Dynamicity to Save Energy in Microrobots Reconfiguration

Hicham Lakhlef, Hakim Mabed, and Julien Bourgeois
UFC/FEMTO-ST, UMR CNRS 6174, 1 cours Leprince-Ringuet, Montbeliard, France
$\{hlakhlef, hmabed, julien.bourgeois\}$ @$femto-st.fr$

***Abstract*—In this paper we present a dynamic self-reconfiguration protocol for MEMS microrobots. The protocol presented in this paper is without map of the target shape which makes it efficient and scalable. In other words, nodes do not store the positions that build the target shape. Consequently, memory usage for each node is reduced to a constant complexity. An algorithm of self-reconfiguration is deeply studied showing how to manage the dynamicity (wake up and sleep of microrobots) of the network to save energy. Our algorithm is implemented in Meld, a declarative language, and executed in a real environment simulator called DPRSim.**

***Index Terms*—MEMS, Distributed Algorithms ; Self-reconfiguration; Physical Topology; Dynamicity; Mobility**

I. Introduction

Micro-electro-mechanical systems (MEMS) microrobots are low-power and low-memory capacity devices that can sense and act. Microrobots systems have a wide range of applications such as odor localization, firefighting, medical service, surveillance, search, rescue, and security. To do these tasks the nodes have to perform the self-reconfiguration [3], [18].

One of the major challenges in developing a microrobot is to achieve a precise movement to reach the destination position while using a very limited power supply. Many different solutions have been studied. For example, within the $Claytronics$ project [1], [2], microrobots can only turn around its neighbor which introduce the idea of a collaborative way of moving. But, even if the power requested for moving has been lowered, it still costs a lot regarding the communication and computation requirements [10].

In the literature, self-reconfiguration can be seen from two different points of view. First, it can be defined as a protocol, centralized or distributed, which transforms a set of nodes to reach the optimal logical topology from a physical topology [9], e.g, the chain represents the worst complexity case with $O(n)$, the square represents the best one with $O(\sqrt{n})$. On the other hand, the self-reconfiguration is built from modules which are autonomously able to change the way they are connected, thus changing the overall shape of the network [7]. This process is difficult to control, because it involves the distributed coordination of a large numbers of identical modules connected in time-varying ways.

Mobility and dynamicity of the system are making the problem even harder to handle as the logical topology of the system has to be stored in a distributed data structure, usually a spanning tree. The range of exchanged information and the number of movements determine the communication and the energy complexity of the distributed algorithm. When the information exchange involves close neighbors, the complexity is moderate and the resulting distributed self-reconfiguration scales gracefully if the algorithm does not need the predefined positions of the target shape. An open issue is whether distributed self-reconfiguration would result in an optimal configuration with a moderate complexity in message, number of movements and memory usage (number of state per node).

This work takes place within the Claytronics project and aims at optimizing the logical topology of the network.

II. Related Works

Many terms refer to the concept of self-reconfiguration. In several works on wireless networks the term used is $self-organization$. This term is also used to express the partitioning and clustering of ad-hoc networks to cliques or clusters. Also, the self-organization and *redeployment* terms can be found in protocols for sensors networks to form a sphere or a polygon from a center node [14], [20]. For self-reconfiguration with robots or microrobots, there are protocols [7], [17] where the desired configuration is grown from an initial seed module. A generator uses a 3D CAD model of the target configuration and outputs a set of overlapping blocks which represent this configuration. In the second step, this representation is combined with a control algorithm to produce the final self-reconfiguration algorithm. Among the centralized algorithms we find centralized self-assembly and/or reconfiguration algorithms [16]. Other approaches give each node a unique ID and a predefined position in the final structure; see for instance [19]. The drawback of these methods is the centralized paradigm and the need for nodes identification. More distributed approaches in [4], [5], [8], [11] and [12].

Claytronics, is a project led by Carnegie Mellon University and Intel corporation. In Claytronics, microrobots called catoms (Claytronics atoms). The idea is to have hundreds of thousands of microrobots forming by self-reconfiguration together objects of any shape or size. Much like the cells in a body or complex organism, each small member of the whole is committed to doing its own part and communication between parts results in a unified form. Many works have already been done within the Claytronics project. In [6], the authors propose a metamodel for the reconfiguration of catoms starting from an initial configuration to achieve a desired configuration using $creation$ and $destruction$ primitives. The authors use these two functions to simplify the movement of each catom.

In [2], a scalable protocol for Catoms self-reconfiguration is proposed, written with the MELD language [1], [15] and using the creation and destruction primitives. In all these works, the authors assume that all Catoms know the correct positions composing the target shape at the beginning of the algorithm and each node is aware of its current position. The first self-reconfiguration without predefined positions of the target shape appears in[11], [13]. However, this solution is not energy-efficient and does not deal with the dynamicity of the network.

III. Contributions

In this paper, we propose a new distributed approach for self-reconfiguration of MEMS microrobots, where the target form is built incrementally, and each node in the current increment acts as a landmark for other nodes to form the next increment, which will belong to the form. We introduce a state model where each node can see the state of its physical neighbors to achieve the self-reconfiguration for distributed MEMS microrobots, using the states the nodes collaborate and help each other. In this paper each node predicts its future actions (movements), so it can compute the energy amount that will spend before the beginning of the algorithm. The prediction property makes the algorithm robust and energy-aware, because the node can make sure that it has correctly followed the algorithm and it is aware of the amount of energy that it will use. Also, to keep the energy and to augment the probability that the node will finish its task, each node is aware of the time slots where its can sleep to save energy.

In the proposed algorithm, the exchange of messages is limited to the construction of the spanning tree. The spanning tree is used to ensure the connectivity of the network and dynamically manage the nodes that can move. Contrary to existing works, in our algorithm each node has no information on the correct positions (predefined positions) of the target shape, the algorithm does not need to know the network size and movement of microrobots is fully implemented.

We propose here an efficient distributed algorithm for nodes self-reconfiguration where each node moves by rotation around their physical neighbors. We study the case of a self-reconfiguration from a chain of microrobots to a square. The performance of the self-organization algorithm is evaluated according to the number of rotations and the time taken. In this paper the MEMS network is organized initially as a chain. By choosing a straight chain as the initial shape, we aim to study the performance of our approach in extreme case. Indeed, the chain form represents the worst physical topology for many distributed algorithms in terms of fault tolerance, propagation procedures and convergence. First, the number of direct contacts between macro-robots is minimal and secondly the average distance between two robots (in terms of number of hops) is of $(n+1)/3$ where n is the number of robots. Also, a chain of microrobots represents the worst case for message broadcasting complexity with $O(n)$. The redeployment into a square organization allows to obtain the best messages broadcasting complexity with $O(\sqrt{n})$.

To assess the distributed algorithm performance, we present the results of the simulations made with Meld [1] and the DPRSim simulator [21].

The rest of the paper is organized as follows: Section 4 discusses the model and some definitions. Section 5 discuss the proposed algorithm and analyzes the number of sent messages, the number of movements, it shows how to manage the dynamicity of the network and discusses the generalization of the algorithm. Section 6 details the simulation results. Finally, section 7 summarizes our conclusions and illustrates our suggestions for future work.

IV. Model and Definitions

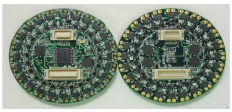

Figure 1. Two catoms.

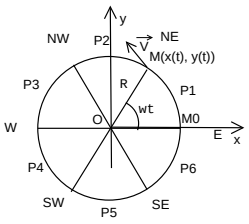

Figure 2. Node modeling, in each movement the node travels the same distance.

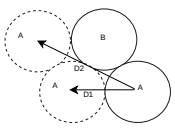

Figure 3. Traveled distance in one movement = 2R, the node A travels 2R in one movement.

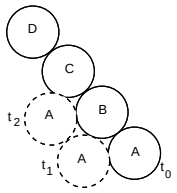

Figure 4. Message transmission, there will be message exchange if the node needs to know the state of a non neighbor node.

Within Claytronics, a catom (figure 1) that we call in this paper a node is modeled as a sphere which can have at most six neighbors. Each node is able to sense the direction of its physical neighbors (east (E), west (W), north-east (NE), south-east (SE), south-west (SW) and north-west (NW)). In this work, the starting physical topology is a chain of n nodes linked together. A chain corresponds to a connected set of nodes where each node has two neighbors excepting the two extremities having only one neighbor. We will take the example of nodes that have neighbors in NW and SE directions and we will show after how to generalize. A node A is in neighbor's list of node B if A is physically linked to B. Direct communications are only possible through contact. We consider the connected undirected graph *G = (V, E)* modeling the network, where $v \in V$, is a node that belongs to the network and, $e \in E$ a bidirectional edge of communication between two physical neighbors. For each node $v \in V$, we denote the set of neighbors of v as $N(v) = \{u, (u,v) \in E\}$. Each node $v \in V$ knows the set of its neighbors in G, denoted $N(v)$. We define the following terminology:

$Connectivity$: in a graph $G = (V, E)$, if $\forall v \in V$, $\forall u \in V, \exists C_{v,u} \subseteq$E : $C_{v,u} = (e_{v,-}, ..., e_{-,-}, ..., e_{-,u})$, with $e_{x,y}$ is an edge from x to y and $C_{v,u}$ represents a path from v to u.

$Snap-Connectivity$: let T be the total execution time of our distributed algorithm DA and $t1..., tm$ are the time slots of execution of DA. There is a Snap-Connectivity in DA with the dynamic graph $G_t(V_t i, E_t i)$ the network state at the instant ti, if $\forall t_i, i \in \{1, ..., m\}$, $G_{ti}(V_{ti}, E_{ti})$ maintains the connectivity.

Spanning tree: is a graph composed of all without any cycle. In the spanning tree, a node is either a child or a parent, a leaf is node without children.

We call the *own movements* of a given node the number of movements it performs.

Consider the figure 2 which represents a microrobot. We say that a microrobot has done a single movement if the distance between its former position and its new position is exactly twice the radius $D1 = 2R$. For example, if the node is in a position at a distance $D2$ (see the figure 3) from the former position it has done two movements. We have 360° can be divided to six equal angles each one has 60°, since the perimeter at an angle a is $P_a = \pi\ Ra/180\ and$ $P = 2\pi R$ we find $P1 = P2 = P3 = P4 = P5 = P6$, this means that the node can have without overlapping at most six neighbors and in each movement the node travels Ra (a=60°) from $m0$ to m. In this paper, we assume that the change of message (consultation) between two physical neighbors is carried without complexity (0 message), while the distance between them is zero. If a node *to decide* needs to know the state of a non-physical neighbor message exchange is required, for example in the figure 4:

- At t_0: the node A needs to know the state of B to move to the new position, this movement is done without message exchange.
- At t_2: if A is in the new position and it needs to know the state of D to move then D sends a message to C informing its state to C that forwards the message to A. So, in this case there is a message exchange and A must wait two rounds to decide.
- But if at t_0 or at t_1 a message has been sent from D to C, so A at t_2 can have the state of D with a simple consultation of C's state.

It is important to minimize the number of movements vis-a-vis the energy and time of execution. And it is important to minimize the space of memory used, therefore the number of states per node.

V. Proposed protocol

A. Dynamic Algorithm with Safe Connectivity (DASC)

In DASC, each node can only move around its physical neighbor. To ensure snap-connectivity only nodes that do not cause network non-connectivity can move around neighbors. Keeping the network connected is very important because the node can move only around its physical neighbor using magnet forces and can communicate only with its physical neighbor, if the property of connectivity is broken nodes will be lost (cannot join the network). For this purpose we introduce the use of the tree to dynamically manage the leaf nodes authorized to move.

Description of the algorithm

The algorithm runs in rounds. In each round, satisfied predicates are executed. In a current round predicates with best priority are executed while others with lowest priority are ignored. We notice that in DASC, the state change actions, represented by predicates labeled P1, are considered more prior than a movement actions represented by P2. The distributed algorithm seeks the desired form by an incrementally process. In a completed increment, the nodes that build it belong already to the form. The initiator which is the root initializes the tree and becomes a parent of itself (5). A node if it does not have a parent becomes a child of one of the neighbor parents (6), a node is a leaf if all its neighbors are parents (7). At the beginning all nodes are initialized with the bad state with predicate (2). The initiator belongs to the target shape, so it changes its state to $good$ (3), it will help its neighbors or future neighbors to take correct positions. The nodes already in the target shape act as a benchmark to neighbor or future neighbor nodes to complete a new layer. The nodes already in the form change their states with the predicate (3) and (8) and they become constant, the node can check if its neighbor have the $good$ state with the predicate (3) and (8). The node that starts the move is the lowest node in the chain that is the first leaf of the first tree built, it rises until the root using motion around other nodes with predicates (11) and (12). The nodes of the current layer (layer being built) may make motion either at left directly or NW directly with the last three predicates. The node can change its state to $good$ with predicate (3) if it cannot move to left or in NW. With the predicate (13) the node moves at left, it will have the neighbor that used it to move at NE direction, it repeats the same motion until it arrives to the diagonal node that have the state spe, it cannot move around this last only if the diagonal node has not a neighbor node in the E direction. Diagonal nodes take the state spe with the predicate (4) and (10), and with (14) the node moves until it takes a correct position. The state change has a priority as the moving actions to avoid bad motion, because of this we introduce the priority in our algorithm. To avoid message exchange the node can change its state to $good$ if it has 3 neighbors having the state $good$ (9) or one neighbor has spe state and has neighbors in the both NE and NW directions with predicate (10).

Complexity of sent messages

DASC needs only $O(n)$ messages to construct the first tree. Avoiding message for state changing will better accelerate the algorithm. In DASC the node can change its state only by looking to the state of its physical neighbors, it does not need to wait for message informing the state of nodes which are not its physical neighbors to make the decision.

Variables and predicates

- v, u, $u1$, $u2$: variables denote a node belongs to the network.
- $\{U\}$:set of nodes.
- $good, bad, spe$: states, a node can take one or two states at the same time, but not spe and bad or $good$ and bad.
- $N_x(v)$: the neighbor in the direction x of the node v: $x \in \{(N), (E), (W), (NE), (SE) or (NW)\}$.
- $connected_v$: $true$ if the node v is connected to the network, $false$ else (Boolean).
- $State_v(k)$: the state of the node v, taking one or two of these values k = $good, bad$ or spe.
- $State_v(s, good)$: the node v has s (s an integer) neighbors that have the $good$ state $State(good)$.
- $moveAroundgood_v(u, P_x)$: move around the neighbor u in such a way that u becomes a v's neighbor in the direction x relative to v.
- $Parent(v, u)$: the node v is parent of node u.
- $isLeaf(v)$: the node v is a leaf in the tree.

Predicates checked only in the first round

1: Initiator$(v) \equiv (\neg N_{nw}(v) =) \wedge connected_v$.
2: State$_v(bad) \equiv connected_v \wedge \neg Initiator(v)$.
3: State$_v(good) \equiv Initiator(v)$.
4: State$_v(spe) \equiv Initiator(v)$.

Predicates checked in each round

5: Parent$(v, v) \equiv Initiator(v)$.
6: Parent$(v, u) \equiv (Parent(w, v), u \neq w) \wedge neighbor(v, u) \wedge State_u(bad) \wedge (\nexists z \in N(v), Parent(v, z))$.
7: isLeaf$(v) \equiv (\forall u \in N(v), \neg Parent(v, u) \wedge \neg Parent(v, v))$.
8: (P1): State$_v(good) \equiv (N_e(v) = u \wedge State_u(good) \wedge \neg N_{ne}(u)) \vee State_v(3, good) \vee (State_v(2, good) \wedge (N_{ne}(v) = u \wedge State_u(spe)) \vee (N_w(v) = u \wedge State_u(good))) \vee State_v(spe)$.
9: State$_v(s, good) \equiv (N_x(v) = \{U\}, |U| = s \wedge State_{\{u\}}(good))$.
10: (P1): State$_v(spe) \equiv (N_{nw}(v) = u1)) \wedge (N_{ne}(v) = u2, State_{u2}(spe))$.
11: (P2): moveAround$bad_v(u, P_e) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{nw}(v) = u \wedge State_u(bad))$.
12: (P2): moveAround$bad_v(u, P_{se}) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{ne}(v) = u \wedge State_u(bad))$.
13: (P2): moveAround$good_v(u, P_{ne}) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{nw}(v) = u \wedge State_u(good))$.
14: (P2): moveAround$good_v(u, P_e) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{ne}(v) = u \wedge State_u(good))$.

the DASC Algorithm .

This is guaranteed with predicate (8) when the node changes its state from bad to $good$. It is obvious that if a node changes its state before it is sure of the good state of other nodes that have moved before it in the current layer, the self-reconfiguration desired will be not achieved. The predicate (8) ensures without exchanging of message that the node changes its state only if all nodes that have moved before changed their states to $good$, therefore the first node that begins the construction of the new layer does not need to wait for the message of the first node that began the previous layer, since the node that is currently checking the predicate (8) can have this information by consulting the state of its neighbor. In other words, the message was being sent before the node needs to know the state of its sender, when the node needs to know this state it will find the message at its physical neighbor. This efficiency is explained by the fact that DASC made synchronization in state changing is not required for nodes that are in the same layer.

B. Predicting the number of movements for each node

To form the matrix of our square with NxN nodes, we begin with an incremental process with a single node that we assume in a correct square 1x1. After, we add each time a new layer contains the number of nodes of the last column plus the number of nodes of the last line of the current square plus one node. Consider the figure 5, the node i will take a position $p + x$. Following the path from top to bottom the node i will never move or move after all nodes after it, so if node A is before B, A will take a position $p + c$, and node B will take a position $p + k$, with $c > k$. Adding layers, each time we add a new layer with number of nodes equal to the number of nodes of the previous layer plus two nodes, this can be expressed on the form of this numerical sequence:

$$U_j = U_{j-1} + 2. \tag{1}$$

Where: U_j is the number of nodes in the layer j and U_{j-1} is the number of nodes in layer $j - 1$.
In the chain we take a partitioning of the nodes to levels, a level can be associated to one or many nodes. The nodes take their levels with this following process: the first nodes that have $i \leq \sqrt{n}$ take the root level (level 0), for the other nodes, the first $x = (2\sqrt{n} - 2)$ nodes after the node $i = \sqrt{n}$ take the first level (level 1), and the second $x - 2$ nodes take the second level and so on (figure 5 shows an example). So each node i gets one level at the end.
The number of movements for each node i of level j can be given with the composition of two sequences $U_{i,j}$ and R_j.

$$R_j = \begin{cases} 0, \, if \, j = 0 \\ 2\sqrt{n} - 5, \, if \, j = 1 \\ R_{j-1} - 2, \, otherwise \end{cases} \tag{2}$$

With R_j is a number associated to nodes having the level j and n is network size.

$$U_{i,j} = \begin{cases} 0, \, if \, i \leq \sqrt{n} \\ 2, \, if \, i = \sqrt{n} + 1, j = 1 \\ U_{i-1} - R_j, \, if \, l(i + 1) \neq j \\ U_{i-1} + 2, otherwise \end{cases} \tag{3}$$

Where: $U_{i,j}$ and $U_j$is the number of movements of node i having level j or the number of movements rounds of nodes having the level j and n is the network size.

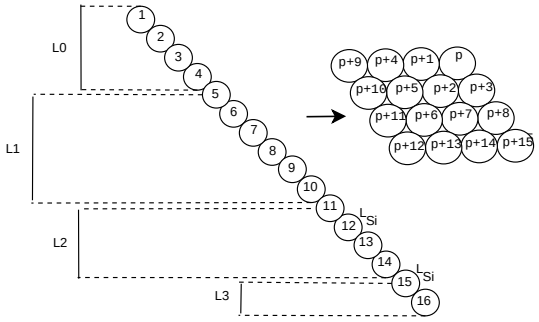

Figure 5. Example of nodes partitioning into levels and the final positions

Theorem 5.1: n is highest number of movements in this algorithm.

Special case This case deals with the situation when the number of nodes is not a square root. We assume it r. To calculate the own movements we take a similar partitioning to the previous. In this special case also r is the number of movements. The next lines are used to express the own movements for each node.

Let $n = \lfloor \sqrt{r} \rfloor$, and $diff = r - n^2$.

R_j has the same definition in (2).

$$U_{i,j} = \begin{cases} 0, if\, i \leq n. \\ diff + 2n - 1, if\, i = n + 1. \\ U_{i-1} - 2, if\, n + diff \geq i > n. \\ diff + 2, if\, i = n + diff + 1. \\ U_{i-1} - R_j, if\, i > n + diff, l(i+1) \neq j. \\ U_{i-1} + 2, otherwise. \end{cases} \quad (4)$$

C. Energy saving

	10	n+14/10	14	n+5/14	16	n/16	1
	n+15/9	n+15/9	n+6/13	n+6/13	n+1/15	n+1/15	n+2/2
	9	n+15/9	13	n+6/13	15	n+1/15	2
	n+16/8	n+16/8	n+7/12	n+7/12	n+8/11	n+8/11	n+9/3
	8	n+16/8	12	n+7/12	11	n+8/11	3
	n+17/7	n+17/7	n+18/6	n+18/6	n+19/5	n+19/5	n+20/4
	7	n+17/7	6	n+18/6	5	n+19/5	4

With:

t/j	i
t/j	t/j

: i is the node, t/j is the time t when the node j changes its state to well and becomes a final neighbor of i

Figure 6. Represents the time when the final neighbor changes its states to *good*. The value in a circle represents the slot time when the node i can enter into sleep state, in the last line the value for some nodes are indicated with arrows

Sleeping state is used to save energy and awake state is used to do the task for each node. The node cannot enter to a sleep state by changing its state to *good* since the node after changing its state to *good* becomes a reference for neighbors or future neighbors and it should help its neighbors so they can take correct positions belong to the final shape. So, it should stay wake to send messages (consultation) to neighbor nodes that need to know its state to decide. The aim of the following functions is to find with an optimal and deterministic method the time slots when the node must wake up to help neighbors and where the node must sleep to save energy. The following functions have a form of mathematical sequences which are in fact messages. Thus, by receiving the information from its neighbor the node can know its value which refers to the time of entering wake or sleep state.

We take a partitioning of nodes into levels, each node will have a level $l(i)$, and some nodes take a special level noticed ls_i.Nodes with $i \leq \sqrt{n}$ take the level 0. For the others nodes: the $x = 2\sqrt{n} - 2$ nodes after $i = \sqrt{n}$ take level 1. After, the following $x - 2$ nodes take the following level (level 2) and so on. A special level ls_i is associated to some nodes: node $4\sqrt{n} - 4$ takes the level ls_i, the following node that takes the level ls_i is the one after $y = 2\sqrt{n} - 5$, and the the following node that will take the level ls_i is the one after $y - 2$ nodes and so on. Figure 5 shows an example. Once the node has taken a child (in the tree) it can enter into the sleep state and it must wake up at the time when it will have new neighbors, it is the time to reach it for nodes that are going up. The root node starts the building of the tree at the round $t0$, it becomes a parent and enters into sleep state to save energy.

n+15	10	n+15	14	n+6	16	n+2	1
n+16	9	n+16	13	n+8	15	n+9	2
n+17	8	n+18	12	n+19	11	n+20	3
n+18	7	n+19	6	n+20	5	n+20	4

With:

t	i

: i is the node and t is the time coredponds to the last neighbor of the node i has changed its state to well

Figure 7. Represents the time of the last neighbor changed its state, so the node i will not have a new neighbor to help it.

(G) S_i=n+15	10	(G) S_i=n+15	14	(E) S_i=n+6	16	(A) S_i=n+2	1
(G) S_i=n+16	9	(E) S_i=n+16	13	(F) S_i=n+8	15	(B) S_i=n+9	2
(G) S_i=n+17	8	(D) S_i=n+18	12	(D) S_i=n+19	11	(B) S_i=n+20	3
(G) S_i=n+18	7	(G) S_i=n+19	6	(C) S_i=n+20	5	(C) S_i=n+20	4

With:

(X) S_i	i

: i is the node and (X) is the action used to calculate $S_i$in (7)

Figure 8. Represents how the last time corresponds to the sleeping time is calculated.

Predicates:
Initiator$^r(v)$: the node has at the beginning only one neighbor in r direction, with $r \in \{nw, ne, w\}$.
Node$^d(v)$: the node was at the beginning in a chain d , with $d \in \{nw-se, ne-sw, w-e\}$.
x: index means for any type of the chain $\in \{nw-se, ne-sw, w-e\}$.

Predicates checked only in the first round

Initiator$^{nw}(v) \equiv (\neg N_{nw}(v)) \wedge (\neg N_{sw}(v)) \wedge (\neg N_{ne}(v)) \wedge (\neg N_e(v)) \wedge (\neg N_w(v)) \wedge (N_{se}(v))$.
Initiator$^{ne}(v) \equiv (\neg N_{nw}(v)) \wedge (\neg N_{se}(v)) \wedge (\neg N_{ne}(v)) \wedge (\neg N_e(v)) \wedge (\neg N_w(v)) \wedge (N_{sw}(v))$.
Initiator$^{w}(v) \equiv (\neg N_{nw}(v)) \wedge (\neg N_{sw}(v)) \wedge (\neg N_{se}(v)) \wedge (\neg N_{ne}(v)) \wedge (\neg N_e(v)) \wedge (\neg N_w(v)) \wedge (N_e(v))$.
Node$^{nw-se}(v) \equiv ((N_{nw}(v) \vee N_{se}(v)) \wedge (\neg N_{ne}(v)) \wedge (\neg N_e(v)) \wedge (\neg N_w(v)) \wedge (\neg N_{sw}(v))$.
Node$^{ne-sw}(v) \equiv ((N_{sw}(v) \vee N_{ne}(v)) \wedge (\neg N_{nw}(v)) \wedge (\neg N_{se}(v)) \wedge (\neg N_e(v)) \wedge (\neg N_w(v))$.
Node$^{w-e}(v) \equiv ((N_w(v) \vee N_e(v)) \wedge (\neg N_{nw}(v)) \wedge (\neg N_{sw}(v)) \wedge (\neg N_{se}(v)) \wedge (\neg N_{ne}(v))$.
State$_v(bad) \equiv Node^x(v) \wedge \neg Initiator^x(v)$.
State$^x_v(good) \equiv Initiator^x(v)$.
State$^x_v(spe) \equiv Initiator^x(v)$.

Predicates checked in each round

Parent(v, v)$\equiv Initiator^x(v)$.
Parent(v, u)$\equiv (Parent(w, v), u \neq w) \wedge neighbor(v, u) \wedge State_u(bad)$.
isLeaf(v)$\equiv (\forall u \in N(v), \neg Parent(v, u) \wedge \neg Parent(v, v))$.
(P1):State$^{nw}_v(good) \equiv (N_e(v) = u1 \wedge State_{u1}(good) \wedge \neg N_{nw}(u1)) \vee State_v(3, good) \vee (State_v(2, good) \wedge (N_{ne}(v) = u1 \wedge State_{u1}(spe)) \vee (N_w(v) = u1 \wedge State_{u1}(good))) \vee State_v(spe) \wedge Node^{nw-se}(v)$.
(P1):State$^{ne}_v(good) \equiv (N_w(v) = u1 \wedge State_{u1}(good) \wedge \neg N_{ne}(u1)) \vee State_v(3, good) \vee (State_v(2,) \wedge (N_{nw}(v) = u1 \wedge State_{u1}(spe)) \vee (N_e(v) = u1 \wedge State_{u1}(good))) \vee State_v(spe) \wedge Node^{ne-sw}(v)$.
(P1):State$^{w}_v(good) \equiv (N_{ne}(v) = u1 \wedge State_{u1}(good) \wedge \neg N_e(u1)) \vee State_v(3, good) \vee (State_v(2, good) \wedge (N_{nw}(v) = u1 \wedge State_{u1}(spe)) \vee (N_{sw}(v) = u1 \wedge State_{u1}(good))) \vee State_v(spe) \wedge Node^{w-e}(v)$.
(P1):State$_v(s,\ good) \equiv (N_x(v) = \{U\}, |U| = s \wedge State_{\{u\}}(good))$.
(P1):State$^{nw}_v(spe) \equiv (N_{nw}(v) = u1)) \wedge (N_{ne}(v) = u2, State_{u2}(spe)) \wedge Node^{nw-se}(v)$.
(P1):State$^{ne}_v(spe) \equiv (N_{ne}(v) = u1)) \wedge (N_{nw}(v) = u2, State_{u2}(spe)) \wedge Node^{ne-sw}(v)$.
(P1):State$^{w}_v(spe) \equiv (N_w(v) = u1)) \wedge (N_{nw}(v) = u2, State_{u2}(spe)) \wedge Node^{w-e}(v)$.
(P2):moveAround$^{nw}bad_v(u, P_e) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{nw}(v) = u \wedge State_u(bad)) \wedge Node^{nw-se}(v)$.
(P2):moveAround$^{nw}bad_v(u, P_{se}) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{ne}(v) = u \wedge State_u(bad)) \wedge Node^{nw-se}(v)$.
(P2):moveAround$^{nw}good_v(u, P_{ne}) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{nw}(v) = u \wedge State_u(good)) \wedge Node^{nw-se}(v)$.
(P2):moveAround$^{nw}good_v(u, P_e) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{ne}(v) = u \wedge State_u(good)) \wedge Node^{nw-se}(v)$.
(P2):moveAround$^{ne}bad_v(u, P_w) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{ne}(v) = u \wedge State_u(bad)) \wedge Node^{ne-sw}(v)$.
(P2):moveAround$^{ne}bad_v(u, P_{sw}) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{nw}(v) = u \wedge State_u(bad)) \wedge Node^{ne-sw}(v)$.
(P2):moveAround$^{ne}good_v(u, P_{nw}) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{ne}(v) = u \wedge State_u(good)) \wedge Node^{ne-sw}(v)$.
(P2):moveAround$^{ne}good_v(u, P_w) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{nw}(v) = u \wedge State_u(good)) \wedge Node^{ne-sw}(v)$.
(P2):moveAround$^{w}bad_v(u, P_w) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{w}(v) = u \wedge State_u(bad)) \wedge Node^{w-e}(v)$.
(P2):moveAround$^{w}bad_v(u, P_{sw}) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{nw}(v) = u \wedge State_u(bad)) \wedge Node^{w-e}(v)$.
(P2):moveAround$^{w}good_v(u, P_{nw}) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{w}(v) = u \wedge State_u(good)) \wedge Node^{w-e}(v)$.
(P2):moveAround$^{w}good_v(u, P_w) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{nw}(v) = u \wedge State_u(good)) \wedge Node^{w-e}(v)$.

The Generalized algorithm of DASC : DGASC

$$T_i = \begin{cases} 7,\ if\ i = 1. \\ T_{i-1} + 4,\ otherwise. \end{cases} \quad (5)$$

Where : T_i is a number associated to node i, $i \leq \sqrt{n} - 2$.

$$I_j = \begin{cases} 2\sqrt{n} - 1,\ if\ j = 3. \\ I_{j-1} - 2,\ otherwise. \end{cases} \quad (6)$$

Where : I_j is a number associated to node i has level $j > 1$.

$$S_i = \begin{cases} n + 2,\ if\ i = 1.(A) \\ S_{i-1} + T_{i-1}, if\ i \leq \sqrt{n} - 1.(B) \\ S_{i-1}, if\ i = \sqrt{n} \vee i = \sqrt{n} + 1.(C) \\ S_{i-1} + 2\sqrt{n} - 4, if\ l(i) = 2 \wedge l(i-1) = 1.(D) \\ S_{i-1} - 2,\ if\ ls(i-1).(E) \\ S_{i-1} - I_i,\ if\ l(i-1) \neq l(i).(F) \\ S_{i-1} - 1, otherwise.(G) \end{cases} \quad (7)$$

Where : $S_i + O(n)$ refers to the sleeping time for node i.

Since the first leaf node will move n rounds, (to become root's neighbor in the E direction) it will be neighbor of the root at the time $n-1$. So, the root adjusts the local clock to wake up at $n-1+O(n)$(with $O(n)$) is the time the first tree) in order to collaborate with its new neighbors. Similarly, other nodes are waiting for the construction of the first tree and enter into sleep sate after having a child, each node located after z nodes from the root enters into awake state at $n-z-1+O(n)$. The sequence S_i expresses in term of n the time when each node can enter into sleep state after helping its neighbors to take correct positions.

Example : figures 6, 7 and 8 present an example showing how the values S_i are calculated.

D. Generalization of the algorithm (DGASC)

We have presented an algorithm that deals with one case of the chain, exactly with the case where nodes can have at the beginning neighbors in directions SE or NW or in both directions at the same time. To show how to generalize the algorithm in order to deal with any chain at the beginning it is important to show how to distinguish the initiator (the root) whatever the case. For other nodes can know what form of chain is by looking at the direction of their two neighbors. The root can be distinguished with principle that it has only one neighbor in the direction SW or SE or E, obviously ,whatever the shape of the chain we cannot find one where another node that has only one neighbor in the direction SW or SE or E, other nodes have two neighbors in the same time one in the direction D and the other in the inverse direction say $-D$ for examples: one neighbor in SE direction and the other in the direction NW (NW-SE), SW and NE (NE-SW) or E and W(W-E). The last node in the chain has one neighbor in the direction NW, NE or W. After recognizing the form of chain, an algorithm similar to DASC presented is called, for example if the chain was with the form where the nodes can only have neighbors in the directions NE or SE or in both directions, we have to call $DASC^{-d}$ if we define $DASC^{-d}$ as the previous algorithm but the move is made from NE to NW or to W or from NW to W. The given DGASC generalizes the algorithm. We note in GASC that the predicates cheeked only in in the first round if they were true it remain always true. For clarity, the predicate indexed with nw means that this predicate is called if the chain at the beginning was NW-SE, the predicate indexed with w means that this predicate is called if the chain at the beginning was W-E, the predicate indexed with nw means that this predicate is called if the chain at the beginning was NE-SW.

VI. Simulation

We have done the simulation with the declarative language Meld using DPRSim. In our simulations the radius of the node is 1 mm. We simulated with a laptop with processor Intel(R) Core(Tm) i5, 2.53 Ghz. The results of these simulations come to agree the results obtained previously, in particular the number of movements for each node and the effectiveness of dynamicity. The nodes applied the procedure of partitioning

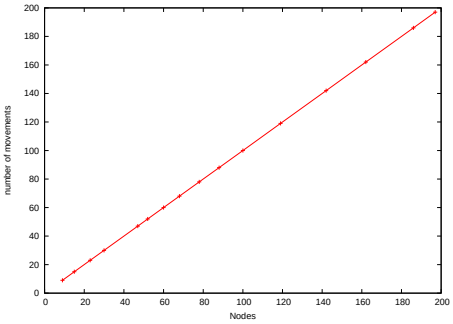

Figure 9. Highest number of movements.

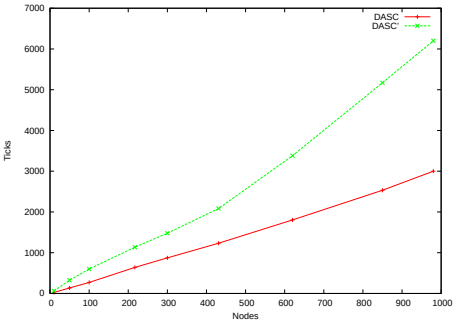

Figure 10. Execution time.

to levels and predicted with the two functions $U_{i,j}$ and R_j the number of movements for each node, at the end of the algorithm each node compares the results of prediction to the results calculated by it. For instance, figure 9 represents simulation results of node performed the highest number of movements. The figure 11 represents the overall number of movements in the networks corresponds to

$$O = \sum_{i=1,j=0}^{i=n,j=\sqrt{n}-1} U_{i,j} \tag{8}$$

The figure 12 represents the average of the overall number of movements corresponds to

$$\frac{O}{n} \tag{9}$$

The nodes applied the procedure of nodes partitioning into levels and obtain with the function discussed previously the time slot when they enter into the sleep state and the wake up state. The figure 10 represents the execution time in ticks by the number of nodes, with counting the tree (DASC') and without counting the tree (DASC). In the curve representing the number of movements, we remark for some values of the network size n, the number is always n as found in theory. For the curves that represents the execution time figure 10, without counting the time of construction of the tree of DASC we see that if the number of nodes increases the time increase. If we count the time of the tree ($O(n)$ time), the execution time of the algorithm increases dramatically. As conclusion, to ensure a Snap-connectivity through all time slots of the algorithm and to manage dynamically the nodes that can move, we have

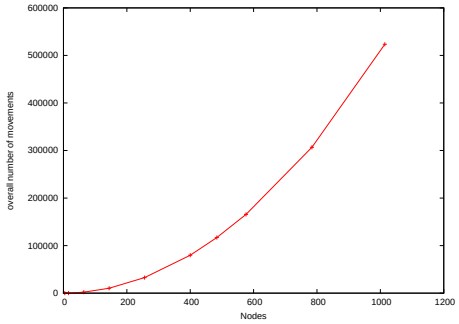

Figure 11. The overall number of movements in the network.

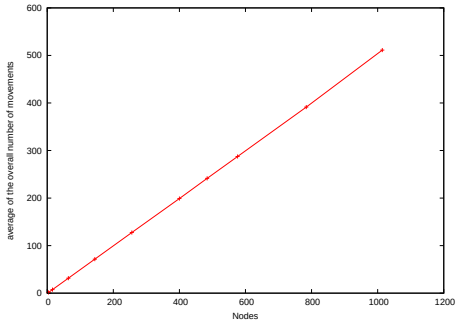

Figure 12. The average of the overall number of movements in the network.

to use the tree, and by using the tree, we need more time to achieve the self-reconfiguration.

VII. Conclusion

In this paper, we presented a new method to complete the self-reconfiguration where the nodes do not know the fixed positions of the target shape but only the aimed shape; nodes collaborate and help each other by analyzing the characteristics of the target shape. Compared the literature works this algorithm is scalable because each node needs only three state to achieve the algorithm. Nodes in our paper can perform algorithm regardless the place where they are because the algorithm is independent of the map, that what we call portability. We have presented a protocol that guarantees the connectivity throughout its execution time. The proposed algorithm is characterized by a constant memory needs and message exchange is limited to neighboring consultations. We presented how to manage the dynamicity of the network to save the energy and how to predict the movements of nodes in order to make the algorithm robust and energy-aware.
However, some open problems remain; we will study the fault tolerance on self-reconfiguration in microrobots networks. Also, the use of tabu algorithms to achieve the self-reconfiguration.

VIII. Acknowledgments

This work is funded by the Labex ACTION program (contract ANR-11-LABX-01-01), ANR/RGC (contracts ANR-12-IS02-0004-01 and 3-ZG1F) and ANR (contract ANR-2011-BS03-005). The authors wish to express their appreciation to the three anonymous reviewers for their constructive comments.

References

[1] M. P. Ashley-Rollman, S. C. Goldstein, P. Lee, T. C. Mowry, and P. Pillai, *Meld: A Declarative Approach to Programming Ensembles,* In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS '07), October, 2007.

[2] M. P. Ashley-Rollman, P. Lee, S. C. Goldstein, Padmanabhan Pillai, and Jason D. Campbell, *A Language for Large Ensembles of Independently Executing Nodes,* In Proceedings of the Int. Conf. on Logic Programming, July, 2009.

[3] J. Bourgeois and S.C. Goldstein. *Distributed Intelligent MEMS: Progresses and perspectives*, 3-rd Int. Conf. ICT Innovations, volume of Communications in Computer and Information Science, Macedonia, 2011.

[4] H. Bojinov, A. Casal, T. Hogg, *Emergent structures in modular self-reconfigurable robots,* Proc. of the IEEE International Conference on Robotics and Automation, vol. 2, pp. 1734-1741. IEEE Computer Society Press, Los Alamitos, 2000.

[5] Z. J. Butler, K. Kotay, D. Rus, K. Tomita, *Generic decentralized control for lattice-based self-reconfigurable robots*, International Journal of Robotics Research 23(9):919-937, 2004.

[6] D. Dewey, S. S. Srinivasa, M. P. Ashley-Rollman, M. D. Rosa, P. Pillai, T. C. Mowry, J. D. Campbell, and S. C. Goldstein, *Generalizing Metamodules to Simplify Planning in Modular Robotic Systems,* In Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, September, 2008.

[7] K.Stoy, R.Nagpal, *Self-reconfiguration using Directed Growth*, 7th International Symposium on Distributed Autonomous Robotic Systems (DARs), France, June23-25, 2004.

[8] C. Jones, M. J. Mataric, *From local to global behavior in intelligent self-assembly.In: Proc. 2003 IEEE International Conference on Robotics and Automation,*, vol. 1, pp. 721-726, Los Alamitos, 2003.

[9] S. Jeon, C. Ji, *Randomized Distributed Configuration Management of Wireless Networks: Multi-layer Markov Random Fields and Near-Optimality* CoRR abs/0809.1916, 2008.

[10] M. E. Karagozler, A. Thaker, S. C. Goldstein, D. S. Ricketts, *Electrostatic Actuation and Control of Micro Robots Using a Post-Processed High-Voltage SOI CMOS Chip*,IEEE International Symposium on Circuits and Systems (ISCAS), May 2011.

[11] H. Lakhlef, H. Mabed, J. Bourgeois, *Distributed and Efficient Algorithm for Self-reconfiguration of MEMS Microrobots*, in the 28th ACM Symposium On Applied Computing, P 560-566, Coimbra, Portugal, March 2013.

[12] H. Mabed, H. Lakhlef, J. Bourgeois *Fully Distributed Redeployment Algorithm for Multi-Robot System.In: 6th Int. Conf. on NETwork Games, COntrol and OPtimization*, NetGCooP'12. IEEE Computer Society, Avignon, France, 2012.

[13] H. Lakhlef, H. Mabed, J. Bourgeois, *Distributed and Dynamic Mapless Self-reconfiguration for Microrobot Networks*, 12th IEEE International Symposium on Network Computing and Applications (NCA 2013), P. 55-60, Cambridge, MA, United States, 2013.

[14] M. Mamei, M. Vasirani, F. Zambonelli, *Experiments of Morphogenesis in Swarms of Simple Mobile Robots*, Journal of Applied Artificial Intelligence, 8(9-10):903-919, Oct. 2004.

[15] M. D. Rosa, S. C. Goldstein, P. Lee, J. D. Campbell, and P. Pillai, *Programming Modular Robots with Locally Distributed Predicates,* In Proceedings of the IEEE International Conference on Robotics and Automation ICRA'08, 2008.

[16] D. Rus, M. Vona, *Crystalline robots: Self-reconfiguration with compressible unit modules*,Autonomous Robots 10(1), 107-124, 2001.

[17] K. Stoy, R. Nagpal, *Self-Repair Through Scale Independent Self-Reconfiguration*, Proc. IEEE/RSJ International Conference on Intelligent Robotsn and systems, Sendai, japan, 2004.

[18] B. Warneke, M. Last, B. Leibowitz, and K.S.J Pister,K.S.J., 2001, *Smart Dust: Communicating with a Cubic-Millimeter Computer*,Computer Magazine, pp. 44-51, 2001.

[19] P. White, V. Zykov, J. C. Bongard, H. Lipson, *Three dimensional stochastic reconfiguration of modular robots* In: Proceedings of Robotics Science and Systems, pp. 161-168. MIT Press, Cambridge , 2005.

[20] F. Zambonelli, M.P. Gleizes, M. Mamei, R. Tolksdorf,*Spray Computers: Explorations in Self-Organization*, Journal of Pervasive and Mobile Computing, Elsevier, Vol. 1, p. 1-20, 2005.

[21] *Physical rendering simulator (dprsim): http://www.pittsburghintel-research.net/dprweb*.