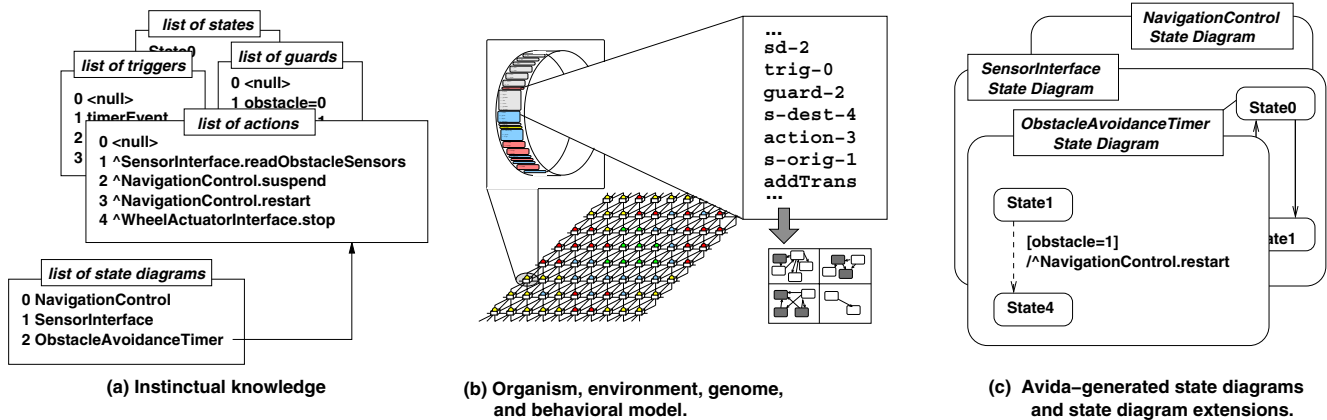
**(a) Instinctual knowledge**

**(b) Organism, environment, genome, and behavioral model.**

**(c) Avida–generated state diagrams and state diagram extensions.**

**Figure 2. An AVIDA organism instinctual knowledge, genome, and generated state diagrams.**

are *deterministic*, where a deterministic state diagram is one in which at most one transition can be taken from each state for a given event and guard combination. Other possible tasks in this category could reward organisms for reducing coupling, and so on.

### 3.3. Generation Guidelines

Although any combination of instinctual knowledge and tasks can be used to drive the evolutionary process, in practice we have found that certain configurations are able to discover compliant state diagrams more rapidly than others. Briefly we describe three recommended configurations and their application for autonomic systems.

**Initial development effort.** This configuration addresses new development efforts in which the class diagram has been constructed and some behavioral information, e.g., scenarios, are known, but no state diagrams have been developed. For this option, the developer provides AVIDA with the class diagram information and uses tasks that check for scenarios. The state diagrams generated by AVIDA organisms should support the execution paths described by the scenarios. This configuration can be used for an autonomic system to generate an initial target system, or to generate a target system that exhibits significantly different behavior than the other target systems.

**New class extension.** This configuration addresses development efforts in which a system is extended to include new behavior that involves the use of a newly added class. For this option, the developer provides AVIDA with the class diagram information and the seed state diagrams for the previously existing classes. The scenario tasks reward organisms for generating state diagrams that include execution paths during which the existing classes interact with the new class. Property tasks are used to constrain how the new class interacts with the existing classes. Namely, these tasks check that the behavioral model continues to meet its previous requirements, and that it meets its new requirement.

Thus, this configuration will generate a state diagram from scratch for the new class and will also extend the seed state diagrams to interact with the new state diagram. This configuration can be used for autonomic systems to generate a new target system by extending a previously developed target system for a new physical configuration, such as the addition of a new monitoring device.

**New behavior for existing classes.** This configuration addresses development efforts in which the current version of the software is being extended with new functionality, but no new classes are added. For example, adding emergency stop behavior to an existing robot navigation system. For this option, the developer provides AVIDA with the class diagram information and seed state diagrams for all the classes. The desired new functionality is specified using a combination of scenario and property tasks. Specifically, the scenario tasks reward organisms for including execution paths that partially implement the new behavior. The property tasks reward organisms for continuing to meet its previous requirements and for satisfying new behavior requirements specified as LTL formulae. Thus, this configuration will extend one or more seed state diagrams to achieve the new behavior without disrupting the system's ability to meet its other requirements. This configuration can be used for autonomic systems to generate a new target system by extending a previously developed target system with a new system-wide behavior, such as a different security policy.

**General guidelines.** For all configurations, we recommend using the software engineering metric tasks. These tasks, in isolation, will not generate compliant state diagrams. However, they assist the organisms in generating compliant state diagrams more rapidly. Additionally, we do not recommend using property tasks without scenario tasks: the set of compliant state diagrams for a given property task is relatively small when compared to the solution space. Thus, we have found that using a scenario task that

describes an execution path that could satisfy a property, in conjunction with the property task itself, accelerates the generation of compliant state diagrams. Additionally, to improve performance, we recommend ordering the tasks. Specifically, in our experiments, we require an organism to generate state diagrams that include all the execution paths specified by the checkScenario tasks prior to checking if the state diagrams satisfy the checkSyntax, checkWitness, and checkProperty tasks.

## 4. Case Study

We illustrate our approach by generating a target system behavioral model for T-Rot, an intelligent robot developed by Kim *et al.* [25] (depicted in Figure 3). Currently, the world's population is aging and the cost of health care is increasing. As a result, the Korea Institute of Service and Technology (KIST) is developing T-Rot to assist in the care and support of the elderly [25], e.g., by performing errands such as retrieving medicine. One particularly challenging problem is to enable T-Rot to autonomously navigate to a designated position.



**Figure 3. T-Rot**

Briefly we describe the software objects that comprise the RNS. The NavigationControl receives a destination from the CommandLineInterface. It plans a path (Navigation-Path) from its current location, identified using the Localizer, to the destination using its map (NavigationMap). The NavigationControl then senses the environment through the SensorInterface and controls the robot wheels through the WheelActuatorInterface. The ObstacleAvoidanceTimer is responsible for detecting obstacles, stopping the wheels, and suspending NavigationControl.

Previously, KIST modeled the RNS with use case diagrams, collaboration diagrams, and a state diagram for the NavigationControl [25] that includes the behavior for moving to a designated position while avoiding obstacles. However, state diagrams for the behavior of the other objects (e.g., the ObstacleAvoidanceTimer and SensorInterface) were not created.

For our case study, we assume that the RNS is resource constrained and thus has two different target systems: navigate and navigate and avoid obstacles. The navigate and avoid obstacles target system is far more computationally expensive and thus the RNS adapts between the two target systems depending upon environmental conditions, such as the presence of an empty room or an infant. We assume that the navigate target system has been developed. However, the navigate and avoid obstacles target system in which T-Rot moves to its destination and avoids obstacles has not been developed.

To demonstrate our technique, we use AVIDA to generate a behavioral model for this target system by extending the navigate target system to avoid obstacles. Specifically, for these experiments, we want to evolve the same obstacle avoidance response as KIST, which is that if the robot encounters an obstacle, then it stops. We formally specify the property that the RNS must avoid obstacles, and we provide AVIDA with two sequence diagrams that refer to two possible new scenarios (with an obstacle and without an obstacle) reflecting the new requirement. We constructed a class diagram for the RNS (that describes the structure of all of the previously mentioned objects) and seed state diagrams for all the RNS classes, except the ObstacleAvoidanceTimer. To construct the seed state diagram for the NavigationControl class, we edited the NavigationControl state diagram created by KIST to remove the elements that corresponded to avoiding obstacles. A successful AVIDA organism must generate a state diagram for the ObstacleAvoidanceTimer from scratch, and extend the seed state diagrams for the NavigationControl and SensorInterface to interact with the ObstacleAvoidanceTimer and thus avoid obstacles.

### 4.1. Experimental Setup

Because we are creating a new target system by extending an existing target system with a new behavior that includes a new class (ObstacleAvoidanceTimer), we use the **New class extension** configuration. Specifically, we provide the AVIDA organisms with instinctual knowledge that includes the UML class diagram for the RNS and the seed state diagrams for all the classes except the ObstacleAvoidanceTimer. We want the AVIDA organisms to extend the seed state diagrams for the NavigationControl and SensorInterface and to create a state diagram for the ObstacleAvoidanceTimer. To drive the evolutionary process to select organisms that generate compliant state diagrams, we used software engineering metric tasks and the following scenario and property tasks.

We defined two checkScenario tasks based upon a collaboration diagram specified by KIST [25]. The first task, checkScenario-obstacle, rewards organisms for generating state diagrams that include the execution path depicted as a

sequence diagram in Figure 4(a). Specifically, in this execution path, the ObstacleAvoidanceTimer detects an obstacle, suspends the NavigationControl, and stops the WheelActuatorInterface. Additionally, the checkScenario-obstacle task provides an additional reward if the SensorInterface portion of the scenario iterates because it is likely that the SensorInterface will be queried multiple times (to gather information about its surroundings). The checkScenario-obstacle task also provides additional rewards if the first states for both the SensorInterface and ObstacleAvoidanceTimer portions of the scenario are their respective initial states (i.e., State0). The second task, checkScenario-no-obstacle, rewards organisms for generating state diagrams that include the execution path depicted as a sequence diagram in Figure 4(b). In this execution path, the ObstacleAvoidanceTimer does not detect an obstacle and thus restarts the NavigationControl . The execution path for the SensorInterface is the same in both scenarios.

Next, we specified two properties that the state diagrams generated by the AVIDA organisms should satisfy:

1. *Globally, it is always the case that eventually the robot's current position will be its destination.*
2. *Globally, it is always the case that if the* ObstacleAvoidanceTimer *detects an obstacle, then eventually the* WheelActuatorInterface *will stop the wheels and the* NavigationControl *will be suspended.*

The first property specifies the desirable general behavior of the RNS. This property was satisfied by the existing navigate target system and should be satisfied by the navigate and avoid obstacle target system. The second property specifies the new behavior: that the ObstacleAvoidanceTimer should stop the robot if it detects an obstacle. These properties are used as checkProperty tasks.

## 4.2. Experimental Results

We ran 100 instances of AVIDA, each with a maximum population size of 3,600 organisms, for 200,000 updates. By the end of the experiment, eight behavioral models, each of which supported the execution paths described by both scenarios and satisfied both properties were generated. These are potential behavioral models for the navigate and avoid obstacles target system. Figures 5-7 depict the compliant state diagrams generated by one of the successful AVIDA organisms. Seed state diagram elements are depicted as solid lines; generated transitions are depicted as dotted lines.

Figure 5 depicts the state diagram generated for the NavigationControl class. This diagram, in conjunction with the rest of the behavioral model, satisfies both properties. Thus, the robot will always arrive at its destination and will always stop if an obstacle is detected. It has achieved the behavior that we specified. However, similar to human-designed models, this generated behavioral model does include addi-

tional, unspecified behavior. Specifically, it includes transitions that connect the initial state to state Idle and state Starting to State12, which is a deadlock state. (Offline we have performed reachability analysis to verify that this transition is never taken, and thus State12 is never reached.) At this point, we could either manually remove the spurious transitions, or modify the property and use AVIDA to automatically generate compliant state diagrams that adhere to this refined property.

Figure 6 depicts the AVIDA organism generated state diagram for the SensorInterface class. The seed state diagram elements describe how the SensorInterface is queried by the NavigationControl for general sensor information. The two new states and three new transitions are used to respond to queries from the ObstacleAvoidanceTimer. This state diagram does not use any extraneous states or transitions.

Figure 7 depicts the state diagram generated from scratch for the ObstacleAvoidanceTimer. In response to a timerEvent(), the ObstacleAvoidanceTimer queries the SensorInterface to determine if an obstacle is present. If so, it stops the wheels and suspends the NavigationControl. This particular AVIDA organism used eight states and eleven transitions to model the desired behavior. Some transitions (connecting State2 to State11, State11 to State12, State11 to State9, and State11 to State3) are never fired and some states (State11, State9, State3) are never visited. In general, if we allowed the experiment to run for longer, then eventually we would expect that the organism and its offspring would optimize the diagram for the min-trans task and these extraneous elements would be removed. However, this state diagram includes the execution paths specified by the scenarios and successfully interacts with the other objects in the system to satisfy the properties. Neither the SensorInterface state diagram, nor the state diagram for the ObstacleAvoidanceTimer were constructed by KIST [25], so we are not able to compare the AVIDA generated solution to a manually created one. However, these state diagrams could be used to further explore the behavior of the RNS.

The compliant state diagrams we have presented constitute one of eight acceptable solutions for the target system generated by the AVIDA organisms. The differing characteristics of the solutions make them amenable for different environmental conditions that were not explicitly stated. For example, we selected a behavioral model to present based on diagram simplicity, others had additional states and transitions that provided greater fault-tolerance through redundancy and security through obfuscation of operational behavior. In general, the developer can browse such solutions, each of which specify different ways of achieving the compliant behavior, or may adjust other properties to further distinguish the solutions. A developer may use these generated solutions in different ways. One possibility is to use these behavioral models to inspire a human-created de-
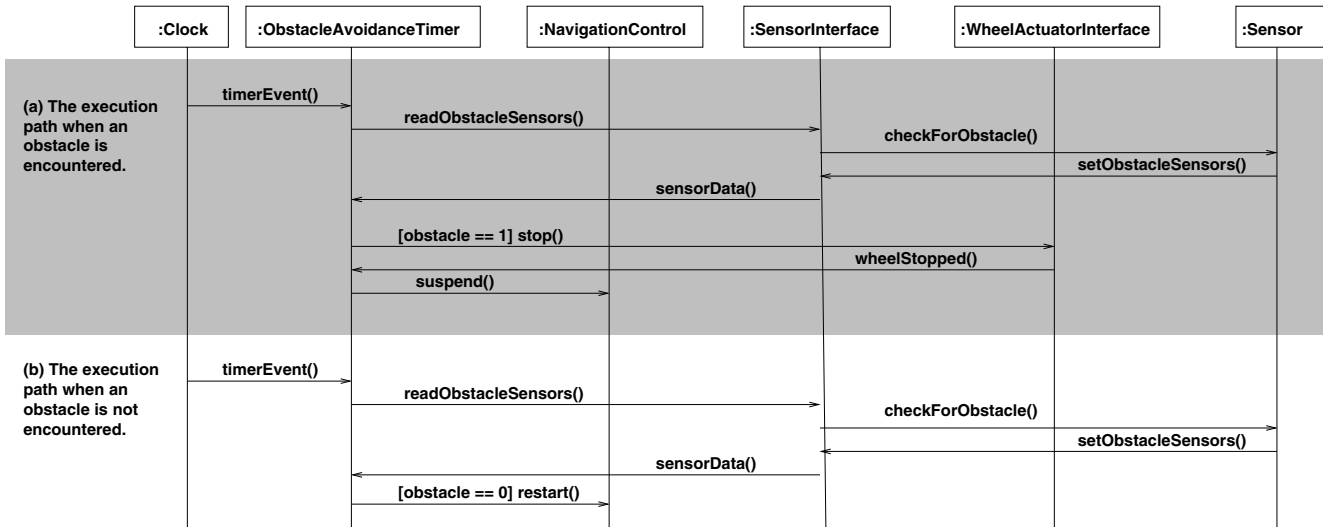
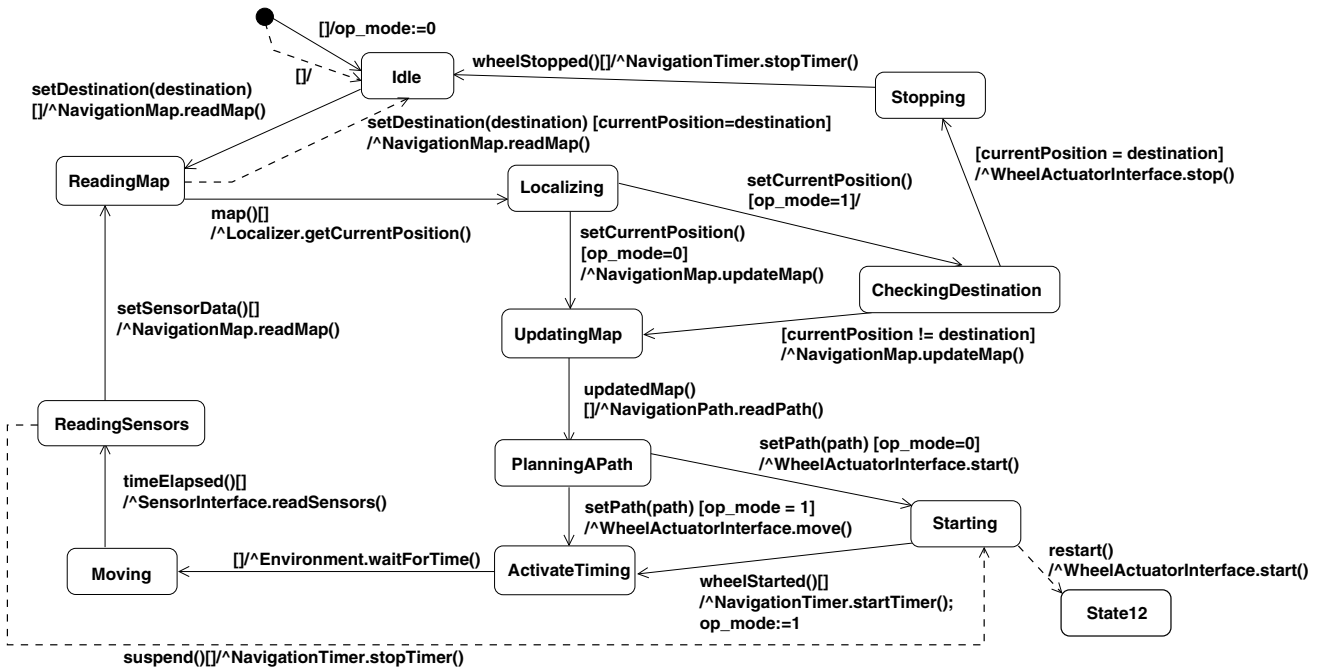**Figure 4. RNS sequence diagrams**



**Figure 5. State diagram for** NavigationControl

sign of the target system. Another possibility is to select one of these behavioral models, improve it manually, and use it as the starting point for the MDD of the autonomic system. Lastly, a developer may use the behavioral model without modification.

## 5. Conclusions and Future Work

We have presented a method for using digital evolution to generate interacting state diagrams that support key scenarios, satisfy functional properties, and extend an existing behavioral model. We applied this method to generating target systems for an autonomic system. In the following, we discuss the ramifications of this approach and also how we plan to address some of its limitations in future work.

**Scalability.** One potential concern is how our approach will scale when used with larger applications. In general, there are two main scalability challenges: (1) enabling the organisms to evolve increasingly complex and large diagrams, and (2) model checking these diagrams to verify
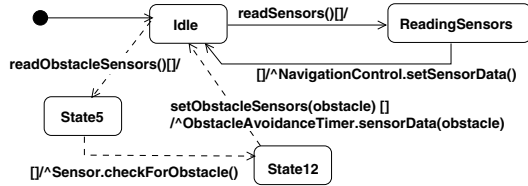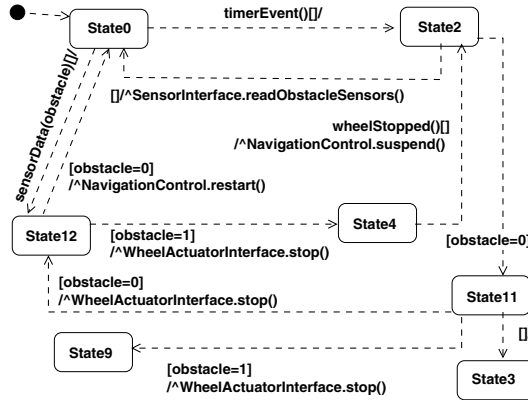
**Figure 6. State diagram for** SensorInterface



**Figure 7. State diagram for** ObstacleAvoidance-Timer

that they adhere to functional properties. We note that the first challenge must also be addressed by all human developers and synthesis algorithms and the second challenge is common to all model checking approaches. Using AVIDA, however, we can potentially reduce the burden on human developers, while automatically exploring innovated solution spaces that would otherwise not be considered due to designer bias. Two methods for addressing these challenges are model abstraction and incremental development [33]. Specifically, a developer may use AVIDA to generate an abstract model of the system in which critical behavior is specified in great detail, whereas less critical behavior is specified at a higher level of abstraction. Additionally, our approach is amenable to incremental development. A developer may divide the scenario and property tasks into multiple increments, where each increment corresponds to a single AVIDA experiment. Thus, the state diagrams generated by AVIDA organisms for an increment will be used as the seed state diagrams for the subsequent increment. Indeed, we were able to generate a set of compliant state diagrams that were comparable to those presented for the case study using the incremental approach.

**Performance.** The performance of an AVIDA experiment is dependent upon the complexity of the tasks, seed state diagrams, and available computational resources. The experiments for the case study generated compliant state diagrams in under 24 hours running on a high performance computing cluster. We have implemented some and are investigating other performance improvements. Currently, we have improved performance using tasks. The selection of tasks is critical to enable the AVIDA organisms to more rapidly generate complex behavioral models. These tasks must reward for partial correctness (i.e., by rewarding for partial scenarios, software engineering metrics, or witness traces). To assist the developer in constructing such tasks, we have provided three generation guidelines. Additionally, we propose a task ordering to further improve performance. Specifically, we can specify that an organism must generate state diagrams that receive the maximum reward for the checkScenario tasks prior to using the checkSyntax, checkWitness, and checkProperty tasks. This strategy improves performance because Spin is a computationally-intensive process, even when not used in combination with AVIDA. Once organisms have evolved to the point where they generate models that are evaluated with Spin, the rate of the experiment slows drastically, as Spin is executed hundreds of times per update. Since we intend for AVIDA to be used offline during development to identify compliant state diagrams, time has not been an issue for our preliminary studies.

**General directions for future work.** There are several possible areas for future work. First, we are interested in creating additional tasks, e.g., checking for coupling, cohesion, and deadlocks, to drive the evolutionary process to select organisms that generate even more optimized models. Second, we are interested in exploring techniques to increase the readability of our generated diagrams, either through additional tasks or post-processing. Additionally, we are investigating the potential of leveraging synthesis (e.g., [8–13,13–15]) and/or machine learning (e.g., [34–36]) approaches to generate an initial behavioral model that is then manipulated by the AVIDA organisms to create potentially more robust and resilient compliant behavioral models. This combination of techniques had the potential to improve the performance of AVIDA, but may also limit the exploration of the solution space. In addition to these improvements and extensions of our work on generating state diagrams, digital evolution might provide a means to explore solution spaces for other software engineering problems, such as generating UML behavioral models for members of a software product line or detecting and mitigating feature interaction.

## Acknowledgements

# References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] D. C. Schmidt, "Model-Driven Engineering," *IEEE Computer*, vol. 39, no. 2, 2006.

[3] R. Allen, R. Douence, and D. Garlan, "Specifying and analyzing dynamic software architectures," *Lecture Notes in Computer Science*, vol. 1382, 1998.

[4] C. Canal, E. Pimentel, and J. M. Troya, "Specification and refinement of dynamic software architectures.," in *WICSA*, pp. 107–126, 1999.

[5] M. S. Feather, S. Fickas, A. V. Lamsweerde, and C. Ponsard, "Reconciling system requirements and runtime behavior," in *IWSSD '98: Proceedings of the 9th International Workshop on Software Specification and Design*, 1998.

[6] A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu, "Towards requirements-driven autonomic systems design," in *DEAS '05: Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software*, (St. Louis, MO, USA), pp. 1–7, 2005.

[7] J. Zhang and B. H. C. Cheng, "Model-based development of dynamically adaptive software," in *ICSE '06: Proceeding of the 28th international conference on Software engineering*, (New York, NY, USA), pp. 371–380, ACM Press, 2006.

[8] J. Whittle and P. K. Jayaraman, "Generating hierarchical state machines from use case charts," in *14th IEEE International Requirements Engineering Conference (RE'06)*, (Washington, DC, USA), pp. 16–25, 2006.

[9] D. Harel, H. Kugler, and A. Pnueli, "Synthesis Revisited: Generating Statechart Models from Scenarios-Based Requirements," in *Formal Methods in Software and System Modeling*, pp. 309–324, 2005.

[10] S. Uchitel, J. Kramer, and J. Magee, "Synthesis of behavioral models from scenarios," *IEEE Trans. Softw. Eng.*, vol. 29, no. 2, pp. 99–115, 2003.

[11] I. Kruger, R. Grosu, P. Scholz, and M. Broy, "From MSCs to statecharts," *In DIPES'98. Kluwer.*, 1999.

[12] R. Alur, K. Etessami, and M. Yannakakis, "Inference of message sequence charts," *IEEE Transactions on Software Engineering*, vol. 29, no. 7, pp. 623–633, 2003.

[13] S. Uchitel, G. Brunet, and M. Chechik, "Behaviour model synthesis from properties and scenarios," in *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pp. 34–43, 2007.

[14] B. Jobstmann and R. Bloem, "Optimizations for LTL synthesis," in *Formal Methods in Computer Aided Design FMCAD '06*, (Washington, DC, USA), pp. 117–124, IEEE Computer Society, 2006.

[15] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive(1) designs," in *in Proc. Verification, Model Checking, and Abstract Interpretation (VMCAI06)*, p. 364380, 2006.

[16] J. R. Koza, M. A. Keane, M. J. Streeter, M. Mydlowec, J. Yu, and G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Springer, 2003.

[17] M. Harman, "The current state and future of search based software engineering," in *FOSE '07: 2007 Future of Software Engineering*, pp. 342–357, 2007.

[18] C. Ofria and C. O. Wilke, "Avida: A software platform for research in computational evolutionary biology," *Journal of Artificial Life*, vol. 10, pp. 191–229, 2004.

[19] R. E. Lenski, C. Ofria, R. T. Pennock, and C. Adami, "The evolutionary origin of complex features," *Nature*, vol. 423, pp. 139–144, 2003.

[20] C. Ofria, C. Adami, and T. C. Collier, "Selective pressures on genomes in molecular evolution," *J. Theor. Biology*, vol. 222, pp. 477–483, 2003.

[21] P. McKinley, B. Cheng, C. Ofria, D. Knoester, B. Beckmann, and H. Goldsby, "Harnessing digital evolution," *IEEE Computer*. In Press.

[22] D. B. Knoester, P. K. McKinley, and C. Ofria, "Using group selection to evolve leadership in populations of self-replicating digital organisms," in *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO-2007)*, (London, UK), July 2007.

[23] H. J. Goldsby, D. B. Knoester, B. Cheng, P. K. McKinley, and C. A. Ofria, "Digitally evolving models for dynamically adaptive systems," in *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, (Minneapolis, Minnesota), May 2007.

[24] G. Holzmann, *The Spin Model Checker, Primer and Reference Manual*. Reading, Massachusetts: Addison-Wesley, 2004.

[25] M. Kim, S. Kim, S. Park, M.-T. Choi, M. Kim, and H. Gomaa, "UML-based service robot software development: a case study," in *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pp. 534–543, 2006.

[26] S. Rasmussen, C. Knudson, P. Feldberg, and M. Hindsholm, "The coreworld - emergence and evolution of cooperative structures in a computational chemistry," *Physica D*, vol. 42, no. 1-3, pp. 111–134, 1990.

[27] T. S. Ray, "An approach to the synthesis of life," in *Artificial Life II* (C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, eds.), pp. 371–408, Reading, MA, USA: Addison-Wesley, 1992.

[28] C. Ofria, C. Adami, and T. C. Collier, "Design of evolvable computer languages," *IEEE Transactions in Evolutionary Computation*, vol. 6, pp. 420–424, 2002.

[29] D. C. Dennett, "The new replicators," in *The Encyclopedia of Evolution* (M. Pagel, ed.), vol. 1, pp. E83–E92, Oxford University Press, 2002.

[30] B. Beckmann, P. K. McKinley, and C. A. Ofria, "Evolution of adaptive sleep response in digital organisms," in *9th European Conference on Artificial Life*, (Lisbon, Portugal), September 2007.

[31] W. E. McUmber and B. H. C. Cheng, "A general framework for formalizing UML with formal languages," in *Proceedings of the IEEE International Conference on Software Engineering (ICSE01)*, (Toronto, Canada), May 2001.

[32] S. Konrad, H. Goldsby, K. Lopez, and B. H. C. Cheng, "Visualizing requirements in UML models," in *Proceedings of the International Workshop on Requirements Engineering Visualization (REV 2006) as part of the 14th IEEE International Requirements Engineering Conference (RE'06)*, (Minneapolis/St. Paul,MN), September 2006.

[33] S. Konrad, H. J. Goldsby, and B. H. C. Cheng, "i$^2$MAP: An incremental and iterative modeling and analysis process," in *Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems*, (Nashville, TN, USA), October 2007.

[34] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987.

[35] S. M. Lucas and T. J. Reynolds, "Learning DFA: evolution versus evidence driven state merging," in *Congress on Evolutionary Computation*, 2003.

[36] S. Luke, S. Hamahashi, and H. Kitano, ""Genetic" programming," in *Genetic and Evolutionary Computation Conference*, 1999.