Distributed and Efficient Algorithm for Self-reconfiguration of MEMS Microrobots

Hicham Lakhlef
FEMTO-ST/DISC, University of Franche Comte
1 cours leprince Ringuet, 25201
Montbeliard, France
hlakhlef@femto-st.fr

Hakim Mabed
FEMTO-ST/DISC, University of Franche Comte
1 cours leprince Ringuet, 25201
Montbeliard, France
hmabed@femto-st.fr

Julien Bourgeois
FEMTO-ST/DISC, University of Franche Comte
1 cours leprince Ringuet, 25201
Montbeliard, France
julien.bourgeois@femto-st.fr

ABSTRACT

Self-reconfiguration for moving MEMS microrobots currently needs a positioning system and a map of the target shape. Traditional positioning solutions like GPS or multilateration are not applicable in the micro-world and maps sharing does not scale. In this paper we present self-reconfiguration method where nodes are unaware of their positions, and where they do not have the final coordinates of each microrobots. In other words, nodes do not store the correct positions that build the target shape. Consequently memory usage for each node is reduced to $O(1)$. This algorithm does not need message exchange to achieve the self-reconfiguration from a chain to square configuration (microrobots are organized in square shape) that represents the optimal logical topology for messages sending and it ensures a *Snap-connectivity*. Obtained results show a rapid convergence of the algorithm. The worst-case time complexity is of n movement rounds, where n represents the number of microrobots. Our algorithm is implemented in the declarative language MELD and executed in the simulator DPRSim.

Keywords

MEMS microrobot,distributed algorithm, self-reconfiguration, physical topology,energy

1. INTRODUCTION

Recent advances in micro and nano-technologies made possible the design and the development of a large variety of Micro Electro-Mechanical Systems (MEMS) which are miniaturized and low-power devices that can sense and act [25, 26]. It is expected that these small devices, that that scaled down to less than 1 mm [10] referred to as MEMS nodes, will be mass-produced, making their production cost almost negligible. Their applications will requires a massive deployment of nodes, thousands or even millions [22] which will give birth to the concept of Distributed Intelligent MEMS[3].

One of the major challenges in developing a MEMS 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, 7, 14], 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 costs. Optimizing the number of movements of microrobots is therefore crucial in order to save energy.

Self-reconfiguration of microrobots is the most useful algorithm and it is used by many applications[11]. Optimizing the energy cost of self-reconfiguration algorithms will therefore have a direct impact on the energy efficiency. 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 form of nodes to reach the optimal logical topology from a physical topology [9]. On the other hand, and in second sense 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[18, 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.

In this work, we focus on a special case of self-reconfiguration where the objective is to optimize the logical topologies such that exchange of messages would be more optimal. We study distributed algorithm of self-reconfiguration for MEMS starting with a form and arriving to a target form while analyzing the memory, exchanging messages complexity and the number of movement rounds.

2. RELATED WORKS

In the literature, 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 or wireless networks to groups called cliques or clusters, where the objective is to organize the various spots where the orga-

SAC'13 March 18-22, 2013, Coimbra, Portugal.

nization is carried by the cluster head that has more amount of energy. While the self-organization term can be found in protocols for sensors networks to form a sphere or a polygon from a center node [13, 24]. The term *redeployment* is also a new term to address self-reconfiguration for sensor networks [12, 17]. For the self-reconfiguration with robots or microrobots there are the protocols [18, 19] where the desired configuration is grown from an initial seed module and a generator takes as input a 3D CAD model of a desired 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.

A growing number of researchers on the self-reconfiguration for microrobots using centralized algorithms are done, among them we find the proposed control algorithms for self-assembly and/or reconfiguration with a centralized manner, see for instance [16]. Other approaches give each unit a unique ID and a predefined position in the final structure; see for instance [23], the disadvantage of these methods is the centralized computing and the need of identities for nodes. More distributed approaches include [5, 8, 20]. In simulation, the authors in [19, 21] have demonstrated algorithms for self-reconfiguration and directed growth of cubic units based on gradients and cellular automata. In [4] the authors have shown how a simulated modular robot (Proteo) can self-reconfigure into useful and emergent morphologies when the individual modules use local sensing and local control rules.

Claytronics, is the name of a recent robotics project by Carnegie Mellon University, in which nanoscale robots called *Catoms*. It is designed by Researchers from Carnegie Mellon University and Intel Corporation that have been working on this project where the idea is to have hundreds of thousands of nano processors (microrobot) form together to create new solid objects in any shape or size. The challenges are vast, but once it becomes a reality it will change the world forever. 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. Within the Claytronics project, many works have already been successfully conducted. In [6] a metamodel for the configuration of catoms beginning from an initial configuration to achieve a second desired configuration using *creation* and *destruction* of catoms, the authors use these two functions due to the inability of motion of catoms in the presence of neighbors that can be considered as barriers. In [7] the authors present a scalable distributed reconfiguration algorithm with the Hierarchical Median Decomposition, to achieve arbitrary target configurations without a global communication, another scalable algorithm is here [14]. In [2] a scalable protocol for catoms self-reconfiguration with the MELD language [1, 15] using the creation and destruction of nodes. In all these works the form target is predefined and the catoms know all corrects position at the beginning and the authors assume that each node is aware of its current position.

3. CONTRIBUTIONS

In this paper, we propose a new approach for MEMS microrobots, self-reconfiguration, where the target form is built incrementally, and each node in the current increment acts as a reference for other nodes to form the next increment, which will belong to the form. We introduce the state model where the node can see the state of its physical neighbor to achieve the self-reconfiguration for distributed MEMS microrobots, using the states the nodes collaborate and help each other. Knowing that the node can get help only from the physical neighbor without global information of the network. We seek in our algorithm to achieve the self-reconfiguration without message exchange except the messages of the tree. Contrary to the existing works, in our algorithm the node has no information on the correct positions (predefined positions) of the target shape, the algorithm does not need to know the size of the network (nodes number). The nodes are unawares of their position on the plan.

We propose here an efficient algorithm for nodes self-reconfiguration where the nodes move by rotation around their physical neighbors. In our paper the MEMS network is organized initially as a chain. By choosing a straight chain as 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, chain of microrobots represents the worst case for messages broadcasting complexity with $O(n)$. The redeployment into a square organization allows to obtain the best case messages broadcasting complexity $O(\sqrt{n})$. The performance of the self-organization algorithm is evaluated according to the number of exchanged messages and the number of movement rounds. Both phenomena (message and movement) impacts on the energy consuming and redeployment time.

To assess the distributed algorithm performance we present the results of the simulations made with the declarative language MELD [1] and the open source simulator DPRSim [27].

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 and movement rounds, and it shows how to generalize the algorithm. Section 6 details the simulation results. Finally, section 7 summarizes our conclusions and illustrates our suggestions for future work.

4. MODEL AND DEFINITIONS

Within Claytronics, a Catom that we call in this paper a node is modeled as a sphere which can have at most six 2D-neighbors without overlapping. 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 representing only one neighbor. A node A is in neighbor's list of node B if A touch physically B. Communications are only possible through contact, which means that only neighbors can have a direct communication.

Consider the connected undirected graph $G = (V, E)$ modeling the MEMS network. $v \in V$, is a node belonging to the network and, $e \in E$ is a bidirectional edge of communication between two physical neighbors. For each node $v \in V$, we

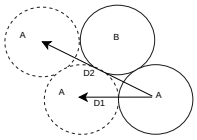

Figure 1: Traveled distance in one round = 2R

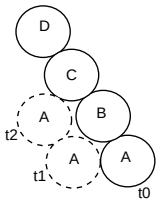

Figure 2: Message transmission

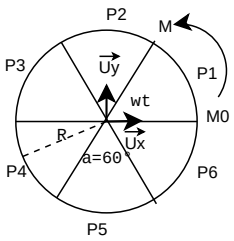

Figure 3: Node modeling

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, $N(v)$. Furthermore, the set $N(v)$ is the unique initial and instantaneous information get by the node. We assume that every node systematically updates the set of its neighbors $N(v)$ after a local change.
We give the following definitions:
Connectivity: a graph $G = (V, E)$ is said connected, if $\forall v \in V, \forall u \in V$, $\exists$ a tuple $C_{v,u} = (e_{v,-}, ..., e_{-,-}, ..., e_{-,u})$, where $e_{x,y} \in E$ is a edge from x to y and $C_{v,u}$ represents a path from v to u.
$Snap - Connectivity$: Let T be the full-time running of a distributed algorithm DA and $t1, t2, ..., tn$ are different instants of DA execution (rounds). Let the dynamic graph $G_t(V_t, E_t)$ the network state at the instant t. DA is said respecting Snap-Connectivity property if $\forall ti, i \in \{1, ..., n\}$, $G_{ti}(V_{ti}, E_{ti})$ maintains the connectivity.
Tree: the tree is any connected graph without simple cycles. In th tree, a node is either a child (leaf) or a parent.

We call *owner movement rounds* of a given node its number of performed movements.In this paper, we present how the node can precacluate the own movement rounds and make sure that it correctly followed the algorithm.

To calculate the owner movement rounds we agree these proposals:

Consider the figures 1 and 3 which represent respectively a neighboring structure and microrobot node model. A node performs a single movement round if the distance between the old and the new positions is exactly twice the radius ($D1 = 2R$).

The node perimeter corresponds to an angle of 360° that can be divided into six equal segments each one of 60°. The perimeter length of a segment with α degree is equal to $P_\alpha = \pi R\alpha/180$. According to figure 3 we prove that $P1 = P2 = P3 = P4 = P5 = P6 = \pi R/3$ and in each round the node travels the same distance.
The movement performed by a node can be represented by the following Cartesian parametric equation:

$$\begin{cases} x(t) = Rcos(wt) \\ y(t) = Rsin(wt) \\ where\, wt \in [0..2\pi[\end{cases} \quad (1)$$

With w is a constant representing the rotation velocity and $x(t)$ and $y(t)$ the coordinates of the M point (see figure 3). The M point represents the contact point between the node in movement and the node around wich it moves. The velocity vector is written:

$$\vec{V} = \begin{pmatrix} -Rsin\ wt \\ Rcos\ wt \end{pmatrix} \quad (2)$$

The arc length from M to $M0$ is equal $\int || \vec{V} ||.dt = Rwt$. So in this paper in one round the microrobot of radius R travels Ra with a=60°.

In this paper, we assume that the message exchange between two physical neighbors is carried without complexity (0 message). By cons the consultation of not neighboring node state, induces message exchange complexity. For example in the figure 2:
- At $t0$, the node A needs to know the state of B to move to a new position (represented by doted circle), which is done without message exchange.
- At $t1$, A is in the new position and it needs to know the state of D. In this case, D informs C of its state, and then C forwards the message to A. So, in this case there is a message exchanging and A must wait 2 rounds to decide.
- If a message has been sent from D to C at $t0$ or $t1$, then A can obtain the state of D at $t2$ with a simple consultation of C's state without messages exchanging.

5. PROPOSED PROTOCOL

5.1 Algorithm with Safe Connectivity (ASC)

As mentioned before, the node can move only around its physical neighbors. To ensure a snap-connectivity of the network, only the leaves are allowed to move.

The distributed algorithm runs in rounds as it is shown in the algorithm hereafter. A priority mechanism is used allowing choosing the most prior satisfied predicate and ignoring the others ones during current round. We notice that in ASC, the state change actions, represented by predicates labeled $P1$, are considered more prior than a movement actions represented by $P2$ predicates. ASC builds the target form by increments. When an increment is completed, its composing nodes are considered as belonging to the final form. At the beginning, only the initiator is set into the final form. Then it helps its neighbors to take corrects positions. In other words, the nodes already in the form act as a benchmark for the neighbor nodes to complete the new layer. To mark its membership to the final form, a node changes it state using the $State_v(well)$ and then it becomes fixed. The first node that can move is the node at the bottom of the chain corresponding to the first leaf of the first tree (with the predicate $moveAroundbad_v(u1, P_{ne})$). The node goes up towards northwest until it reaches the root node by moving around intermediate nodes using the second predicate ($moveAroundbad_v(u1, P_e)$) in ASC. The nodes of the layer under construction may make motion either at left (W) or NW directly using the last two predicates

$moveAroundwell_v(u1, P_{ne})$ and $moveAroundwell_v(u1, P_e)$, given in the algorithm ASC. The node changes its state to *well* when it cannot move to left (W) or to NW. The node uses the predicate
$moveAroundwell_v(u1, P_{ne})$ to move to left(W), which induces that pivot node becomes its NE neighbor. This motion is repeated until that the node reaches the diagonal position and set the *spe* state.

Variables and Predicates
- v, u, $u1$, $u2$: denote a node belongs to the network.
- $\{U\}$:set of nodes.
- $well, bad, spe$: states, a node can take one or two states at the same time, but not *spe* and *bad* or *well* 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 $k = well, bad$ or spe.
- $State_v(s, well)$: the node v has s (s is an integer) neighbors that have the *well* state $State(well)$.
- $moveAroundwell_v(u, P_x)$: move around the neighbor u in such a way that u becomes in the direction x regarding to v.
- $Parent(v, u)$: means 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

$$\text{Initiator}(v) \equiv N_{nw}(v) = \oslash \wedge connected_v.$$
$$\text{State}_v(bad) \equiv connected_v \wedge \neg Initiator(v).$$
$$\text{State}_v(well) \equiv Initiator(v).$$
$$\text{State}_v(spe) \equiv Initiator(v).$$

Predicates checked in each round

$$\text{Parent}(v, v) \equiv Initiator(\text{v}).$$
$$\text{Parent}(v, u) \equiv (Parent(w, v), u \neq w) \wedge neighbor(v, u) \wedge State_u(bad).$$
$$\text{isLeaf}(v) \equiv (\forall u \in N(v), \neg Parent(v, u) \wedge \neg Parent(v, v)).$$
$$\text{(P1.1): State}_v(well) \equiv (N_e(v) = u1 \wedge State_{u1}(well) \wedge N_{ne}(u1) = \oslash) \vee State_v(3, well) \vee (State_v(2, well) \wedge (N_{ne}(v) = u1 \wedge State_{u1}(spe)) \vee (N_w(v) = u1 \wedge State_{u1}(well))) \vee State_v(spe).$$
$$\text{(P1): State}_v(s, well) \equiv (N_x(v) = \{U\}, |U| = s \wedge State_{\{u\}}(well)).$$
$$\text{(P1): State}_v(spe) \equiv (N_{nw}(v) = u1)) \wedge (N_{ne}(v) = u2, State_{u2}(spe)).$$
$$\text{(P2): moveAround}bad_v(u, P_{ne}) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{nw}(v) = u \wedge State_u(bad)).$$
$$\text{(P2): moveAround}bad_v(u, P_e) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{ne}(v) = u \wedge State_u(bad)).$$
$$\text{(P2): moveAround}well_v(u, P_{ne}) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{nw}(v) = u \wedge State_u(well)).$$
$$\text{(P2): moveAround}well_v(u, P_e) \equiv isLeaf(v) \wedge State_v(bad) \wedge (N_{ne}(v) = u \wedge State_u(well)).$$

ASC.

The node can move around the diagonal nodes using $moveAroundwell_v(u1, P_{ne})$ only if the diagonal nodes have not a node at the E direction. All diagonal nodes get *spe* state with the predicate $State_v(spe)$, and with $moveAroundwell_v(u1, P_e)$) the node moves until it takes a correct position. The state changing is more prior than moving actions in order to avoid bad motion. To avoid messages exchanging, the node can change its state to *well* if it has either 3 neighbors at the *well* state ($State_v(n = 3, well)$) or one neighbor at *spe* state and two neighbors in both NE and NW directions.

5.2 ASC and Snap-Connectivity property

ASC maintains the snap-connectivity property since it uses the tree mechanism where only the leaves can move. Since only leaf nodes can move, moved nodes will not generate a hole between nodes connected through it. We can divide the move impact into two categories. A first case happens when no new neighbor is appearing after the movement. In this case there is no ti when the message cannot be sent because during the motion it was being always the neighbor of the node used it to move. The second case appears when the moved node gets a new neighbor. In this case, before it let its neighbor it becomes a neighbor node with another node that is connected since the graph was connected at beginning, so there is another route for the message of $C_{v,u}$ which will not be blocked for all ti, $i \in \{1, ..., n\}$.

5.3 Movement rounds complexity analysis

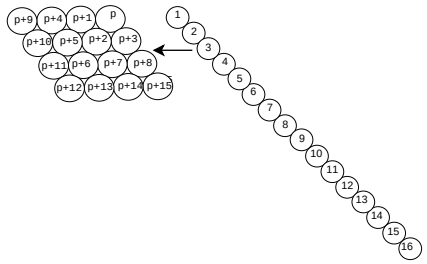

Figure 4: Nodes positioning into the final square shape

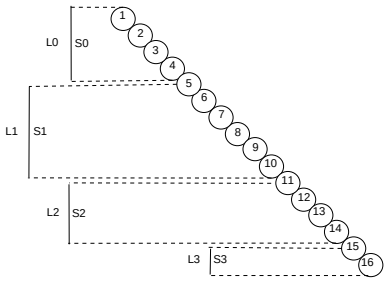

Figure 5: Nodes partitioning into levels

To form the NxN square matrix, starting from n nodes ($\sqrt{n} = N$), we use an incrementally process with a single node that we assume in a correct square 1x1. Then we add, each time, a new layer that 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. For example to reach the square 2x2 we have to add a new layer with three nodes. Considerer the figure 4, the node i will, at the end, take the position $p+x$. Following the path from up to down the node

i still fix until that all node $j > i$ been moved. Consequently, if A is before B, and node A takes the position $p + c$ and node B takes the position $p+k$. We deduce then that $c > k$. The number of nodes into each layer follows an arithmetic sequence:

$$U_j = U_{j-1} + 2. \tag{3}$$

Where: U_j is the number of nodes in the layer j and U_{j-1} is the number of nodes in layer $j-1$. Now, the nodes of the initial chain are partitioned and a rank or level is associated with each subset. The nodes partitioning is obtained as follow: the first $\sqrt{n}$ node on the top of the chain get the level 0. Then the next $\gamma = (2\sqrt{n}-2)$ nodes are assigned the level 1. Level 2 is assigned to the following $\gamma = \gamma - 2$ nodes and so on (Figure 5 shows en example). We notice $L(i)$ the level of the node i.

The number of movement rounds for each node i of level j can be given with the composition of two sequences $U_{i,j}$ and S_j.

$$S_j = \begin{cases} 0, \, if \, j = 0. \\ 2\sqrt{n} - 5, \, if \, j = 1. \\ S_{j-1} - 2, \, otherwise. \end{cases} \tag{4}$$

With S_j is a number associated to nodes that have the level j.

$$U_{i,j} = \begin{cases} 0, \, if \, j = 0. \\ 2, \, if \, i = \sqrt{n} + 1 \wedge j = 1. \\ U_{i-1} - S_j, \, if \, L(i+1) \neq j. \\ U_{i-1} + 2, otherwise. \end{cases} \tag{5}$$

Where $U_{i,j}$ and U_j represent respectively the number of movement rounds of the node i of a level j and the number of movement rounds of each node of a level j.

THEOREM 5.1. *The highest number of movement rounds induced by ASC is equal to n.*

Special case
We study here the case where the number of nodes n is not a square of an integer number. To calculate the owner movement rounds of a given node we consider a similar partitioning system as before. The highest number of movement rounds to reach the final form remains n.
Let $q = \lfloor \sqrt{n} \rfloor$, and $diff = n - q^2$.

$$U_{i,j} = \begin{cases} 0, \, if \, i \leq q. \\ diff + 2q - 1, \, if \, i = q + 1. \\ U_{i-1} - 2, \, if q + diff \geq i > q \\ diff + 2, \, if \, i = q + diff + 1. \\ U_{i-1} - S_j, \, if \, i > q + diff \wedge L(i+1) \neq j. \\ U_{i-1} + 2, otherwise. \end{cases} \tag{6}$$

5.4 Three states necessary and sufficient

ASC proceeds using three different states *(well, bad, spe)*. In this section we prove that three states are a minimum to obtain the algorithm convergence. Obviously, with a single state, nodes have no way to distinguish whether they are in a good position or not and therefore if the node should move or not. Supposing we seek a distributed algorithm using two states, with two states a node knows whether it is in a good position or not. Node moves to West or North-West around nodes that have a *well* state (figure 6.a). However, at the early rounds, the nodes forming the diagonal will be authorized to move around nodes having the state *well* and have not a neighbor at NE and NW direction, wich will give another chain, and we loss the aim see figure 6.b) .In other hand, in the algorithm the node can change its state to *well* if it has three neighbors having the *well* state, but the node that is in W direction from the diagonal node cannot move to NW and has only two neighbors having *well* state, and it must change its state. There is no benchmark to distinguish those situations only with adding a special state to nodes forming the diagonal. On the other hand, to avoid message exchange, the node can change its state to *well* if it has three neighbors having *well* state, but while building the new layer diagonal node has only two neighbors having *well* state while the same case are other nodes not illegal to change its state, so these can be solved by adding this *spe* state. A third state is essential to differentiate the node that can move around a node having state *spe*. Therfore, it can move around it, if and only if it has no right neighbor (figure 6.c).

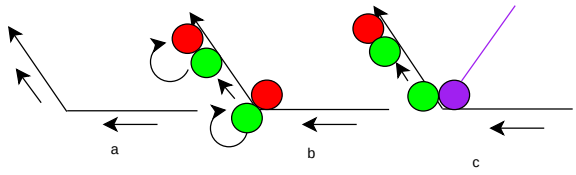

Figure 6: States of nodes

5.5 Null complexity of sent messages

The most interesting action for message changing in the algorithm is the one activated by state changing predicates, from the *bad* state to *well* ($State_v(well)$), it is obvious that if a node changes its state before it be sure of the good state of other nodes that have moved before it in the current layer, the process will completely go in the opposite direction of the desired objective and self-reconfiguration desired, the predicate $State_v(well)$ ensures without exchanging of message that the node changes its state only if all nodes that have moved before it have changed their states, 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 $State_v(well)$ can have this information by simply consulting (message) the state of its former neighbors. In other words, the message was being sent before that the node needs to know the state of its sender, when the node needs to know it, the node will find the message at its physical neighbor. So we do not need to transmit information from the node blocked necessarily in a good position with the *well* state to other nodes which are forming the new layer which explains that throughout the algorithm in any case we do not need to transmit information between two non-neighboring node of the new layer, this efficiency is

explained by the fact that synchronization in state changing is not required for nodes that are in the same layer. We would note that we do not count the messages for the tree construction of ASC.

5.6 Generalization of the algorithm

Presented ASC is specific to a chain case where nodes form initially a straight line oriented toward SE-NW directions. In this section we describe how the algorithm can be generalized to any kind of initial straight chain with any direction as shown in figure 7. We start by explaining how the initiator (root) node is selected whatever is the direction of the straight chain. For this end, the node with only one neighbor situated either in SW, SE or E direction is chosen as root node. Every node in the chain can deduce the orientation of the chain (one of the three cases represented in figure 7) by analyzing the orientation of its neighbors. For example, if a node corresponds to an extremity node (one neighbor) where the direct neighbor is on the E side, the node deduces that the straight line is oriented E-W. The same thing is happened on the middle nodes, which use the orientation of their two neighbors to determine the orientation of the formed chain. Generally, every node after the detection of the chain orientation, noted D-$\overline{D}$, runs a variant of the ASC algorithm depending of the orientation $D \in \{W, NW, NE\}$. The variant of ASC algorithm, ASC^D, represents an adaptation of the the original ASC algorithm (corresponding to ASC^{NW} to the two other possible orientations. For instance if the initial chain is oriented NE-SW, the algorithm ASC^{NE} is called, and the square form is realized using moves of type $moveAroundbad_v(u1, P_w)$, $moveAroundWell_v(u1, P_w)$ and $moveAroundwell_v(u1, P_{nw})$. The usage of these three predicates is described in figure 8.

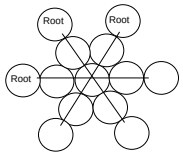

Figure 7: The three possible cases of a straight Chain.

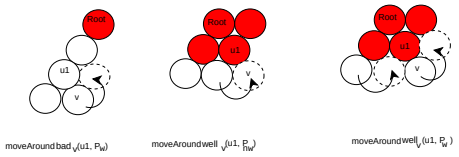

Figure 8: Moves adaptation in the case of NE-SW chain. Dark nodes represent the well-state node.

6. SIMULATION

We have done the simulation with the declarative language Meld that uses the DPRSim simulator. In our simulations the radius of the node is 1 mm[1]. We simulated with a laptop with processor Intel(R) Core(Tm)i5, 2.53 Ghz

[1]The time of one movement round depends on the size (the diameter) of the microrobot, as shown in section 4.

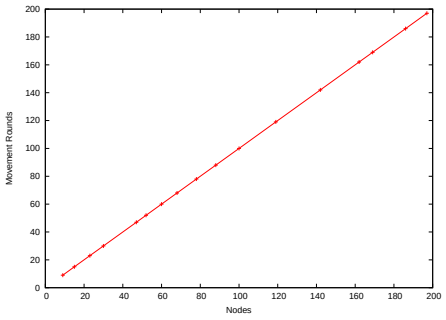

Figure 9: Highest movement rounds

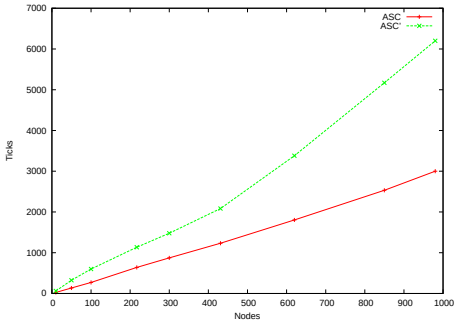

Figure 10: Time of execution

with 4 Go of memory. The results of these simulations come to agree the results obtained previously, in particular as regards the owner movement rounds, figure 9 represents the number of movement rounds by number of nodes and figure 10 represents the execution time in ticks by the number of nodes, with counting the tree and without counting the tree. We see that the time of the tree has an impact on the total time of the algorithm.

In the curves that represents the movement rounds, we remark for some values of the network size n, the number is always is n. For the curves that represent the times of execution figure 10, without counting the time of construction of the tree of ASC we see that if the number of nodes increases the time increase. We see, if we count the time of the tree ($O(n)$ time), the time execution of the algorithm increases dramatically.

7. CONCLUSION

In this paper, We proposed a distributed and efficient algorithm for Self-reconfiguration of MEMS microrobots. We presented a new method to complete the self-reconfiguration where the nodes do not know the fixed positions of the target form but only the aimed shape. We have presented a protocol that ensures the connectivity throughout the execution time of the algorithm. The proposed algorithm is characterized by a linear time complexity regarding to the system size (microrobots number) with a constant memory needs. Messages exchange is limited to neighboring consultations except the messages of the first tree. Consequently system reconfiguration is quite fast.

However, some open problems remain. We study the conception of an energy-efficient algorithm when the starting form may be any connected shape, in wish we predict the

loss of these previous characteristics found in this paper, in particular the number of states of each node and the messages exchanging. In our algorithm, the number of owner movement rounds is calculated (predicted) according to the size of the network. The question is then to know if it is possible to calculate the owner movement rounds without knowing the size of the network. finally, is it possible to generalize the idea of this paper to any targeted shape?

8. ACKNOWLEDGMENTS

This work is funded by the UMR CNRS 6174. The authors wish to express their appreciation to the three anonymous referees for their constructive comments.

9. 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 International Conference on Logic Programming (ICLP '09), July, 2009.

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

[4] H. Bojinov, A. Casal, T. Hogg, *Emergent structures in modular self-reconfigurable robots,* Proceedings 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 Proceedings of IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems IROS '08, September, 2008

[7] S. Funiak, P. Pillai, M. P. Ashley-Rollman, J. D. Campbell, and S. C. Goldstein, *Distributed Localization of Modular Robot Ensembles,*In Proceedings of Robotics: Science and Systems, June, 2008.

[8] C. Jones, M. J. Mataric, *From local to global behavior in intelligent self-assembly.In: Proceedings of the 2003 IEEE International Conference on Robotics and Automation*,ICRAM 2003, vol. 1, pp. 721-726. IEEE Computer Society Press, 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] K. Kotay, D. Rus, M. Vona, and C. McGray, *The Self-reconfiguring Robotic Molecule*,in Proceedings of the 1998 IEEE International Conference on Robotics and Automation, Leuven, 1998.

[12] F. Kribi, P. Minet, A. Laouiti, *Redeploying mobile wireless sensor networks with virtual forces*, IFIP Wireless Days, Paris, France, December 2009.

[13] 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.

[14] R. Ravichandran, G. Gordon, and S. C. Goldstein: *A Scalable Distributed Algorithm for Shape Transformation in Multi-Robot Systems,* In Proceedings of the IEEE International Conference on Intelligent Robots and Systems IROS '07, October, 2007.

[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] R. Soua, L. Saidane, P. Minet, *Sensors deployment enhancement by a mobile robot in wireless sensor networks*,IEEE ICN 2010, Les Menuires, France, April 2010.

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

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

[20] W. Shen, P. Will and A. Galstyan, Hormone-inspired self-organization and distributed control of robotic swarms. Autonomous Robots 17(1), 93-105, 2004.

[21] K. Stoy, *Using cellular automata and gradients to control self-reconfiguration*, Robotics and Autonomous Systems 54(2), 135-141, 2006.

[22] 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.

[23] 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.

[24] 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.

[25] *http : //www.stanford.edu/class/mems/ee321*

[26] *http://www.xs4all.nl/ganswijk/chipdir/m/sensor.htm*

[27] *http://www.pittsburgh.intel-research.net/dprweb.*