Experiments in Design Synthesis when Behavior is Determined by Shape

Eric Schweikardt

Cornell Computational Synthesis Lab

Department of Mechanical and Aerospace Engineering,

Cornell University

(+1) 303 517 4826

ees68@cornell.edu

Mark D. Gross

Computational Design Lab

School of Architecture

Carnegie Mellon University

(+1) 412 268 2767

mdgross@cmu.edu

Abstract

As we rapidly approach the day of transitive materials, made of individual elements that sense and actuate and can be programmed and reprogrammed it is time to think about how to design things using these new materials. Our roBlocks construction kit toy teaching children about emergent behavior in complex systems serves as an example for investigating the challenges of designing things made of transitive materials. The roBlocks kit comprises heterogeneous modular robotics components that exhibit modularity, one-to-one mapping between form and behavior, and non-hierarchical control; and these features make it appropriate for experimenting with emergent behavior. However as the numbers of robotic components scales to the orders of magnitude needed to consider them as material these same features also make it difficult to apply traditional methods to design constructions with desired behaviors. To understand this design space we built the Erstwhile Agent, which uses an evolutionary approach to automatically synthesize roBlocks constructions to meet specified desiderata.

Keywords programmable matter, evolutionary algorithms, material computing, automated design synthesis

1. Introduction

Today we are on the threshold of having computational materials whose physical properties are mutable, programmed by code that runs on units of the material to produce —in aggregate—emergent behavior. Recently several research groups have experimented with building and programming ensembles of modular robots [1-6]. As work on modular robotics advances, yielding smaller more mechanically sophisticated and more easily programmed units it will be appropriate to think of the ensembles of robots as material, or "programmable matter."

We have been working with a kit of modular robots that we designed as a construction kit for children. We called our kit "roBlocks": Our purpose was to introduce young people to computational thinking and elementary robotics, without requiring prior in-depth knowledge of electronics, mechanics, and programming. In the spirit of constructionist learning [7-9], we intend that through this playful activity children will acquire fundamental insights into how behavior emerges in complex systems.

Our RoBlocks kit comprises many different kinds of modules, each with a predefined behavior. Different sensors, actuators, and calculating functions are encapsulated into individual physical blocks, and the behavior of a robot construction emerges from the combination of local interactions between its modules. By snapping blocks together, users simultaneously construct the physical robot and specify its behavior.

When form and behavior are tightly coupled like this we start to observe interesting system properties. Certain kinds of configuration produce certain behaviors; control algorithms take spatial form. And because control is distributed, when hardware is removed from an ensemble of blocks, performance degrades gracefully. However, users find that as they add modules to a construction, behavior becomes more difficult to predict. As the blocks are cubes, each new module can have up to six new connections, one on each face. With multiple, and often cyclic, data paths traditional conditional instructions give way to complex, interlocked chains of causality. The blocks are simple and easy to work with and capable of complex emergent behavior but designing a construction to satisfy a specific non-trivial requirement can be challenging.

We began to see our construction kit as more than a children's toy. We began to see it as a prototype for a programmable matter, in which objects and materials are composed of large numbers of individual unit robots. Our experiences building small roBlocks constructions led us to think about how one might design for this new medium made of modular robots.

Intuition and conventional design expertise serves when working with small constructions; but it is unlikely to scale up for constructions comprised of hundred or thousands of modular robot units. The usual way to design large systems is by constructing hierarchies of assemblies and subassemblies. However, the distributed nature of control in roBlocks, and the way in which global behavior emerges from local conditions, challenges the usual strategy of hierarchic design.

To understand how one might manage the complexity inherent in constructions made of material like roBlocks, we therefore investigated strategies for automating the design of roBlocks constructions. Specifically, we developed an evolutionary algorithm to generate and test roBlocks constructions to produce desired behaviors. We call this program the Erstwhile Agent. The Erstwhile Agent uses evolutionary techniques to design and evaluate constructions that satisfy requirements set by a user. We found that the Erstwhile Agent can produce successful designs.

In the following sections we report on our journey from building a children's robot construction kit toy to understanding this kit as a model for computational material, and on our investigation of strategies for designing with this new material. In section 2 we describe the roBlocks construction kit: the rationale behind it, the kit itself, and children's experiences working with the kit. Then in section 3 we describe the Erstwhile Agent, an evolutionary algorithm for designing with roBlocks. The Erstwhile Agent manipulates a software simulation of roBlocks to generate constructions that meet requirements we specify in a fitness function. We present results from our first experiments with the Erstwhile Agent. Based on these results we reflect, in section 4, on the nature of designing with materials that consist of large numbers of modular units whose characteristics are similar to roBlocks.

2. roBlocks – a Heterogeneous Modular Robot Construction Kit

2.1 The roBlocks Kit

Our roBlocks construction toy consists of a variety of small (40 mm) plastic blocks (see figure 1). Blocks snap together with magnets that are embedded in each face. They pass power and data through the magnets and gold spring probes that are also embedded in each face. The kit has three main kinds of blocks. *Sensor blocks* take input from the physical environment, for example detecting light, sound, distance, or motion. *Action blocks* act on the physical environment, producing light, sound, or motion. *Think blocks* do not sense the environment or act on it, but operate on information that the robot has taken in. In addition to sensor, action, and think blocks, the kit also provides a battery block that powers a construction, and blocker blocks that limit information flow. With a small complement of these blocks, a child can build a construction that meets our definition of a robot: a machine that *senses* its environment, *operates* on the information it has taken in, and *acts* to change the state of the environment.

We were inspired by other efforts to build modular construction kits, for example the Activecubes [10], self-describing building blocks [11], and McNerney's tangible programming bricks [12]. The design of roBlocks is subtly but importantly different from these previous projects in that roBlocks embodies a local and distributed computing model. Detailed descriptions of the design, implementation, and testing of the roBlocks construction kit can be found in [13-17]

The roBlocks kit uses a simple diffusion model to pass data from block to block within a construction. Each sensor block transduces information from its environment into an integer from 0-255. A light-sensing block, for example, generates a zero if it is in the dark, and 255 in the presence of bright light. A sensor block continually passes its number value to all its neighboring blocks. The number is wrapped in a packet that also includes its unique block identity (but not a description of its particular type), a hop-count that reflects how far the packet has travelled in the construction, and a timeout integer that decrements periodically to prevent data from persisting in the system after a block is disconnected. When the packet reaches an action block, the action block uses the hop-count to weight the number value, and then it uses the weighted value to generate a signal. For example, if an action block receives a number value of 255 with a weight of 1, it will light its lamp at full brightness, or run its motor as fast as it can.

A roBlocks construction, then, consists of a set of blocks that are constantly passing values around from block to block. The sensor blocks produce values, the action blocks consume them, and the think blocks change values as they pass through. For example, a *minimum* think block takes the smallest of the number values it receives from its neighbors and it passes only that number along. A *sum* think block adds all the number values it receives, and passes that along. As a packet travels from block to block, its hop-count and timeout are decreased. This serves three functions. First, a block that receives two packets from the same sensor origin that have traveled different routes through the construction can determine which value represents the shortest route and discard the other. Second, a block can differentiate older sensor values from new ones. Even though the sensor value from a moment ago (when the room was dark, for example, before the lights were turned on) is still circulating throughout the system, the timeout value allows the block to take only the most recent sensor data into account. Third, each action block uses the hop-count in each packet it receives to weight the number value in that packet. The farther away the originating Sensor block, the weaker its contribution to the action block's behavior.

With only a few blocks we can build interesting simple Braitenberg-like [18] robots. For example, a light sensor block atop a tractor action block (with a battery block to power it) moves toward or away from light. The light sensor produces a number that the tractor block uses for speed control. If there is no light, the sensor produces zero and the tractor remains still. If there is a lot of light, the sensor produces a high value and the tractor motor runs fast. (Whether it goes toward or away from light depends on the relative orientation of the light sensor and tractor action blocks.)

To extend this example, we can build two of these light-sensor, tractor-action towers and connect them. The two towers act as before, each driving (say) toward light. Now, though, if the light is on one side or the other, the connected robot will turn away from the light because the light sensor closer to the light will drive its motor faster than the motor on the darker side. This difference in motor speed causes the robot to turn.

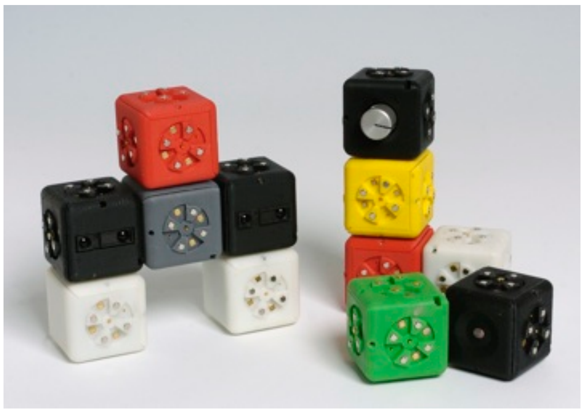

Figure 1. roBlocks is a heterogeneous modular robot construction kit. The white blocks are 'action' blocks, the black blocks are 'sensor' blocks, and the colored blocks are 'think' or

operator blocks. They snap together and transmit power and data through the magnets and spring probes on their faces.

2.2 Playing and Learning with roBlocks

After we built a functional prototype of the roBlocks kit with several different kinds of sensor and action blocks, we wanted to see whether children would find them as interesting and provocative as we had planned. At least we wanted to see whether (and at what age) children would be able to construct working robots. We also wondered what they would understand about how an assembly of roBlocks works, and whether they would grasp the properties of modularity and hierarchy that are necessary to "think in roBlocks".

We conducted a series of informal user studies with fifteen children with ages ranging from 5 to 13 years. Eleven boys and four girls participated in our studies, which took place at the children's homes, at their school during lunch hour, or in one case at their mother's work place. We introduced the blocks and explained briefly about sensor, action, and think blocks and how they worked together; we built a few simple robots (see figure 2 for the simplest robot), and then allowed the children to play freely with the blocks. We did not coach or try to get the children to accomplish certain goals; however, when asked we answered specific questions.

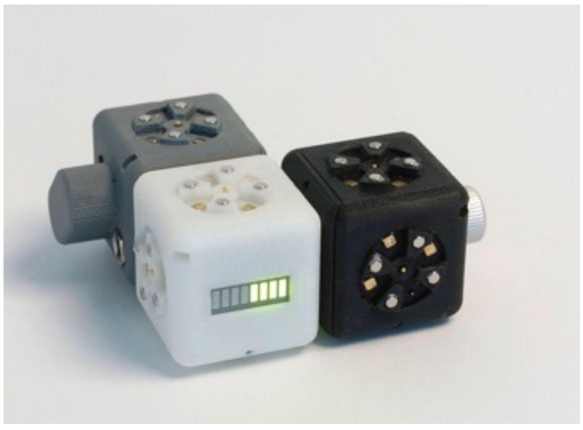

Figure 2. The simplest robot: a twist sensor block (the black block with a knob on the right) and a bar-graph action block. The gray block in back is a battery block.

All the children enjoyed playing with roBlocks and their undirected play sessions with roBlocks lasted from twenty to forty-five minutes. Except for the five year old, who mostly seemed to enjoy snapping the blocks together magnetically and pulling them apart, all the children engaged with the ideas of sensors, actions, and operators.

Children easily grasped the concept of modularity that is inherent in the design of roBlocks. One seven-year-old observed that the bar graph block and the numeric readout block were the same—that is, both Action blocks make visible the data they receive. The same child later pointed out that "really, all the white [action] blocks are the same".

Children also built modular assemblies, or "meta-modules" that exhibit specific local behaviors, and then combined these hierarchically to build larger robots. An eleven-year-old built a small assembly of a distance sensor block, an inverter think block, and a tractor action block. By itself, the assembly is a robot that slows down when it approaches an object. She then used this assembly in various robots, including a robot made of two of these meta-modules that approaches an object and then turns away when it gets close.

Our play sessions with children were not intended to demonstrate that by using the roBlocks kit to build robots, children learn specific mathematics or engineering concepts. Although we believe that future testing will support this hypothesis we merely wanted to see whether the kit would engage children in meaningful play, and whether they could grasp the underlying themes of hierarchy and modularity that are essential elements of the kit.

On one hand our play testing of roBlocks has encouraged us to proceed with extending the kit of blocks, adding different kinds of sensor and action blocks, as well as to produce the kits in larger numbers. On the other hand, even with small numbers (6-10) of blocks in a construction children begin to encounter complex behaviors that may be difficult to manage with larger constructions. These complex behaviors arise because of the emergent and distributed nature of computation in the roBlocks. We see this not as a flaw in the design of the kit, rather as an inherent property of this class of heterogeneous modular programmable matter. This led us to ask: What design strategies might be appropriate for materials made of large numbers of heterogeneous modular robots? In the following sections we consider this question.

3. Design Strategies for roBlocks Constructions

We turn now, away from the details of the roBlocks kit and our experience with young users, to consider roBlocks as a model for material computing. We begin by examining the specific material properties of our kit. We then describe a simulation environment we built in order to experiment with automated design of roBlocks constructions using an evolutionary design program, the Erstwhile Agent. We report on our first experiences using the Erstwhile Agent to synthesize constructions that meet specific desiderata.

3.1 Material Properties of the roBlocks Kit

Most obvious, when we consider roBlocks as a building material, is that at least in their current size the blocks are fairly large. Apart from this, what properties of this material affect the designs that can be made with them, and what design strategies could be effective for large-scale configurations of roBlocks?

First, and most obvious, the blocks are *modular*; we have already alluded to this. Their modularity is both physical and functional. Physically, all blocks are the same size and shape, except for the special face on each sensor and action block—the tractor treads, the photocell, the rangefinder, etc. All blocks also have the same physical interface. They connect face-to-face, orientation does not matter, and every connecting face is the same. There are no 'male' or 'female' faces. Functionally, the blocks are also modular. All sensor blocks behave the same way, regardless of what particular physical signal the block is designed to sense; and from the point of view of the construction, action blocks are all the same as well—they transduce data into physical effect. Only the think blocks are functionally different—and viewed from a higher level of abstraction, they too, are all the same—they all operate on data.

A second property of the roBlocks kit is that it is *heterogeneous.* Although the kit is modular, and therefore all the blocks are the same at one level of abstraction, there are different types of blocks. By comparison, a material made of robots like Goldstein's Claytronics [1] would be homogeneous: the units are all identical.

A third property of the roBlocks kit is that *functional behaviors are tightly coupled, one-to-one, with physical elements* of the kit. The tractor action block does only one thing: it moves. The light sensor block does only one thing: it captures a light level signal from the environment. This discrete matching of shape and behavior makes each unit relatively simple—it only has to do one thing. This also requires the kit to be heterogeneous, because more than one behavior is needed in order to accomplish any interesting task.

A fourth property of roBlocks is that they *propagate information locally* throughout a construction. A simple integer transmitted locally, from block to block through adjacent faces is the only information used. Every sensor block is an information source, and every action block is an information consumer. There is no hierarchic command and control, nor is there message passing. Information is not directed at a target, but simply suffuses across the construction.

Finally, roBlocks are *concurrent*; they operate autonomously and asynchronously; there is no clock or central scheduler. Each block executes its own process, operating on information it receives from its neighbors.

3.2 Simulating Behaviors of roBlocks Constructions

Even with small numbers of roBlocks we found we could build robots that exhibit interesting behaviors. We wanted to see what it would be like to design and build with large numbers of roBlocks. As we do not have large numbers of roBlocks, we decided to build a simulator that we could use to experiment with designing and testing larger scale constructions.

Our simulator is built in Microsoft Robotics Development Studio (MRDS) and it simulates the basic physics of roBlocks as well as the way information flows through a roBlocks construction. For example, simulated gravity makes unsupported blocks fall and unbalanced constructions tip over; simulated magnetism snaps blocks together and keeps them together. Sensor inputs travel from block to block in a graph structure that is determined by the adjacencies of roBlocks, just as in the physical kit.

We chose MRDS for its service-oriented architecture; its framework for managing concurrency is ideal for experiments involving multiple robots. Each roBlock in a simulated construction runs as an independent service, and they communicate asynchronously in the same manner as the real world hardware system. Each roBlock service is also paired with a simulation entity. The physics engine treats each roBlock as a simple cube (with a mass that approximates a roBlock's), but they are rendered graphically using a triangular mesh and color-coded to match the real world blocks. The roBlock entities in a construction attach to their neighbors by breakable joints that simulate the magnetic connections between blocks. The joints also serve as interfaces between the services: As in the real world system, physically connected blocks also communicate computationally.

The various roBlock hardware types are simulated using a variety of techniques. For example, the simulator approximates the behavior of distance sensor blocks by casting rays in a narrow cone and determining collision points. Light sensor blocks use the MRDS stock webcam component and average the pixel values of an acquired image to calculate a brightness level. Tractor blocks use wheels to mimic the mechanics of the physical system, and the Think blocks run the same C code that is embedded in each roBlock's microcontroller.

We verified the simulator by building roBlocks constructions and comparing their performance in the simulator with their physical world counterparts. Once convinced that we could rely on the simulator to accurately predict the behavior of roBlocks in the physical world, we began to investigate strategies for automated design.

3.3 the Erstwhile Agent: an Evolutionary Design System

The Erstwhile Agent (EA) is an evolutionary design system. Like all evolutionary design systems, in an attempt to design a construction that satisfies requirements set by a user it maintains a large set (population) of roBlocks constructions (candidates) and operates on them over several iterations (generations). Each candidate receives a single score (fitness) based on how well it performs on a particular evaluation. The algorithm generates new candidates by combining (mating) and randomly altering (mutating) strong candidates from the previous generation. Sometimes a small number of highly-fit candidates live on (elitism) to the next generation.

Many interesting automatic design systems have been built [19, 20]. Because of the great variety in their purpose and implementation, it is helpful to discuss the Erstwhile Agent through the four part framework of representation, generation, evaluation, and guidance [21].

In contrast to the simulator's 3D physical representation of a construction, the EA *represents* each candidate roBlocks construction as a graph: a set of blocks and a set of connection objects between those blocks. (Connection objects contain information about which faces they connect and at what orientation.) Nothing in the representation prevents the EA from creating and operating on constructions that cannot be physically built (where two blocks would occupy the same space, for instance, or constructions containing 3-block triangular cycles). We chose to allow these impossible representations and later check whether a construction can be built. We did this because the graph structure lends itself to crossover and grafting operations more easily than, say, a three dimensional array that would also represent the construction's physical instantiation as blocks.

Also, instead of using a higher-level (i.e., genotype) language to describe the desired features of a design, the representation of a construction is the phenotype itself. Although this direct encoding may not easily give rise to appealing design outcomes such as symmetry or self-similarity, building these operators into the representation seems an overt attempt to manipulate the final designs.

From an initial population of random candidates, the EA *generates* offspring using mating and mutation. Two parent candidates are mated using crossover: each graph representation is broken along a branch and joined with a piece of the other, creating two offspring. Infrequently, a candidate is mutated: a random block is rotated, replaced by a block of a different type, or removed altogether.

Evaluating the performance of each construction is challenging. Although each construction has some easily observable properties that can be assessed just by parsing the representation (How many blocks does it contain? Are there any power blocks?), the interesting properties of a construction are emergent. They are not easily determined simply by examining the physical construction; we must run the construction to see how it will behave. For example, if we require a safety-conscious construction that beeps when it backs up, the best way to evaluate that criterion is to move it in reverse and find out. With thousands of candidates in an evolutionary run of thousands of generations, however, actually building each candidate construction and testing it in the real world is impractical. Instead, the EA evaluates constructions in simulation using the MRDS system described above. Relatively inexpensive computing power enables us to run many simulations in parallel and in accelerated time.

All our evaluations were specifically written tests that assign a fitness score to each candidate. As we were trying to test the global, emergent behavior of parallel systems like roBlocks, another service is created: a third-person evaluation service that takes an omniscient perspective and evaluates the behavior of the entire construction. In an attempt to design a construction that glows in the dark, for instance, the EA instantiates an evaluation service that adjusts the light level in the environment and monitors the glow level of the construction. This evaluator watches each candidate construction and assigns it a fitness score that rates how well it performs.

A challenge in designing with a concurrent material involves translating from a distal to a proximal description [22]. The distal description ("glows in the dark") emerges based on the proximal description (the arrangement of modules in the construction). Taking a third person perspective, the evaluation service tests each proximal description against the distal description. We discuss some challenges in creating these fitness functions in the discussion section below.

Testing in simulation gives the EA a third-person view of how the construction interacts with its environment. Constructions that respond and react to their surroundings are fully embodied [23]; the embedded sensors and actuators mean that behavior makes sense only in relation to the environmental inputs that the construction receives. Unlike pure software or other more abstract systems, the behavior of computational-physical systems like roBlocks emerges from the ways a construction interacts with its environment.

After the EA evaluates the entire population of a generation, the fitness score of the individual candidates provides *guidance* to the algorithm as it creates the next generation. Nature uses the process of natural selection: candidates sort things out on their own, and the more successful ones (from nature's point of view) generally have more offspring. But unlike nature, the EA is teleological. We want a particular outcome. So, like a dog breeder, we select the offspring that we like best to mate and possibly mutate when creating the next generation. The EA uses stochastic universal sampling [24] as a form of artificial selection where fitter candidates are more likely to mate with others and carry their attributes into the next generation.

3.4 Some Evolutionary Runs

Our first attempts, like other efforts to evolve dynamic physical form [25, 26], aimed at designing foraging robots, that is, constructions that move quickly. The evolutionary runs were successful but unsurprising. As the only motorized action blocks available to the simulation were tractor blocks, the best possible foraging robot is a balanced construction with tractor block(s) on the bottom and a distance sensor aligned to output a consistently high (near) value.

Some unsatisfying results arose because the vast majority of constructions do not move at all; they receive a fitness score of zero. Many trials carried on for hundreds of generations without marked improvement. Additionally, due to an oversight in the evaluation framework several successful evolutionary "jumps" did not persist: when two constructions mate, they do not necessarily both preserve their orientation. The offspring of a successful construction with several tractor blocks, for example, may end up "on its back" with its actuators pointed in the air instead of down, rendering it unable to move. To address this shortcoming the EA could use a more controlled mating process, but that might restrict its creativity. Alternatively, each construction could be placed into the simulation and evaluated at each of the six possible orientations.

In one set of experiments, we challenged the EA to design a roBlocks construction that was somehow mindful of state. Without memory, roBlocks constructions are purely *behavior-based* [27]; they process inputs and outputs in real time regardless of previous experience. We hoped that the EA would manage to create a primitive Turing machine, a construction that (for example) modified its surroundings in order to keep track of state. Our simple evaluation function for this first queried each actuator block for its current data value, then placed a large object in the scene. The aim was to trigger any distance sensors that might be part of the simulated construction. The evaluation function waited a few seconds, removed the object, then queried the actuators again. Constructions with a larger difference between the first and second queries were deemed to have 'noticed' and 'remembered' the object's presence in the scene; the evaluation function gave them a high fitness value.

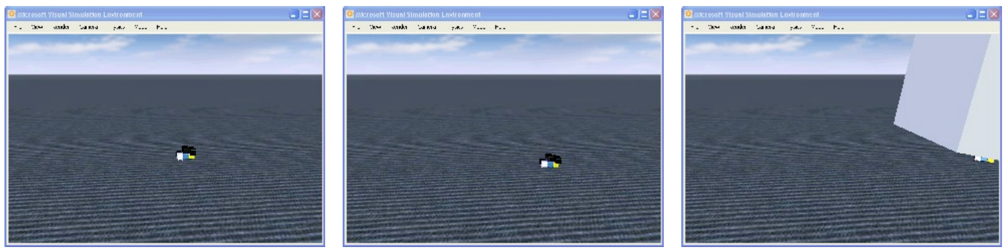

Figure 3. Screen captures from a video of a highly fit "stateful" candidate. It exploits the fitness function by moving quickly to the right so that it is violently thrown when the large box is added to the scene.

In response to this challenge, the EA designed a number of fit constructions. Its designs were based on two different patterns: some constructions were very large, and others immediately drove quickly to the right (see Figure 3). In both cases all or part of the construction was located where the large object was added; the object would collide with the construction and bounce it away. The displaced construction would land at a different orientation; its originally down-pointing distance sensors, which produced a high (near) value that would drive the robot quickly, would now point away from the ground plane, reading a low (far) value. Although these results did not represent the types of solutions we had hoped for, they do demonstrate the power of evolutionary algorithms to create novel designs and exploit environmental conditions to their benefit.

4. Discussion

4.1 The Limits of Simulation

Testing a material in simulation to determine its performance in the real world raises its own set of potential problems. As materials with sensors and actuators are so clearly embodied, their behavior in a simulated "blocks world" may differ drastically from their behavior in, say, a classroom. Say we want a construction that is flexible in the dark and becomes rigid when exposed to light. Given an appropriate kit of parts, we might expect an evolutionary algorithm to design a solution that fulfills our requirement, but quite possibly the solution will exploit some feature of the simulation environment, as our "stateful" construction did. Perhaps its light sensors are mounted on the bottom of the construction making it dependent on a ground plane with high reflectivity. It will fail in a carpeted room. Can we get around these problems by expanding our evaluation to test each candidate in a variety of different simulation environments? How, then, to structure those environments and testing?

4.2 Modularity and Hierarchy

The world is filled with complex systems. Human designers, whether of buildings, businesses, or products, have developed methods to manage complexity that don't rely on explicitly evolutionary methods. When these types of complex systems become unwieldy, designers manage by using modularity and hierarchy. When software designers, for example, create functions and objects, they are building reusable modules with a simple, specific interface to the rest of the system. At a higher level, database or user-interface modules may be separate from backend transaction processing so that they can be modified independently; a clearly specified interface between modules allows a designer to modify one part of a system without keeping the whole system in mind. Governments, another example of a complex system, show hierarchy along with modularity. Its branching tree structure allows a single entity to effectively be "in charge" of a vast number of other entities without managing each of those entities directly. We've seen young children (unprompted) successfully use modularity and hierarchy to manage their roBlocks constructions. Using these techniques with a discrete, highly connected material could lead to interesting outcomes.

In a system like roBlocks, where the form of a construction determines its behavior, constructions based on modularity and hierarchy express those organizing principles in their physical form. In Figure 4, the construction on the left is organized into two meta-modules. Each meta-module functions in relative isolation; their only communication is through the single-block-wide "bridge". The construction on the right of Figure 4 is hierarchical. Its various parts combine in prescribed ways, handling interaction between parts locally and then passing a calculated output down the chain to the next part.

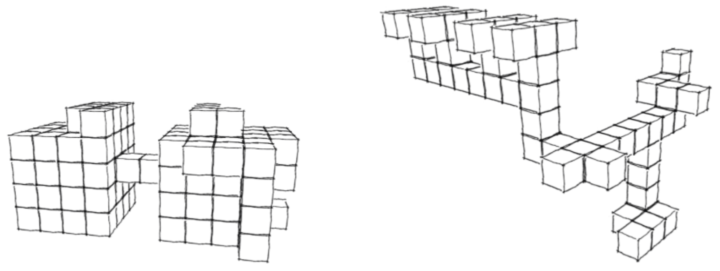

Figure 4. Constructions that express modularity (left) and hierarchy (right) in both their behavioral programming and their physical shape.

The point of these simple examples is that standard techniques for managing complexity now create highly salient features in the form of a construction, which may or may not be a good thing. The modularity construction, on the left of Figure 4 might behave drastically differently if we only made the interface (the bridge) two blocks wide instead of one. More data would be passed and more dependencies would be created. Physically, the construction would be less likely to fall apart. In roBlocks, there are ways to manage these issues (by using "Blocker" blocks that do not transmit data, for instance), but these solutions are not universal to all discrete materials.

A great difficulty with discrete materials is that designers must simultaneously reconcile the shape and behavior of a construction. Traditional systems, by contrast, have "very high level modules" of separate mechanical design and software design, but constructions made of discrete heterogeneous robots do not afford that luxury. Perhaps a designer requires the behavior of the construction in Figure 4 (left) but needs a flat (2-D) construction. Perhaps she requires the hierarchical organization of Figure 4 (right) but needs the construction to have a minimal surface area. Satisfying these requirements may be possible but it is likely not easy.

4.3 Representing Constructions: Genotype vs. Phenotype

An area for improvement in the Erstwhile Agent is in the representation of constructions themselves. The simple, direct representation we used in these experiments is incapable of the tremendous power of a genetic language that can specify (for example) reuse, modularity, symmetry, and self-similarity with a minimal amount of information. We attempted to design a language like this [13] but realized that our language would give rise only to constructions with the attributes we designed into it. We concluded that what is really needed is a genetic encoding that can itself evolve, incorporating new operators on its own. This remains to be done.

4.4 To Design, or to Design Requirements?

Evolutionary algorithms are demonstrably a viable method of design in many domains. We can have them design things —even robots—for us. But we must design the evaluation ourselves, which is generally not trivial. Our attempts to get the Erstwhile Agent to create 'stateful' constructions highlight the first difficulty inherent in automating design: design problems are open-ended. If we can specify requirements so precisely that they can be tested in analytical simulation, then basically we have designed a solution.

We have, of course, seen responses to the same problem in the field of software engineering. Long requirement specifications lead to a series of unit tests and specialized tools; requirements engineering can become the most time and labor intensive components of a development project. Although you can specify the software so completely that it can be automatically generated and verified, at that point you have done as much work as writing the software yourself.

The question is whether by painstakingly writing the specification (evaluation criteria) you have gotten anywhere. The answer is clearly "yes." Even if you do the design work yourself, the evaluation criteria still must be specified. Without it, there is no way to know whether your designs are satisfactory. The challenging work of creating design requirements that we have discussed is not unique to automated design systems: no matter who does the work, there must be an evaluation framework to determine the quality of the designs.

Perhaps this is the future for automated design. Before this is possible, however, much remains to be accomplished in *representing* design requirements. Software requirements start with a long list of "shall" statements, but our computer simulations cannot parse these imprecise descriptions. In order to be useful for automated synthesis design requirements must take the form of tests that can be carried out and scored, and these tests often deal with 3D geometry, real-world scenes, and complex actions. The tests are not easy to write, even for seasoned programmers.

It's also hard to imagine designers being comfortable writing a series of unit tests in order to create the framework to evaluate automatically generated designs made of modular robots. It may be easier to imagine if we can provide an interactive simulation environment. Designers could create virtual spaces and sensor stimuli using a familiar palette of digital modeling tools, and then specify requirements as a series of constraints. Size constraints could be specified with dimensioned outline limit drawings, path constraints with boundary lines drawn on a floor plan. In a scheme like this, however, we are still limited to specifying primitive behavior. Any desired behavior more complicated than simple action-reaction requires programming.

Automated design methods like the evolutionary techniques described here are appropriate only for certain problems: those where the evaluation criteria can be described algorithmically. The evaluation challenges we have outlined are specific to constructions made of robotic materials, no matter who, or what, designs them.

5. Conclusion

We came from our modular robotics construction kit toy for children to the question of how to design things made of components like our kit. The design of things depends, to a large measure, on the properties of the materials they are made of. This will be no less true of things made of modular robots than of wood, metal, paper, and plastic. The structure of materials matters. Although modular robot materials will be programmable, that does not make them indefinitely mutable. The fundamental characteristics of the modules will strongly influence the properties of the material they make up.

We used our roBlocks construction kit as a starting point to think about the design of things made of modular robot materials, specifically robots whose characteristics are like those of roBlocks: modular, heterogeneous, closely coupled behavior and form, and governed by diffuse and non-directed information flow. Other systems of modular robots with quite different characteristics are being developed, and these different characteristics will likely give rise to materials with quite different emergent properties.

Interestingly, the same properties of roBlocks that make them provocative learning toys in children's hands—they are not programmed top-down, each module provides one and only one behavior—make it difficult to think about designing with large ensembles. Still, experiences with constructions that have small numbers of blocks suggest that material made of robots like roBlocks will exhibit interesting emergent behaviors. The art and methods of design developed over centuries for a world of materials with relatively static properties, may well not serve. This led us to experiment with automated design synthesis using the approach of evolutionary algorithms.

Our journey from children's blocks to automated design has highlighted one specific challenge: how to manage the complexity that arises when designing things made of robot building blocks. It seems likely that we will begin to see materials that are programmable, and that there may well be many forms of programmable matter. Work is underway on the physics, material science, and software architectures for material computing: It is time to think about how to design with this new medium.

Acknowledgments

This work was supported in part by the National Science Foundation under Grant ITR-0326054.

References

1. Goldstein, SC, Campbell JD, and Mowry TC, Programmable Matter. IEEE Computer, 2005. 38(6): p. 99-101.
2. Murata, S, et al., M-TRAN: self-reconfigurable modular robotic system. Mechatronics, IEEE/ASME Transactions on, 2002. 7(4): p. 431 - 441.
3. Nagpal, R, Self-Assembling Global Shape, using Ideas from Biology and Origami, in Origami3:3rd International Meeting of Origami Science, Mathematics and Technology (3OSME), T. Hull, Editor. 2002, A.K. Peters. p. 219-231.
4. Rus, D and Vona M, A physical implementation of the self-reconfiguring crystalline robot, in Intl Conf Robotics and Automation (ICRA). 2000. p. 1726-1732.
5. Støy, K, Lyder A, Garcia RFM, and Christensen D, Hierarchical Robots, in Workshop on Self-Reconfiguring Robots at Intelligent Robots and Systems (IROS). 2007, IEEE: San Diego, USA.
6. Yim, M, Duff D, and Roufas K, PolyBot: A Modular Reconfigurable Robot, in Intl. Conf. on Robotics and Automation (ICRA). 2000, IEEE. p. 515-519.
7. Harel, I, Children Designers: Interdisciplinary Constructions for Learning and Knowing Mathematics in a Computer-Rich School. 1991, Norwood, NJ: Ablex Publishing Corporation.
8. Papert, S, Mindstorms: children, computers, and powerful ideas. 1980, New York: Basic Books, Inc.
9. Resnick, M and Silverman B, Some reflections on designing construction kits for kids, in Interaction Design and Children (IDC). 2005, ACM: Boulder, USA. p. 117-122.
10. Watanabe, R, et al., The Soul of ActiveCube - Implementing a Flexible, Multimodal, Three-Dimensional Spatial Tangible Interface, in Proc. of ACM SIGCHI International Conference on Advanced Computer Entertainment Technology ACE 2004. 2004. p. 173-180.

11. Anderson, D, Frankel, J., Marks, J., Agarwala, A., Beardsley, P., Hodgins, J., Leigh, D., Ryall, K., Sullivan, E., Yedidia, J., Tangible Interaction + Graphical Interpretation: A New Approach to 3D Modeling, in SIGGRAPH 2000. 2000, ACM. p. 393-402.
12. McNerney, TS, From turtles to Tangible Programming Bricks: explorations in physical language design. Personal Ubiquitous Computing, 2004(8): p. 326-337.
13. Schweikardt, E, Designing Modular Robots. 2008, PhD Dissertation, Computational Design, Carnegie Mellon University
14. Schweikardt, E and Gross M. The Robot is the program: interacting with roBlocks. in Tangible and Embedded Interaction. 2008. Bonn, Germany: ACM.
15. Schweikardt, E and Gross MD, roBlocks: a robotic construction kit for mathematics and science education, in Intl. Conf. on Multimodal Interfaces (ICMI). 2006, ACM: Banff, Canada. p. 72-75.
16. Schweikardt, E and Gross MD, A Brief Survey of Distributed Computational Toys,, in First IEEE workshop on Digital Game and Intelligent Toy Enhanced Learning (DIGITEL). 2007: Jhongli Taiwan. p. 57-64.
17. Schweikardt, E and Gross MD, Learning about Complexity with Modular Robots, in The 2nd IEEE International Workshop on Digital Game and Intelligent Toy Enhanced Learning. 2008, IEEE: Banff. p. 116-123.
18. Braitenberg, V, Vehicles: Experiments in Synthetic Psychology. 1984, Cambridge, MA: MIT Press.
19. Bentley, P, ed. Evolutionary Design By Computers. 1999, Morgan Kaufmann.
20. Funes, P and Pollack J, Computer Evolution of Buildable Objects for Evolutionary Design by Computers, in Fourth European Conference on Artificial Life, I.H. Phil Husbands, Editor. 1997, MIT Press.
21. Cagan, J, Campbell MI, Finger S, and Tomiyama T, A Framework for Computational Design Synthesis: Model and Applications. Journal of Computing and Information Science in Engineering, 2005. 5(3): p. 171-181.
22. Sharkey, NE and Heemskerk J, The neural mind and the robot, in Neural Network Perspectives on Cognition and Adaptive Robotics, A.J. Browne, Editor. 1997, IOP Press: Bristol, UK. p. 169-194.
23. Dourish, P, Where the Action Is: The Foundations of Embodied Interaction. 2001, Cambridge, MA: MIT Press.
24. Baker, JE. Reducing Bias and Inefficiency in the Selection Algorithm. in Proceedings of the Second International Conference on Genetic Algorithms and their Application. 1987: Hillsdale.
25. Hornby, GS, Lipson H, and Pollack JB, Generated Representations for the Automated Design of Modular Physical Robots. IEEE Transactions on Robotics and Automation, 2003. 19(4): p. 703-719.
26. Sims, K. Evolving Virtual Creatures. in International Conference on Computer Graphics and Interactive Techniques. 1994.
27. Brooks, R, Intelligence Without Representation. Artificial Intelligence, 1991. 47: p. 139-151.