In Proceedings of the 2009 IEEE Symposium on Artificial Life, Nashville, Tennessee, March 2009.

Applying Digital Evolution to the Design of Self-Adaptive Software

Benjamin E. Beckmann, Laura M. Grabowski, Philip K. McKinley, and Charles Ofria
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824
{beckma24,grabow38,mckinley,ofria}@cse.msu.edu

***Abstract*— As software developers, we strive to create computational systems that are as robust and versatile as biological organisms have evolved to be in nature. We propose a software development methodology capable of producing self-adaptive software, using digital evolution to discover behaviors and optimize solutions. Employing this methodology we present an example behavioral concept from inception to fruition on physical hardware, as a proof of concept of the approach. We evolve environmentally-aware motility behaviors through digital evolution, automatically translate the evolved programs into C code, and compile and load the programs onto mobile robots.**
Keywords: digital evolution, evolutionary computation, autonomic computing, software development, self-*, cooperation, microrobot.

I. Introduction

To design dynamic and resilient computational systems, researchers often take inspiration from nature. Living organisms exhibit an amazing ability to adapt to a changing environment, both in the short term (phenotypic plasticity) and in the longer term (Darwinian evolution). Moreover, many organisms exhibit traits that are desirable in self-managing computing systems: system monitoring (senses, awareness); short-term changes in priorities (reflexes, sleep); system reconfiguration (muscle growth, calluses); self repair (blood clotting, tissue healing); intrusion detection/elimination (immune systems). For this reason, many researchers turn to biologically-inspired methods to construct highly adaptive systems [1]–[3], both by mimicking the designs produced by nature, and by evolving new ones. While biomimetic approaches have been successful at addressing aspects of system design [2], [4], they are limited to codifying behaviors found in nature *today*, imitating the results of evolution without accounting for the process that optimized these systems to fit their environment. Evolutionary computation, on the other hand, tends to go to the opposite extreme of abstracting out the evolutionary process and applying it in a purely algorithmic form to problem solving.

A fundamentally different approach to building robust, flexible computational systems is *digital evolution* [5]. In this method, 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" are provided with limited resources whose use must be carefully balanced if they are to survive. Over generations, these organisms evolve to optimize resource usage and thrive if they are able. The environment of these organisms can be crafted such that the organisms are faced with problems comparable to the one that the researcher wants to solve. Unlike mere numerical simulations, Avida organisms possess the ability to evolve in an open-ended manner, often revealing unexpected and strikingly clever solutions. The most widely used digital evolution platform is Avida [6], developed by Ofria and colleagues, which has been used to conduct extensive research in the evolution of biological complexity [7].

In this work we investigate the application of digital evolution to the design of software exhibiting self-adaptive properties, based on high-level goals provided by the developer. A developer will be able to use this methodology to construct and optimize solutions for a target platform and environment. Specifically, we apply the methodology to the design of control software for mobile robots. We evolve behaviors in Avida, and then automatically translate them into C programs, which are compiled and loaded onto iRobot^{TM} Create mobile robots. Experiments demonstrate that the robots exhibit the desired behaviors and can be used to solve problems.

The main contribution of this paper is to demonstrate a "proof of concept" of the proposed approach. We show that this method is capable of (1) synthesizing and optimizing solutions based on high-level behavioral specifications, (2) generating adaptive behaviors, and (3) facilitating reimplementation when the target platform or environment changes.

II. Software Development Model

As described by Kephart and Chess [8], autonomic computing focuses on creating computer systems that manage themselves according to an administrator's goals. Our approach uses digital evolution to help design computational behavior needed in such systems. However, rather than addressing only runtime behavior, we also apply autonomic computing concepts to help guide the software development process. Evolving behaviors in virtual entities whose capabilities closely match those of the target platform can produce well adapted solutions that can be directly mapped into a target environment. Moreover, by introducing noise and uncertainty into the virtual environment, the evolutionary process will select solutions that are robust, efficient, and acceptable as real-world solutions.

To create such a system, physical, biological, and evolutionary principles must be packaged into a single development methodology. The system must also enable verification and validation of the resulting software, with a feedback loop to the developmental process. We propose a four-stage development model, depicted in Figure 1, to meet this need. The stages are: cultivation, translation, simulation, and deployment. Each plays a significant role in the development of robust and effective real-world solutions.

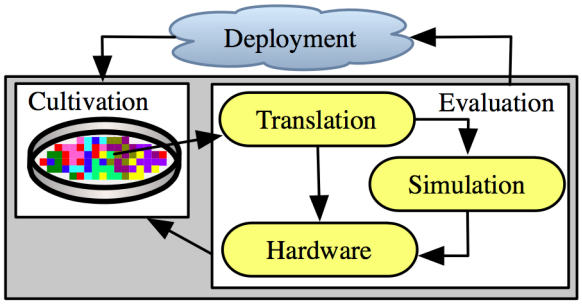


Fig. 1. Development process [9].

During the cultivation stage, digital organisms with capabilities similar to a target platform are evolved in the Avida digital evolution system, described in Section III. Effectively, Avida provides a "digital Petri dish" for evolving new computational behaviors based on high-level human guidance. Thus the solutions are produced *autonomically* – once initiated, the population lives and evolves without human interaction. By abstracting development in this manner, a developer can focus time and effort on the desired behavior of the system rather than low-level implementation.

Avida produces digital organisms whose genomes – programs coded in a specially designed language – have evolved to fit their environment. The second stage of the methodology translates one or more of these genomes (typically the most prevalent ones in the population) into a format compatible with target hardware platforms. For our prototype system we developed a tool that translates Avida genomes into C code. C was chosen due to the availability of compilers for a wide variety of architectures. The C code produced by the translator can be compiled, without modification, to execute on a multitude of computing devices, from a resource-heavy server to a constrained sensor device or even a microrobot.

The simulation and deployment stages of the development methodology are both used to test the effectiveness of evolved solutions. Any shortcomings observed in either of these stages are fed back into the cultivation stage to refine the evolved model. Simulation is often used during the early stages of development, followed by experimentation on actual hardware. Moreover, parts of the simulation stage can be automated, as in [10], to further reduce the time and cost of testing. Finally, the translated code is tested in simulation and on real-world devices to detect any inaccuracies, which feed into the cultivation and simulation stages as adjustments to the model. Through the use of this multi-stage development methodology, a developer can focus time and energy on determining desired behaviors and rely on the evolutionary process, combined with the translator, to produce a low-level implementation. Additionally, a developer can integrate existing tools such as model checkers into the cultivation stage, in order to verify that the software adheres to specified properties [11], [12].

III. Digital Evolution and the Avida System

Digital evolution (DE) [5] is a type of evolutionary computation (EC) focused on studying evolutionary processes in a virtual environment. DE allows biologists to hypothesize and test generalities of the evolutionary process, as well as for problem-solving applications. Traditional EC methods consist of iterative stochastic search through a simulated landscape guided by selection, based on a predefined fitness function. The power of the EC approach is well documented, as EC methods have been used in the design of distributed systems [13], and have been shown to be competitive with humans in many other areas of design and engineering [14].

Unlike many EC methods, digital organisms in systems such as Avida self-replicate and evolve in an open-ended manner, reminiscent of natural organisms. However, because the evolution occurs *in silico*, Avida allows complete transparency of the entire population and enables fundamental questions to be distilled into a pure form and studied free of the constraints that often hinder the study of natural organisms. Digital evolution can also be "harnessed" to help solve problems in science and engineering, including the design of computing systems that would benefit from behaviors similar to those exhibited by natural organisms [15]. Recently, Avida has been used to cultivate digital organisms that exhibit self-organizing [16], [17], self-healing [16], and adaptive resource aware behavior [18], as well as automated state-diagram construction [11] used in requirements engineering. For the study described here, we have extended Avida to handle movement of embodied digital organisms, and taken care to ensure that evolved solutions produced by Avida can be directly mapped and tested in both simulation and real-world hardware.

A. Avida Operation

In Avida, a population of individual digital organisms (or Avidians) compete for space in a fixed size lattice of *cells*. Each cell can contain at most one organism comprised of a circular list of assembly-like instructions (its genome) and a virtual CPU, as shown in Figure 2. A standard virtual CPU is made up of three general purpose registers (AX, BX, and CX), two stacks, and two general purpose heads. One head (Flow) is used as a target for jumps and the other (IP) is an instruction pointer. When executed, the instructions in an organism's genome operate on elements of its virtual CPU. Avida instructions can perform flow control operations, manipulate the content of an organism's virtual CPU, collect information from the environment, or cause the organism to move around the lattice. Instruction execution costs the organism both virtual CPU cycles and energy. The number of virtual cycles and energy expended depends on the instruction

being executed. For example, executing an instruction that activates a motor consumes more energy than executing an arithmetic instruction that manipulates registers.

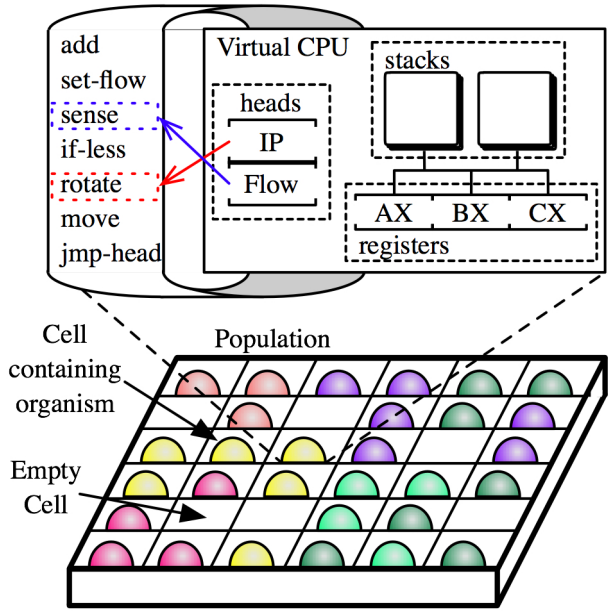


Fig. 2. Avida population (bottom) and composition of a digital organism: genome (top left), virtual CPU (top right)

Typically, Avidians exist in a single population, evolving from an ancestral organism capable only of replication. This organism's genome consists of a copy loop and a large number of "no-operation" instructions (used as blank tape). An organism replicates by copying its genome and placing the offspring into a random neighboring cell, often overwriting the previous inhabitant. During copying, mutations occur probabilistically, in the form of instructions being inserted, deleted, or replaced. The Avida instruction set is designed to be resilient: any program is syntactically correct, though it many not have a useful function [19]. All behavior beyond replication must enter the genome through mutations. Organisms gain energy by performing *tasks* to metabolized user-specified resources, which typically give them a competitive advantage.

An alternative to evolving a single, large population is to divide the Avida world into independent regions called *demes* (see Figure 3). This approach is useful when evolving group-level cooperative behaviors or, as in this paper, to explore movements of individual organisms. When an organism is injected into a cell within a deme, it receives a fixed energy budget and its metabolic rate is set using equation $MetabolicRate = \frac{Energy}{InstructionsBeforeZeroEnergy}$. An organism's metabolic rate is used by the Avida scheduler to divide virtual CPU cycles among the organisms in the population, and to adjust the energy cost associated with executing instructions. The higher an organism's metabolic rate, the more virtual CPU cycles it executes per unit time, and the higher the energy cost per instruction. For the runs presented in this paper, an organism's metabolic rate was set so that $10,000$ of the simplest arithmetic instructions can be executed before an organism's energy is depleted.

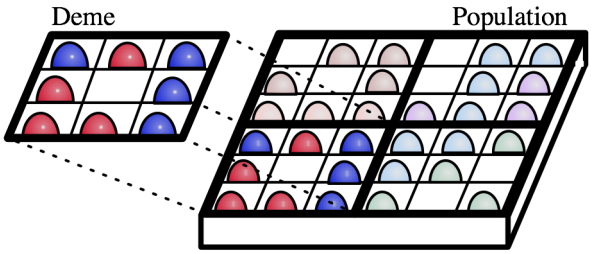


Fig. 3. Structure of an Avida population including demes and cells.

To facilitate deme-level evolution, we turned off individual replication of organisms and instead replicated them into another deme whenever a deme-level task (or *predicate*) has been satisfied. The more quickly a deme satisfies a predicate, the faster it will replicate and spread its genome throughout the population.

Initially, each deme is assigned a copy of an empty genome, containing only no-ops, as shown on the left of Figure 4. Once a deme has either reached an age limit or satisfied a deme-level predicate, the deme is replicated. As in individual replication, mutations can occur during deme replication. In this study, there is a 0.75% probability that the newly created genome is different from its parent. A copy of the new genome is then inserted into a random deme, and both the original deme and the one containing the offspring are restarted.

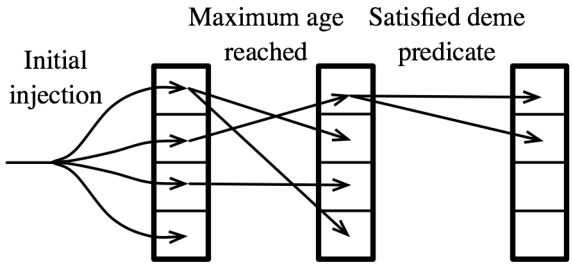


Fig. 4. Example showing deme initialization and replication.

In our case study, we evolved mobility behaviors by limiting each deme to a single organism. This deme configuration provided the organisms with space in which to live, move, and eventually evolve the desired behaviors, which were specified as deme-level predicates.

B. Genome Translation

We are exploring two different approaches to translating Avidian behaviors into software for real-world systems. In one approach, described in [12], Avidians evolve to construct state diagrams describing behaviors. These diagrams are then translated into code using traditional code generation techniques. In the other approach, used in the study presented here, the genomes themselves describe the behaviors of interest, and we translate them directly into C code.

The virtual machine used within Avida has a von Neumann architecture and can easily be transformed into C, however, other more advanced architectures can be used. As each instruction is either atomic or modified by no-operation instructions that immediately follow it in the code, we can translate a genome in a single pass. To perform the translation we use FLEX, a freely available lexical analyzer generator, and BISON, a parser generator, along with a custom generator function. Once the translation is complete, the generated code and virtual machine are cross-compiled for the target platform. By translating the code rather than executing a virtual machine, we achieve increased execution speed and decreased memory usage, allowing limited resource devices to execute the translator's output.

IV. Methods

To demonstrate the feasibility of the proposed development methodology, we present an example behavior from concept to fruition on physical hardware. We provide guidance for construction of the final product only through a single deme-level predicate. Multiple predicates and other organism-level tasks could be added to obtain more complicated behavior. However, our goal in this paper is to show proof of concept, rather than a production-level system. We demonstrate that through the use of the development methodology, adaptive behaviors can be produced for real-world hardware, specifically an iRobot™ Create system.

The goal of the case study is to evolve control software for robots, of varying capabilities, that allows them to search for and move to a light source. Event detection and location discovery are common problems addressed in adaptive behavior and evolutionary robotics literature [20], [21]. We will present evidence that an embodied agent's capabilities can be accurately mapped into the development methodology and produce effective results. In the following experiments an agent will be evolved and embodied in virtual hardware during the cultivation stage. Once cultivation has completed, the evolved solution is translated and loaded onto an iRobot™ Create, and its behavior is evaluated.

A. Relevant Avida Instructions

In this study, an individual organism can interact with its local environment in one of two ways: movement and sensing. An organism can move to a neighboring cell by executing the MOVE instruction. The destination cell is determined by which of its eight neighboring cells an organism is facing, as shown in Figure 5. An organism can change its facing by executing a rotate instruction, which causes the organism to rotate its facing to the right or left by one cell. Besides movement, an organism can also interact with its environment through sensing. By executing a sense instruction, an organism can gain information about the current environmental conditions. An example sense instruction is SENSE-CELL-DATA, shown in Figure 5. The SENSE-CELL-DATA instruction is similar to a majority of Avida instructions in that its execution can be modified by a no-op instruction that immediately follows the SENSE-CELL-DATA instruction in a genome. By default, the SENSE-CELL-DATA instruction will place a sensed value into an organism's BX register. However, if a no-op instruction immediately follows the SENSE-CELL-DATA instruction, then the sensed data will be placed in the register specified by the no-op. For example, if a NOP-A instruction immediately follows a SENSE-CELL-DATA instruction then the sensed data will be placed in the AX register. By allowing instructions to be modified in this manner the Avida designers have removed the need for instruction-level arguments. This novel representation increases the robustness of the Avida language to mutations, which would be disastrous to most general purpose programming languages. It also implicitly disallows syntactically incorrect genomes because an instruction's format can never be violated.

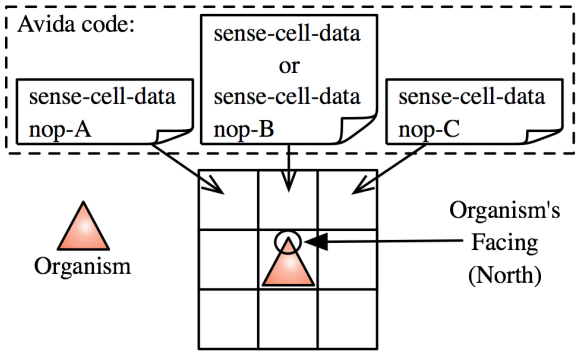


Fig. 5. Depiction of SENSE-CELL-DATA

To increase similarity between the real-world and the Avida world, all of the movement and sensing instructions used in the following experiments have a higher energy cost than other instructions that do not require external devices, such as sensors and motors. In addition to increased cost, three separate environmental sensing facilities were used during cultivation depending on the capabilities of the target platform. The same base platform, an iRobot™ Create, is modeled and used for comparison. However, separate treatments with various sensing capabilities are presented to demonstrate the flexibility of the development methodology.

B. Treatments

We experimented with different sensing capabilities in our Avida runs, illustrating the effectiveness of the methodology in producing results for various embodied agents. The first of these sensing capabilities uses the COLLECT-CELL-DATA instruction. This instruction enables an organism to sense the intensity of light at its current position. The second capability enables an organism to separately sense the intensity of a light source in three directions using the SENSE-CELL-DATA instruction described above. Specifically, the organism can sense the light intensity directly in front, in front and to the left, and in front and to the right of its current location. With the appropriate arithmetic instructions, these sensed measurements can be compared to determine in which direction the light

is the most intense. The final sensing capability combines directional sensing with a rotate operation. Using the SENSE3-AND-ROTATE instruction, an organism can sense the light intensity within its environment and rotate to face the highest intensity. As will be shown, this level of functionality is highly suitable to the target task, and can be used as a building block for more complex evolved behaviors.

To encourage the evolution of object detection and location behaviors we apply selection pressures that reward organisms for efficiency in reaching the target, a simulated light bulb. Since instructions have different energy costs, variations in execution sequences can produce diverse energy consumption among organisms. Once an organism reaches a target, that organism's genome is replicated and placed into the current deme and into another randomly selected deme. A bonus is given to the new organisms based on the amount of energy remaining in the organism. By applying selective pressures in this way, we reward individual organisms that can detect and move to the simulated light bulb in an efficient manner. In the following section we discuss the experimental results of these three treatments. Each treatment consists of 20 replicate runs, each containing 500 single-organism demes and lasting for $250,000$ updates. Each individual is allowed to live for 1000 updates or until its deme is replaced. On average, an organism will execute a single instruction during an update.

V. Experimental Results

Initially we expected the evolved solutions that were exposed to the SENSE-CELL-DATA instruction to exhibit better performance than those exposed to the COLLECT-CELL-DATA instruction, due to the increased functionality of the former. However, this was not the case. Even though SENSE-CELL-DATA enables an organism to sense its environment in more detail, we never observed an evolved organism that used the functionality entirely. We speculate that this effect arose due to the energy cost associated with executing this instruction and the simplicity of the environment. Specifically, organisms evolved that could perform the target task with only partial information about the environment. This success is encouraging to the extent that the evolutionary process did not need complete information of an organism's environment to complete the task. Actually, in both the COLLECT-CELL-DATA and SENSE-CELL-DATA treatments, evolved organisms exhibited arc-like movement, where an organism would either move straight ahead if conditions were improving, or turn if not. In general, dominant genomes produced by these runs contained a loop, similar to the sample code shown in Table I. The organism senses the environment and moves or rotates depending on whether or not the light intensity is increasing. Over the course of evolution, these loops were optimized to consume less and less energy, and we can see from Figure 6 that on average both treatments require a similar amount of time to detect and move to the simulated light source.

In addition to the total time required for an organism to move to the target location, we also tracked individual movments. Plotted in Figure 7 is the average number of

TABLE I

SAMPLE PORTION OF EVOLVED GENOME USING COLLECT-CELL-DATA

COLLECT-CELL-DATA NOP-C	sense intensity in cell store in CX
MOVE	move one cell
COLLECT-CELL-DATA	sense intensity in new cell store in BX
SWAP-STK	change stack; no effect
PUSH NOP-B	push BX on to stack no effect
IF-LESS	is BX < CX
ROTATE-RIGHT	then rotate right; else skip
ADD NOP-B	overwrite BX has no effect
MOV-HEAD	repeat

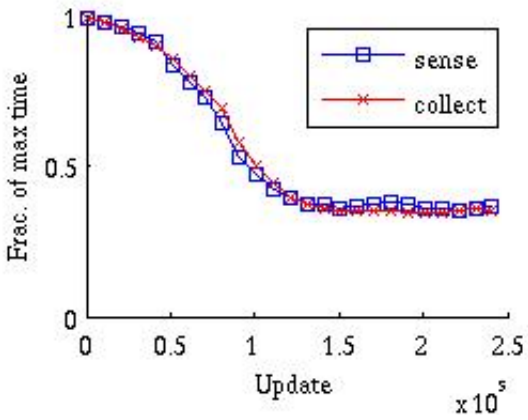


Fig. 6. Average fraction of total allowed time required for a deme to be replicated in both the COLLECT-CELL-DATA and SENSE-CELL-DATA treatments.

organisms in the SENSE-CELL-DATA treatment populations that moved toward, away, and neutrally with regard to the location of the light source. A similar graph was also generated for the COLLECT-CELL-DATA treatment, but the results were not significantly different, and are not shown here. A notable aspect of this graph is that, in general, the same number of organisms are moving toward, away and neutrally with respect to the light source. This result is intuitive since mobility is dependent on an organism's memory of sensed values from multiple locations. In order for the organism to change its heading it must be subjected to a movement that either was neutral or away from the target. Thereby, the organism depends on this type of "bad" movement to find the target, regardless of its adverse effect on energy consumption.

Building on the results seen in the two previous treatments and the prospect of evolving more complex behaviors in the future, we exposed populations of organisms to the SENSE3-AND-ROTATE instruction. As described above, the SENSE3-AND-ROTATE instruction automatically rotates an organism to face the cell with the highest measured light intensity. This

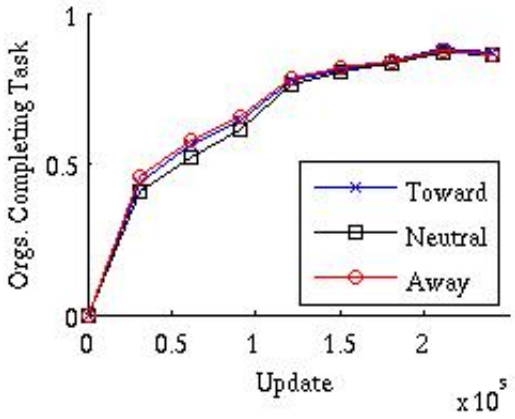


Fig. 7. Average fraction of organisms in a population that have moved toward, away, and neutrally with respect to the light source in the SENSE-CELL-DATA treatment.

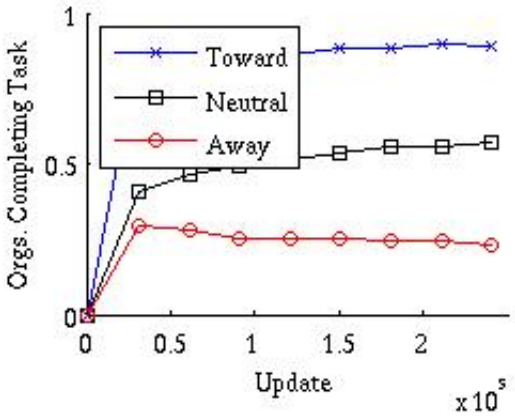


Fig. 9. Average fraction of organisms in a population which moved toward, away, and neutrally with respect to the light source in the SENSE3-AND-ROTATE treatment.

instruction is an example of a building block that a developer might consider adding to a system based on the problem domain. It also illustrates the flexibility of the methodology to handle various levels of target device capabilities. With the addition of the SENSE3-AND-ROTATE instruction the average fraction of the maximum time allow to complete the task is drastically reduced (Wilcoxon rank sum test $\alpha = 0.01$ $p = 6.8 \times 10^{-8}$), as shown in Figure 8. Also, the average fraction of organisms in the population that move toward the light is significantly higher than those that move away (Wilcoxon rank sum test $\alpha = 0.01$ $p = 6.73 \times 10^{-8}$), as shown in Figure 9.

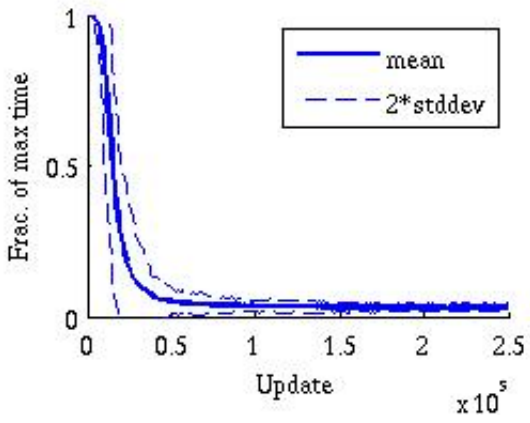


Fig. 8. Average fraction of total allowed time required for a deme to be replicated in the SENSE3-AND-ROTATE treatment.

Table II shows a portion of an evolved genome, produced by the cultivation stage using the SENSE3-AND-ROTATE instruction. This partial genome is shorter and simpler than the partial genome shown in Table I, but accomplishes the same task. Even when energy costs for executing movement and sensing instructions are increased by an order of magnitude, this treatment still produces solutions capable of performing the task. However, the other two treatments are inhibited by such an increase in energy cost because of the increased number of instructions required to successfully complete the task.

TABLE II
PORTION OF EVOLVED GENOME EXPOSED TO THE SENSE3-AND-ROTATE INSTRUCTION

sense3-and-rotate	rotate in direction of highest intensity
sense3-and-rotate	
move	move straight
mov-head	repeat

Figure 10 presents a sample of the Avida results in a 50×50 grid. The figure shows a three dimensional representation of the gradient used in the experiments. The black lines represent paths of 3 different organisms that used the SENSE3-AND-ROTATE instruction to follow the gradient. Depending on the organism's initial facing, the beginning of a path may display a few movements that allow the organism to orient itself to face the high point of the gradient, and then move in that direction. Since an organism is rewarded for energy conservation, more direct paths are preferred.

We translated several evolved genomes into C code, compiled them, and loaded them onto an iRobot™ Create robot. We then placed the robot in a room with a light source and filmed the resulting behavior. Four sample clips from one of these movies are shown in Figure 11. The images show a sequence of clips from the robot's initial position, orientation, its progress toward the light source, and then finally touching the light.

VI. RELATED WORK

In this paper we proposed a methodology capable of producing autonomic behaviors in software systems, by using digital evolution to realize a *developer's* high-level goals. Many traditional approaches to autonomic computing, where an *administrator's* goals guide the runtime behavior of the

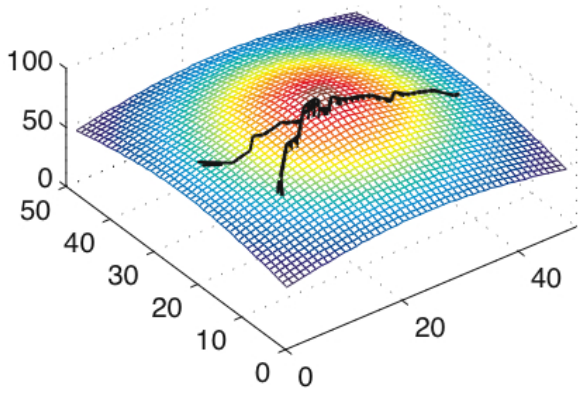
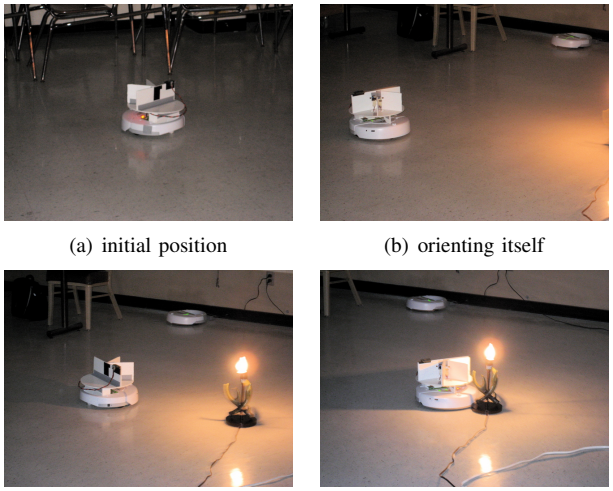


Fig. 10. Example paths of dominant evolved genomes from the SENSE3-AND-ROTATE treatment, superimposed on the gradient used in the cultivation stage [9].



(a) initial position (b) orienting itself

(c) moving toward higher intensity (d) light source found

Fig. 11. Clips from a movie that shows translated code produced by the SENSE3-AND-ROTATE treatment executing on an iRobot Create system [9]. The video is available at www.cse.msu.edu/thinktank/mobility

system, have been proposed, including those based on architectural models [22], [23], infrastructure for engineering emergent behavior [24], model-driven development [25], and design patterns [4]. In addition, the development of emergent systems [26] has been shown to produce robust, scalable, self-* systems [2], [3], [27]. Control over the population of agents [28], and methods to quantify their behaviors [29], show considerable promise. Our work complements these methods by using digital evolution to explore, as part of the software development process, possible solutions that realize autonomic behavior.

The case study presented in the paper is related to work in evolutionary robotics [1], which addresses the automatic generation of autonomic robots, and has been used to investigate questions in cognitive and neuroscience [30]. Work in this area has provided insight into the evolutionary conditions necessary for emergent communication [31], as well as communication protocols [32], multi-robot cooperation [33], and the concurrent design of a robot's morphology and its controller [34]. By combining autonomic computing with evolutionary computation, specifically evolutionary robotics, the proposed methodology provides a means to harness the power of evolution and apply it to the development of robust, scalable, self-* systems.

VII. Conclusions and Future Directions

We have described an autonomic software development model based on Darwinian evolution that is capable of producing adaptive and autonomic behaviors. Through the use of this methodology we have shown that various device capabilities can be successfully modeled and cultivated to produce solutions that, once translated, can execute directly on real-world hardware. We have also shown that a developer can direct the process by providing high-level requirements for the evolved solution, and easily handle a change in the target platform's morphology by altering the capabilities modeled in the evolutionary process.

Employing evolution as the "designer" of software is not without its limitations. Well crafted high-level goals and building blocks that allow evolution to produce desired results are needed. Also, the evolutionary system may present hurdles to overcome. For instance, the virtual CPU architecture presented in this paper is capable of computing complex solutions, however the number of instructions required may be too large for the evolutionary process to stumble upon. More capable virtual architectures are under development. Even with these limitations, our process has been shown to produce results acceptable for deployment of real-world devices.

We intend to extend this work to include evolved behaviors for swarm robots. We are beginning to work on problems such as division of labor, communication optimization, and predator avoidance. We also plan to address evolution of the morphology of the target platform along with its controller. By adding this dimension we hope to produce cohesive robot-controller combinations that are competitive with human-designed systems.

Traditional software development methods are being challenged by the ever-increasing complexity of today's software systems. As the cost of developing these systems grows, alternatives that construct adequate solutions with less manpower are appealing. Evolution provides us with a method capable of producing solutions for increasingly complex environments. Enabling software development methods to harness this power, through digital evolution, may provide a means to produce economical software solutions that exhibit the robustness, flexibility, and adaptability that abound in solutions produced by natural evolution.

Further Information. Papers on digital evolution and the Avida software are available at `http://devolab.msu.edu`. Information on evolving adaptive and cooperative behavior can be found at `http://www.cse.msu.edu/thinktank`.

Acknowledgments. The authors gratefully acknowledge the contributions of Andres Ramirez and Wesley Elsberry. This research was supported by a grant from the Cambridge Templeton Consortium,"Emerging Intelligence: Contingency, Convergence and Constraints in the Evolution of Intelligent Behavior," by NSF grants CCF-0523449, CCF-0750787, CNS-0751155, CCF-0820220, and EIA-0219229, by U.S Army grant W911NF-08-1-0495, and by the MSU Quality Fund.

REFERENCES

[1] S. Nolfi and D. Floreano, *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines*. Cambridge, MA: MIT Press, 2001. 2001 (2nd print), 2000 (1st print).

[2] R. J. Anthony, "Emergence: A paradigm for robust and scalable distributed applications," in *Proceedings of the First International Conference on Autonomic Computing*, (Los Alamitos, CA, USA), pp. 132–139, IEEE Computer Society, 2004.

[3] T. H. Labella, M. Dorigo, and J. L. Deneubourg, "Division of labor in a group of robots inspired by ants' foraging behavior," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, no. 1, pp. 4–25, 2006.

[4] O. B. et al., "Design patterns from biology for distributed computing," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, no. 1, pp. 26–66, 2006.

[5] C. Adami, C. A. Ofria, and T. C. Collier, "Evolution of biological complexity," *Proceedings of the National Academy of Sciences*, vol. 97, pp. 4463–4468, April 2000.

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

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

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

[9] B. Beckmann, L. M. Grabowski, P. McKinley, and C. Ofria, "An autonomic software development methodology based on Darwinian evolution (poster summary)," in *The 5th IEEE International Conference on Autonomic Computing*, (Chicago), June 2–6 2008.

[10] J. Bongard, "Synthesizing physically-realistic environmental models from robot exploration," in *Advances in Artificial Life - Proceedings of the 9th European Conference*, vol. 4648 of *Lecture Notes in Computer Science*, pp. 806–815, Springer, 2007.

[11] H. J. Goldsby, D. B. Knoester, B. H. C. 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*, (Minneapolis, Minnesota), May 2007.

[12] H. J. Goldsby, B. H. C. Cheng, P. K. McKinley, D. B. Knoester, and C. A. Ofria, "Digital evolution of behavioral models for autonomic systems," in *Proceedings of the 5th International Conference on Autonomic Computing*, (Chicago, Il), June 2008.

[13] S. Pierre and G. Legault, "A genetic algorithm for designing distributed computer network topologies," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 28, pp. 249–258, Apr 1998.

[14] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, Mass.: MIT Press, 1992.

[15] P. McKinley, B. Cheng, C. Ofria, D. Knoester, B. Beckmann, and H. Goldsby, "Harnessing digital evolution," *Computer*, vol. 41, pp. 78–87, January 2008.

[16] 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*, (London, UK), July 2007.

[17] B. Beckmann, P. K. McKinley, D. B. Knoester, and C. Ofria, "Evolution of cooperative information gathering in self-replicating digital organisms," in *Proceedings of IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, (Boston, MA, USA), pp. 65–74, MIT Press, July 2007.

[18] B. Beckmann, P. K. McKinley, and C. Ofria, "Evolution of an adaptive sleep response in digital organisms," in *Advances in Artificial Life - Proceedings of the 9th European Conference*, vol. 4648 of *Lecture Notes in Computer Science*, pp. 233–242, Springer, 2007.

[19] C. Ofria, C. Adami, and T. C. Collier, "Design of evolvable computer languages," *IEEE Transactions in Evolutionary Computation*, vol. 17, pp. 528–532, 2002.

[20] J. Blynel and D. Floreano, "Levels of dynamics and adaptive behavior in evolutionary neural controllers," in *Proceedings of the 7th International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, (Cambridge, MA, USA), pp. 272–281, MIT Press, 2002.

[21] R. D. Beer and J. C. Gallagher, "Evolving dynamical neural networks for adaptive behavior," *Adaptive Behavior*, vol. 1, no. 1, pp. 91–122, 1992.

[22] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart, "An architectural approach to autonomic computing," in *Proceedings of the First International Conference on Autonomic Computing*, (Los Alamitos, CA, USA), pp. 2–9, IEEE Computer Society, 2004.

[23] J. Strassner, "Focale - a novel autonomic networking architecture," in *First Latin American Autonomic Computing Conference (LAACS)*, July 2006.

[24] J. Beal and J. Bachrach, "Infrastructure for engineered emergence on sensor/actuator networks," *IEEE Intelligent Systems*, vol. 21, no. 2, pp. 10–19, 2006.

[25] R. Calinescu, "Model-driven autonomic architecture," in *4th International Conference on Autonomic Computing*, (Los Alamitos, CA, USA), p. 9, IEEE Computer Society, 2007.

[26] S. Johnson, *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. Scribner, 2002.

[27] M. Jelasity and O. Babaoglu, "T-Man: Gossip-based overlay topology management," in *Proceedings of Engineering Self-Organising Applications (ESOA'05)*, July 2005.

[28] T. White, B. Pagurek, and D. Deugo, "Management of mobile agent systems using social insect metaphors," in *21st IEEE Symposium on Reliable Distributed Systems*, pp. 410–415, 2002.

[29] T. D. Wolf, G. Samaey, T. Holvoet, and D. Roose, "Decentralised autonomic computing: Analysing self-organising emergent behaviour using advanced numerical methods," in *Proceedings of the Second International Conference on Autonomic Computing*, (Los Alamitos, CA, USA), pp. 52–63, IEEE Computer Society, 2005.

[30] I. Harvey, E. D. Paolo, R. Wood, M. Quinn, and E. Tuci, "Evolutionary robotics: A new scientific tool for studying cognition," *Artificial Life*, vol. 11, no. 1-2, pp. 79–98, 2005.

[31] D. Floreano, S. Mitri, S. Magnenat, and L. Keller, "Evolutionary conditions for the emergence of communication in robots," *Current Biology*, vol. 17, pp. 514–519, March 2007.

[32] D. Marocco and S. Nolfi, "Emergence of communication in teams of embodied and situated agents," in *Proceedings of the 6th Evolution of Language Conference*, 2006.

[33] M. D. et. al., "Evolving self-organizing behaviors for a swarm-bot," *Auton. Robots*, vol. 17, no. 2-3, pp. 223–245, 2004.

[34] H. Lipson and J. B. Pollack, "Automatic design and manufacture of robotic lifeforms," *Nature*, vol. 406, pp. 974–978, August 2000.