Autonomous Discovery and Functional Response to Topology Change in Self-Reconfigurable Robots

Behnam Salemi
Department of Computer Science
USC/Information Sciences Institute
Marina del Rey, California, USA
salemi@isi.edu

Peter Will
Department of Computer Science
USC/Information Sciences Institute
Marina del Rey, California, USA
will @isi.edu

Wei-Min Shen
Department of Computer Science
USC/Information Sciences Institute
Marina del Rey, California, USA
shen @isi.edu

Abstract— **The topology of a self-reconfigurable Robot can change at anytime. This can be as a result of the failure of some modules of the robot, joining new modules to the robot, displacement of some modules from one location to another caused by the self-reconfiguration task or any combination of these cases. Considering that the process of selecting relevant behaviors to accomplish a given task is based on the current topology of the self-reconfigurable robot, modules must be able to detect and respond to any changes to the robot topology. When changes to the topology of the robot are detected, modules can investigate new ways of accomplishing the given task. This paper presents a distributed solution, FEATURE algorithm, to the problem of autonomous discovery and functional response to topology change. The result is experimentally verified and demonstrated on the CONRO self-reconfigurable robots.**

key words: Self-Reconfigurable Robots; Modular Robotics; Distributed Robotics; Intellignet Ditributed Control.

I. INTRODUCTION

A self-reconfigurable system is a special type of complex systems that can autonomously or manually rearrange its software and hardware components and adapt its configuration (such as shape, size, formation, structure, or organization) to accomplish difficult missions in dynamic, uncertain, and unanticipated environments. A self-reconfigurable system is typically made from a network of homogeneous or heterogeneous *reconfigurable modules* (or *agents*) that can autonomously change their physical or logical connections and rearrange their configurations. Self-reconfigurable robots [1-4] are examples of such systems that consist of many autonomous modules that have sensors, actuators, and computational resources. These modules are physically connected to each other in the form of a configuration network. Since the topology of the network may change from time to time, the controller of the robot must be distributed and decentralized to avoid single-point failures, communication bottleneck among modules and to accomplish the given task.

These modules must have some essential capabilities in order to accomplish complex tasks in dynamic and uncertain environments. The capabilities that we addressed in our previous work were: (1) distributed task negotiation [5] – allowing modules to agree on a global task to accomplish, (2) distributed behavior collaboration [6] – allowing modules to "translate" a global task into local behaviors of modules; (3) synchronization – allowing modules to perform local behaviors in a coordinated and timely fashion; In these previous works we assumed the network of modules can have any initial topology but it remains unchanged during accomplishing a task.

Here we relax this assumption and allow the topology of the network of modules to change at any time including in the middle of accomplishing a task. Our solution is a distributed approach inspired by the concept of hormones [9] and is based on 1) giving the ability of detecting local changes in the topology of the network to the modules and 2) letting them coordinate their activities based on the new topology such that the given global task is accomplished.

The related approaches for solving similar problems include Role-based Control [7] and stochastic approaches for self-repair such as [8]. The first approach is based on detecting the changes in local relationships of the immediate neighboring modules. This approach requires less computational power. However, it is an open-loop approach and might not be very flexible for accomplishing complex tasks. The second approach has been applied to lattice-based self-reconfigurable robot which their configuration space is much smaller than that of the chain-type self-reconfigurable robots such as CONRO.

This paper is organized as follows: Section 2 defines the problem of autonomous discovery and functional response to topology changes and uses the CONRO self-reconfigurable robot as an illustrative example; Section 3 presents the idea of probing; Section 4 presents the FEATURE algorithm and reviews its sub-algorithms; Section 5 gives an example of applying the FEATURE algorithm to real CONRO modules, and Section 6 concludes the paper and specify possible future research directions.

II. AUTONOMOUS DISCOVERY AND FUNCTIONAL RESPONSE TO TOPOLOGY CHANGE

The problem of autonomous discovery and functional response to topology change can be defined as follows: Given a global task and a set of self-reconfigurable modules, coordinating global responses to local changes in the topology of the network of modules in order to produce the desired global effects. Local changes include adding or deleting new modules or communication links to/from the network of modules.

This problem is very challenging due to several reasons: relationships among modules may change anytime; changes in configuration is locally detectable but a coordinated global response is required; the number of modules in the robot is not known; modules have no unique global identifiers or addresses; modules do not know the global configuration in advance, and can only communicate with their immediate neighbors. Generally, accomplishing a given global task is dependent on the topology of the network of modules [6]. Modules can accomplish a global task by selecting correct *behaviors* in coordination with other modules and performing them synchronously. As a result, changes in the topology will directly influence the selected behaviors and the order they should be performed.

Here, the nodes and links represent the modules and the communication links between them, respectively. Note that the size of the network is dynamic and unknown to the individual nodes; also the index numbers are only used for defining the problem and not used in the solution. Under these circumstances, a satisfactory solution to this problem must be distributed. Modules must detect local changes in the configuration graph and inform the rest of the modules in order to let them change their internal states.

To illustrate the problem, we use the CONRO self-reconfigurable robot as an example. CONRO is a chain-type self-reconfigurable robot developed at USC/ISI (http://www.isi.edu/robots). Figure 1 shows the schematic views of CONRO module and a six-legged CONRO robot. Each CONRO module is autonomous and contains two batteries, one STAMP II-SX micro-controller, two servomotors, and four docking connectors for connecting with other modules. Each connector has a pair of infrared transmitters/receivers, called *outgoing-Links* and *incoming-Links*, to support communication as well as docking guidance.

Each module has a set of open I/O ports so that various sensors for tilt, touch, acceleration, and miniature vision, can be installed dynamically. Each module has two Degrees Of Freedom: DOF1 for pitch (about 0-130° up and down) and DOF2 for yaw (about 0-130° left and right). The range of yaw and pitch of a module is divided to 255 steps. The internal state of each module includes the current values of the yaw, pitch of a module, and the number of the sent and received messages. The modules' *actions* consist of moving the two degrees of freedom to one of the 255 positions, attaching to or detaching from other modules, or sending messages to the communication links through the IR senders.

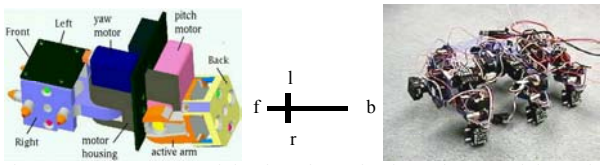


Figure 1: A CONRO module, the schematic view of one module, and a hexapod (insect) configuration with 9 modules.

Modules can connect to each other by their docking connectors. Connected docking connectors are called *active* connectors. Docking connectors are located at either end of each module. At one end, labeled *back* (*b* for short), there is a female connector, consisting of two holes for accepting another module's docking pins. At the other end, three male connectors of two pins each are located on three sides of the module, labeled *left* (*l*), *right* (*r*) and *front* (*f*).

III. PROBING AND COMMUNICATION

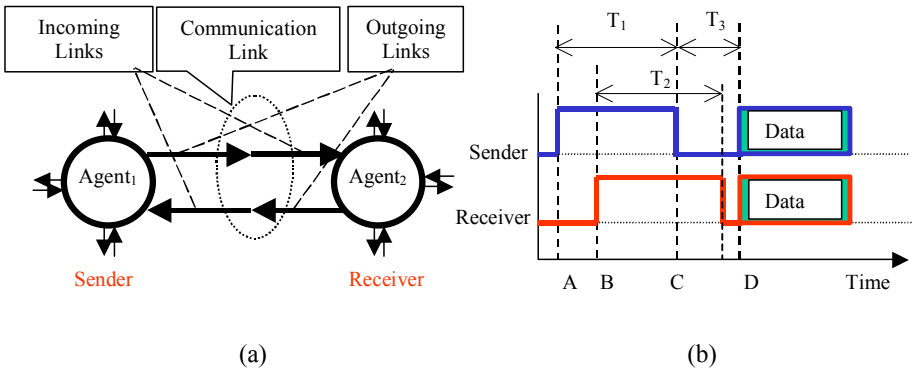
For each module, the first step in responding to the network topology changes consists of detecting local changes. Instances of local changes are: 1) A new module connects to one of the modules in the network, 2) An existing module disconnects from all other modules in the network, 3) An existing module establishes a new connection with another module in the network, and 4) A module disconnects some of its connectors from other modules in the network. In situations 1 and 2 the number of nodes and links and in situations 3 and 4 the number of links in the configuration network changes.

Based on what we said above, changes in the number of active links of a module is equivalent to the changes in the topology of the network of modules Therefore, modules can detect the changes in the topology of configuration network by monitoring the changes in the number of their local active links. Modules perform this task by periodically monitoring their active docking connectors for disconnections and/or inactive docking connectors for new connections. We will call this action *probing*. In order to detect all the above-mentioned cases of the topology change efficiently, we propose two types of probing: 1) Probing by communicating and 2) Probing by sending *probing signals*.

A. Probing by Communiation

The communication protocols between modules that use handshaking signals for sending and/or receiving messages can be used for probing the active connection links between modules. A successful communication action over an active connector shows that the connection is still active. Similarly, an unsuccessful communication action shows the disconnection of an already active connector.

Figure 2 shows an asynchronous communication protocol that was implemented in CONRO modules. $Agent_1$ is the sender and $agent_2$ is the receiver. What follows is a brief description of the handshaking sequence of this protocol:



(a) (b)

Figure 2: (a) The communication link between two agents. (b) The asynchronous communication protocol between two agents

1) The sender requests to send a message by making its outgoing link 'High', point A. and then frequently checks its incoming link for receiving a 'High' signal.

2) The receiver responds by making its outgoing link 'High', point B, and waits.

3) After receiving the 'High', sender makes its outgoing link 'Low', point C, and starts sending the message (Data) after some delay, point D. This short delay is called *preparation time* (T_3), which gives a chance to the receiver to prepare for receiving Data. Data is communicated using RS232 asynchronous communication protocol. T_1 and T_2 are sender's and receiver's timeouts, respectively.

This simple handshaking protocol successfully completes if and only if *both* modules actively respond. Therefore a successful communication verifies an active link between two modules. Similarly, an unsuccessful communication confirms that the receiving module is not present and the link is inactive.

Probing by communication, however, is not an efficient way of probing the inactive connectors since modules have to wait for every sent message to an inactive docking connector to timeout. Also, in situations where two modules do not communicate for longer than a pre-specified monitoring period, the communication based approaches will not work. In such situations we will use a different probing method based on sending *probing signals*.

B. *Probing Signals*

Probing signal are narrow pulses that are periodically sent to inactive connections or active connections if no communication is going to occur on them for a long periods. The width of probing signals is narrow enough such that they can be distinguished and filtered from the communication protocol signals. Figure 3 compares the probing and the communication protocol signals widths.

Figure 4 shows the block diagram of a module's connection link. The 'Probing Signal Filter' on the incoming link separates the probing signals from the communication signals. On the outgoing link, the communication and probing signals are merged on a single output line.

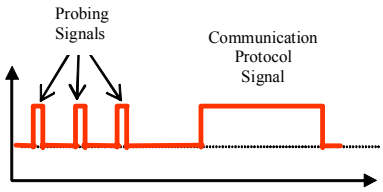


Figure 3: Probing and Communication protocol signals

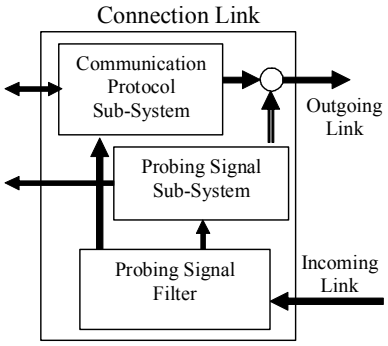


Figure 4: Joining and separation of the probing and communication signals

C. *Probing Algorithm*

Figure 5 describes the probing algorithm for detecting local changes in the topology of the network. This algorithm consists of two procedures. The first procedure, **GenerateProbe**, is called for generating probing signals on the inactive connectors or the active connectors that have not communicated for longer than 'monitoring period'.

The second procedure, **CheckTopology**, is called for detecting changes in local topology of the network based on the received probing signals or the recent communicated

messages. This procedure returns a true value if the topology has been changed.

```
when GenerateProbe () do
  for each C ∈ Connectors do
    if (C = Inactive) or (NoComm (C, Period) = true) do
      send ProbingSignal to C;
end do; end do; end do;

when CheckTopology () do
  TempLocalTopology = CurrentLocalTopology;
  TopologyChanged = false;
  for each C ∈ Connectors do          //reset
    CurrentLocalTopology (C) = Inactive; end do;
  for each C ∈ Connectors do
    if (CommOccurred (C) = true) or
      (Probe Signal Received (C) = true) do
      CurrentLocalTopology (C) = active;
  end do; end do;
  if (TempLocalTopology ≠ CurrentLocalTopology) do
    TopologyChanged = true;
  end do;
  return TopologyChanged;
end do;
```

Figure 5: The probing Algorithm

IV. FUNCTIONAL RESPONSE TO TOPOLOGY CHANGE USING PROBING

Our solution for the functional response to topology change problem in self-reconfigurable robots relies on our previous work on distributed control for self-reconfigurable robots. Specifically, the 'distributed task negotiation' and 'distributed behavior collaboration' problems. In this section we will briefly describe these problems and their proposed solutions. Then we will present our probing-based algorithm for solving the functional response to the topology change problem.

A. Distributed Task Negotiation

Distributed Task Negotiation is a process by which modules in a self-reconfigurable robot can negotiate and select a single coherent task among many different and even conflicting choices.

Formally, a distributed task negotiation problem consists of a tuple (*P, L, T, S*), where *P* is a list of nodes, p_i, such that $i \in \{1,..., N\}$; *L* is a list of communication links, l_{jk}, such that $j,k \in \{1,..., N\}$; *T* is a list of tasks, t_m, such that $1 \le m \le N$, and *S* is a set of task selection functions, $S_i: (T') \rightarrow t_i$, such that $i \in \{1,...,N\}$ and $T' \subset T$. Each node has a task selection function that can select a single task from a set of given tasks. A distributed task negotiation problem is solved when all nodes have selected the same task from *T*, called t^*, and have been notified that the negotiation process is terminated. Note that the index numbers assigned to *P* are only used for defining the problem and not used in the negotiation process. In addition, the size of the network is unknown to the individual nodes.

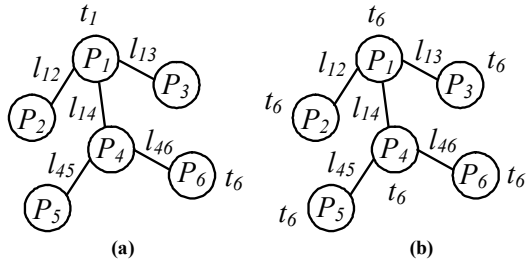


Figure 6: An example of a distributed task negotiation problem. **a**) Initially p_1 and p_6 initiated two tasks (t_1, t_6). **b**) A solution, when all agents have selected $t^* = t_6$.

To illustrate the above definition, consider the example in Figure 6(a), where $P= \{p_1, p_2, p_3, p_4, p_5, p_6\}$, $L = \{l_{12}, l_{14}, l_{13}, l_{45}, l_{46}\}$, $T = \{t_1, t_6\}$, and *S* is a selection function that prefers tasks with greater indexes and shared by all nodes. Initially, node p_1 and p_6 have initiated two tasks, t_1 and t_6, respectively, and the rest of the nodes are waiting to receive tasks. Figure 6(b) depicts a solution for the given problem where all nodes agreed on task t_6.

In [5] we presented the DISTINCT algorithm as a solution for the distributed task negotiation problem. The main idea is that all modules work together to build global spanning trees and each tree is associated with a task. Initially, all modules that have their own competing tasks start building their own trees, but as they exchange messages for tree building, most modules will give up their "root" status and participate in building trees for other tasks. In this process, modules report their status to their parent module in the tree that they participate, and the module that does not have parent but received reports from all its children is the root for the entire network of modules. When this happens, this root module can conclude that the negotiation process has succeeded and all modules in the tree have agreed on the same task. An embedded synchronization algorithm detected the termination of the negotiation process.

Important characteristics of this solution are: 1) modules do not require having unique Ids. This is an important requirement for a truly homogenous system; 2) ensures that all nodes will select the same task coherently; regardless of the number of competing tasks initiated in the network; and more importantly 3) it is not dependent on the topology of the network of modules.

B. Distributed Behavior Collaboration

The problem of distributed behavior collaboration can be defined as follows: Given a global *task* and a *group behavior*, selecting a correct set of local behaviors at each module and coordinating the performance of the selected behaviors to produce the desired global effects.

Formally, the problem of distributed behavior collaboration is a tuple (*P, Q, C, A, B, t, GB*), where *P* is a list of nodes p_i; *Q* is the list of the internal state, q_i, associated with each node p_i, such that $i \in \{1,..., N\}$; *C* is a list of labeled physical or logical links, c_j, such that $j \in \{locally\ unique\ labels\}$; *A* is a set of actions a_s that a node

can execute, $s \in \{1,..., S\}$; B is a set of behaviors in the form of $b_m = (a_x, a_y a_z,\ldots)$, such that $m \in \{1,..., M\}$; t is the global task given to all nodes, and GB is the desired group behavior. A distributed behavior selection problem is solved if and only if $\beta = GB$, where $\beta = \beta + b_{pi}$, $i \in \{1,..., N\}$; meaning that the ordered sequence of the selected behaviors of all nodes over time is equal to the desired group behavior. The *configuration graph* of the network of modules is a graph consists of P nodes and C edges.

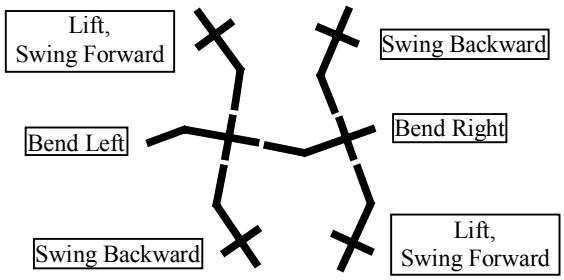


Figure 7: Selected behaviors of modules in a quadruped configuration to accomplish a locomotion task.

Figure 7 shows a snapshot of the selected behaviors in a quadruped shape self-reconfigurable robot performing a locomotion task. The 'Lift, Swing Forward' is an example of a behavior that consists of two actions, 'Lift' and 'Swing Forward', which are simultaneously performed by the pitch and yaw joints of the module, respectively.

In [6] we presented D-BEST algorithm as a distributed solution for the problem of distributed behavior collaboration. It is a new approach based on the concept of "path" to represent extended neighborhood topology at the connector level. This allows modules to select appropriate local behaviors for a given global task for a given configuration, based on the relative location of the modules in the configuration graph. D-BEST algorithm utilizes the modules' *extended type* for behavior selection. The extended type contains information about the way a module is connected to its extended neighbors

A group behavior is represented in the form of a set of *decision rules*. These rules are mappings from nodes internal states, and their selected task to behaviors, $(Q,t) \rightarrow B$, and represent the relevant behaviors for accomplishing a given task. The required number of communicated messages of this algorithm is of the order $O(N)$ (where N is the number of modules). D-BEST algorithm uses the same synchronization algorithm that is used in DISTINCT algorithm for synchronizing the order of execution of the performed behaviors.

C. The FEATURE Algorithm

In this section, we will describe the FEATURE algorithm that brings all the above-mentioned pieces together and solves the problem of functional response to topology change in self-reconfigurable robots. This will be the algorithm that will run on all modules of the self-reconfigurable robot to ensure the homogeneity of all modules. Figure 8 depicts this algorithm.

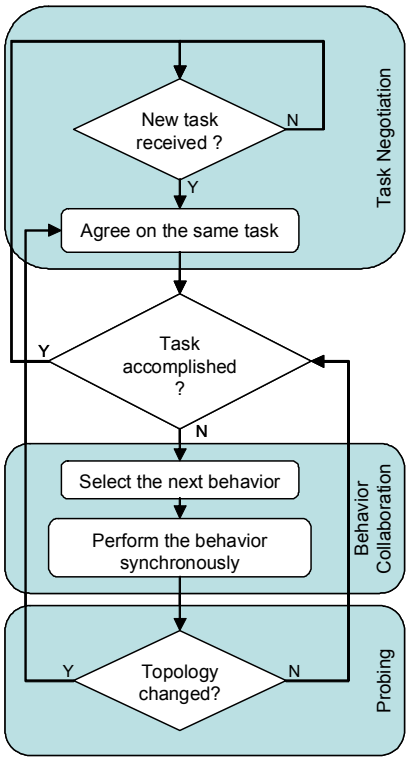


Figure 8: The FEATURE algorithm

Initially all modules will wait to receive a new task. The new task can be initiated by an outside controller or a sensor on one of the modules. Receiving a new task initiates a distributed negotiation process among all modules in the robot. This is necessary to ensure that 1) all modules know what task they accomplishing and 2) in cases where multiple modules have initiated more than one task, all modules will agree on accomplishing the same task that has the highest priority. The above-mentioned process is controlled by the DISTINCT, task negotiation algorithm shown on top of the figure 8.

If the selected task is not already accomplished, modules will generate a set of relevant behaviors. The relevant behaviors are represented by a set of decision rules that have been uploaded in all modules in advance. The execution of the selected behaviors will be coordinated by an embedded distributed synchronization mechanism. The above-mentioned process is controlled by the D-BEST, behavior collaboration algorithm shown in the middle of the figure 8.

The topology detection process is monitored by the probing algorithm shown on bottom of the figure 8. If the topology of the network of module changes, while modules are performing their behaviors, the modules which detected the local changes, will initiate the DISTINCT algorithm and use their currently selected task in the negotiation process. As a result of this process, a new spanning tree will be dynamically created for the new topology that will synchronize and coordinate the initiation of new sets of behaviors based on the current topology of the network.

V. EXAMPLES

We have implemented and tested the FEATURE algorithm and all of it sub–algorithms on the CONRO self-reconfigurable robots. All modules are running as autonomous systems without any off-line computational resources and are loaded with the same control program and decision rules. For economic reasons, the power of the modules is supplied independently through cables from an off-board power supplier.

In our experiment we gave a 'Move' task to a quadruped CONRO robot. The robot initiated a 'Four-Legged Walking' gait. While performing the gait, we detached the two spine modules. The resulting configuration was two separate T-shape robots. In this situation, each T-shape robot continued the locomotion by executing the 'Butterfly Stroke' gait. Later, two T-shape robots were re-connected and the resulting four-legged robot re-initiated the 'Four-Legged Walking' gait.

For the snake configuration, we have experimented with caterpillar movement with different lengths ranging from 1 module to 10 modules. With no modification of programs, all these configurations can move and snakes with more than 3 modules can move properly as caterpillar. The average speed of the caterpillar movements is approximately 30cm/minute. In this experiment, we have dynamically "cut" a 10-module running snake into three segments with lengths of 4, 4, and 2, respectively. All these segments adapt to the new configuration and continue to move as independent caterpillars. We also dynamically connected two or three independent running caterpillars with various lengths into a single and longer caterpillar. The new caterpillar adapted to the new configuration and continued to move in the caterpillar gait. These experiments show that the described approach is robust to changes in the length of the snake configuration.

To test this approach for self-reconfiguration from a Snake to T-shape, a self-reconfiguration task was manually given to one the middle modules of a snake-shape robot consisting of seven modules. After completion of the self-reconfiguration task the new topology of the robot was detected and a butterfly gait for the T-shape robot was generated.

The videos of these experiments are available at http://www.isi.edu/robots.

VI. CONCLUSION AND FUTURE WORK

This paper presented the FEATURE algorithm that combines a set of distributed algorithms for accomplishing global tasks in chain-type self-reconfigurable robots consisting of multiple modules with dynamic topology.

The FEATURE algorithm uses probing for detecting the local topology changes. This information is used for coordinating the changes in the behavior of the modules. The FEATURE algorithm was implemented on the real CONRO modules and the experimental results were presented.

As the future work, we will study the conditions for performing successful self-reconfiguration and locomotion tasks based on the received messages and will test the performance of the FEATURE algorithm in multiple joints connection/disconnection self-reconfiguration tasks.

ACKNOWLEDGMENT

We are grateful that this research is sponsored by AFOSR under award numbers F49620-01-1-0020 and F49620-01-1-0441.

REFERENCES

[1] Yim, M., Y. Zhang, D. Duff, *Modular Robots*. IEEE Spectrum, 2002

[2] Rus, D., Z. Butler, K. Kotay, M. Vona,, *Self-Reconfiguring Robots*. ACM Communication, 2002.

[3] Shen, W.-M., B. Salemi, and P. Will., *Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots*. IEEE Transaction on Robotics and Automation, 2002. 18(5): p. 700-712.

[4] Shen, W.-M. and M. Yim (editors), *Special Issue on Self-Reconfigurable Modular Robots*, IEEE Transactions on Mechatronics, 7(4), 2002.

[5] Behnam Salemi, Peter Will, Wei-Min Shen, Distributed Task Negotiation in Self-Reconfigurable Robots, International Conference on Intelligent Robots and Systems. Las Vegas, October 2003.

[6] Behnam Salemi, Wei-Min Shen. "Distributed Behavior Collaboration for Self-Reconfigurable Robots". International Conference on Robotics and Automation. April-May 2004, New Orleans, LA, USA.

[7] Stoy, K., Shen,WM., Will, P.,, Using Role-Based Control to Produce Locomotion in Chain-Type Self-Reconfigurable Robots. IEEE/ASME Transactions on Mechatronics, 2002. **7**(4): p. 410.M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[8] Murata, S., H. Kurokawa, E. Toshida, K. Tomita, and S. Kokaji. A 3-D self-reconfigurable structure. in ICRA. 1998.

[9] Salemi, B., WM. Shen and P. Will. *Hormone Controlled Metamorphic Robots*. in *ICRA*. 2001.