Active Self-Assembly

Daniel Arbuckle and Aristides A. G. Requicha
Laboratory for Molecular Robotics
University of Southern California
Los Angeles, USA
darbuckl@usc.edu   requicha@usc.edu

Abstract—**Self-assembly is expected to become a dominant fabrication technique for the nanodevices and systems of the future. Traditional, or passive, self-assembly techniques have great difficulty in producing the asymmetric structures needed by the applications. This paper discusses self-assembly methods that use active assembly agents (robots). It shows that swarms of such robots that communicate only by very simple messages can be programmed to form either wholly or partially specified structures, with the construction process possibly involving sacrificial components or scaffolds. The assembly agents have small memory and communication requirements, and interact only when they are in contact. They are good models for future nanorobots, which are likely to communicate chemically.**

Keywords-nanorobotics; distributed robotics; reconfigurable robotics; swarm robotics; intelligent self-assembly; nanofabrication; state-space reduction; partially specified structures; assembly from primitive shapes; sacrificial structures

I. Introduction

Nanotechnology is widely recognized as a crucial technology for the 21st century. However, the fabrication of structures at the nanoscale (1-100 nm) remains a difficult problem. Most of the nanostructures and nanodevices built until now have been assembled by using nanomanipulation with Scanning Probe Microscopes (SPMs), or fabricated by electron-beam or SPM lithography [1]. All of these processes are inherently sequential and inappropriate for mass production. SPM methods may be parallelized by using multi-tip arrays instead of single tips [2], but parallel SPM operations are still slow for industrial purposes.

Complex systems are built in nature by self-assembly, a process in which components autonomously assemble themselves. For example, many life processes involve the construction of biomolecules from other molecules that recognize each other when they meet under thermal agitation. Assembly of larger components under surface tension is an interesting example of an artificial version of self-assembly [3]. The known examples of self-assembly rely on the environment to position the various components. These are *passive*, and are capable only of recognizing and attaching themselves to their mating components (thereby producing a configuration with lower energy).

Self-assembly is inherently parallel, and therefore suitable (in principle) for the mass production of nanodevices and systems. However, the artificial structures produced by self-assembly until now tend to be symmetric, while most applications (e.g., nanoelectronics) require asymmetric systems. For example, a typical self-assembled monolayer covers uniformly a given surface. In addition, the components of passive self-assembled systems are "programmed in hardware". In other words, the components themselves must be built with the right connecting sites for the desired system to be formed. In spite of very interesting work on the theory of self-assembly by Len Adleman's group at USC [4, 5] and by others, it is still very difficult to design a set of components that produces a specified structure by self-assembly. To construct asymmetric structures, one might try to assemble a set of components on a pre-patterned surface. Unfortunately, this reduces the problem of constructing a nanostructure to that of patterning a surface at the nanoscale, which is not much simpler.

The research reported in this paper is based on a simple idea: why not use self-assembling components that are active, i.e., not only they sense when they meet others, but also they propel themselves and make decisions? In other words, why not use robots? These robots (or a subset of them) move autonomously into a desired configuration and *become the structure themselves*. Active self-assembling components are programmable in the usual sense, which is a major advantage over their passive counterparts. This is an idea whose time has come, at least at USC, where three groups hit upon it largely independently: the distributed robotics group, under Maja Matarić and Gaurav Sukhatme [6], the reconfigurable robotics group under Wei-Min Shen and Peter Will [7], and the Laboratory for Molecular Robotics (LMR) under Ari Requicha, Bruce Koel and Mark Thompson.

Additional motivation for this work comes from our desire to understand how to program swarms of nanorobots, and the capabilities and limitations of such systems. Nanorobots of the future are expected to have limited capabilities. An individual nanorobot, because of its minute size, will not be able to do much on its own, but complex behaviors may be achievable by the coordinated actions of many such robots. Nanorobots are most likely to be able to communicate only by using chemical signals, which is what their counterparts in nature normally do. Chemical communication requires *contact* between robots (or between robots and molecules secreted by other robots). This is a significant limitation of nanorobot swarms when compared to other systems such as current mobile robots, which can gather information about objects at a distance, without touching them. Can interesting behaviors still be achieved under such

The research reported in this paper was supported in part by the NSF under grants IIS-99-87977, EIA-01-21141 and DMI-02-09678, and Cooperative Agreement CCR-01-20778.

limitations? How? These are the primary issues addressed in the remainder of the paper.

II. Related Work

The concept of active self-assembly was introduced by our USC colleagues in [6] and [7]. Previous work is well discussed in [6] where the term "Intelligent Self-Assembly" as used instead of "Active Self-Assembly". Here we will provide a brief summary.

Jones and Matarić [6] discussed a particular self-assembly technique that is applicable to any fully specified and fully connected structure in the plane. This technique places a state machine in each of the Assembly Agents. The Assembly Agents bind to one another when their states are compatible according to the transition rules of the state machine, and then transition to new states. The transition rules are automatically generated by a compiler which takes the desired planar structure as its input. The problem with this approach, from a nanotechnology perspective, is that it requires at least one state and associated transition rules in its state machine for every position of an Assembly Agent in the structure. For a complex structure, this results in an unacceptably large memory requirement being placed on the Assembly Agents.

The present paper owes much to the concept of hormone-based control as described for example in [8] and [9]. The "hormones" presented in these papers are messages which trigger different actions in different places. Although these papers discuss self-reconfigurable robotics rather than self-assembly, the idea of hormone messages is very applicable to the domain of Active Self-Assembly. Wei-Min Shen, Peter Will and Berok Khoshnevis have applied hormone-based control to self-assembly, but in the context of spaceborne operation [7] where communication between non-neighbor Assembly Agents is practical, and complex Assembly Agents are acceptable. The work presented in this paper makes extensive use of hormones, but does so in a context where Assembly Agents must be simple and long-range communication is unavailable.

There is also related work in the study of biological swarm intelligence, with perhaps the nearest analogue to the present paper being the SWARM-BOT of [10]. In many ways, the descriptions of pattern-forming behavior in insects and Active Self-Assembly are different ways of looking at the same phenomenon. See [11, 12] for more on this perspective.

The general theory of self-assembly has seen a lot of work, for example [4, 5], as has assembly by teams of robots [13].

III. Assembly Agent Requirements

For the class of algorithms described here to operate, each Assembly Agent must have the following capabilities:

- Store and execute a finite state machine
- Communicate the current state to adjacent Assembly Agents in the structure
- Accept a message from an adjacent Assembly Agent and forward it to N adjacent Assembly Agents
- Maintain a hop counter on messages
- Physically attach to other Assembly Agents
- Perform periodic actions
- Perform a random walk

The examples provided in this paper also make use of a further ability: the ability to move along the surface of the structure. This ability is not critical to the function of the system, but it can allow for faster construction by turning the whole surface of the structure into a collector that funnels Assembly Agents toward areas of active construction.

IV. Terminology

Assembly Agent: a single entity that takes part in the construction of a structure by actively becoming part of that structure.

Swarm: A collection of Assembly Agents. Swarms are often considered as single objects.

Seed: An Assembly Agent that is triggering the growth of a structure. Seeds can either be injected into the swarm a priori, or an Assembly Agent can become a seed when it senses a particular property of the environment.

State Memory: The part of an Assembly Agent's memory that is time dependent.

State Space: The set of meaningful values that can be stored in state memory.

V. Communication

When Assembly Agents have the power to communicate with their neighbors, the swarm gains significant new capabilities.

The current state of an Assembly Agent is always communicated to its connected neighbors. In addition, Assembly Agents can send messages. Messages are defined as nonstate information that is passed from one neighbor to another. The messages used in this paper are elements of a predefined subset of the counting numbers, but messages containing data fields are conceivable.

Messages are distinct from communicated state in that individual Assembly Agents do not send the same message in all directions. Assembly Agents usually transmit messages to the neighbor on the opposite side from where the message originated. This provides the messages with a property of directionality that is necessary for the techniques in this paper to work.

VI. Advantages of Communication

By allowing the Assembly Agents to locally pass messages, we enable a number of interesting capabilities. We have used communication to reduce the state memory requirements of Assembly Agents, to enable the assembly of partially specified structures, and to construct structures by building and disassembling a series of primitive shapes in a known sequence.

A. Reducing the State Space of Assembly Agents

We reduce the state space of Assembly Agents by dividing the state space into a number of non-overlapping sets of states and requiring the Assembly Agents to remember only in which set is their current state. From this point onward, when we say "state" we mean a unique value in state memory, which in our technique usually represents a set of physical states. This is a gain relative to Jones' and Matarić's work, since it allows several positions in the structure to be represented by a single value in state memory, as opposed to having several values in state memory for each position in the structure.

To illustrate this, consider the assembly of a square from Assembly Agents that are themselves square. While this task could certainly be performed by assigning several states to each position in the final square (for example, states that mean "waiting for a connection on the north side" or "waiting for a connection on the east side"), we would like to reduce the amount of state memory required.

We can reduce the state space required to assemble a square by noting that a square can be recursively defined as what you get when you take a smaller square and perform a Minkowski dilation on two sides. In other words, a square consists of a nested set of L shapes and a single atomic square. (See Fig. 1)

Since our Assembly Agents are themselves squares, we only need to provide for the L shapes in order to be able to create a square. An L can in turn be broken up into two rows of Assembly Agents, so the question becomes "how many states do we need to represent a row of N assembly agents?"

At one end, we have a seed, which is periodically sending a message out of one face. We can interpret this message as meaning "build a row." Other Assembly Agents randomly attach to the structure. When one of these agents receives the "build a row" message, it transitions to a new state, which we can call "in the row." When an Assembly Agent that is "in the row" receives a "build a row" message with a hop counter equal to the desired length of the row (a parameter determined at the time of state machine creation), that agent can conclude that the row is complete. Thus, building a row requires only the seed, random movement and "in a row" states, though more can be added for convenience or to improve various features of the construction.

The end result is that we can build a square using only three states. Similar reduction techniques can be used for other shapes (approximate triangles, rectangles, etc). These shapes can then be used as primitives for assembling more complicated shapes. We have built up a small library of these primitives.

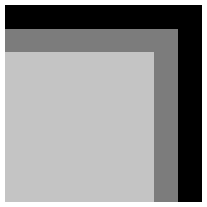
Figure 1. Recursivly defined square

B. Assembling Partially Specified Structures

In section VI.A, we had to add complexity to the Assembly Agents in order to fully specify the shape. In particular, the messages we sent had to have a hop counter attached to them, so that we could tell the Assembly Agents to build lines of a specific length. If we stop using the hop counter, we lose the ability to say "build a line of length N" but we gain the ability to say "build a line, of any length." We then have a state machine that specifies not a single shape, but a set of shapes.

This concept allows us to specify shapes that have connectivity requirements instead of positional requirements. In other words, message propagation without hop counting can specify shapes in terms of what is connected to what, instead of what is placed where.

For an example of the utility if this idea, consider the task of self-assembling a road between two regions. We don't care about what shape the road takes, except that it is necessary that the road intersect both of the desired end regions and it cannot intersect any obstacles. It is in fact impossible for us to specify the precise shape the road is to take unless we know the precise locations of the end regions relative to each other. By employing partially specified self-assembly, we can describe the road well enough for the Assembly Agents to create it even without precise positional information.

After an Assembly Agent has become a seed, it periodically sends out "direction" messages to its neighbors, which forward the messages further. This directionality is used to move Assembly Agents that have attached to the structure but which do not yet have a final location away from the anchored end of the path to one of the construction ends.

Once these free Assembly Agents reach an end of the structure, they simply sit there until they receive a "direction" message, at which point they transition to a state meaning "finalized position." If an Assembly Agent in the "finalized position" state determines that it is in contact with the second endpoint for the path, it transitions into a state meaning "on the path." If an Assembly Agent in the "finalized position" state sends a "direction" message to an "on the path" Assembly Agent, it too transitions to "on the path."

In this way a single unbranching path is traced backwards through the tree from destination to start. Because the Assembly Agents are unable to penetrate obstacles, the path is guaranteed not to intersect any obstacles. Finally, a message is sent out which causes those agents not "on the path" to revert to their original random walk state. When the process is over, a line of Assembly Agents which forms a path remains. Figs. 2 (A, B & C) are sequential snapshots of this process in operation.

C. Sequential Construction and Sacrificial Structures

Of particular interest in the road building scenario is that the self-assembly process has two stages: a first stage in which something that is a superset of the desired shape is built, and a second stage in which messages passed through the structure

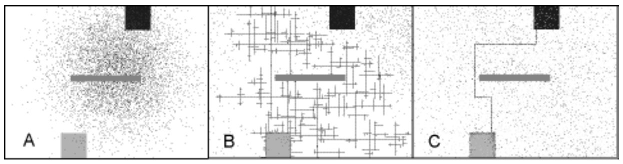

Figure 2. Road building sequence

cause unwanted pieces to remove themselves from the structure.

This highlights two nice features of communication in self-assembly. One of these features is that a global sequence can be imposed on the construction by means of messages passed from neighbor to neighbor. The other nice feature is that this self-assembly technique supports the creation and subsequent removal of sacrificial structures.

Imposing a global ordering on the construction of a structure can be very important, because the order of construction can affect whether or not is it possible to complete a structure. For example, consider assembling the shape represented by the black area in Fig. 3 in a two dimensional space. If the assembly technique does not provide ordering, it is possible for the outer shell to be completed first, leaving too few Assembly Agents inside to complete the rest of the structure. There are many analogous structures in three dimensions.

Communication is not unique in its ability to support global ordering ([6] supports it as well), but by using communication is it possible to consider global ordering on the more coarse (and thus less memory intensive) level of sequencing the construction of primitive shapes, instead of on the level of sequencing individual Assembly Agent binding events. Each primitive shape generates a message when it is complete, which can be used to command one of the Assembly Agents taking part in the structure at a uniquely identifiable location to become a seed for the next primitive shape to be added.

On a square, the corners are such uniquely identifiable locations, because the agents in those positions can recognize that fact from local topology and message traffic.

Using communication allows us to treat each of the primitive shapes we have thus far constructed as a modular component with attachment points for other components. The process of creating a state machine to assemble a structure made from these primitives is only a matter of stringing together the state machines, one after the other, by turning Assembly Agents into seeds when they receive the appropriate "shape completed" message.

Related to the ability to perform global ordering is the ability to generate and then remove sacrificial structures. Sacrificial structures are structural elements that are needed during some phase of the construction but which are not intended to be part of the final structure. Scaffolds and temporary supports are examples of sacrificial structures. Sacrificial structures depend on the ability to globally order events, since a sacrificial structure that is not present when it is needed, or which remains present after it is no longer needed, results in a failure to construct the desired structure.

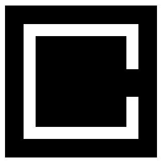
Figure 3. Problematic structure

The ability to build sacrificial structures expands the range of shapes that can be self-assembled. With sacrificial structures, it is possible to self-assemble objects that are not fully connected. For example, it is possible to construct the shape represented by the black area in Fig. 4 by using a sacrificial structural element to connect the two parts. Once both parts are built, the sacrificial element disassembles itself, leaving the desired structure in place. If they are not fully enclosed, the Assembly Agents that made up the sacrificial structure can wander away and take part in another construction project.

VII. Validation

Several validations have been performed on the concepts presented in this paper. We have implemented state machines which perform partially specified road building in both discrete and continuous simulated environments, we have written state machines which result in the construction of primitive shapes, and we have connected the state machines representing primitive shapes together to form state machines that construct more complicated structures.

A. *Building roads*

We have created and tested a state machine that results in the creation of partially specified roads, as described earlier. This state machine has 16 states, and its transition table can be stored in 158 bytes.

A state machine designed in the paradigm described in [6] to generate an optimal road in our test environment would have approximately twice as many state bits and twice as many bytes in the transition table. These factors increase as the distance between the end points increases, and the state machine is valid only when the end points have the precise relative positions that the state machine was designed for.

Our state machine is smaller, has memory requirements that are independent of the size of the built structure, and is robust in that it builds a correct structure regardless of the relative positions of the endpoints.

We have not evaluated the reliability of the system, but present as anecdotal evidence that we have never seen it fail except in cases where it was starved for Assembly Agents.

B. *Primitive Shapes*

We have created and tested state machines that construct the primitive shapes square, rectangle, and approximate triangle. The state and transition table memory requirements of each of these state machines are independent of the size of the built shape. The state machines can be parameterized to construct shapes of any size.

Each of these primitive shapes has several locations that can become seeds when the construction of the primitive is finished. A portion of a state machine for constructing a primitive shape is detailed in the appendix.

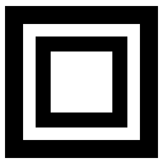
Figure 4. Disconnected structure

1) *Squares*

The state machine that creates squares causes the structure to grow from a seed in one corner of the square. In Fig. 5, the seed is in the

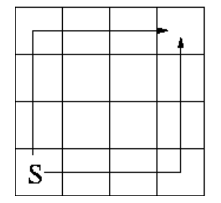

Figure 5. Square completion criterion

square marked "S." The seed grows two perpendicular rows of Assembly Agents (in the up and right directions in Fig. 5), whose length is controlled by counting how many hops a message takes when it travels along them.

Once each row is of the right length, its members change to a new state in which they forward the seed's "grow row" messages, with a modified hop count, along the axis perpendicular to their original direction of travel. This recursively creates a square by creating nested perpendicular rows.

The square is known to be completed when messages can propagate along the two paths represented by arrows in Fig. 5. When an Assembly Agent with two neighbors receives "check for completion" messages from both neighbors, it knows that the square is finished and that it may generate the "shape completed" message.

2) Rectangles

Rectangles grow outward from a seed at one end. The seed generates a single row of Assembly Agents aligned with the long axis of the desired rectangle. Once that row is in place, layers are added to it on each side until the rectangle is of the appropriate width. Width is measured by sending out messages perpendicular to the long axis of the rectangle from each Assembly agent in the center row, and checking the hop count on those messages. Each layer starts at the Seed end, so the last two Assembly Agents to join the structure will be in the two corners of the rectangle that are furthest from the seed.

The rectangle is known to be completed when a message can be sent from the seed along the length of the rectangle, then sent outward in both directions to the corners and back in to the middle, as shown in Fig. 6. When an Assembly Agent receives both the version of the completion message sent "up" and the version sent "down," it knows that the rectangle is complete and that it may send the "shape completed" message.

3) Right Triangles

Triangles grow from a seed in their right-angled corner. Like a square, a triangle grows by creating nested L shapes, but in the case of the triangle each arm of the L is two Assembly Agents shorter than the one it is nested within, instead of being one shorter as in the case of the square. Message propagation is handled in the same way as with a square; the difference is in how the hop counter is modified when its direction is changed.

Determining that a triangle is complete requires a bit more cleverness than does determining that a square or rectangle is complete, because its hypotenuse-edge is an even number of message hops away from the seed when the length of the other edges is odd, and an odd number of hops away when the edge length is even. This makes finding a single

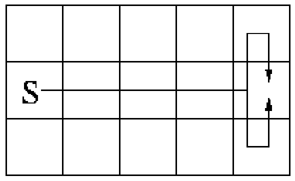

Figure 6. Rectangle completion criterion

pattern of message flow that uniquely identifies a completed triangle harder. One solution is to make the hypotenuse edge of the triangle reflective to completion check messages. Then, a triangle can be known to be finished when two completion check messages with hop counters either a) equal to the edge length or b) equal to the edge length minus 1 are received by an Assembly Agent. At this point, the agent can declare the triangle finished and send the "shape completed" message for the triangle. These criteria are illustrated in Fig. 7.

4) Compound shapes

We have aggregated state machines for primitive shapes into a state machine that generates a compound shape. This required the addition of a single extra transition to the system per joint between primitives. These extra transitions are triggered by the "shape completed" messages, and result in an Assembly Agent that is located at a uniquely identifiable location within the structure becoming a seed for a new shape.

The memory requirements of these aggregate state machines are equal to the sum of the requirements of the primitives that make them up, plus one transition table entry per primitive. For example, the memory required for a square+triangle compound shape is equal to the memory required for the square and the memory required for the triangle, plus a single extra transition entry to turn one of the members of the square into a seed for the triangle.

VIII. Conclusions and Future Work

Swarms of very simple robots can be programmed ro self-assemble into interesting structures such as wires between given shapes, primitive shapes including rectangles and triangles, and compositions of primitive shapes. The robots have small memories, simple execution mechanisms, and communicate with very few messages and only when they are in physical contact. These are characteristics that chemically-communicating nanorobots of the future are expected to have. Therefore our work shows that it is likely that complex, asymmetric nanostructures may be built by active self-assembly, which is an inherently parallel process well suited to mass production.

There are many avenues for future research. The most exciting would be to demonstrate active self-assembly with physical nanorobots, but this must wait until the state of the art in nanorobotics has advanced significantly. Extensions that are possible now include 3-D self-assembly (in simulation), and CAD tools that allow a user to create structures by defining primitive shapes, positioning and scaling them, linking them to produce composite objects, and in turn linking these together, much like the mechanical CAD systems of today. The finite state machines and other information required to self-assemble the structures in the proposed CAD tool would be generated automatically by the tool.

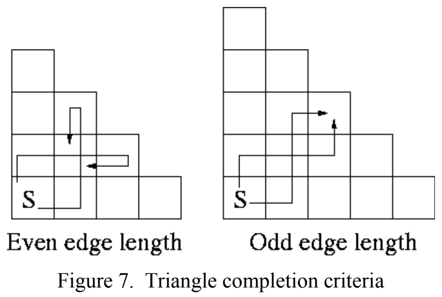

Figure 7. Triangle completion criteria

```
TRANS WANDER WANDER    TIMER                      REORIENT
TRANS WANDER PLACED    TOUCH+SOUTH=SEED           GRAB/SOUTH
TRANS WANDER PLACED    TOUCH+WEST=SEED            GRAB/WEST
TRANS WANDER MATERIAL  TOUCH+SOUTH=PLACED  GRAB/SOUTH
TRANS WANDER MATERIAL  TOUCH+WEST=PLACED          GRAB/WEST
TRANS WANDER MATERIAL  TOUCH+NORTH=PLACED  GRAB/NORTH
TRANS WANDER MATERIAL  TOUCH+EAST=PLACED          GRAB/EAST
TRANS WANDER MATERIAL  TOUCH+SOUTH=FINAL          GRAB/SOUTH
TRANS WANDER MATERIAL  TOUCH+WEST=FINAL           GRAB/WEST
TRANS WANDER WANDER    ELSE                       MOVE
```

Figure A.1 Excerpt of state machine code for square

References

[1] A. A. G. Requicha, "Nanorobots, NEMS and Nanoassembly", *Proc. IEEE, Special Issue on Nanoelectronics and Nanoprocessing*, Vol 91, No. 11, pp. 1922-1933, November 2003

[2] D. J. Arbuckle and A. A. G. Requicha, "Massively Parallel Scanning Probe Nanolithography", *Proc. 3rd. IEEE Intl. Conf. on Nanotechnology*, S. Francisco, CA, Vol. 1, pp. 72-74, August 12-14, 2003.

[3] R. R. A. Syms, E. M. Yeatman, V. M. Bright and G. M. Whitesides, "Surface-tension-powered self-assembly of microstructures – The State of the Art", *J Microelectromechanical Systems*, Vol. 12, No. 4, pp 387-417, August 2003.

[4] Adleman, L., "Toward a mathematical theory of self-assembly," Technical Report 00-722, Department of Computer Science, University of Southern California. 2000.

[5] L. Adleman, Q. Cheng, A. Goel and M. Huang, "Running time and program size for self-assembled squares", *Proc. ACM Symposium on Theory of Computing*, Heraklion, Greece, pp. 740-748, July 6-8 2001

[6] Chris V. Jones and Maja J. Mataric´. "From Local to Global Behavior in Intelligent Self-Assembly". *Proc. IEEE Intl. Conf. on Robotics and Automation*, Taipei, Taiwan, pp. 721-726, Sep 14-19 2003

[7] Wei-Min Shen, Peter Will, Berok Khoshnevis, "Self-Assembly in Space via Self-Reconfigurable Robots", *Proc. IEEE Intl. Conf. on Robotics & Automation*, Taipei, Taiwan, pp. 721-726, Sep 14-19 2003.

[8] Shen, W.-M., Y. Lu and P. Will, "Hormone-based control for self-reconfigurable robots.", *Proc. Intl. Conf. Autonomous Agents*, Barcelona, Spain, pp. 1-8, June 3-7 2000.

[9] B. Salemi, W.-M. Shen and P. Will, "Hormone Controlled Metamorphic Robots", *Proc. IEEE Intl. Conf. on Robotics and Automation*, Seoul, Korea, Vol. 4, pp. 4194-4199, May 21-26 2001.

[10] E. Sahin, T. H. Labella, V. Trianni, J-L Deneubourg, P. Rasse, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, M. Dorigo, "SWARM-BOT: Pattern Formation in a Swarm Of Self-Assembling Mobile Robots" , *Proc. IEEE Intl. Conf. on Systems, Man and Cybernetics*, Hammamet, Tunisia, Vol. 4, Oct. 2002.

[11] E. Bonabeau, M. Dorigo, and G Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford: Oxford University Press, 1999.

[12] E. Bonabeau, G. Theraulaz, E. Arpin and E. Sardet. "The Building Behavior of Lattice Swarms." *Proc. Fourth International Conference on Artificial Life*, MIT Press, Cambridge, MA, pp. 307-312, 1994.

[13] Jens Wawerla, Gaurav Sukhatme, and Maja J. Mataric´. "Collective Construction with Multiple Robots." *Proc. IEEE/RSJ International Conference on Robotics and Intelligent Systems*, Lausanne, Switzerland, Vol. 3, pp. 2696-2701, Oct 2002

Appendix: Example State Machine Code

The code in Fig. A.1 is excerpted from the state machine that controls the construction of our square primitive shape. This particular instance has been parameterized to represent a square with its seed in the "lower left" corner. The complete state machine has 46 transitions between 6 states. The excerpted code represents the transitions from the random walk state, called "wander." The syntax "trans *state1 state2*" means that the line describes a transition from the state called "state1" to the state called "state2".

The state machine transitions between states when there is a transition condition that matches the current state. For example, in the code shown, the syntax "TOUCH+WEST=final" means that the transition should take place when the neighboring Assembly Agent in the direction called 'west' is in the state "final." When the state machine transitions, it may send control signals such as "grab/WEST," which means that the Assembly Agent should grab its neighbor to the west.