SOMIK: System Of Mobile Interactive Knowledge

Marat Fayzullin
University of Maryland
fms@cs.umd.edu

December 29, 1999

1 Motivation

Over the years of human history, man-made systems reached higher and higher levels of complexity, as we wanted them to perform more and more complicated tasks. The predominant way humans use to build their machines is to construct them as more or less rigid structures of modules, each performing a predefined task, with a minimal number of connections between modules. In fact, this modular structure is considered necessary by the rules of correct system design taught to engineers, computer programmers, and other technical types.

Why do we always choose such a rigid structure for our machines? Clearly, the main virtue of a rigid system is that it is easy to comprehend and troubleshoot if a problem occurs. Modules can also be reused in different machines, given that they perform general enough functions. Rigid systems have their problems though:

- The number of elements in the system rises proportionally to the complexity of the task. If the task complexity rises exponentially, then so does the number of elements.
- The probability of a fault rises exponentially as the number of elements rises linearly, because of possibly faulty connections between elements.

Based on those problems, one can deduce that designing a complex yet reliable machine becomes a difficult task. Indeed, in current designs reliability is often achieved through complete duplication of functions, which is very costly and raises complexity even further.

Because of this complexity crunch, it seems likely to me that rigidly designed systems will sooner or later give way to other designs. To get an insight into what those designs may be, let us look at Nature, which was doing its own design and optimization for millions of years. By looking at biological organisms, we can see several main differences they have from the man-made systems:

- Functions are distributed among many components.
- Components are similar and simple.
- Components are strongly interconnected.
- Connections between components may change.
- Self organization is done through minimization of some function.
- Goal achieved through minimization of some function.
- "Best effort" performance.

In this paper, I am proposing a basis for a man-made system design exhibiting features observed in biological systems. The system will consist of many identical or similar nodes communicating with each other to achieve some goal defined by the rules that have to hold the observed state of the environment. A violated rule will cause the system to act toward correcting the environment state.

2 Overall Design

Let us first define a *node* which has following features:

- Some processing capabilities.
- Some means to change node's location in space.
- Some sensors whose readings constitute node's *local state* S.
- Storage for the *observed state* G.
- Set of predicates $r_1 \dots r_n$ on a given location, known as *rules*, with attached *actions* $a_1 \dots a_n$.
- Transceiver for communicating with other nodes with controlled level of transmission signal v.
- Location of node in space l.
- Time source t.

The transceiver can only be used to broadcast messages and listen to messages broadcast by other nodes, i.e. it is not suited for directed transmissions. There are three types of messages in the system:

1. S*-message* carries the local state of a node.
2. $R(a, l_{act})$*-message* carries a request to perform action a on location l_{act}.

3. $A(a, l_{act})$-*message* carries an acknowledgement of action a being performed on location l_{act}.

Nodes have no IDs of any kind, but each message also carries $< l, v, t >$ of a node that sent it. When a message is received, its incoming signal strength v' and reception time t' are added to produce $< l, v, t, v', t' >$. At this point, we will postpone the discussion of time synchronization and location discovery by nodes.

It is worth noting that in our system node's location effectively acts as its ID, the only attribute that makes it different from other nodes. Each system action comes with a *location* at which this action has to be executed. This closely resembles real life, where actions are always applied *to something, somewhere.*

We also *do not* require each node to hear *all* other nodes. In fact, such a capability may be harmful because of scarce communication bandwidth and limited processing capabilities of nodes. It has been shown that biological systems follow the *locality principle*, where each element only knows about its immediate neighbors.

3 Node Operating Cycle

Each node repeatedly goes through an *operating cycle* during which it exchanges information with its neighbors and decides on actions to take:

1. Broadcast local state using S message.
2. If executing action $a(l_{act})$, broadcast $R(a, l_{act})$ message.
3. Listen for incoming messages collecting them into three sets M_S, M_R, M_A.
4. Interpolate node location l based on $< l, v, t, v', t' >$ data from received messages.
5. Generate observed state

$$G = S + < l, v, t > + \sum_{j \in M_S} (S_j + < l, v, t, v', t' >_j)$$

6. Set $M_O = \emptyset$.
7. For each $r_i(G, l_i) = true$, add $R(a_i, l_i)$ to M_O.
8. If executing action $a(l_{act})$ then
   (a) If there is $A_j(a, l_{act}) \in M_A$ such that $d(l, l_{act}) > d(l_j, l_{act})$, stop executing $a(l_{act})$.

   else

   (a) Select $R(a, l_{act}) \in M_R \cup M_O$ such that

- $a(l_{act})$ is executable
- $d(l, l_{act})$ is minimal
- There is no $A_j(a, l_{act}) \in M_A$ such that $d(l, l_{act}) > d(l_j, l_{act})$

(b) Remove $R(a, l_{act})$ from its original set.

(c) Start executing $a(l_{act})$.

9. For each $R(a, l_{act}) \in M_O$ such that $A(a, l_{act}) \notin M_A$, broadcast $R(a, l_{act})$ message.

As can be seen, the node's first obligation is to inform surrounding nodes about things node sees and action it executes (1,2). For simplicity, we will assume that *each node can only execute one action at a time*, and *each action requires only one node to execute it.*

After sending its vital state information, the node goes into the listening phase, collecting all received messages (3). At the end of this phase, the node tries to discover/clarify its own location (4) and create a "world view" G from this location. It seems logical to use a hierarchical data structure (quadtree?) to store G, "fusing" received sensor readings from lower levels of the tree to the higher levels.

At the next stage of the algorithm, the node goes through the observed state G, checking whether any of $r_1 \ldots r_n$ predicates are satisfied at any locations. Hierarchical G structure should make this procedure relatively inexpensive. Satisfaction of a predicate r_i at location l_i means that conditions at this location strayed from the norm, and our system must correct them. Therefore, we generate a *request for action* message $R(a_i, l_i)$ (6,7).

If our node is not busy executing an action, we should now go through all received (M_R) and generated (M_O) R messages and see whether there is an action request we want to satisfy (8). Of course, this means that multiple nodes may decide to execute the same action at the same location. This is not a problem for the system though. Each node executing an action constantly checks for action acknowledgement messages generated by other nodes (8) and aborts if there is a *better positioned* node executing the same action.

As it was decided that the system will be based strictly on the spatial differences between otherwise identical nodes, the node's position is "better" if it is closer to the action location.

Finally, the node sends out action requests it generated, that haven't yet been satisfied by any other nodes in the vicinity (9).

4 Location Discovery

As was said earlier, the nodes in our system do not have any IDs. They are differentiated solely by their location in space. It was not explained though how a node, with its meager amount of on-board hardware, can determine its own location.

When a node receives some message j, it obtains the $< l, v, t, v', t' >_j$ structure from that message, where

- l_j is a location of the sending node
- v_j is the original transmission volume at the sending node
- t_j is the transmission time at the sending node
- v'_j is the reception volume at the receiving node
- t'_j is the reception time at the receiving node

The node location l can be interpolated based on locations l_j of sending nodes and distances to them. The distance to a sending node $d(l, l_j)$ can be computed in two ways:

1. $d(l, l_j) = c(t'_j - t_j)$ where c is the signal propagation speed. In our case $c = 3 \cdot 10^8 m/s$ i.e. the speed of light.
2. $d(l, l_j) = k\sqrt{v_j/v'_j}$ where k is some constant, as the energy of a transmission decreases proportionally to the squared distance from the point of origin.

Let us look at the first method of computing $d(l, l_j)$ and say that we want to determine the distance with precision of $1cm$:

$$10^{-2}m = \delta_d = c\delta_t = \delta_t \cdot 3 \cdot 10^8 m/s$$

$$\delta_t = (1/3) \cdot 10^{-10}s$$

i.e., effectively, the node needs a clock frequency of 30GHz, and this clock has to be synchronized with the clock at the sending node. This, of course, is unrealistic. Thus, the first method doesn't look too useful for us, although it cannot be ignored.

Now, consider the second method of computing $d(l, l_j)$. This method is local, as each node only hears the nodes in its vicinity. It is much more precise than the propagation delay method though, because of the fact that v'_j changes sharply with the distance.

The *location discovery* algorithm for a node would then look in a following way:

1. $D = \emptyset$
2. For each received message j, if $v'_j < v_j$ then add $< l_j, k\sqrt{v_j/v'_j} >$ to D.
3. Interpolate l based on D

To determine its location, the node needs to know locations of at least three neighbors located in different directions from the node, preferably at 120^o from each other. As our nodes are mobile, we cannot guarantee that such configuration always occurs. Therefore, the system requires to have some nodes with known fixed positions acting as a coordinate grid to which mobile nodes are tied. The sparseness of this grid is one of the factors affecting system reliability.

Assume that each node can hear neighbors in the d_{max} radius. Then the optimal grid will have triangular cells (120^o), with vertices at the distance of

$$d_{base} = 2d_{max}cos(60^o) = d_{max}\sqrt{3}$$

from each other. How can we create such a grid? There are two ways:

1. Put a grid of dedicated stationary nodes increasing their transmission power to make grid more sparse.

2. Make idle mobile nodes converge on the grid vertices and stay there serving as grid nodes.

Each approach clearly has its strong and weak points. The first approach may be too expensive and vulnerable, making grid nodes "sitting ducks". Also, increasing transmission power allows for a more sparse grid but makes distance measurements less precise.

The second approach requires more dense grid and may tie up too many mobile nodes. It still requires at least a few stationary nodes to start interpolation from.

It seems to me that the optimal solution would involve a few (exactly three?) powerful stationary nodes surrounding system's deployment area and some mobile nodes acting as temporary grid centers.

There should also be a way for a *lonely* node that cannot determine their locations to find other nodes, to be discussed later in this paper.

5 Spatial and Temporal Organization

So far, it has been assumed that the media used to broadcast messages between nodes has infinite capacity. No matter how loudly each node shouts and how many of them are there, all messages will be heard by any node in the transmission range.

Unfortunately, this is not a realistic assumption. Electromagnetic waves are perhaps the only practical way of broadcasting in real-life environment, so we can assume for now that our system uses radio transceivers to communicate. But two entities cannot broadcast on the same radio channel in the same time, as their signals will interfere and scramble each other. Therefore, there should be a protocol to share the radio channel between its users.

The solution, used in digital wireless telephony and other applications, is a TDMA scheme with a common transmitter sending out marks or "cells" to a channel [1]. A sending node will wait for an empty cell and fill it up with its own message. The same TDMA cell marks can be used to synchronize node clocks.

Of course, this limits the capacity of a channel to a certain number of cells of a certain size, in a certain time. The cell allocation algorithm is also a problem on its own, but it is out of this paper's scope.

5.1 Scattering

When using TDMA, it may well happen that a node will not be able to send all the messages it wants to send to the channel in required time. It seems logical to at least prioritize messages, sending unique S and A messages first, and then all the R messages that fit, hoping that some other nodes will duplicate missing R messages.

In the same time, when receiving messages, a node may get so many of them that node's internal buffers are overrun and excessive messages have to be dropped.

But what is the actual meaning of the channel cluttering for us? A node that fails to send or receive messages because of the channel cluttering is not going to kill our system, as there is a crowd of other nodes around (why would the channel be busy otherwise?). It is clear though that this particular node becomes useless in this particular point of space.

Consider what happens with a biological system, such as a colony of germs, mice, or roaches, when its living space becomes too crowded. The colony then spreads out in space, with each member trying to attain the most comfortable position, not too crowded, but not too empty either. By growth, the colony also achieves bigger area of influence. Isn't it a desirable property for our system, too? I think it is.

I will call the process of spreading the system in space *scattering*. Each node not currently executing any action should scatter, i.e. try to maintain optimal distances from its neighbors located by their S messages. Nodes at the edge of the crowd would then move away from the crowd, while nodes inside the crowd would try to maintain equal distances from their neighbors.

Finally, there is a question of what would a node do if it cannot change its location, but the channel is still cluttered. This may occur because of several reasons:

- A node is moving to execute an action.
- A node has reached a stable position at equal distances from all its neighbors.
- A node is immobilized in some way.

In any of these cases, a node can decrease its transmission volume hoping that the rest of nodes around it will do the same. This, in effect, will decrease channel cluttering while also decreasing connectivity between nodes. This behavior may lead to a potentially messy situation when node a hears node b, but b does not hear a, because a reduced its transmission volume.

5.2 Gathering

Imagine now an opposite situation of a node finding the channel almost or completely empty. Unless such "lonely" or "lost" node is executing some action, it is of not much use for us, as it can't communicate with the rest of the system. Even worse, a lonely node may not have enough data to reliably determine own location.

The first obvious action for a lonely node to take would be to increase the transmission volume, hoping that the other nodes scattered around it will do the same.

If pumping up the volume hasn't produced any improvement though, the node has to find where the rest of the nodes are. We will call this behavior *gathering* and note that it is exactly opposite to *scattering*. There may be several different ways of gathering:

- Define a location l_c in space, which is a logical *center* of the system, and make every idle lost node move toward l_c. This strategy is good for systems whose overall location in space does not change with time (security systems, etc.).
- Define multiple locations $l_{c0} \ldots l_{cn}$ and make every idle lost node move toward l_{cj} closest to the node's last known position. This strategy is good for systems which have a fixed location but take wide areas of space (agricultural systems tied to a certain field, etc.).
- Make each node memorize place with the biggest concentration of nodes and move toward that place when a node is lost or lacks company. This is a good strategy for mobile systems, where entire collection of nodes may shift to a new location over time (i.e. systems performing search).

6 Applying Control And Getting Response

Given that the proposed system is highly distributed, it remains a question of how we can set tasks for it and get some useful responses.

In many cases we do not need to apply any control after the system is deployed. In other cases (sometimes coinciding with the first), we do not expect any explicit feedback from the system. Consider, for example, an agricultural system where each node carries some amount of chemicals and releases these chemicals when and where required by conditions in the field. After deployment of such a system, it can be left to run on its own until nodes exhaust their batteries and supplies of chemicals. The nodes can then be harvested from the field the same way golf balls are collected from a golf court.

Let us now consider a case where we *need to control* our nodes after their deployment. Earlier in this paper, it was stated that the TDMA communication scheme employed by the system requires a centralized transmitter sending cells. Control can be easily applied to the system by filling some of these cells with S messages changing system's behavior. We can even change the system itself by transmitting specialized P messages with new sets of rules and action algorithms.

Getting response from the system is a more complicated task though. While we can rely on some remote (and presumably safe) omnipotent transmitter to send control messages, response messages have to originate from the system itself. In a conventionally designed system, that would require dedicated node(s) whose responsibility is to talk to the system user. All responses would then be forwarded to the dedicated nodes and then retransmitted outside with greater strength.

The dedicated node strategy is clearly undesirable for our design though. The goal, stated in the beginning of this paper, is to create a system of *similar* or *identical* components. Having dedicated retransmission nodes defeats this goal. Similarly to the positioning grid centers, retransmission nodes become sitting ducks, a vulnerability for the system as a whole.

More appropriate idea would be to make *all* nodes work together to send responses. Each node has only limited transmission range, but if we can make them all transmit the same message in the same time, the entire system becomes a huge transmission antenna.

To implement this principle, let us introduce B messages containing information to be sent out of the system. On reception of a yet unseen B, each node stores this message and rebroadcasts it. Thus, the system gets flooded with B messages, and they get stored on as many nodes as possible. When the user wants to get response from the system, he uses TDMA cell transmitter to send an S message containing the time to send the response. When a TDMA cell corresponding to that time comes, each node will send all B messages it accumulated so far, in the order of original creation times. After sending all B messages, the node flushes them from its internal buffers and the cycle repeats.

It has been shown that with the right encoding, the distributed antenna array has an ability to send large amounts of information using narrow band signals [4]. This is clearly beneficial for our system, as the narrower band transmission consumes less power and is less likely to be detected.

7 Reliability Analysis

To understand the reliability of the proposed design, let us first look at how reliability is assessed in the traditional rigid designs. Theoretically, we have to find fault probabilities for each element of the system and use them to compute the overall fault probability. This may be a very difficult problem though, as the designer has to know how elementary faults are related to each other, whether they are independent, mutually exclusive, or causing one another. So, in practice, it is easier to measure how frequently system fails and then duplicate the system enough times to bring the overall fault probability below the required limit. This duplication is very wasteful, but it achieves the goal.

In the proposed distributed system, all components of the system are the same, given that each component has the same set of rules and actions. Therefore, it should be much easier to predict the fault probability in our system.

Consider that we have a system of uniformly distributed $n\ nodes/m^2$ with communication and sensing ranges of $r\ meters$ and want to compute how many nodes will react to an environment change at some location. Responding to a change, the system goes through three phases:

1. *Detecting a Change*
   Assuming that a node successfully detects a change in the environment with p_{det} probability, the number of nodes that detect the change will be
   $$N_{det} = p_{det} n \pi r^2$$
2. *Informing of a Change*
   All nodes that detect a change are going to send action requests. Assuming that a node succeeds to receive an action request with p_{rec} probability, the probability of being informed by j signaling neighbors becomes
   $$p_{inf} = max(p_{rec} j, 1) = 1 - (1 - p_{rec})^j$$
   To find out the actual number of nodes receiving requests, we integrate:
   $$N_{rec} = 2n\pi \int_0^{2r} (1 - (1 - p_{rec})^{p_{det} n S}) x dx$$
   $$N_{rec} = 4n\pi r^2 - 2n\pi \int_0^{2r} (1 - p_{rec})^{p_{det} n S} x dx$$
   where S is the intersection between the reception area of a node and the area from which all action requests originate:
   $$S = 2r^2 acos(\frac{x}{2r}) - x\sqrt{r^2 - \frac{x^2}{4}}$$
3. *Reacting to a Change*
   At last, the nodes that receive the request must act on the change. Assuming that a node acknowledges action request with p_{act} probability, the number of nodes that react to the change will be
   $$N_{act} = p_{act} N_{rec}$$

As it is assumed that only one node is required to execute any action, the goal is to have $N_{act} \geq 1$. With given failure probabilities, this can be done by choosing high enough node density n.

In reality, the uniform distribution is not guaranteed, but may be achieved by using scattering behavior discussed in this paper. The proposed approach also doesn't consider that nodes are discrete. It is assumed that the node density n is high enough to view the problem as continuous.

8 Applications

So far, the proposed system was described in very general terms, without making clear what practical actions nodes can execute and how given system design can be applied to the real life. In this section, you will find several examples of such applications, from conventional to somewhat futuristic. It is necessary to say though, that one of the goals of this paper was to provide a design one can use *in practice, today, with current technologies.* In spite of somewhat fantastic application examples, I believe that there are enough cheap technologies available today to make all mentioned applications practical.

8.1 Data Acquisition Systems

A *distributed data acquisition system* is basically exactly the same as the "abstract" design proposed in this paper. It consists of many tiny devices with sensors spread over an area from which we intend to collect information. The devices have limited sensor and transmission ranges. Their goal is to accumulate sensor readings and send them back to the system user. Such a system may be used for many different purposes:

- First, there is a military use, for eavesdropping on the enemy, providing battlefield information, or treaty control [2]. The devices can be spread by detonating a cluster bomb, creating a difficult to discover and even more difficult to destroy intelligence grid of thousands elements.
- Second, information gathering systems can be used in agriculture. Imagine a field seeded with sensors reporting temperature, humidity, acidity, and other vital parameters across the entire field.
- Third, the system can be used in science, such as meteorology, to keep track of atmospheric conditions [2]. For example, instead of a single weather balloon, meteorologists can use multiple balloons with sensors anchored at different locations and returning data on the large volume of space.

Distributed data acquisition system nodes would have following characteristics important for making hardware/software choices during the design:

- Nodes are usually not mobile.
- Nodes are usually expendable.
- There are many nodes.
- Nodes are assumed to be quite sparse, requiring relatively long (comparing to the total deployment area) transmission range.
- Power source of each node has a limited capacity.
- Ability to extract power from the environment is beneficial.

8.2 Transport Systems

A *distributed transport system* involves a number of automated vehicles (trucks, mechanized carts) transporting parts to/from the construction line. Traditionally, there would be dedicated (and manned) trucks for each stage of the construction line.

It seems natural to make each truck a node of a distributed system and allow it to carry any kind of parts. We can then populate the construction area with enough trucks to satisfy all the transport needs.

Distributed transport system nodes would have following characteristics important for making hardware/software choices during the design:

- Nodes are highly mobile.
- There are few nodes.
- Power source of each node has virtually unlimited capacity, as far as communications are concerned.

8.3 Robots and Construction Machinery

Consider a robotic hand with five fingers. Each finger has some sensors and servos. Now, let us create a system of five nodes, one for a finger, and attach each finger's sensors/servos to an appropriate node. Note that the nodes should not necessarily be located at the fingers. In fact, "nodes" can live on a single computer simulating broadcast communications between them. By letting close fingers "talk" to each other, we obtain spatial coordination between them. Spatial coordination is very difficult to achieve in traditional designs.

In fact, this example can be extended to any piece of construction machinery consisting of many parts. Wire each part's sensors and motors to a separate node, and parts get ability to organize their actions between themselves. To increase reliability, multiple nodes can be used to drive each part of the machine.

Robot control nodes would have following characteristics important for making hardware/software choices during the design:

- Nodes can be "virtual", residing in a centralized control unit.
- Communication bandwidth is limited, as we are simulating broadcasts on a traditional sequential computer.
- Location discovery is trivial, as nodes represent more or less rigidly moving elements.
- Power supply is not a problem.

8.4 "Compugrapes"

Some applications of the proposed system design may not, in fact, involve any interaction with physical environment at all. Imagine a small spherical node, known as a *"compugrape"*, about the size of a real grape. The node lacks any sensors, motors, or mechanical appendages. It does contain memory and a CPU. To perform a computation, the user pours a bunch of compugrapes into a bucket, sends the initial input and the distributed algorithm code using an external transmitter, and then periodically checks whether the result is ready.

As each node still has broadcasting abilities, such a "parallel computer" has a great advantage in connectivity over current parallel designs. It is also easy to scale: just throw in more nodes.

Compugrapes would have following characteristics important for making hardware/software choices during the design:

- Nodes are not mobile.
- Nodes don't need to be aware of their absolute positions, just positions relative to each other.
- It would be beneficial to find a way to provide power externally (microwave?), but recharging will also work.

9 Related Works

Today, there are quite a few places researching systems of miniature self-organizing nodes. Most researchers use approaches sufficiently different from the one proposed in this paper though. Also, the bigger part of research seems to be conducted on the engineering side, while ignoring the computer science side of the problem. Below is a list of several projects that may be of interest in relation to this paper:

- *Smart Dust Project*, University of California at Berkeley
  This project, headed by Dr. Kris Pister, is an attempt to create a system of dust-sized nodes (known as "motes") for data acquisition [6]. Because the small size limits power consumption to $10\mu Wt$ [3], micro motes cannot use radio communications. Instead, it is proposed to use miniature steered mirrors to reflect laser beam from an external source. Unfortunately, this limits the system to directed communications. The project also involves designing mote delivery mechanisms (miniature radio controlled aircraft proposed) and larger "macro motes" with radio capability and bigger processing power.
- *Multi-Antenna Wireless Communications*, University of California at Berkeley
  Dr. Joe Kahn, who is also participating in the Smart Dust Project, has coauthored several papers on encoding data for transmission with multi-element antennas [7].

- *Micro Motion Systems*, Instrumentation Laboratory, Royal Institute of Technology, Sweden
  Thorbjorn Ebefors, Johan Ulfstedt Mattsson, Edvard Kalvesten, and Goran Stemme are working on the micro motion systems based on arrays of movable silicon legs [8]. The legs are raised using polyimide micro-joints shrinking with temperature. Current prototypes of walking microrobots reach speeds of $12mm/s$ and are able to carry a mass of $2g$ [8].

- *Digital Micro-Propulsion Project*, California Institute of Technology
  The Digital Micro-Propulsion Project is a joint venture of DARPA, TRW, The Aerospace Corporation, and Engineering Design Research Laboratory at Caltech with the goal of producing a miniature thruster system for micro satellites. The current design is a thruster-on-a-chip that produced 0.1 milli-Newton-seconds of impulse and about 100 Watts [5].

10 Conclusion: Things Overlooked

In this paper, I describe what I consider to be a new approach to the system design. Basic system architecture, useful behaviors, and interaction with the system were discussed. I have also tried to quantify the reliability of the proposed distributed system. The next logical step would be to model the system in a "virtual world" on a computer to test and refine the discussed algorithms.

Below is a list of some problems which haven't been discussed in this paper.

- In addition to its own position, a mobile node should also know the direction to the center of coordinate grid. It wasn't explained how nodes obtain this direction.

- To simplify things, it was assumed that each node can only perform one action at a time, and each action only requires one node. The cooperation between nodes can be achieved by attaching a participant counter to each R message telling how many nodes action requires.

- It was assumed that an action is applied to a *point* in space. In fact, many actions need to be applied to an *area*, making the self-organization more difficult. What happens when two nodes running to execute the same action converge on the area from different directions and do not hear each other?

- In the course of the discussion, it came up that we need an external (relative to the system) transmitter to generate TDMA cells. There is also a requirement for at least a few positioning grid nodes. Additional care should be taken to make these dedicated elements as safe and reliable as possible.

- I haven't discussed the communications security issues assuming that all communications are scrambled. The TDMA transmitter may also add noise to the channel to masquerade communication activity.

- The discussion of TDMA cell allocation protocols was avoided, as it is a very broad and actively researched topic.

11 References

References

[1] L.P.Clare, G.J.Pottie, and J.R.Agre. Self-Organizing Distributed Sensor Networks. manuscript, 1999.

[2] K.S.J.Pister, J.M.Kahn, and B.E.Boser. Smart Dust: Wireless Networks of Millimeter-Scale Sensor Nodes. *Electronics Research Laboratory Research Summary*, University of California Berkeley, 1999.

[3] J.M.Kahn, R.H.Katz, and K.S.J.Pister. Mobile Networking for Smart Dust, Proc. of ACM/IEEE Intl. *Conf. on Mobile Computing and Networking*, Seattle, 1999.

[4] D.Shiu and J.M.Kahn. Scalable Layered Space-Time Codes for Wireless Communications Performance Analysis and Design Criteria". *Proc. of IEEE Wireless Communications and Networking Conference*, New Orleans, 1999.

[5] Digital Micro-Propulsion Project.
http://design.caltech.edu/micropropulsion/

[6] Smart Dust Project.
http://robotics.eecs.berkeley.edu/ pister/SmartDust/

[7] Dr. Joseph M. Kahn Homepage.
http://alder.EECS.Berkeley.EDU/ jmk/

[8] Micro Motion Systems.
http://www.s3.kth.se/instrlab/research/projects/polyimideactuators.html