Cluster-based tangible programming

Andrew Cyrus Smith
CSIR Meraka Institute and University of South Africa
Pretoria, South Africa
acsmith@csir.co.za

***Abstract*—Clustering is the act of grouping items that belong together. In this paper we explore clustering as a means to construct tangible program logic, and specifically as a means to use multiple tangible objects collectively as a single tangible program parameter. We introduce T-logo, a simple tangible programming environment developed to test the concept of cluster-based programming. Although the principle of cluster-based programming is technology agnostic, we describe it here by means of a vision-based system. We further introduce the concepts of Cluster Marker, Cluster Marker Position, and Cluster Marker Zone to describe the principles on which cluster-based programming is based.**

Keywords—construct, physical program logic, intangible program logic, tangible program logic, free-form construction, cluster.

I. INTRODUCTION

T-Logo is based on the Logo [1] programming language and serves as a vehicle to test the tangible clustering concept. Although T-logo makes use of the reacTIVision framework [3], many other position and orientation mechanisms could be used with adaptation, for instance the Microsoft Surface [2].

To construct tangible program logic, a tangible object with a fiducial attached to its underside is placed on a construction surface. A camera is located below the construction surface, pointing up towards the surface and detects the fiducial through the translucent glass. Images captured by this camera are relayed by the camera control software to the reacTIVision image analysis software where the position and orientation of the fiducial are determined. The position and orientation of all the fiducials visible to the camera are then sent by means of the TUIO [3] protocol for processing by the T-Logo application software.

This paper's main contribution is an alternative physical programming mechanism based on clusters of objects positioned on a flat surface.

This paper is structured as follows. Section II briefly lists some tangible programming systems and their dimensionality. Section III is an overview of the methodology we followed in our research project. Section IV provides a high-level overview of the project. Section V describes the concept of clustering and the associated technical terminology. In section VI we describe T-logo and its implementation and an example of how it is used. Section VII describes the modes of the T-logo programming environment. Section VIII gives an overview of our tests. Section IX gives our initial observations of this project, and Section X concludes.

II. RELATED TANGIBLE PROGRAMMING SYSTEMS

Tangible program logic can be classified along many lines. Here we have classified them according to the number of Cartesian dimensions in which the logic is interpreted. In order to provide context, we give examples of intangible program logic for the one and two-dimensional cases.

A. One Cartesian dimension (one-dimensional)

Intangible textual program logic, such as constructed using the C-language, is one-dimensional. Examples of one-dimensional tangible program logic are those constructed using programming environments such as TORTIS – slot machine [4], GameBlocks [5], T-Maze [6], Electronic blocks [7], Navigational Blocks [8], and Robo-Blocks [9].

B. Two Cartesian dimensions (two-dimensional)

Two-dimensional intangible program logic can be constructed in graphic form using for example the Pd programming environment. Two-dimensional tangible logic can be created with programming environments such as Algoblocks [10], Flow blocks [11], Questzal [12], Tern [13], SmartTiles [14], QuiltSnaps [15], Tangible Programming Brick [16], reacTable [17], and Turtan [18]. reacTable and Turtan are interactive programming systems in which the results of program changes are immediately visible to the user.

C. Three Cartesian dimensions (three-dimensional)

Three-dimensional tangible program logic can be created using roBlocks [19].

All the above examples seem to interpret each tangible object as an independent entity. What is missing is a mechanism that can interpret a cluster of similar tangible objects as a single entity.

III. METHODOLOGY

We followed a design research approach in reaching our research objective. Our first design was influenced by the 'maker' community in developing regions. This design was evaluated and followed by additional iterations. The current iteration as discussed here is the result of learning from prior iterations and the researchers' subsequent exposure to contemporary positioning mechanisms.

A number of iterations were initiated, with some completed, and others abandoned or adapted prior to completion, before we have reached the current design as described here.

IV. Tangible Program Logic and Execution: T-Logo

In describing T-logo we use expressions based on human language studies. For what is usually described as commands in ‘traditional’ programming languages, we use the term verb. The object on which the verb acts is called a noun. If needed, an adverb optionally describes a verb and fulfills a role similar to the parameter used in traditional programming languages. In this paper we limit our discussion to verbs and nouns.

In this section we describe the design principles and the derived engineering requirements that were applied in reaching our research goal of the current T-logo tangible programming and execution environment.

Fig. 1. For humans, it’s a simple task to cluster kitchen items (left) and smaller objects (right).

A. T-Logo Design Principles

T-logo was designed to simplify programming for the novice artist. Our approach to simplification was to reduce the T-logo ‘instruction set’. Because our target group is the artist who is looking for a tool to support her creativity as supposed to a novice whose aim is to become a professional programmer, the justification of this constraint can be argued.

This target group would very likely not have had prior programming experience or exposure to a programming environment. This afforded us an opportunity to experiment with mechanisms that deviate significantly from existing programming approaches. Design principles were specifically developed to suit this target group:

1) Clustering
Cluster concepts in a way similar to how physical items in the real world would are clustered; items that belong together are placed together. It appears that people experience it as ‘natural’ to put things together that ‘belong’ together. For example, when clearing one’s office desk the pens and pencils are kept together but separate from the piles of printed journal pages.

2) “Soft” Failure
Syntax errors should not prevent the program from executing. Consider for example how a paint-by-numbers painting can be completed and viewed even when the incorrect color/number mapping has been used at places.

3) Tangible Artifacts
Combinations of tangible artifacts represent the logic of the computer program instead of intangible symbols on a two-dimensional computer screen.

B. T-logo Engineering Requirements

Once we had established the design principles we derived engineering requirements that would adhere to these principles. We next describe the resulting engineering requirements that address each of the three design principles.

1) Clustering
Assuming that the programmer clusters objects as they are perceived to belong together, we designed the programming environment that uses a marker to denote a cluster. We call this the “cluster marker” (CM). Around this marker a circular region of a fixed radius is identified. This circular region is also called the “zone of influence”. Our algorithm assumes that all objects located within this circular region are to be “associated” with each other, and any object beyond this circular region not to be associated with the objects inside this region.

2) Soft Error Handling
We engineered recovery from anticipated errors by assigning fixed default values for adverbs if they are not explicitly present in the tangible pro-gram logic. Conversely, in the case when no verb is present and only adverbs given, then such a cluster is simply ignored and not interpreted further.

3) Physical Artifacts
Our design makes use of three-dimensional objects which the user places on a two-dimensional construction surface. Underneath each physical artifact is attached an optical marker (fiducial) whose value and position is detected and stored when the object is placed onto the construction surface. Depending on its intended function in the program logic, a physical object may represent either a verb or an adverb.

V. Clustering

Clustering is a method of grouping tangible objects that seemingly belong with each other (Fig. 1).

Fiducials attached to the physical objects identify the tangible objects to the computer process tasked to cluster objects. Each fiducial is distinguished from others by its identification number (ID), its orientation, and its two dimensional position on the construction surface.

A. Cluster Marker, Position, and Zone

Clustering is achieved by first identifying all fiducials with ID = 0. This specific ID number is reserved for this purpose only. We call a tangible object with this associated ID a Cluster Marker (CM). The positions of the remainder fiducials are in turn compared with the position of each CM.

The position of each CM is mapped to a plane in the digital domain and we call this position the Cluster Marker Position (CMP) (Fig. 2, Fig. 3, and Fig. 4). We name the area with radius R around the CMP the Cluster Marker Zone (CMZ). The CMZ specifies the CMP's 'area of influence'.

Another tangible object, with an attached fiducial, may be placed in close proximity to the CM. If this object falls within the CMZ then it is assumed to be associated with the CMP.

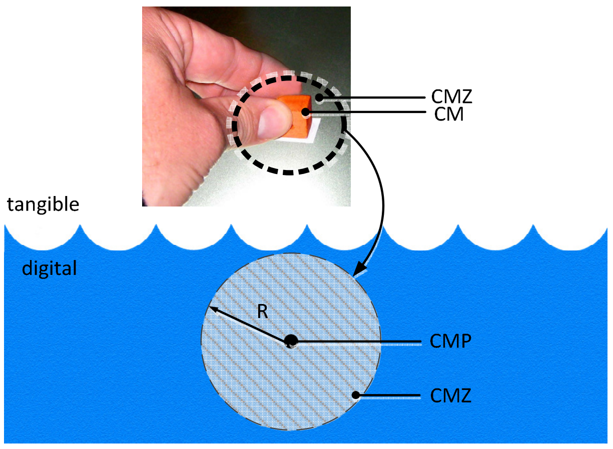


Fig. 2. A CMZ with a radius of R is centred on the CMP (diagram design based on [20]).

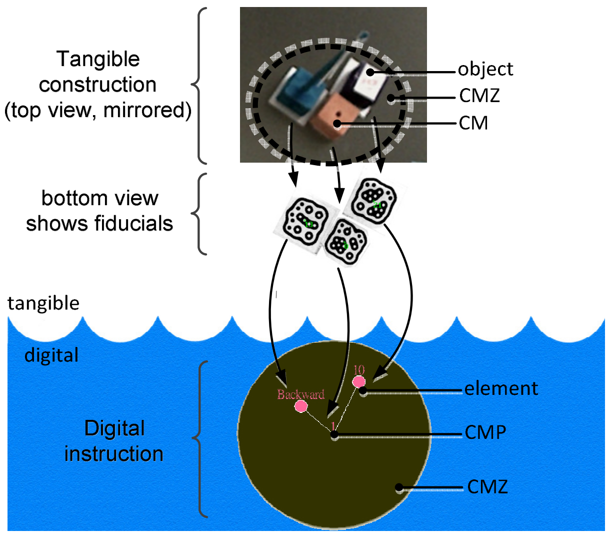

Tangible construction (top view, mirrored)
object
CMZ
CM
bottom view shows fiducials
tangible
digital
Backward
10
element
CMP
CMZ
Digital instruction

Fig. 3. A tangible construction and equivalent digital instruction (diagram design based on [20]).

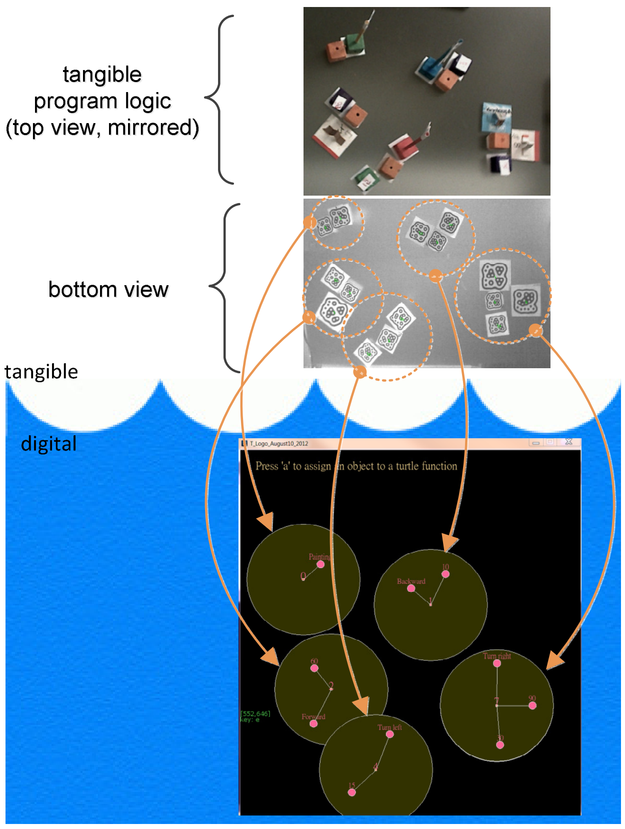

tangible program logic (top view, mirrored)
bottom view
tangible
digital
Press 'a' to assign an object to a turtle function
Backward
10
Turn right
90
Turn left
15
Forward
60

Fig. 4. The tangible construction is replicated in the digital domain.

VI. T-Logo

The T-logo system consists of tangible objects with attached fiducials, the construction surface, a light source, a web camera, camera control software, image analysis software, and a display.

Clusters consist of at least one tangible object, called an element. A single element can either represent an action or a numerical value. Tangible objects are identified by their fiducial identity (ID) and a look-up table is maintained in computer memory which maps multiple fiducials to multiple verbs or values. Mapping makes provision for multiple ID's to be mapped to the same action or value.

Clusters are sorted according to a top-bottom, left-right sequence. Fig. 5 illustrates the top left to bottom right scanning sequence used for detecting CMP's in the source image. In Fig. 5, the order in which the CM's are processed is first CM #1, then CM #2, and then CM #3.

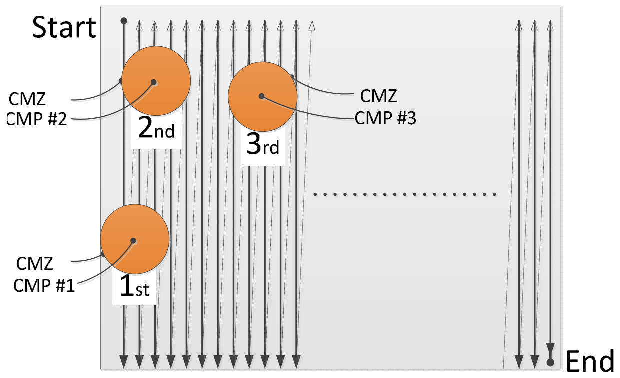
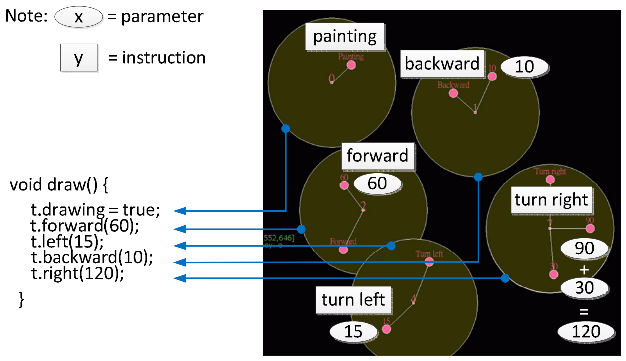


Fig. 5. The scanning and processing sequence.

The compilation process consists of four steps. (1) A predefined code template is written to the target file. (2) For each sorted Cluster, the elements contained are evaluated (Fig. 6, right) in sequence. (3) When multiple numerical values are present in one Cluster, the values are summed and the numerical result used in further operations. Fig. 6 shows an example of where 90 and 30 are added to produce the result of 120. (4) A line of text code is sent to the target file for each action type detected, along with the numerical summed value if required (Fig. 6, left). Values are simply ignored when none are required.



```
void draw() {
    t.drawing = true;
    t.forward(60);
    t.left(15);
    t.backward(10);
    t.right(120);
}
```

Fig. 6. Clusters (right) are interpreted and text added to a program file (left).

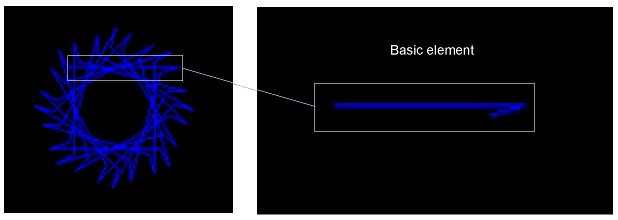


Fig. 7. The resultant display after repeatedly executing the five instructions in Fig. 6.

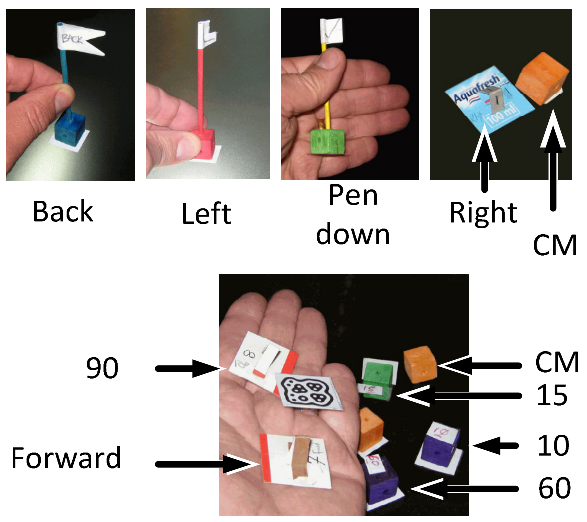


Fig. 8. Tangible objects and what they represent.

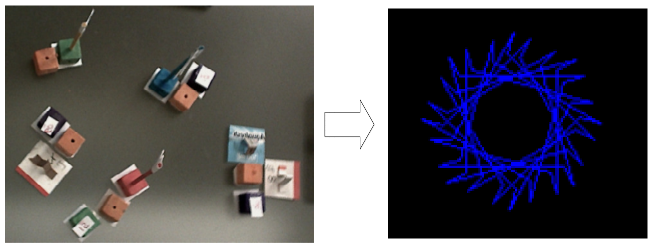


Fig. 9. Tangible program logic and intangible result.

VII. T-Logo Modes

The T-Logo software executes in three modes: mapping, construction, and execution. We describe these modes next.

A. *Mapping Mode*

Clusters consist of at least one tangible object, called an element. A single element can either represent a verb or a numerical value that elaborates a verb. When the T-logo software operates in mapping mode, a tangible object is bound to a predefined program action or numerical value.

Tangible objects are identified by their fiducial identity (ID) and a look-up table is maintained in computer memory which maps multiple fiducials to multiple verbs or values. Mapping makes provision for multiple ID's to be mapped to the same verb or value. However, it is possible that a particular ID is mapped to one verb or value only.

Mapping is achieved as follows: First, the user places an element on the construc-tion surface. Next, T-logo decodes the fiducial marker attached to the element and prompts the user with seven options on the computer screen. These options are 'hard coded' and cannot be changed by the user:

- Two options relate to the ability of the drawing turtle. One of these is equivalent to the Logo Pen Down command, and the other to the Pen Up command.

- Four options set the direction in which the drawing turtle will move. These are equivalent to the Logo Forward, Back, Left, and Right commands.

- One option allows for a numerical value to be mapped. This option provides the ability for the user to attach a positive integer to the element.

Finally, when the user has made the selection, the ID and selection are stored as a data pair in computer memory.

B. Construction Mode

Construction in a tangible programming environment is roughly equivalent to the act of programming in 'traditional' programming environments when text/graphic editors are employed. In this mode the user makes use of a CM to identify which combination of IDs should be clustered together. Clustering is done according to the position of an element relative to the CM's associated CMP. If an element resides within a particular CMZ (Fig. 3), the element is assumed to be associated with that particular CMP.

C. Execution Mode

The execution mode follows three sequential steps: sort, compile, and launch. We describe these steps next.

1) Sort

The construction surface is considered to be a two-dimensional plane. Clusters are sorted according to a top-bottom, left-right sequence. Fig. 5 illustrates the top left to bottom right scanning sequence used for detecting CMP's in the source image.

Scanning is done to the same resolution as the image received from the camera. As used in this research, the resolution may be considered as infinitely fine.

2) Compile

Fig. 5 shows the sequence in which the example CMZ's are processed when the following steps are executed.

The compilation process consists of four steps. (1) A predefined code template is written to the target file. (2) For each sorted Cluster, the elements contained are evaluated in sequence. (3) When multiple numerical values are present in one Cluster, the values are summed and the numerical result used in further operations. (4) A line of text code is sent to the target file for each verb detected, along with the numerical summed value if required. Values are simply ignored when none are required.

Fig. 7 is the result when the tangible program, shown of Fig. 4, is executed.

3) Launch

An instance of the execution environment is launched with the target file as parameter. The result is the execution of the target file as an independent process. Visual output is shown on a display (Fig. 9, right).

VIII. T-LOGO TESTING

T-logo was designed to take as input tangible objects that define program logic and display the result of program execution on a computer display. We conducted laboratory tests to identify and rectify problems. Steps (2) and (3) below were repeated with various combinations of the tangible objects.

The test required the sequential activation of the three T-logo modes as follows:

(1) Using the mapping mode, we associated tangible objects with program elements. Program elements include verbs, adverbs, and CM's. For the purpose of our test we mapped the objects to elements as indicated by the annotations in Fig. 8. Some of the objects were cut from cardboard stock while others were made from colored wooden cubes. Fiducials in the form of printed paper patterns were attached to the bottom of each object.

(2) T-logo was then placed in the construction mode. In this mode we arranged the 15 objects on the glass construction surface into clusters to represent the desired program logic. Fig. 9 (left) shows the construction ready to be interpreted.

(3) With the construction completed we placed T-logo in interpretation mode. This resulted in the display as shown in Fig. 9, right.

IX. OBSERVATIONS

In this section we provide our initial observations in developing T-logo cluster-based tangible programming environment:

CMZ: The size of the CMZ is 'hard coded' in the T-logo software and we had to adjust the CMZ multiple times to achieve a usable system. Three variables have to be considered when choosing a CMZ: (a) the CMZ should be large enough to encompass multiple tangible objects in program constructions, (b) the CMZ should not occupy a large portion of the construction surface so as to leave space for other CMZ's, (c) the minimum size of a fiducial is determined by the usable resolution of the camera and lighting conditions.

Using our setup, we found that we could fit five clusters in an area of approximately 400x400mm.

Some tangible objects were marked using paper flags to indicate their intended function (Fig. 8). The extension dowel on which the flags were attached served as unexpected useful handles with which to move the objects. Some cubes did not have these extensions and required careful manipulation so as not to disturb the surrounding objects already on the construction surface.

The following result is not a consequence of the design but rather the implementation thereof. A notable delay is evident between the time when the execution mode is activated and when the result is visible on the display. This can be explained as follows: The current implementation makes multiple calls to the underlying operating system. Every call adds a delay of approximately one second to the time before the program output is made visible. We anticipate that this delay can be reduced by an order of magnitude by applying appropriate software engineering thinking to the implementation.

X. Conclusion

We have presented and explained an approach to constructing tangible program logic which makes use of 'clustering'.

All the features of the fiducial design are not yet fully exploited by T-logo. One such feature is the ability to derive the orientation of the object to which the fiducial is attached. If exploited, the program construction can be decoupled from the current 'scanning' mechanism used in the T-logo 'sort' mode. Instead, it will then be possible for the tangible objects themselves to determine the order of interpretation by means of their orientation relative to other objects.

It is possible that two or more CMZs intersect and we still have to devise a mechanism to deal with such a situation.

In conclusion, we have developed an approach to tangible programming which makes use of clusters of objects grouped in close proximity to each other so that they are interpreted collectively as a single program parameter.

Acknowledgment

This research was sponsored in part by the South African Department of Science and Technology.

References

[1] B. Harvey, "Logo," in *Encyclopedia of Computer Science*, Chichester, UK: John Wiley and Sons Ltd., 2000, pp. 1035–1038.

[2] J. Wall, "Demo I Microsoft Surface and the Single View Platform," in *Collaborative Technologies and Systems, 2009. CTS '09. International Symposium on*, 2009, pp. xxxi–xxxii.

[3] M. Kaltenbrunner, "reacTIVision and TUIO: a tangible tabletop toolkit," in *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, 2009, pp. 9–16.

[4] R. Perlman, "Using computer technology to provide a creative learning environment for preschool children.," MIT Artificial Intelligence Lab, May 1976.

[5] A. C. Smith, "Using magnets in physical blocks that behave as programming objects," in *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction*, 2007, pp. 147–150.

[6] D. Wang, C. Zhang, and H. Wang, "T-Maze: A tangible programming tool for children," in *IDC2011*, 2011.

[7] P. Wyeth and H. C. Purchase, "Tangible programming elements for young children," in *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, 2002, pp. 774–775.

[8] K. Camarata, E. Y.-L. Do, B. R. Johnson, and M. D. Gross, "Navigational blocks: navigating information space with tangible media," in *IUI '02: Proceedings of Intelligent user interfaces*, 2002, pp. 31–38.

[9] A. Sipitakiat and N. Nusen, "Robo-Blocks: designing debugging abilities in a tangible programming system for early primary school children," in *IDC*, 2012, pp. 98–105.

[10] H. Suzuki and H. Kato, "Interaction-level support for collaborative learning: AlgoBlock - an open programming language," in *CSCL '95: The first international conference on Computer support for collaborative learning*, 1995, pp. 349–355.

[11] O. Zuckerman, S. Arida, and M. Resnick, "Extending tangible interfaces for education: digital Montessori-inspired manipulatives," in *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, 2005, pp. 859–868.

[12] M. S. Horn and R. J. K. Jacob, "Tangible programming in the classroom: a practical approach," in *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, 2006, pp. 869–874.

[13] M. S. Horn and R. J. K. Jacob, "Tangible programming in the classroom with tern," in *CHI '07: CHI '07 extended abstracts on Human factors in computing systems*, 2007, pp. 1965–1970.

[14] N. Elumeze and M. Eisenberg, "SmartTiles: mobility and wireless programmability in children's construction and crafts," in *WMTE '05: Proceedings of the IEEE International Workshop on Wireless and Mobile Technologies in Education*, 2005, pp. 230–237.

[15] L. Buechley, N. Elumeze, C. Dodson, and M. Eisenberg, "Quilt Snaps: a fabric based computational construction kit," in *IEEE International Workshop on Wireless and Mobile Technologies in Education*, 2005, p. 3pp.

[16] T. S. McNerney, "Tangible Programming Bricks: An approach to making programming accessible to everyone," Massachusetts Institute of Technology, 1999.

[17] S. Jorda, M. Kaltenbrunner, G. Geiger, and R. Bencina, "The reacTable," in *Proceedings of the International Computer Music Conference (ICMC 2005)*, 2005.

[18] D. Gallardo, C. F. Julia, and S. Jorda, "TurTan: A tangible programming language for creative exploration," in *TABLETOP 2008. 3rd IEEE International Workshop on Horizontal Interactive Human Computer Systems*, 2008, vol. 10, no. 4, pp. 89–92.

[19] E. Schweikardt and M. D. Gross, "The robot is the program: interacting with roBlocks," in *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction*, 2008, pp. 167–168.

[20] H. Ishii, "Human-computer interaction: design issues, solutions and applications," A. Sears and J. A. Jacko, Eds. CRC Press, 2009, pp. 141–159.