Pers Ubiquit Comput
DOI 10.1007/s00779-010-0315-7

ORIGINAL ARTICLE

Hyperform specification: designing and interacting with self-reconfiguring materials

Michael Philetus Weller · Mark D. Gross · Seth Copen Goldstein

Received: 1 February 2009 / Accepted: 1 September 2009

Abstract We are on the verge of realizing a new class of material that need not be machined or molded in order to make things. Rather, the material forms and re-forms itself according to software programmed into its component elements. These *self-reconfiguring materials* are composed of robotic modules that coordinate with each other locally to produce global behaviors. These robotic materials can be used to realize a new class of artifact: a shape that can change over time, i.e., a four-dimensional shape or a *hyperform*. Hyperforms present several opportunities: objects such as furniture could exhibit dynamic behaviors, could respond to tangible and gestural input, and end-users could customize their form and behavior. To realize these opportunities, the tangible interaction community must begin to consider how we will create and interact with hyperforms. The behaviors that hyperforms can perform will be constrained by the capabilities of the self-reconfiguring materials they are made of. By considering how we will interact with hyperforms, we can inform the design of these systems. In this paper, we discuss the life cycle of a hyperform and the roles designers and end-users play in interacting with hyperforms at these various stages. We consider the interactions such a system could afford as well as how underlying hardware and software affect this interaction. And we consider the extent to which several current hardware systems, including our own prismatic cubes (Weller et al. in Intelligent Robots and Systems. IEEE, 2009), can support the hyperform interactions we envision.

Keywords Modular robotics · Tangible interaction · Hyperforms · Programmable matter

M. P. Weller (✉) · M. D. Gross
Computational Design Lab, Carnegie Mellon University, Pittsburgh, PA 15213, USA
e-mail: philetus@cmu.edu

M. D. Gross
e-mail: mdgross@cmu.edu

S. C. Goldstein
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA
e-mail: seth@cs.cmu.edu

1 Introduction

1.1 A new kind of material

Making things—giving objects form—has traditionally involved a manufacturing process such as machining a block of material or pouring molten material into a mold. However, recent advances in robotics herald a new kind of material that can receive a digital description of a desired form and arrange itself into that shape. One implementation of such a *self-reconfiguring material* is an ensemble of robotic modules. Each module runs a small program; together the programs encode the ensemble's behavior. By coordinating with neighbors, modules respond to external stimuli and arrange themselves into a potentially vast number of forms.

Self-reconfiguring materials promise to revolutionize the creation and distribution of physical objects much as digital audio files have revolutionized the distribution and content of music. The digital description of a chair or a bottle opener could be downloaded and then realized from a reservoir of self-reconfiguring material. Unused objects return to this reservoir to provide raw material for other objects.

1.2 Form in four dimensions

More significantly, an object composed of this new material need not be limited to a single static form. Instead, the currently running program can change its form. We call a form that varies in the four dimensions of space and time a *hyperform*; a single hyperform expresses itself as different shapes at different times. A new hyperform is realized just by loading a new program into the material.

For example, the 'social table' hyperform automatically expands as more people sit down (Fig. 1). The table grows to make space for additional guests (drawing more material from a household reservoir as needed); as people get up from the table after dinner, the social table melts away leaving only a small dinette with a single empty chair. The material that had been part of the expanded table returns to the reservoir and later renders a sofa and coffee table for guests to relax after dinner.

We call the description of a hyperform a *hyperform specification*; it defines a set of states with individual forms and behaviors, instructions for transitioning between these states, and constraints to maintain during transitions.

1.3 A challenge for tangible interaction

Clearly, hyperforms will require changes in the way we create and interact with things. Fortunately, hyperforms can also provide a powerful new class of design tools: we can build full-scale functional prototypes from the outset; and we can use a hyperform to customize its own specification—its form and behavior could be directly manipulated [16] through tangible interaction. This combination of self-reconfiguring materials and tangible modeling techniques will make everyone a potential designer.

In this paper, we envision a future hyperform ecosystem in which the current flow of objects from design to manufacture to consumption has been disrupted by this new technology. Our aim is to engage the tangible interaction community in exploring this space.

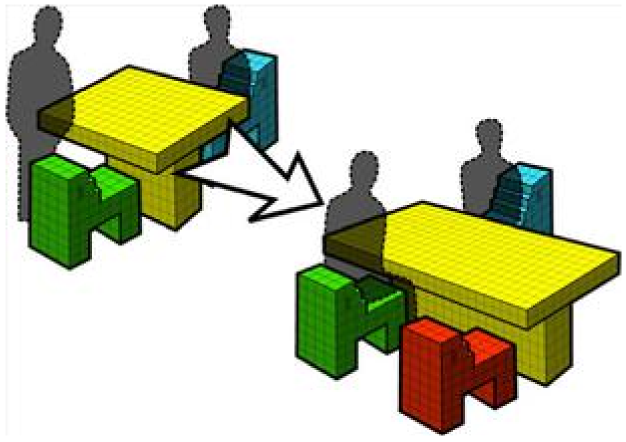


Fig. 1 A 'social table' hyperform that automatically grows as more people sit at the table so one empty chair is always available. Additional material to grow the form is drawn from a household reservoir

The behaviors that hyperforms can realize will be constrained by the capabilities of the self-reconfiguring materials they are made of. Some first and critical decisions about the implementation of modular robotics are already being made. By addressing the question of how we will interact with self-reconfiguring materials to create hyperforms, and how we will interact with hyperforms to control their behavior, the tangible interaction community can guide the development of this emerging technology.

Hyperforms are a radical vision of pervasive and ubiquitous computation: not only is computation embedded in everyday objects, everyday objects are *made of* computational materials; and the primary mode of interacting with this embedded computation is through tangible interfaces and voice and gesture commands. To realize this vision, researchers will need to understand the constraints and affordances of self-reconfiguring materials.

We begin (in Sect. 2) with a scenario illustrating the potential for self-reconfiguring materials to disrupt current practice in creating and distributing objects. Through developing the 'social table' hyperform scenario further we provide a (wildly speculative) vision of the different roles people could play in the hyperform life cycle.

To ground further discussion, in the following sections, we address the state of the art in self-reconfiguring materials and tangible interaction. In Sect. 3, we give a brief description of the hardware and software components that comprise a modular robotics ensemble and discuss the way the limitations of these systems constrain the behavior of hyperforms. In Sect. 4, we outline a framework describing the affordances of ensembles of robotic modules relevant to supporting tangible interaction.

Section 5 introduces a specific hardware and software prototype that we have been working on, the *prismatic cubes*. We discuss features we could add to our this system to support hyperform interaction. In Sect. 6 we present a second, (slightly) less ambitious hyperform scenario that serves two purposes: to illustrate the systems and modes of interaction we have introduced in the previous sections in context; and to identify some of the challenges in realizing these behaviors. In the final section, we revisit the dimensions of this new interaction design space, and suggest promising areas for further research.

2 The hyperform life cycle: a scenario

The following (speculative) scenario illustrates the potential for hyperforms to revolutionize the way we think about creating, acquiring and using objects. While hyperforms

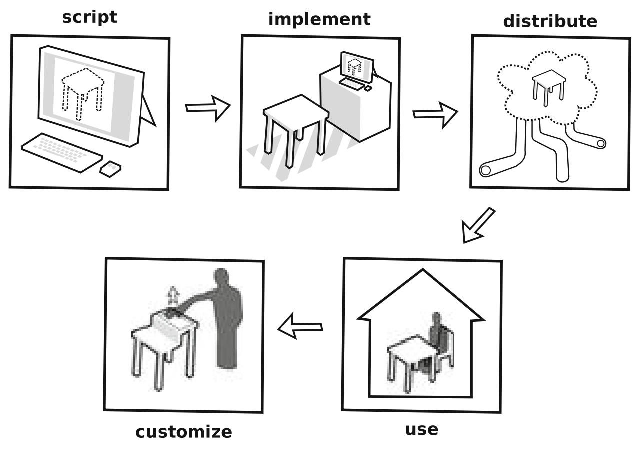


Fig. 2 The stages of the hyperform life-cycle (clockwise from *upper left*): *scripting* defines reusable parametric forms and behaviors; *implementation* specifies a complete hyperform at a hyperform workstation; specifications are *distributed* by posting them online; to *use* a hyperform the specification is downloaded and realized from a local reservoir of material; and end-users can *customize* a form and post the new specification for others to download

will allow anyone to participate in the design process, due to the technical complexity of self-reconfiguring materials some aspects of hyperform specification will require specialized knowledge. We can manage this complexity by dividing the life cycle of a hyperform into stages defined by the different roles people play in interacting with it (Fig. 2). Designers will need to consider the novel complexities of forms that change through time. They will be supported in this task by the ability to build and manipulate full working prototypes, and by screen-based tools that expose the constraints and algorithms that control system behavior. They will be expected to define not only the form and behavior of an object, but also interfaces to support customization by end-users. For end-users, objects will no longer be products in a box, but digital files to download, customize, and share with others.

2.1 The composition of a hyperform

Our scenario describes the creation, distribution and customization of the 'social table' hyperform mentioned above (see Fig. 1). The social table has three distinct components: the table form that grows and shrinks as people sit at and leave the table; chair forms that track the number of places at the table; and a serving platform that materializes on demand to convey serving platters and dishes. We call each such component of a hyperform with a recognizable form and behavior a *phrase* after the minimal recognizable component of music composition. The metaphor with music is intended to highlight that these components are not necessarily static but can vary through time. Also, components may be repeated several times within a larger hyperform composition, often with a particular rhythm, e.g., a table hyperform might feature pairs of leg phrases at regular intervals, with more legs added as the table expands.

2.2 Behaviors are defined as hybrid automata

To make the design of forms that vary through time more manageable, we define their behavior in terms of static (or relatively stable dynamic) states, as well as transitions between these states, and events that trigger transitions, following the formalism of hybrid automata [8]. We define the social table's state space in terms of the number of places at the table. When the table senses that someone has joined the meal, the table determines the desired static configuration of the new state with one more place, and then transitions to the new state. To manage complexity, we decompose designs into phrases (as mentioned above, a recognizable combination of form and behavior). The serving platform in the center of the table would be defined as a phrase anchored to the table phrase's top surface. It has its own states and transitions which are independent of the main table phrase. We might define the state of the serving platform in terms of its velocity: stationary or moving along the table. This simple state transition hybrid automata framework can cover a wide range of hyperforms.

2.3 Stages of hyperform design: scripting, implementation, and use

Figure 2 illustrates the several stages of design and implementation of hyperforms. *Hyperform scripters* will fashion reusable routines to define behaviors such as a table adding legs as it gets longer. Later, *hyperform implementers* will draw on these routines to incorporate behaviors and phrases into individual hyperform specifications. Finally, *end-users* interact with these hyperforms to control transitions between states. They may also customize the behavior of a hyperform through interfaces established by scripters and implementers. We do not intend to suggest that these stages must be implemented by different people; rather we envision these stages as different roles a person can play in interacting with a system. For example, the same person who adopts the role of an implementer to design a new table hyperform will adopt the role of an end-user when she sits at the table to have a meal.

2.3.1 Scripting generic behaviors

In the first—meta-design—stage, hyperform scripters create and package libraries of reusable behaviors and phrases. For example, to protect end-users from issues such as structural failure during transitions, a scripter might specify generic table phrases with geometric and functional constraints. The length and width of the table surface are defined parametrically, and the rest of the shape, through functional constraints: (1) tile the top surface of the table; (2) maintain structural stability; and (3) leave clear a ‘leg envelope’ under the table. These abstract definitions form the basis for more complete specifications that the implementers craft, in the next stage, to protect the integrity of the table while allowing end-users to customize aspects of a specification.

2.3.2 *Implementing specific instances*

Building on the work of the scripters, hyperform implementers create new specifications at a *hyperform workbench*—a personal computer running screen-based tools with a work surface next to a reservoir of self-reconfiguring material. Depending on the scale of objects, the work surface may be an actual workbench, the floor, or even an outside work yard.

The hyperform workbench allows the implementer to define and edit states either by defining parameters and constraints onscreen (automatically generating the specified shape on the work surface) or by directly manipulating material set in a special ‘free modeling’ mode. For example, the initial one-person table is defined by importing an abstract table phrase and then manipulating it in free modeling mode. The implementer can create larger versions of the table that accommodate more people by adjusting the length parameter for the desired number of places.

To establish transitions, the implementer uses an onscreen state flow interface to connect states in various sequences. State transitions—triggered by timeouts, sensor events, or gestures—are managed by one of several low-level reconfiguration planners built into the underlying software; the particular planner chosen depends on the desired behavioral characteristics. The implementer uses a timeline interface to inspect the transition behavior generated by the planner, and to place additional constraints on this transition or vary parameters over time in order to tune its behavior.

Once the chair and table forms are modeled, the implementer crafts a sensor profile to determine whether a chair is occupied and then tests it by actually sitting in the chair. In the state flow diagram, changes in chair occupancy increment or decrement the table’s length parameter, triggering reconfiguration. Next, to define the serving platform, the implementer puts the table in free modeling mode and selects a rectangular area in the middle of the table, pulling it up (as discussed in Sect. 4) to create a raised platform. Then, the implementer defines a pushing gesture to trigger a transition into sliding mode that continues until a user makes a stop gesture, or the platform reaches the end of the table.

2.3.3 *Use*

Once a hyperform has been implemented it is ready for use. The end-user downloads the social table specification and then activates it to assemble an empty one-place configuration using material from the household reservoir. In this stage of the table’s life cycle it is responsive to both changes in its environment as well as explicit tangible and gestural commands.

As people sit down for a meal the table automatically expands. When the table senses a serving platter with food, a platform rises up underneath it. As people serve themselves they slide the platform over to the next person by making a pushing gesture. After everyone has eaten, as people rise from the table they place their dishes on the serving platform to return them to one end of the table while the other contracts.

During use, explicit gestures made at the table control its behavior. For example directing a pushing gesture toward one side of the serving platform causes it to glide down the table. For finer-grained control, we use the ‘sticky hands’ interaction technique[1]; when you hold your hand near, but not touching, one side of the platform for a few seconds it begins to follow your hand, maintaining a constant distance, so you can position the platform more precisely. To serve yourself, you hold your hand near the inside face of a platform carrying a serving platter until the platform “sticks,” then pull it closer to make the food easier to reach.

2.3.4 *Customization*

End-users cannot only interact with a hyperform in predetermined ways; they can also customize its behavior. By putting a hyperform phrase into ‘editing mode’ with a voice or gesture command, an end-user can customize shapes and behaviors within parameters established during the previous implementation phase. For example the social table allows end-users to customize the shape of its legs. After putting a leg into editing mode a foot is first molded at the bottom, then propagated to all the other legs. Or an end-user can add a new phrase to the social table, for example, downloading a ‘basin’ phrase from a library and associating it with a ‘double fist’ gesture. When someone

[1] Although in the future a hyperform may be able to sense someone pushing on its surface and actuate to create the impression that the shape is being physically manipulated in real time, implementing such a behavior demands sophisticated sensing and actuation. The ‘sticky hands’ technique is a useful approximation that can be implemented with simple controllers and inexpensive sensors.

spills a drink, quickly making a double fist gesture over the spill now generates a depression to contain the mess. A serving platform raises around the basin and slides to the end of the table. Finally, we can share this customized version of the social table with others by posting its digital description.

2.4 Rethinking design

As this scenario illustrates, self-reconfiguring materials have the potential to disrupt current systems for creating and distributing objects. Rather than acquiring manufactured goods delivered in a box, specifications for objects are downloaded and realized from a reservoir of reusable material. Instead of being finalized in the first stage of creation, designs are refined in stages all the way down to end-users who can post customized forms online to share with others. We urge other researchers to consider how this technology could reshape the way we think about the design process at the same broad scope; and to envision how new processes could give people greater control over their built environment.

3 Constraints of modular robotics

The behavior of a hyperform is constrained by the properties of the self-reconfiguring material of which it is made. Therefore a basic understanding of the properties of these materials can help members of the ubiquitous computing and tangible interaction communities both to envision how these technologies can be applied and to propose how these materials could better support envisioned behaviors. Below we give a brief introduction to one promising technology for realizing this kind of material, modular robotics. We briefly describe the hardware and software components typical of such a system and discuss the potential limitations of the current state of the art. We focus here not on realizing new self-reconfiguring materials, but on envisioning how we could create and interact with them.

3.1 Modular robotics as a self-reconfiguring material

Over the past 15 years a significant amount of work on self-reconfiguring modular robots has resulted in a number of working prototype systems [26]. Some systems support *articulated structures* with functional limbs, while others support *lattice structures* that can be arranged in a variety of shapes. Polybot [25], an instance of the former, is a kit composed of actuated elbow modules with connectors at each end and cubic hub modules with a connector on each face. An ensemble of Polybot modules can reconfigure itself between various articulated structures such as a snake, a loop, and a quadruped to locomote with different gaits.

Although self-reconfiguring articulated structures have many potential applications, lattice systems (Fig. 3) are a better choice for self-reconfiguring materials as they can realize (nearly) arbitrary 3D forms. Two varieties of lattice module have been particularly successful. *Rotating lattice modules* such as M-TRAN [10], Molecube [29] and ATRON [5] feature powerful actuators that can rotate several modules around an axis at once as shown in Fig. 3a. *Prismatic lattice modules* such as the Telecube [18] and our own prismatic cubes [24] (described later, in Sect. 5) extend and retract their faces to latch to neighbors. When connected they form a cubic lattice, and by coordinating with neighbors a module slides over one space within the lattice (Fig. 3b).

3.2 The composition of an ensemble of modular robots

The software architecture for controlling an ensemble of modular robots typically has several interface layers (Fig. 4). At the top level is a specification of the ensemble's hyperform. A planner takes this specification as input and generates a distributed program to run on individual hardware modules. The control abstraction layer hides the messy details of controlling physical motion. It takes movements specified by the planner and converts them into low-level instructions to actuate the hardware modules. Together these layers comprise a complete self-reconfiguring material that can realize hyperforms.

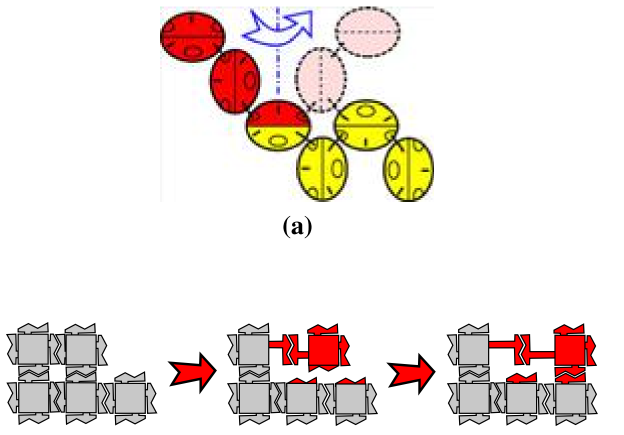


Fig. 3 Lattice modules can implement self-reconfiguring materials. By coordinating with nearby modules to move between the cells of a lattice, these systems can realize almost any configuration. **a** Rotating lattice modules such as ATRON [5] move groups of neighboring modules to a new lattice position by rotating them around an axis. **b** By extending adjacent faces modules in prismatic lattice systems such as the prismatic cubes [24] can slide across one lattice cell

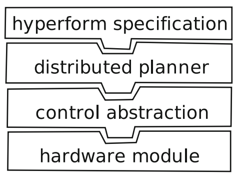


Fig. 4 Layered architecture of an ensemble of modular robots, from high-level description of behavior down to underlying hardware

3.3 Constraints of hardware self-reconfiguration

Two significant constraints affect the behavior of this kind of ensemble. The first constraint has to do with the control abstractions designed to manage the actuation and latching schemes by which modules self-reconfigure. Most algorithms in the control abstraction layer do not allow modules to move individually, but rather move groups of modules at once. For example, the 'sliding cube' control algorithm [2] (Fig. 5a) treats groups of modules as a single cube metamodule that can move one lattice space sideways or diagonally. To enable rendering higher granularity structures with ATRON hardware modules, Christensen et al. [3] developed the idea of an emergent metamodule in which the structure is not subdivided into larger metamodules a priori, but rather individual modules opportunistically recruit neighbors to create a temporary metamodule when they need to move, and then return to the general pool when they reach their destination. Similarly, the prismatic cubes' movement primitives [24] allow an ensemble to be reconfigured by applying production rules as shown in Fig. 5b and c. Applying one of these motion primitives triggers the targeted modules to temporarily group together to perform the requested transition.

These control abstractions restrict the way individual modules can move through a lattice and thus constrain the shapes that an ensemble can realize. For example, sliding cubes and motion primitives can transfer modules through the center of an occupied lattice, whereas ATRON's emergent metamodules require an adjacent empty region of the lattice to rotate through. These low-level constraints need not be explicitly addressed by a hyperform implementer, but are handled by the planning layer. An example of one way these constraints would manifest is that a system would refuse to realize small-scale features that its planner could not reach.

The second significant constraint on ensemble behavior is imposed by the distributed algorithm employed to control the behavior of the modules, as we discuss below.

3.4 Constraints of distributed planning algorithms

The demands of controlling an ensemble of independent robotic agents constrains the kinds of behaviors that an ensemble can realize in several ways. As with computer-controlled fabrication, designers working with self-

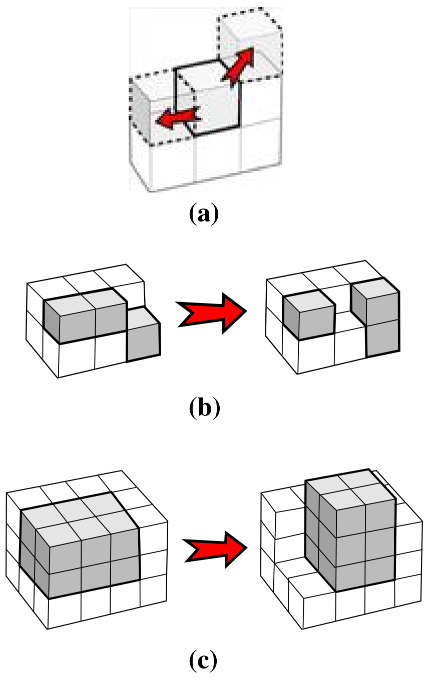


Fig. 5 The sliding cubes metamodule (**a**) is a simple control abstraction but often requires many hardware modules to implement a single movement; motion primitives (**b**, **c**) provide finer-grained control by representing transitions as production rules [24]. **a** A sliding cube metamodule can move one space either straight or diagonally in any direction. Each metamodule is composed of several hardware modules. **b** The slide-up motion primitive moves a hardware module straight one space (corresponding hardware movements shown in Fig. 3b). **c** The round motion primitive rotates a $2 \times 2 \times 3$ group of hardware modules

reconfiguring materials produce instructions for a machine to follow rather than a drawing to guide a machinist in a shop. However, with tools such as a computer-controlled mill the underlying manufacturing process has not changed; the mill executes more or less the same operations as a machinist would, it just requires more detailed instructions. In contrast, this sort of detailed centralized planning scales poorly to large numbers of robotic modules. Instead, a distributed plan must guide each individual module to respond to its local circumstances to produce a cohesive behavior for the entire ensemble.

Different planning algorithms produce hyperform transitions with different characteristics, so specifying hyperforms will require a basic familiarity with the properties of different styles of planning algorithms. Below we describe several relevant properties of hyperform transitions that depend on the choice of planning algorithm.

3.4.1 Structural stability

One obvious concern is that during a reconfiguration a structure could become unstable and collapse. This is a non-trivial problem and there are several strategies for addressing it. One is to have the ensemble run a distributed structural simulation to test the safety of each movement before it is executed. Simulating a structure in real time in this manner would allow customization but could consume system resources and slow reconfiguration. Other strategies are to only allow transitions that have been simulated beforehand to assess their safety (which would limit customization), or to develop techniques for guaranteeing that a particular control algorithm can never reach an unstable configuration.

One example of distributed planning that produces stable structures is the stigmergic algorithms that guide wasp swarms' nest-building [19]. Each individual wasp follows the same global set of rules—each rule describes a particular arrangement of cells adjacent to the empty cell currently occupied by the wasp, and a type of cell to be added in this circumstance. When a wasp sees that the local structure of the nest matches one of the rules' preconditions it adds a new cell specified by that rule's postcondition. As wasps build up the nest, new conditions are created that trigger other rules, coordinating the parallel activity of many wasps without explicit communication. Certain classes of rulesets (such as the rulesets for actual wasps' nests) reliably produce structurally stable forms.

3.4.2 Reconfiguration speed

A critical aspect of a state transition is the amount of time it takes to complete. The social table must be able to create enough places for everyone to sit down before the food gets cold. A disadvantage of wasps' local rule-based algorithms is that the wasps move randomly over the structure, increasing the amount of time it takes to form. One strategy for speeding up reconfiguration is to distribute information about which areas of a structure are growing. In Stoy and Nagpal's gradient-based directed growth planner [17] an ensemble of modules reconfigures into a goal shape by having individual modules move around the surface until they find a growing region and attach themselves there. Unattached modules follow gradients broadcast by attached modules adjacent to growing regions. By taking advantage of local communication to propagate gradients, this technique reduces the time required to reconfigure an ensemble.

3.4.3 Surface turbulence

Some algorithms route modules along the surface of a structure during transitions, while others route modules internally until they reach areas that are growing or shrinking. When transitions take place near people or objects it can be desirable to limit surface turbulence. For example, when the social table expands it is important that modules do not collide with people's legs under the table, and it would be inconvenient for modules to crawl across the surface of the table while people are eating.

DeRosa et al.'s Hole Motion Planner [13] reconfigures an ensemble from an initial shape to a goal shape by growing bubbles on expanding surfaces (Fig. 6) and then propagating these voids through the center of the ensemble until they reach a shrinking surface, where each void 'pops' leaving a small crater. Through the action of these voids modules propagate through the center of the ensemble towards expanding regions. This mechanism limits surface turbulence to areas that are growing or shrinking.

3.4.4 Precompilation

Rather than distribute the desired goal state to each module some algorithms compile local instructions based on the goal state and then only distribute these instructions. This style of algorithm has certain advantages such as minimizing required communication, but it is poorly suited to guiding transitions during customization as the goal state is not known beforehand. In contrast, the Hole Motion Planner requires no precompilation, making it suitable for realizing customizations with the 'sticky hands' technique.

3.5 Summary

Hyperform implementers may not need to know the details of individual hardware systems and reconfiguration algorithms, but they must understand the tradeoffs between different desirable characteristics of the underlying hardware and algorithms. Familiarity with the properties of self-reconfiguring materials can help tangible interaction researchers in developing models and techniques for

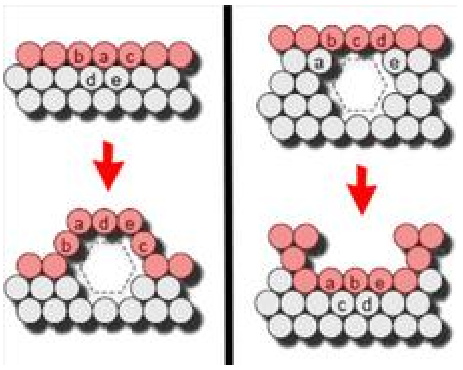


Fig. 6 DeRosa et al.'s Hole Motion Planner creates bubbles on expanding surfaces (*left*) and propagates the resulting void through the center of the ensemble until it reaches a shrinking surface where it pops, leaving a crater (*right*)

interacting with hyperforms, and inform a discussion of how to improve the design of these materials.

4 Affordances of ensembles of robotic modules

In the previous section, we discussed some ways that the underlying hardware and software of a self-reconfiguring material constrains the behavior of hyperforms. There we focused specifically on a particular technology, self-reconfiguring modular robotics. We step back now to examine the different kinds of interaction that self-reconfiguring materials potentially afford. The scope of relevant work is broader, encompassing modular robotics as well as other tangible interaction systems that are composed of groups of modules.

Here, we outline a comprehensive framework to account for the varieties of interaction affordances needed to achieve scenarios like the one in Sect. 2. Our purpose is not to provide a summary of related work, but to illustrate our framework with a few well-known examples. By describing the breadth of affordances available to hyperforms, we aim to foster discussion of which would be most useful in the modules of self-reconfiguring materials.

Our framework identifies three complementary pairs of input/output interactions. The first two pairs concern interactions with the external geometry of a hyperform; the third concerns interaction with a hyperform's internal state. The interaction affordances are:

placing adding, moving or removing modules to alter a form's shape; and
self-reconfiguring actuating a form to shuffle modules around to alter its shape; and
posing bending, squeezing or stretching a form to alter its shape; and
self-posing actuating a form to bend and stretch to alter its shape; and
commanding issuing symbolic instructions to a form to alter its behavior; and
signalling displaying symbolic information with modules to indicate internal state.

Figure 7 illustrates the three input modes of our interaction categories: placing, posing, and command.

No single system that we know of features all six categories of affordances in our framework. Many tangible interfaces today are limited to either input or output only. In the future useful implementations of self-reconfiguring materials may feature only a subset of these affordances. However, there is an advantage to tangible modeling interfaces providing complementary input and output modes; supporting both modes of a pair promotes bidirectional communication between the system and a designer.

4.1 Placing and self-reconfiguring

Placing and self-reconfiguring both facilitate transforming a structure into a new shape. A simple example of placing as input is stacking ordinary wooden blocks or plastic bricks to build a 3D form. The canonical example of self-reconfiguring as output is a set of blocks that can reconfigure itself into a previously demonstrated 3D form. These two interaction modes are fundamental to specifying and realizing hyperforms.

In systems supporting *bidirectional reconfiguration*, modules can be arranged either through manual placement or self-reconfiguration. Most systems that support self-reconfiguration can already sense the ensemble's current shape. All that is required for a self-reconfiguring module to also support input by placing is an interface to manually trigger connecting to and disconnecting from its neighbors. However, we are unaware of any system that fully supports both input by placing and output by self-reconfiguration.

4.1.1 Input by placing

For a hyperform to support input by placing, a user must be able to remove modules from an ensemble and place them somewhere else; and the modules must be instrumented to detect their relative position and determine the ensemble's new shape. For example, in Fig. 7a modules placed on the surface of the social table hyperform define a serving platform. It is relatively straightforward to build modules that afford input by placing: Each module must be able to mechanically latch to its neighbors and communicate through a data link to identify itself.

One tangible interface that supports input by placing is MERL's self-describing building blocks [1]. They are Lego-like blocks, instrumented to communicate the geometry of a model to applications running on a personal computer. Each block has a male connector on top and a female connector on the bottom, just like a Lego brick except that the pins and sockets also form electrical connections. One example application generates geometry for a first-person-shooter game; reconfiguring the blocks changes the layout of the game.

Other tangible interaction projects also support input by placing. They employ various hardware technologies. ActiveCubes [20] and roBlocks [15] have magnetic latches with electrical connectors on all six faces of cubes; Glume [11] models shapes with sticky blobs that communicate with neighbors using a capacitive connection; Senspectra's [7] hubs and struts connect with headphone jacks; and Posey's [21] hubs and struts feature optocoupled ball-and-socket connectors [23].

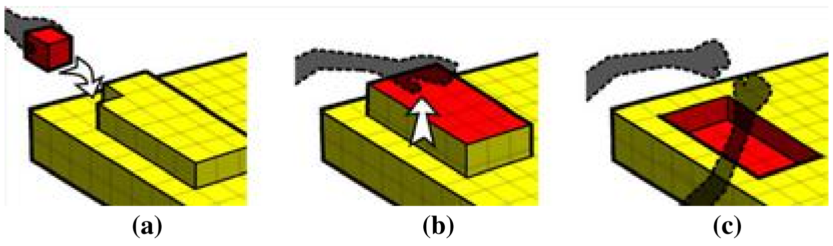


Fig. 7 Three types of input that modules of a self-reconfiguring material afford. **a** Forming a raised platform by placing modules. **b** Posing a platform with the sticky hands technique. **c** Creating a basin with a double fist gesture command

4.1.2 Output by self-reconfiguration

The complement of input by placing is output by self-reconfiguration. Self-reconfiguration, the most technically challenging aspect of realizing hyperforms, requires modules to coordinate with their neighbors to arrange an ensemble into different shapes. It has largely been the province of modular robotics systems; we are unaware of tangible interaction work that has attempted self-reconfiguration. As discussed in Sect. 3, modular robotics systems such as ATRON [5] and our prismatic cubes [24] coordinate to move modules between nearby cells of a lattice. Each system operates under particular constraints due to its reconfiguration mechanism and corresponding control abstractions. For example, only a few prismatic cubes are needed to coordinate to move laterally, but six modules must coordinate to move around a convex corner [24]. Although it is difficult to implement, this is the defining affordance of a self-reconfiguring material.

4.2 Posing and self-posing

Whereas the input and output modes of placing and self-configuring facilitate constructing new shapes, posing and self-posing facilitate adjusting an existing shape. A tangible interface that affords posing can sense when a form is bent, stretched, twisted or otherwise deformed. Modules with actuated degrees of freedom can self-pose. Posing and self-posing allow minor adjustments to the geometry of form and can express motion.

To support posing, the individual modules must either have sufficient degrees of freedom to move fluidly, or they must coordinate with nearby modules to deform the ensemble's geometry. MERL's Lego-style bricks mentioned above cannot be posed at all; they can only be placed. Below we describe several articulated systems with hinges or sockets that afford posing or self-posing or both.

4.2.1 Input by posing

Input by posing does not entirely change the shape of an ensemble but merely alters its posture. Modules that sense how the geometry of a structure is bent, squeezed or twisted support input by posing; each poseable degree of freedom must be sensed. Many tangible interaction projects support some form of posing. For example, the Monkey [4] is a poseable humanoid doll designed to help professional special-effects artists animate computer graphics characters. Each of its one-dimensional rotational joints is instrumented with a potentiometer to detect the current joint rotation.

Our own Posey's [21] hubs and struts connect with ball-and-socket joints. Arrays of optocoupled infrared emitters and sensors embedded in each ball-and-socket detect the identity of neighboring modules as well as the roll, pitch and yaw of each connection. This instrumentation supports input both by placing and posing. For example, with Posey's Puppet Show application you first build the skeleton of a creature by placing the hubs and struts of the Posey kit, and it appears onscreen. You then skin this skeleton with an onscreen interface to create a puppet; and you animate the puppet on the screen by posing the model. This general interaction style of indicating basic three-dimensional forms and behaviors with a tangible interface, then specifying further details through an onscreen interface, is a promising model for hyperform specification tools.

4.2.2 Output by self-posing

In self-posing an ensemble moves under its own power; a quadruped walks across the floor; a chair stretches to raise a child to the height of a table. An ensemble retains its current form and actuates some portion of it. Providing this output affordance is less challenging than providing full self-reconfiguration; nonetheless it can support a wide variety of useful devices.

4.2.3 Bidirectional posing

As with placing and self-reconfiguring, a significant payoff comes when a single device combines both the posing and self-posing input and output modes. Modules that thus afford *bidirectional posing* can sense being bent, stretched or squeezed and then perform the demonstrated deformation. For example the well-known Topobo toy's [12] 'active' modules have poseable sockets with one rotational degree of freedom. Each socket has both a sensor to detect current rotation and a motor to actuate to new rotations.

Pressing a button on an active module puts it into 'record mode': wiggling the creature's legs and torso programs a gait. Then, pressing the button again switches Topobo to 'play' mode and Topobo replays the motion that was just demonstrated.

Another instance of a system that supports bidirectional posing is CKbot [27], an updated version of Polybot composed of actuated elbow joint modules with three connectors on each side. CKbot is intended to support the construction of useful robots and as with Topobo, robots built with CKbot can be posed to demonstrate gaits. Going beyond Topobo, CKbot captures the full geometry of a robot and communicates its configuration and pose to an onscreen gait programming environment. As a robot is built with CKbot modules the onscreen interface displays the robot's current configuration. Keyframes are defined by posing the robot, and then can be shuffled and edited. The resulting gait definition runs on the physical robot and can be further refined. It illustrates the dual utility of self-posing: first as a display during the prototyping stage; and then as a means to accomplish tasks during the use stage.

4.2.4 *Lattice posing*

Both examples (Topobo and CKbot) are articulated structures, but lattice structures can also support posing and self-posing. In the context of a lattice structure, posing could allow a form to be stretched and molded. With the 'sticky hands' technique discussed above, surfaces of a form can be grabbed and then stretched or squeezed.[2] For example Fig. 7b shows the top surface of a platform being stretched upwards.

In a lattice structure, posing can also demonstrate movement. For example, the speed of the serving platform could be adjusted with a double-sticky-hands gesture by placing one hand on each side of the platform and sliding it forward to demonstrate the desired speed.

4.3 Commanding and signalling

The previous two input/output pairs—placing and self-reconfiguring, posing and self-posing—are means to interact with a hyperform's external geometry. To control the behavior of a hyperform through time, it can also be helpful to interact with its internal state. The third input/output pair concerns this interaction. For example, Topobo's command interface is composed of a single button that cycles through three internal states—off (for building), record, and play—and to signal the current state Topobo lights a multicolor LED.

[2] Although the sticky hands gesture could be seen as signalling a command, it is useful to distinguish posing gestures used to directly manipulate the geometry of a structure from more abstract signalling gestures used to trigger arbitrary behaviors.

4.3.1 *Command interfaces*

Commands that can be delivered to the modules of a distributed system are of two types: global commands broadcast to all modules at once, or local commands sent directly to one or a few neighboring modules. In Topobo, on each 'active' module a local command button cycles the module through its three states. However, we often want to think of an entire Topobo creature as being in a single global state. To achieve this effect, Topobo comes with cables to connect all the active modules together, so that button commands received on one module are broadcast across the system.

There are several ways to issue a command to a system. A button is sufficient for simple local commands—more sophisticated sensors installed in each module can support different styles of local commands. For example each tile of the Siftables kit [9] detects gesture commands with an embedded accelerometer. Modules with cameras could capture, recognize, and interpret hand gestures; custom gestures could be recorded and associated with commands such as the double fist gesture for indicating the 'basin' command illustrated in Fig. 7c.

A remote control such as an application running on a cell phone could broadcast commands to an entire system—the phone interface might serve as a magic wand that, pointed at a furniture hyperform, would turn it into a table, couch or bed. Voice recognition could allow spoken commands to be quickly and intuitively broadcast throughout a system.

4.3.2 *Signalling interfaces*

A signal allows a module to communicate its internal state. A simple signal is a multicolor LED that indicates a small number of states. This could be used to indicate, for example, a surface that is currently 'grabbed' with sticky hands. To send more complex signals, modules could incorporate full-color screens as do the Siftables tiles [9].

4.4 Summary

We presented a framework of three pairs of input/output affordances. To realize hyperforms, a hardware module need not provide all these affordances. A subset of these affordances can combine to realize a variety of behaviors. For example Topobo [12] does not afford placing or self-reconfiguring, and only some, 'active', modules contain electronics. Its only degrees of freedom are the rotational joints of the active modules, so Topobo can sense whichever part of the model is manipulated and reproduce the motion. Another example is a self-balancing table [28] with legs composed of simple chain-style modules (similar to Polybot [25] without self-reconfiguration). The table can be placed on an uneven surface or in someone's lap, and through a distributed algorithm the modules that make up the legs coordinate to keep the surface of the table level. However, to realize a self-reconfiguring material a system must at a minimum afford self-reconfiguration.

5 Supporting hyperform interaction with the prismatic cubes

So far we have sketched a speculative scenario of hyperform use, considered a life cycle model for hyperform specification, examined how the underlying hardware and software constrain hyperform behavior, and outlined a framework for tangible interaction with hyperforms. In this section, we ground this discussion in a particular implementation of a lattice style system that we have been working on, the *prismatic cubes*.

Modular robotics research has largely been concerned with demonstrating self-reconfiguration: our prismatic cubes have demonstrated the low-level transitions that could support self-reconfiguration on a handful of modules; and other hardware systems [5, 10] have demonstrated self-reconfiguration on somewhat larger ensembles. Below we briefly describe our current implementation of the prismatic cubes hardware (for more detail see [24]). Then, we discuss features we could add to this system to better support hyperform interaction.

5.1 Prismatic cubes implementation

To support self-reconfiguration with the prismatic cubes, we have focused on structural robustness and simplicity of control. Ensembles of prismatic cubes achieve structural robustness through their close-packed cubic lattice and the powerful intermodule bond provided by their electrostatic latch [6]. To simplify control, we designed modules to dock passively and attempted to minimize the number of modules that need to coordinate to implement the cubes' control abstraction, movement primitives.

5.1.1 Hardware design

For our prismatic cubes, we have chosen a module size amenable to rendering interactive furniture as outlined in our scenario above; each is roughly the size of a brick. We adopt the morphology pioneered by the Crystalline Atom [14] and the Telecube [18]; each face of a cubic module can extend independently and latch to a neighboring module (Fig. 8). With faces closed the modules stack in a cubic lattice 125 mm on center. By coordinating with its neighbors, a module can move itself one space over in this lattice as discussed in Sect. 3—see Fig. 3b

Each face is outfitted with an electrostatic latch [6] made of a mylar-wrapped extruded comb that can be charged with static electricity. When two faces mate the combs interlace, guided by a passive alignment mechanism. Once charged, strong electrostatic forces keep them latched (without additional power) until the electrostatic charge is drained.

In most situations the combination of the dimensional constraints of the cubic lattice and the passive self-alignment built into our electrostatic latches suffice to guide docking with no explicit planning beyond extending two neighboring faces.

An essential aspect of self-reconfiguration is moving a module from one location to another. Independently actuating each face of the prismatic cubes makes the individual modules more complex but it allows modules to slide in any direction by coordinating with only a few neighbors.

5.1.2 Control abstraction

The low-level details of coordinating a group of modules to unlatch, actuate and then relatch in order to move a module into a neighboring lattice cell can be complex. To simplify planning, instructions are often given in terms of a control abstraction that specifies an atomic movement of a small group of modules to a nearby location. We created *movement primitives* [24] as an abstraction layer for the prismatic cubes. Each module can choose when to trigger a movement primitive that maps the current local geometry to a new geometry.

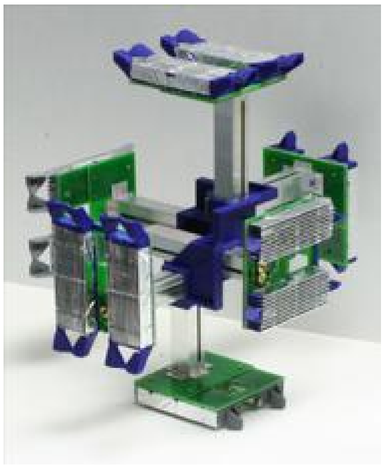


Fig. 8 Current prototype prismatic cube module showing core and six latch faces

Moving the prismatic cubes vertically or horizontally is fairly straightforward; the most difficult transition to accomplish with this morphology is moving around a convex corner. The *round* movement primitive (Fig. 5c) does this using twelve modules. Executing this primitive places the most strain on the hardware; as shown in Fig. 9 it requires one module to hold two others cantilevered at a distance of one cell.

Our prismatic cube movement primitives lend themselves to planning methods such as the Hole Motion Planner [13] that transition through the interior of an ensemble, an important feature for practical transitions in spaces shared with objects and people.

5.2 Future work in supporting hyperform interaction

To allow us to experiment with different modes of tangible and gestural interaction, each face of the prismatic cubes has an expansion bay for a daughter board. Here, we discuss some different sensors and actuators we could include on an expansion board and the modes of interaction they would support.

5.2.1 Intermodule communication

One feature we are including on the expansion board is intermodule communication. While needed for self-reconfiguration, some implementations could also be useful for supporting tangible interaction. Two leading contenders are infrared transceivers and electrical spring probes. Spring probes detect when latches are fully engaged, while infrared transceivers can be tuned to communicate with neighboring modules before latching.

5.2.2 Face coloring

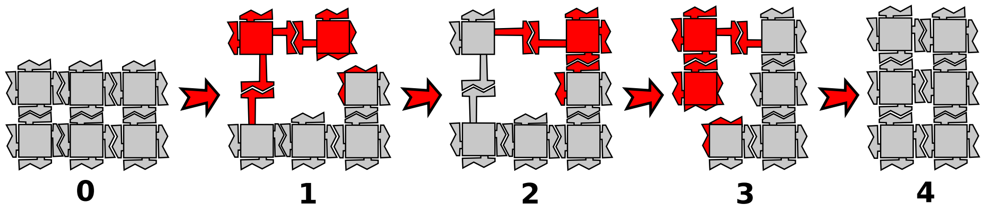
We would like to control the color of individual faces. This could be accomplished with diffuse multicolor LEDs or electrochromic paint (that changes color when a charge is applied). Faces could use color to indicate a change in state, for example when a surface is selected with the 'sticky hands' technique (Fig. 7b). Or each module in a surface could serve as a pixel to create a low-resolution screen.

5.2.3 Grasp detection

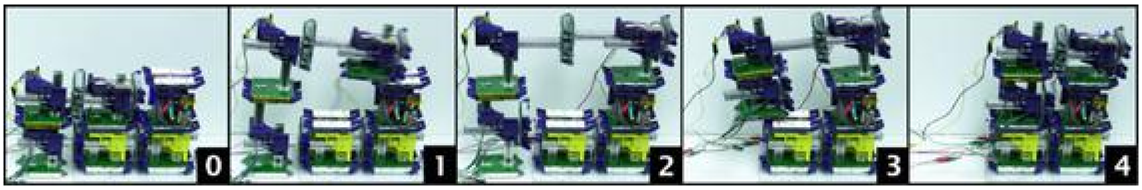
We would like to detect when a cube is grasped, for example to initiate 'placing mode' as shown in Fig. 7a. This could be implemented in several ways: a simple but inelegant solution would provide a button on each face that you press to "grasp" a module. A more elegant solution would detect (with capacitive sensing) when the comb of a face is touched. If a camera or distance sensor is already included for gesture sensing it could also be used to detect grasping; however with a distance sensor it could be difficult to distinguish a hand from a foreign object.

5.2.4 Latch engagement

When a module is manually placed next to a new module, we would like to detect when the latches are fully engaged and then actuate them to hold the module in place. A spring probe communication link could do this. Or a simple limit switch could detect when another face touches it. Alternatively, a distance sensor together with infrared communication would enable modules to detect each other even if



(a)



(b)

Fig. 9 The round motion primitive moves two modules around a convex corner; it places the most strain on the hardware [24]. **a** Diagram of round primitive actuation. **b** Round primitive being executed on hardware

they were only nearby, and could then actuate to fully engage.

5.3 Ensemble surfaces

The prismatic cubes' form is well suited for tangible spatial modeling; the hand-sized blocks can be stacked manually or can respond to tangible and gestural input to self-reconfigure. However, a limitation of our cube hardware (as well as other hardware systems) is the quality of the surface of an ensemble; the surfaces of most current systems are dominated by latching mechanisms. The faces of the current cube prototype do not offer a particularly appealing surface for sitting or laying (see Fig. 8). To realize furniture and structures, there are a variety of surface properties that would be desirable, e.g., softness (for sitting or laying), a watertight seal (for tables and basins and roofs) and smoothness (for tables and counters). Future hardware modules could feature more discreet latching systems that allow greater control over surface properties, or ensembles could feature specialized skin modules that provide desirable surface properties.

6 Scenario illustrating hardware implementation

In Sect. 2, we envisioned a 'social table' hyperform to illustrate the life cycle of objects constructed with self-reconfiguring materials. Then in Sect. 5, we discussed how our prismatic cube modules could be adapted to support hyperforms. Here, to illustrate how the choice of hardware system affects hyperforms, we envision a second, simpler scenario; a couch that turns into a bed when someone lays down on it, and then back into a couch when there is no longer anyone laying on it. We describe this scenario in the terminology developed above. Then, we examine in more detail how several behaviors described in this scenario could be implemented with the improved prismatic cubes, illustrating how the choice of hardware system could support desirable behaviors and thus inform system design.

6.1 Fold-out couch hyperform

The hyperform implementer begins to design the fold-out couch hyperform by selecting an abstract couch routine at her hyperform workbench; material climbs out of the reservoir to realize a basic couch on the work surface on the floor next to her. She adjusts parameters on the screen to modify the 'couch' state, and then creates a new 'bed' state and puts the ensemble into free modeling mode to shape the bed. With a combination of *posing* and *command* gestures she widens the seat to make enough room to sleep, pulls the arms up into a headboard and footboard, and squashes down the back of the couch. To get the top edge of the headboard just right, she adjusts several individual modules by grasping and *placing* them.

Once satisfied with the shape of both the couch and bed she selects both states on the screen and adds a transition from the couch to the bed. She selects a posture identification routine from a sensor library (built by a hyperform scripter) that determines whether a person is laying, sitting or standing anywhere on the structure; when a person's posture changes the routine generates events. She sets the 'laying on' event to trigger the transition. She adds a transition from the bed state back to the couch and sets it to trigger when there is no longer anyone laying on the bed.

She selects the couch-to-bed transition on the screen and opens the customization interface. She selects *minimizing surface turbulence* as the most important consideration (there is someone laying on the ensemble!). The system selects a distributed planning algorithm to attempt to satisfy this requirement and shows a simulation of the transition under that planner on the screen.

To further protect the person occupying the hyperform during reconfiguration, she applies a library routine that restricts module movement near people. The routine tracks the position of the occupant and routes transitioning modules around a constraint envelope.

To test the behavior she has just developed, our implementer now lays down on the couch. The posture detection routine senses this and triggers the transition to the couch state. The distributed planner identifies areas of the structure that need to grow and routes modules from areas that need to shrink; as more modules are needed they are drawn from the reservoir. As the back of the couch begins to melt away a tower of modules under her arm blinks bright yellow; where her arm is laying across the back of the couch the modules are forbidden to move by the 'occupant constraint envelope' routine. Once she repositions her arm the transition completes. Satisfied, she applies an alarm routine that will gently shake the occupant awake after a short nap.

6.2 Implementation scenarios

We now describe how several of the hyperform behaviors described above could be realized.

6.2.1 Implementation of placing

To adjust the details of the headboard, the hyperform implementer adjusts the position of individual modules by manually placing them. To allow our prismatic cubes to be placed as illustrated earlier in Fig. 7a, each face would feature an infrared transceiver to communicate with neighbors and a sensor to trigger a 'placing mode' when a

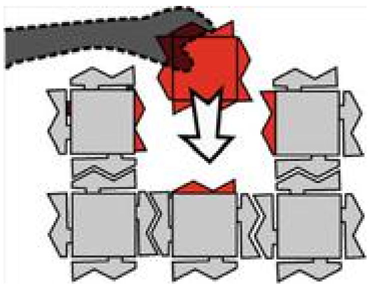


Fig. 10 A prismatic cube in placing mode being inserted into a lattice

cube is grasped—the module disconnects from its neighbors and retracts its faces so that it can be manually removed from the ensemble. The (now retracted) faces of the grasped module also turn red to indicate that the module is in placing mode. When the module is placed near a new position it communicates with its potential new neighbors over infrared; valid nearby positions are indicated by turning those faces red also. As shown in Fig. 10, faces adjacent to a valid position retract to allow the cube to be placed. Once inserted the cube latches to neighboring cubes and leaves placing mode, returning all the cubes' faces to their default colors.

6.2.2 *Implementation of sticky hands*

To reshape one arm of the couch form into a headboard, the hyperform implementer uses posing gestures such as 'sticky hands'. She holds one hand over the top surface of the arm until it turns red (to indicate it is selected) and pulls it up to establish the height of the headboard. She then uses a few other command and posing gestures to establish a plane of symmetry and further mold the shape. To realize the 'sticky hands' behavior (Fig. 7b), each cube's face would be equipped with a distance sensor to sense that a hand is being held nearby. As above, with placing, a specialized distributed algorithm colors other modules in the

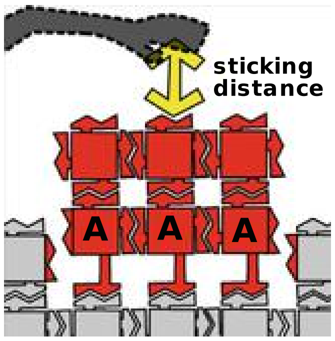


Fig. 11 Modules beneath top layer (marked with an *A*) recruited to maintain 'sticking distance' with hand held over top surface of serving platform

same surface red. These then actuate in parallel to move the surface to follow her hand.

The modules in the selected surface recruit the next layer of modules beneath them (marked *A* in Fig. 11) to participate in raising and lowering the surface. As the surface is raised or lowered further a planning algorithm such as the Hole Motion Planner [13] grows or shrinks the area beneath the surface so that the surface boils upward to follow, and craters downward as it is pushed back.

6.2.3 *Implementation of gesture commands*

One tricky aspect of hyperform interactions that we have only hinted at is the question of how to explicitly control transitions. For example, although the couch described here only responds to changes in the environment (someone laying on it), we suggest in Sect. 2 that when a drink is spilled on the table a 'double fist' gesture could command the table to explicitly form a basin to contain the mess. To trigger this style of command, we must both select a desired behavior and indicate where it should happen. Hand gestures are a sort of 'command line' for tangible interaction: a minimal and powerful interface, which however requires you to commit commands to memory. Another possibility is that a handheld device (such as a cell phone) could present a menu of possible behaviors, and then be pointed to indicate a site of operation.

6.2.4 *Implementation of reconfiguration near people*

This scenario highlights a complication of hyperform furniture—the need to protect the safety of nearby people. One aspect is assuring that structures are statically stable and will not collapse or overturn, even as they reconfigure; another is assuring that moving modules do not strike or pinch nearby people.

Figure 12 shows a diagram illustrating the cross-section of the couch hyperform during transition. A constraint envelope surrounds the occupant and prevents any immediately adjacent modules from moving. The profiles of the couch and bed are shown in dotted lines, and the cube modules are moving to vacate the back of the couch (on the left) and fill in the extended profile of the bed (to the right). However, the person laying on the couch has thrown her arm over the back of the couch, preventing a column of modules from moving away.

An initial strategy to resolve conflicts between people and modules that are attempting to move is to blink the modules to indicate that they are stuck (marked *S*), and then wait for the person to move out of the way.

An advantage of the prismatic cubes is that they can transfer through the center of a lattice. However, structures could be destabilized by this internal reconfiguration. For

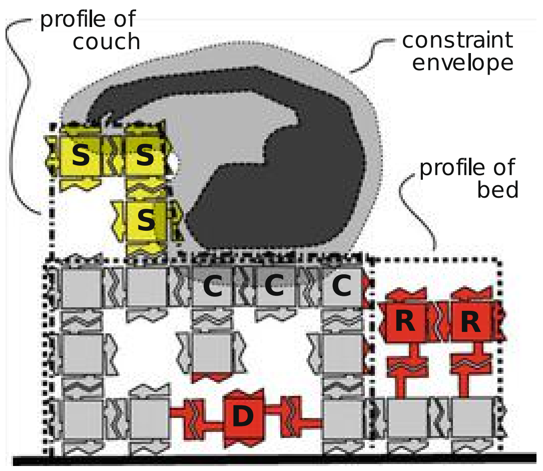


Fig. 12 Cross-section of a hyperform as it transitions from couch to bed. Modules marked *C* are constrained to stay where they are as they fall within the constraint envelope surrounding the person. Modules marked *S* need to move but are stuck as they also fall within the constraint envelope. Modules marked *D* and *R* are currently moving to reconfigure into the bed state

example, the module directly under the person in the middle of reconfiguring (marked *D*) is potentially destabilizing the structure by abandoning the load-bearing module overhead. A challenge is to model the loads on a structure from nearby people even during reconfiguration.

6.3 Summary

With current modular robotics systems and tangible interaction techniques, we can begin to think about realizing some of the most basic hyperform behaviors. Directly placing modules is within reach, and a somewhat fragile version of sticky hands could be realized on our current prismatic cubes hardware. Transitioning ensembles between shapes such as the couch and bed states described above is still limited to simulation due to the difficulty and expense of making that many modules. Even with a sufficient number of modules, much more work on control algorithms is necessary before it would be safe to sit on a couch hyperform.

7 Discussion and future work

We have presented a vision of a technology that could revolutionize the ways we create and interact with the objects that comprise our built environment. Self-reconfiguring materials such as a more advanced and refined version of our prismatic cubes could disrupt the current manufacturing cycle by digitizing the distribution of objects and including a larger swath of society in the design process. The challenges of interacting with and designing hyperforms present a radical ubiquitous computing research program in which our everyday objects are *composed of* tiny robots and our primary mode of interaction with them is through tangible and gestural interfaces. In surveying technologies that could support this vision, and presenting scenarios illustrating how they could be combined to realize hyperforms, we have attempted to outline the space of this project. Here, we first suggest some directions for future research toward self-reconfiguring materials. Then, we suggest some opportunities for the tangible interaction community to inform these next steps.

7.1 Next steps toward self-reconfiguring materials

Although great progress has been made toward functional self-reconfiguring materials, challenges remain to be overcome. Below we suggest some next steps that could be taken. Four areas that merit particular attention are manufacturing, planning, sensing and verification.

7.1.1 Manufacturing

We are beginning to be able to build modules capable of self-reconfiguring; yet it is still laborious and expensive to produce them in quantity. This impedes our ability to produce ensembles to serve as testbeds for further development. Developing modules optimized for inexpensive manufacture could improve this situation.

7.1.2 Planning

Although attempts to realize distributed planners that can arrange an ensemble into a shape specified by a 3D model have met with some success, full 3D models are an impoverished format for representing individual states. More work needs to be done to establish planning algorithms to transition between states defined in terms of constraints and parameters, as well as algorithms to define dynamic states.

7.1.3 Sensing

To operate safely among us, hyperforms will need to sense both people and objects and distinguish between the two. Possibilities include arrays of infrared sensors to cameras embedded in each module, as well as more exotic solutions such as identifying people by smell in order to avoid encroaching.

7.1.4 Verification

A particularly important area of research is to be able to predict how these complex systems will behave, and to verify that the behavior specified for a particular hyperform will not put people in danger. These systems could range from a simple distributed structural model running on a hyperform to test the safety of potential reconfigurations on the fly, to a sort of automated building code that would take a hyperform specification file and test all possible behaviors to guarantee its safe performance.

7.2 Challenges for the tangible interaction community

The scenarios we have presented here are not intended as a research program but as an invitation to the community to consider this space. By engaging with the issues of how to create and interact with hyperforms, the community can help to guide the development of self-reconfiguring materials.

One of the biggest obstacles to participating in developing methods for interacting with hyperforms is building or acquiring self-reconfiguring materials. However, there are several strategies for validating novel interaction techniques without an ensemble of self-reconfiguring hardware. To demonstrate the viability of interaction techniques designed to work in a distributed fashion on large ensembles, we can use computer simulations. It is not necessary to implement self-reconfiguration to explore methods of interacting with self-posing objects; as mentioned above both Topobo [12] and the self-balancing table [28] explore novel modes of interacting with ensembles of robotic modules. It is not necessary to implement any electronics at all to explore the behavior of distributed algorithms—by participating in the Human Hive [22] a group of people execute the rules of a stigmergic algorithm (each person has a card with a single rule) to build a large hive structure. Below we consider some of the areas of exploration that could inform the future development of hyperforms.

7.2.1 The hyperform life cycle

Self-reconfiguring materials have the potential to completely revolutionize the way we create and acquire things. There are many possibilities to explore in this space. It could be that instead of having expert hyperform scripters and hyperform implementers create specifications, all we need is a simple construction interface and people will design their own things and trade them with their friends in a sort of physical embodiment of MySpace. Or it could be that we are too optimistic about developing automated safety-assurance routines for hyperforms; after a few unfortunate mishaps people will only be willing to download hyperforms from large and reputable brokers that vouch for the safety of each hyperform specification (and provide little opportunity for hazardous customizations).

7.2.2 Tangible spatial modeling

This is an area that has begun to be explored but still holds a lot of potential. Is the ‘sticky hands’ technique suggested here a good model for describing forms? Perhaps we will not directly interact with self-reconfiguring materials, but rather we will sketch forms in space with a stylus; we will then fill these forms with a 3D ‘paint bucket’ tool that gives us control of the material properties.

7.2.3 Hyperform command and control

Once we are able to create hyperforms, will it be worth the trouble of interacting with them? There are many possible models for triggering hyperforms to alter their behavior. We could explicitly control changes in behavior with the analog of a command line or a manual transmission for a car. Hyperform implementers could compete to provide empathetic hyperforms: by sensing our emotional states these structures could anticipate when to alter their behavior without our having to ask. Or the objects surrounding us could become sophisticated pets that we train from “puppies” to behave as we would like, with varying degrees of success.

7.3 Our next steps

We are still working towards manufacturing enough of our prismatic cubes to serve as a testbed for experimentation with control software and interaction techniques. In parallel, we are considering distributed algorithms for supporting tangible interaction with larger ensembles in simulation. The first steps toward realizing hyperforms have been taken, but there is a great deal of work to be done before we will be eating at a social table.

Acknowledgments This research was supported in part by the National Science Foundation under Grants ITR-0326054 and CNS-0428738.

References

1. Anderson D, Frankel J, Marks J, Leigh D, Ryall K, Sullivan E, Yedidia J (1999) Building virtual structures with physical blocks. In: User interface software and technology (UIST). ACM, pp 71–72
2. Butler Z, Kotay K, Rus D, Tomita K (2002) Generic decentralized control for a class of self-reconfigurable robots. In: International conference on robotics and automation (ICRA). IEEE, pp 809–816
3. Christensen DJ, Østergaard EH, Lund HH (2004) Metamodule control for the atron self-reconfigurable robotic system. In: Intelligent autonomous systems (IAS), pp 685–692
4. Esposito C, Paley WB, Ong J (1995) Of mice and monkeys: a specialized input device for virtual body animation. In: Symposium on interactive 3D graphics, pp 109–114

5. Jorgensen MW, Østergaard EH, Lund HH (2004) Modular atron: modules for a self-reconfigurable robot. In: Intelligent robots and systems (IROS). IEEE, pp 2068–2073
6. Karagozler ME, Campbell JD, Fedder GK, Goldstein SC, Weller MP, Yoon BW (2007) Electrostatic latching for inter-module adhesion, power transfer, and communication in modular robots. In: Intelligent robots and systems (IROS). IEEE, pp 2779–2786
7. LeClerc V, Parkes A, Ishii H (2007) Senspectra: a computationally augmented physical modeling toolkit for sensing and visualization of structural strain. In: Human factors in computing (CHI). ACM, pp 801–804
8. Lynch N, Segala R, Vaandrager F (2003) Hybrid I/O automata. J Inf Computing 185(1):105–157
9. Merrill D, Kalanithi J, Maes P (2007) Siftables: towards sensor network user interfaces. In: Tangible and embedded interaction (TEI). ACM, pp 75–78
10. Murata S, Yoshida E, Kamimura A, Kurokawa H, Tomita K, Kokaji S (2002) M-tran: self-reconfigurable modular robotic system. Trans Mech 7(4):431–441
11. Parkes A, LeClerc V, Ishii H (2006) Glume: exploring materiality in a soft augmented modular modeling system. In: Ext. abstracts of human factors in computing (CHI). ACM, pp 1211–1216
12. Raffle H, Parkes A, Ishii H (2004) Topobo: a constructive assembly system with kinetic memory. In: Human factors in computing (CHI). ACM, pp 647–654
13. Rosa MD, Goldstein SC, Lee P, Campbell JD, Pillai P (2006) Scalable shape sculpting via hole motion: motion planning in lattice-constrained module robots. In: International conference on robotics and automation (ICRA). IEEE, pp 1462–1468
14. Rus D, Vona M (2001) Crystalline robots: self-reconfiguration with compressible unit modules. Autonomous Robots 10(1):107–124
15. Schweikardt E, Gross MD (2006) roblocks: A robotic construction kit for mathematics and science education. In: International conference on multimodal interfaces (ICMI). ACM, pp 72–75
16. Shneiderman B (1983) Direct manipulation: a step beyond programming languages. Computer 16(8):57–69
17. Stoy K, Nagpal R (2004) Self-reconfiguration using directed growth. In: Distributed autonomous robotic systems (DARS), pp 1–10
18. Suh JW, Homans SB, Yim M (2002) Telecubes: mechanical design of a module for self-reconfigurable robotics. In: International conference on robotics and automation (ICRA). IEEE, pp 4095–4101
19. Theraulaz G, Bonabeau E (1995) Modelling the collective building of complex architectures in social insects with lattice swarms. J Theor Biol 177(4):381–400
20. Watanabe R, Itoh Y, Asai M, Kitamura Y, Kishino F, Kikuchi H (2004) The soul of activecube—implementing a flexible, multimodal, three-dimensional spatial tangible interface. In: Advanced computer entertainment technology (ACE). ACM, pp 173–180
21. Weller MP, Do EY-L, Gross MD (2008) Posey: instrumenting a poseable hub and strut construction toy. In: Tangible and embedded interaction (TEI). ACM, pp 39–46
22. Weller MP, Do EY-L, Gross MD (2009) Exploring architectural robotics with the human hive. In: Creativity and cognition (C&C), ACM (to appear)
23. Weller MP, Do EY-L, Gross MD (2009) An optocoupled poseable ball and socket joint for computationally enhanced construction kits. In: Robot communication and coordination (RoboComm). IEEE, pp 1–6
24. Weller MP, Kirby BT, Brown HB, Gross MD, Goldstein SC (2009) Design of prismatic cube modules for convex corner traversal in 3D. In: Intelligent robots and systems (IROS), IEEE (to appear)
25. Yim M, Duff DG, Roufas KD (2000) Polybot: a modular reconfigurable robot. In: International conference on robotics and automation (ICRA), pp 514–520
26. Yim M, Shen WM, Salemi B, Rus D, Moll M, Lipson H, Klavins E, Chirikjian GS (2007) Modular self-reconfigurable robot systems. Robotics Automation 14(1):43–52
27. Yim M, Shirmohammadi B, Sastra J, Park M, Dugan M, Taylor C (2007) Towards robotic self-reassembly after explosion. In: Intelligent robots and systems (IROS). IEEE, pp 2767–2772
28. Yu CH, Willems FX, Ingber D, Nagpal R (2007) Self-organization of environmentally-adaptive shapes on a modular robot. In: Intelligent robots and systems (IROS). IEEE, pp 2353–2360
29. Zykov V, Mytilinaios E, Desnoyer M, Lipson H (2007) Evolved and designed self-reproducing modular robotics. Trans Robotics 23(2):308–319