Coordination and Computation in distributed intelligent MEMS

J. Bourgeois[1,4], J. Cao[2], M. Raynal[3], D. Dhoutaut[1], B. Piranda[1], E. Dedu[1], A. Mostefaoui[1], H. Mabed[1]

Abstract— **Over the last decades, research on microelectromechanical systems (MEMS) has focused on the engineering process which has led to major advances. Future challenges will consist in adding embedded intelligence to MEMS systems to obtain distributed intelligent MEMS. One intrinsic characteristic of MEMS is their ability to be mass-produced. This, however, poses scalability problems because a significant number of MEMS can be placed in a small volume. Managing this scalability requires paradigm-shifts both in hardware and software parts. Furthermore, the need for actuated synchronization, programming, communication and mobility management raises new challenges in both control and programming. Finally, MEMS are prone to faulty behaviors as they are mechanical systems and they are issued from a batch fabrication process. A new programming paradigm which can meet these challenges is therefore needed. In this article, we present CO2Dim, which stands for Coordination and Computation in Distributed Intelligent MEMS. CO2DIM is a new programming environment which includes a language based on a joint development of programming and control capabilities, a simulator and real hardware.**

I. INTRODUCTION

New technologies create new scientific fields and this is especially true in communication networks. Local area networks and then Internet have created many of them, and later on, wireless communications also raise new possibilities and therefore new challenges that have been tackled by new research domains. A new technology called Distributed Intelligent Microelectro-mechanical systems (DiMEMS) [1] is currently emerging. DiMEMS can be defined as an ensemble of MEMS units where each unit can sense, act, process data and communicate.

This emergence is due to different factors. The main one is the progresses of MEMS technologies which is now a mature technology. A second factor, is the integration of MEMS and intelligence that is actually pushed further by research labs and companies [19].

DiMEMS are challenging systems as they can integrate many units, hundred thousand or even millions seems realistic. Scalability impacts programming, communication management and even the simulation system. Furthermore, due to this high number of units, fault tolerance has to be taken into account in each step of the process, from early detection of hardware failure to fault-tolerant algorithm and software.

This article presents the COordination and COmputation in Distributed Intelligent MEMS (CO_2Dim) project in detail and gives perspectives on how to deal with these systems.

II. APPLICATIONS OF DIMEMS

The very first application of distributed MEMS was about objects conveyance. This research has developed different types of MEMS actuator arrays, based on actuators either pneumatic [20, 8, 15], servoed roller wheels [17], magnetic [16] or thermobimoph and electrostatic [22]. More recently, sensors have also been integrated [18]. Within the Smart Surface project[1] a single surface composed of MEMS sensors and actuators, intelligence and communication capabilities has been proposed to sort and to convey different kinds of objects. The follow-up of Smart Surface named Smart Blocks project aims to build a MEMS-based modular and self-reconfigurable surface for fast conveying of fragile objects and medicinal products, composed of centimeter-size cubes where each of them comprises MEMS sensors and actuators, processing unit and communication capabilities.

Programmable matter is the most ambitious idea using distributed intelligent MEMS. The objective is to design matter that can be programmed to change its shape. Several approaches exist, the Claytronics project [9] proposes to use millimeter-size silicon balls that can move around each other thanks to electrostatic actuation. The software environment is particularly advanced as it includes two programming languages and two different simulators [21].

Among these two major fields of applications, distributed MEMS are also used in lots of different applications like atomic force microscopes (AFM) arrays [11], boundary layer control either on aircraft (AeroMEMS I and II projects) or on cars (ANR CARAVAJE project), flying drone from Silmach company for example (see Figure 1), and smart dusts [12]. Furthermore, many distributed macro sensor/actuator array like acoustic impedance control [3], see Figure 1, could be applied using diMEMS systems.

[1] UFC/FEMTO-ST, UMR CNRS 6174, France.
2 Hong-Kong Polytechnic University, China
3 IRISA, France.
4 Carnegie Mellon University, USA

[1] http://www.smartsurface.cnrs.fr

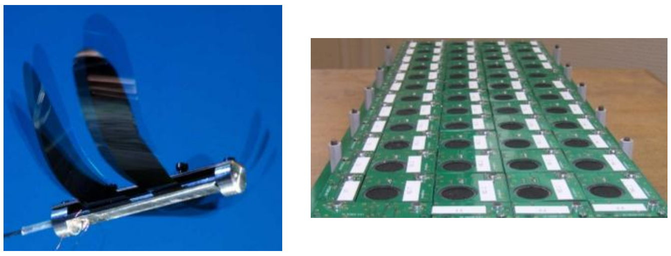


Figure 1: Potential applications for diMEMS, on the left, the Silmach dragonfly air drone and, on the right, acoustic impedance control (M. Collet and al., FEMTO-ST)

As it can be seen in these examples, the potential of diMEMS is huge and we think it will give birth to even more new applications as well. The maturity of these projects is varying a lot. While the first car equipped with an array of MEMS pneumatic actuators has shown a drag coefficient benefit of more than 10%, Silmach dragonfly will need substantial efforts to become a reality. Impacts will therefore range from short to long term.

Despite the economic crisis, the MEMS market has grown from \$6.9 Billion in 2009 to \$8 Billion in 2010 and the progression of the co compound annual growth rate is expected to increase up to 25% until 2015 [19].

To maintain this progression new types of MEMS have to be developed, and, by including intelligence inside distributed MEMS, it's exactly what we're looking for.

III. PRESENTATION

The problems and requirement listed above raise challenging questions concerning actuators, sensors, processing, communication and modules design.

The objective of CO_2Dim is to propose a software environment for programming and controlling distributed intelligent MEMS. This objective raises challenges in the following fields: scalable distributed programming, fault detection, fault tolerance and distributed coordination.

The next sections detail our proposition to solve this challenges so that the programming model can scale up to millions of units, faults can detected thanks to a k-set agreement in an asynchronous message passing environment, robust methods will deal with faulty units and communications and that the co-design between distributed computing and control will allow to manage millions of sensors/actuators.

IV. PROGRAMMING MODEL

Each unit of the DiMEMS system can be viewed as an autonomous system because it has a processing unit, communication, sensing and actuation capabilities. We will call such units ubiquitous interacting objects (UIOs). Every UIO has certain attributes and provide some functions (actuations and sensing) to other UIOs. UIOs interact with each other and build contextual relationships among themselves in order to discover services provided by other UIOs and to compose higher level services required by other UIOs or by human users. Coordinating interaction of innumerable UIOs in a decentralized manner in order to achieve some common objective is a challenge in the macro world which tantamount to the one faced by the DiMEMS in the micro world.

A. Definition of a programming model

1) Presentation

The key challenges include (1) achieving a generic programmability of the integrated DiMEMS-UIO system, and (2) communication between multiple DiMEMS-UIO entities.

The topology of DiMEMS can be either static or dynamic depending on the nature of the MEMS objects. While it is easy to handle static DiMEMS, managing mobile DiMEMS is really difficult because any fixed coordination system will fall short in the face of sheer dynamicity. Different distributed coordination algorithms are required to address the coordination challenges in the macro and micro worlds. So, the crucial issues we want to look into are (1) how to construct and dynamically maintain a group of collaborative DiMEMS-UIOs, in order to fulfil high-level tasks, (2) how to identify the functionalities and to develop an interface for interaction between UIO, (3) how to control large ensembles of actuators.

The second challenge is about facilitating communication between multiple DiMEMS-UIOs. UIOs are often contextually connected and this can generate a context-based overlay above the physical DiMEMS-UIO communication network. Contextual interconnection can be due to two or more UIOs sharing the same location or belonging to the same owner. Any number of UIOs owned by the same person can then form links among themselves based on matching owner attribute. However, any pair of such UIOs may be physically connected through multiple intermediaries and depend on multihop communication. In this research we need to delve deep into the issue to find out the possible requirements to build such an overlay structure which can help DiMEMS-UIOs to interact easily and with minimum cost.

Other issues that require investigation include: (1) whether it is possible to program the DiMEMS-UIO systems together using the same programming model and/or language, (2) whether any existing programming model can be used to program the unified system or new tools are to be developed, and (3) develop network protocols adapted to such systems.

2) Implementation

The challenge here is how the DiMEMS-UIOs can be programmed in order to achieve specific functionalities. Generally, there is a tradeoff between fast execution of algorithms and fast prototyping of the program [24]. So a major issue is how to easy programming of users, what provide to users as development environment. The goal is to enable users to efficiently program the DiMEMS-UIOs for fast and easy development of smart applications which require inter-UIO coordination, without sacrificing much of the execution speed.

V. Distributed Coordination

Distributed coordination means matching real-time constraints raised by coordinated actuation and programmation of distributed systems. In order to deal with real-time constraints performance awareness of the system has to be proposed. This is a real challenge which has been raised by J.P. Hespanha et al. who said in [10] that performance awareness is a future challenge to be tackled by control methods. In our opinion, performance awareness is a future challenge for both control methods and programming languages because the only way to dynamically solve time constraints raised by the controllers is to take into account the performance of the whole controlled system, including quantization, communication, computation and control.

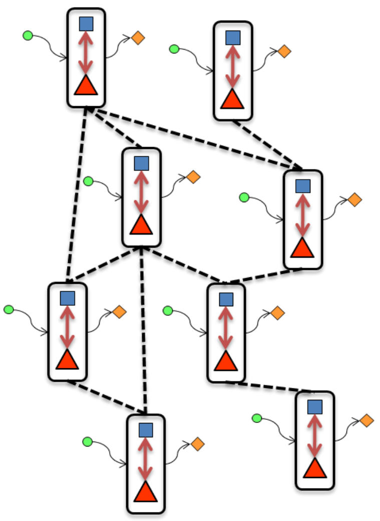


Figure 2: Representation of the targeted distributed system

The contributions of CO$_2$Dim within the field will be to propose the a programming and controlling language for large-scale systems. In order to do so, CO$_2$Dim will be performance-aware in order to meet the real-time constraints of control. The novelty is to have an integrated approach between control and programming which is only possible because the performance will drive the whole system.

A. *Performance awareness*

Performance awareness objective is to evaluate the current state of the performance, to forecast the possible performance until the next step and to send this forecast to the control so that it can adapt its control law regarding the performances. The performance awareness module is the central point of this architecture. It will intercept the instructions coming from the program executing the program and it will update the current performance state. By looking ahead in the program, it can then be able to have an estimate of the performance of the next operations. As the execution platforms will be quite simple ones compared to modern processors and modern communication networks, we propose to extend a micro-benchmarking method that we already used for modeling CPU in P2P distributed programs [7]. If the linearity of the execution is not met, we will apply a parametric block benchmarking method. It consists of benchmarking a larger portion of codes while reducing the number of iterations for loops with a static analysis of the source code [2].

As the networking stack is limited to the minimum, communications could be modeled with a simple αβ+γ model together with a congestion model taken from [23]. Benchmarks on real available systems will also be used if more precision is needed.

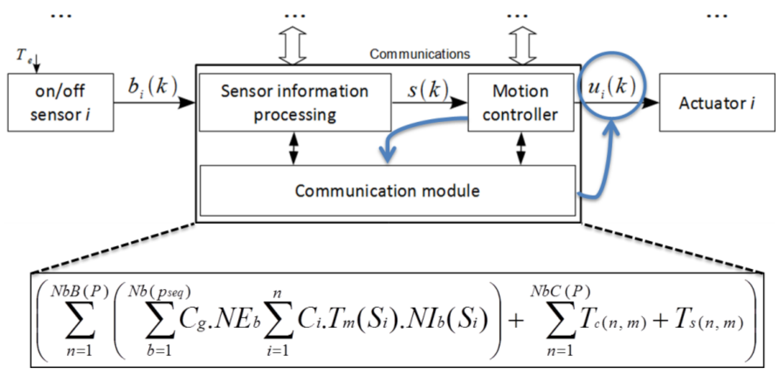
B. *Implementing distributed coordination*

Distributed coordination takes care of the atomicity, synchronization, scalability and network congestion inside the system.

The atomicity for timed operation has to be ensured. The system is dynamic and during the completion of a communication or of a task, the system should stay more or less identical, inside known limits.

Units are distributed and have a local vision, comprising only their neighborhood. This means that convergence time for achieving a specific result could increase. To reduce it, a larger vision could be envisaged [4]. In this context, the synchronization among neighbors of units is a key feature. Indeed, for control systems of microscopic size, coordination and synchronization among units are essential. For example, moving an object to some destination requires coordination and synchronization of several actuators on the path to destination.

The scalability deals with programming and controlling thousands or even millions of units. This is very different that programming a few parallel objects. Indeed, the interactions among objects are greater than usually by several orders of magnitude. This requires a major shift in distributed algorithm design. An additional issue is that there is no central point. For example, both the multi-threading and message passing programming paradigms [6] are generally



$$\left(\sum_{n=1}^{NbB(P)} \left(\sum_{b=1}^{Nb(p_{seq})} C_g.NE_b \sum_{i=1}^{n} C_i.T_m(S_i).NI_b(S_i) \right) + \sum_{n=1}^{NbC(P)} T_{c(n,m)} + T_{s(n,m)} \right)$$

Figure 3: Performance awareness module will interact with control law

based on tasks which are created and dispatched on several units whose result comes back to some central point, so it is not well adapted in our distributed context. A possible solution to deal with scalability is to use a hierarchical programming, where units are grouped and a leader is chosen for each group.
The scalability leads to a network congestion phenomenon. Nowadays, variants of TCP congestion control are widely used to take care of network resources. They are well adapted to Internet, which has several specific characteristics: the topology (networks of autonomous systems), the type of traffic generated, in time and space, no real-time guarantees and so on. Distributed intelligent MEMS need specific protocols to deal with these issues. DCCP [14] is in a good position, as it features a more real-time shape and has several congestion controls built-in. Some data could also be more important than other, in which case it should be prioritized, such as in [5].

VI. : FAULT-DETECTION AND FAULT-TOLERANCE

A. *Fault-detection of MEMS cells*

The objective is to develop approaches that are able firstly to detect faults, more precisely system misbehaviors, by observation in a large-scale distributed MEMS system. In fact, the faults can emanate from wrong design and/or assemblying, material ageing, corrosion, damages, etc. The unpredictable nature of faults occurrence in MEMS and the difficulty in ensuring their proper behavior come mainly from the limited information about the microscopic failure mechanisms which differ from the “commonly” known macroscopic mechanisms. To this end, it is necessary to maintain a kind of continuous observation of the overall behavior of the system in order to be able to detect failures or at least to highlight some system misbehaviors. Such observation must take into account not only the individual behavior of components but extends it to all components. To illustrate let’s consider the following example on convoying microscopic objects in a MEMS system. We suppose that an object must be “moved” from point (entrance) to point (exit) and that we have a set of available modules (Figure 4). Each module can have an individual action on the object i.e., changing object’s position and speed according to a given law. From the system behavior observation, we can see that some objects (red arrows) are not convoyed to the right exit as illustrated into Figure 4, while black arrows are the suitable objects trajectories. This system misbehavior could suggest many failure reasons (sensor failure, air pulse failure, air rate failure, etc.) and even more combination of them e.g., wrong sensor detection and right air pulsing.
In other words, the research challenge we are facing is how to detect the possible origin(s) or the possible combinations of the factors leading to this misbehavior? The problem is even more complicated in fully distributed system when each module has only a partial/local view of the system misbehavior.

Another important issue, particularly prominent in MEMS systems, is the localization of modules responsible of this misbehavior. In fact, failure localization in MEMS system is as important as its detection because, as we are dealing with microscopic mechanisms, the maintenance task is very costly and tedious. Being able to identify and localize the origins of failures will certainly make the maintenance much easier and effective.

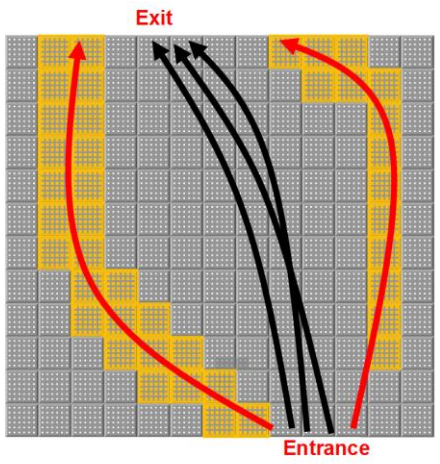


Figure 4: Illustration of the misbehavior problem

Within this project, we plan to investigate a *decentralized asynchronous message-passing approach* that is a natural candidate in large-scale distributed micro-controlled system, as it is the case in MEMS systems. In fact, by reason of microscopic environment of the latters, it is very hard to ensure synchronicity at a reasonable cost in terms of communications and failure detection decision time. Furthermore, asynchronous approaches are much more robust that the centralized ones and more importantly they ensure scalability which is an important requirement in MEMS system.

Briefly, we highlight below the different steps of our approach:

- Each module can get locally two kind of information: (a) information derived from the control function implemented on each module (which results from a training process) and (b) information provided by the sensing devise. From these two data, each module could constructs firstly a local view and decides if there is misbehavior or not i.e., if the sensed data does not match, to a certain extent that has to be investigated more in details later on, the expected values from the control function (yellow blocs in the figure above).
- If a positive decision has been taken by a module, the latter communicates it to its immediate neighbors in order to “enlarge” its local view.
- Upon receiving notifications from its neighbors and using its local information, a module can then “refine” its local decision and decides to propagate this information to its neighborhood.

In such an approach, a central question remains: how to decide “globally” of the failure detection and its localization. This is the main research subject of the sub-task below.

B. *k-simultaneous consensus in an asynchronous message passing system*

Once the system misbehavior has been highlighted locally by some modules, the second important step is to "decide" globally of the failure occurrence and to localize it. In other terms, the modules that have detected the failure locally must "converge" to a global consensus on the detection and localization of the failure.

At first sight, we plan to investigate simultaneous consensus approaches in distributed systems that better fulfill our research problem requirements. In fact, in such approaches, each unit participates at the same time in k independent consensus instances until it decides in any one of them. However, in message-passing systems, as it is typically the case in MEMS systems, the k-simultaneous consensus problem remains an open research issue. The objective of this subtask is therefore to weaken the consensus problem in a k-set agreement problem where up to k different values can be decided. k-set agreement problem can be solved despite asynchrony and unit failures when $k > t$ (where t is the maximum number of units that can be faulty), but it has been shown that it has no solution when $t >= k$.

As we have seen in part 1, MEMS batch production process is prone to failures and its likely possible that t could be greater than k. This case has then to be studied. Equivalence of the k-consensus problem to a k-set agreement problem already exist in the case of $t<N/2$ (with N the number of units). But, currently no solutions exist for t>=N/2, we therefore propose here to study the equivalence between the k-set-agreement problem and the binary k-simultaneous problem if more than half of the units crashes.

The novelty of our approach is to lead units to a decentralized asynchronous consensus on the detection of failures and at the same time their localization in an asynchronous message passing system.

VII. Simulation Platform

The goal of this subtask will be to define the requirements and then the means to simulate diMEMS in a generic, adaptable way.

A. *Defining the requirements for a generic simulator*

We will have first to specify the processing and memory capabilities, along with the communications, sensing and actuation capabilities of the simulated diMEMS. The following constraints will serve as strong guidelines for this work.

- Processing and memory capabilities are to be rather small, considering the context of low individual cost of diMEMS.
- Communications capabilities should be direct contact, low bandwidth and error prone interfaces. Wireless is also an option, but in a given application not all elements may have it.
- Sensing abilities should allow simple measurement to be taken at one or possibly multiple points of each independent element (such as boolean contact sensing or temperature sensing)
- Actuation capabilities should include a way to change its visual aspect (color or LEDs control) or the ability to interact with the physical world by moving itself or adjacent objects.

Absolute or relative positioning API are of great importance, but will depend on choices made concerning the physical capabilities of our diMEMS.

From a networking designer point of view, choices will be made whether to support very generic protocols stacks or not. Using stacks such as tcp/ip, utp/ip or dccp/ip would indeed offer an almost unlimited application support, at the price of a relatively large overhead considering the large number of simulated diMEMS. Building more specific, task-optimized protocols, is also a possibility which will have to be carefully evaluated.

Last but not least in this subtask is the specification of the families of error our diMEMS simulator should be able to handle. At a minimum our work should take into account:

- Communications / networking errors
- Sensor readings errors
- Actuation errors
- Complete failure of some diMEMS in the ensemble.

Being able to cope with memory / storage errors and computation errors may also be investigated.

B. *Building tools and simulating the communications.*

Generic simulators already exist but most tend to focus on robotic simulation with bigger and more complex simulated elements [robot3D]. Those simulators rely on very detailed physics but lack in scalability.

On the other hand, DPRSIM [DPRSIM] was designed for claytronic diMEMS and is able to simulate in a distributed way very large numbers of simple elements (up to millions of them). But this is done at the cost of lack of precision in the simulation and communication models (with a strong artificial per tic synchronization) and the lack of simulation determinism (due to the use of operating system's scheduler to handle parallel execution).

The way communications will be simulated is of critical importance as the behavior of the network has a strong influence on the way errors will be perceived at other layers. Many powerful networks simulators exist. They are largely used in the community for their extended functionalities along with the de facto scientific comparisons standard status of some of them. But they obviously lack the physical simulation we require along with the sensing and actuating capabilities.

A too simple modeling of the processing and communication capabilities hurts the reliability of the simulation, but at the same time allows the scaling up. So we intend to build a simulation core that can be configured for both use. It would provide simple and very fast internal communication capabilities, along with the ability to be interfaced with dedicated and recognized external simulators such as NS2 or NS3.

Using very detailed network simulations nonetheless brings the strong drawback of the scalability. Going over a few tens of thousands elements should not be practical. We would miss one of our important goals which is to evaluate very large ensembles.

We thus intend to use a three steps approach for the network part of the simulations :
Firstly by simulating at relatively small scale with a high precision level
Secondly validating lighter and much faster models by comparing them to the precise and complex ones on small to medium scale scenarii.
Thirdly simulating at full scale with the faster models.

We consider important to retain a deterministic behavior in our simulator, so we intend to use a dedicated scheduling mechanism. A deterministic behavior means a simulation run should produce completely predictable results. Multiple simulations runs using the same random number's seed should produce identical results. Final results should be computed statistics from a sufficient number of runs using different seeds.

Simulating physics is also very important but has the same shortcomings as the precise network simulation. Detailed physic is very demanding in computational resources. We thus intend to implement an optional fall back to simpler or even no physic at all mode. Small and detailed simulations would help to validate simpler models used for large scale scenarii. Capabilities in this field should also include the possibility to interface with real sensors / actuators, to build hybrid scenarios between real world and simulation.

Analysing and understanding what is happening is a very important and sometimes underestimated part of any simulation work. To help analyzing simulated code behavior, we intend to provide a comprehensive 3D view along with full logging capabilities. It will be able to show actuations effects, changes in the topology and communications between elements. Configurable real-time view should be a very useful tool both for design and debugging purposes, and should also help with the disseminations of our results.
An example has been developed in a preliminary version of our simulator. It simulates the conveying of small objects using pneumatic actuators driven by a network of diMems. Movements of these objects are detected by sensors to which embedded simulated softwares reacts. In this case, the physical simulation computes the acceleration of the objects under the effect of the pneumatic actuators (air jets) and their speed variations taking into account many physical parameters such a s mass, friction, damping …

Two operating modes are implemented: a real time mode showing the real running speed and a maximum speed mode allowing to quickly check the program termination. A replay capability should also be added to the simulator as a part of the helping tools suite.

To provide a maximum portability, the simulator was developed in C + + using the OpenGL graphics library which is available on any systems.

C. *Handling scalability*

Validating our approach of diMEMS would not be complete without large scale simulations. Numbers in the hundreds of thousands and over are not attainable by actual experimentations and have to be simulated. But even traditional simulation would have problem with such numbers.

This means working on the parallelization of our software. Along with the three steps approach for the network simulation mentioned earlier, all of our software will have to be adapted to provide a usable simulation and validation environment.

Two main shortcomings should limit the final scalability and prevents us to simulate as many elements as a very scale focused simulator such as DPRSIM, but we consider them as reasonable drawbacks for the corresponding advantages.
The first comes from keeping the determinism of the simulation, making it harder tho parallelize the simulation core. The second one comes from the complex physical, sensing and actuation part of the simulation.
For the later, as we previously explained and as it was done for DPRSIM, we nonetheless intend to allow the optional deactivation or simplification of the physics, greatly improving the speed.

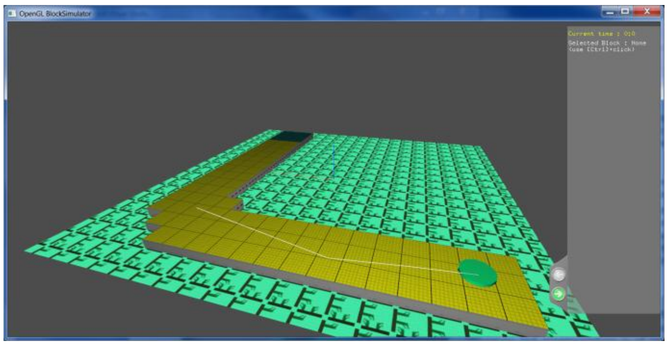


Figure 5: A screen snapshot of the first version of the simulator.

VIII. Demonstrators

A. *Acquiring and adapting multiple hardware platforms.*

To demonstrate our approach and also validate our simulations, a physical demonstrator will be build. To minimize the risk with still experimental hardware, we intend to use two different hardware platforms.
The first one would be taken from the SmartBlocks project[2]. Those blocks will have the ability to move by using neighboring blocks as support points. They will have sensors

[2] http://smartblocks.univ-fcomte.fr

and communication interfaces on the sides, along with sensors and actuators on the top.
The second one will be the Blinky Blocks [13] developed at Carnegie Mellon University. Those blocks do not move by themselves but can be manually rearranged as they are maintained by strong magnets. They have the capability to communicate with their direct neighbors, to emit light of various colors and to detect small impacts and fingers interactions through embedded accelerometers.

B. Implementation of those hardware platforms into the generic simulator

The specificities of each hardware platforms will be implemented in out generic simulator. As previously hinted, hardware and software implementations will be complementary, the hardware demonstrating the reality of our work, the simulations allowing to evaluate it at much larger (and cheaper) scales.

C. Hardware-in-the-loop for development and debugging purposes.

Our experience shows that going back and forth between simulations and experimentations is of great utility to the development process. We intend also to minimize the risk on the hardware part of the project by enabling an "emulation" mode. In this mode the real sensors and actuators will not be required and the simulator will be used to provide the required information to the code in the real physical blocks.

IX. Conclusion

This paper has presented a new approach for dealing with distributed intelligent MEMS which handles many challenges from theoretical studies like studying the equivalence between the k-set-agreement problem and the binary k-simultaneous problem if more than half of the units crashes, to more practical problem of building efficient simulation systems. The originality of the approach is also to bridge the gap between simulation and real testbeds in distributed intelligent MEMS.

The efficiency of our approach still needs to be demonstrated,

References

[1] Julien Bourgeois and Seth Goldstein. Distributed intelligent mems: Progresses and perspectives. In Ljupco Kocarev, editor, *ICT Innovations 2011*, volume 150 of *Advances in Intelligent and Soft Computing*, pages 15–25. Springer Berlin / Heidelberg, 2012.

[2] Bogdan Cornea and Julien Bourgeois. A framework for efficient performance prediction of distributed applications in heterogeneous systems. *The Journal of Supercomputing*, pages 1–26. 10.1007/s11227-012-0823-5.

[3] Petr David, Manuel Collet, and Jean-Marc Cote. Experimental implementation of acoustic impedance control by 2d network of distributed smart cells. In IEEE CPS, editor, *Proc. of the 1st Workshop on hardware and software implementation and control of distributed MEMS (dMEMS2010)*, 2010.

[4] Eugen Dedu. *Design of a Simulation Model of Multi-Agent Systems, and its Parallel Algorithmic and Implementation on Shared-Memory MIMD Computers: ParSSAP Model*. PhD thesis, University of Versailles Saint Quentin, France, March 2002.

[5] Eugen Dedu and Emmanuel Lochin. A study on the benefit of TCP packet prioritisation. In *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, 17, pages 161–166, Weimar, Germany, February 2009. IEEE.

[6] Eugen Dedu, Stéphane Vialle, and Claude Timsit. Comparison of OpenMP and classical multi-threading parallelization for regular and irregular algorithms. In *Proceedings of Software Engineering Applied to Networking & Parallel/Distributed Computing (SNPD)*, pages 53–60, Reims, France, May 2000.

[7] J.-B. Ernst-Desmulier, J. Bourgeois, and F. Spies. P2PPerf: a framework for simulating and optimizing peer-to-peer distributed computing applications. *Concurrency and Computation: Practice and Experience*, 20(6):693–712, 2008.

[8] Hiroyuki Fujita. Group work of microactuators. In *International Advanced Robot Program Workshop on Micromachine Technologies and Systems*, pages 24–31, Tokyo, Japan, October 1993.

[9] Seth Copen Goldstein and Todd C. Mowry. Claytronics: A scalable basis for future robots. In *RoboSphere 2004*, Moffett Field, CA, November 2004.

[10] J.P. Hespanha, P. Naghshtabrizi, and Yonggang Xu. A survey of recent results in networked control systems. *Proceedings of the IEEE*, 95(1):138 –162, jan. 2007.

[11] Hui Hui, Michel Lenczner, Emmanuel Pillet, and Scott Cogan. A two-scale model for one-dimensional arrays of cantilevers and its verification. *Mechatronics*, (0):–, 2011.

[12] R. H. Katz J. M. Kahn and K. S. J. Pister. Mobile networking for smart dust. In *ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99)*, 1999.

[13] Brian T. Kirby, Michael Ashley-Rollman, and Seth Copen Goldstein. Blinky blocks: a physical ensemble programming platform. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, CHI EA '11, pages 1111–1116. ACM, 2011.

[14] Eddie Kohler, Mark Handley, and Sally Floyd. Datagram Congestion Control Protocol (DCCP). IETF standard, March 2006. RFC 4340.

[15] S. Konishi, Y. Mizoguchi, M. Harada, and K. Ohno. Experimental investigation of a distributed conveyance system using air flow. In *Micromechatronics and Human Science, 1998. MHS '98. Proceedings of the 1998 International Symposium on*, pages 195 –200, nov 1998.

[16] C. Liu, T. Tsao, P. Will, Y.C. Tai, and W.H. Liu. A micromachined permalloy magnetic actuator array for micro robotics assembly systems. In *The 8th International Conference on Solid-State Sensors and Actuators*, 1995.

[17] J. E. Luntz and W. Messner. A distributed control system for flexible materials handling. *IEEE Control Systems*, 17(1), February 1997.

[18] M. Mita M. Ataka and H. Fujita. Stack-integrated micro actuator/sensor array for 2d planar micro manipulator. *submitted to IEEE/ASME Transactions on Mechatronics*, *:*, 2012.

[19] E. Mounier. MEMS markets and applications 2011-2017, an overview. In *dMEMS'12, 2nd workshop on design, control and software implementation for distributed MEMS*, Besançon, France, 2012. IEEE Computer Society Press.

[20] K.S.J. Pister, R. Fearing, and R. Howe. A planar air levitated electrostatic actuator system. In *IEEE Workshop on Micro Electro Mechanical Systems*, pages 61–71, 1990.

[21] Benjamin D. Rister, Jason Campbell, Padmanabhan Pillai, and Todd C. Mowry. Integrated debugging of large modular robot ensembles. In *ICRA*, pages 2227–2234, 2007.

[22] J.W. Suh, S.F. Glander, R.B. Darling, C.W. Storment, and G.T.A. Kovacs. Combined organic thermal and electrostatic omnidirectional ciliary microactuator array for object positioning and inspection. In *Solid State Sensor and Actuator Workshop*, 1996.

[23] Pedro Velho and Arnaud Legrand. Accuracy study and improvement of network simulation in the simgrid framework. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques for Communications, Networks and Systems, SimuTools*, page 13, 2009.

[24] Stéphane Vialle and Eugen Dedu. Long parallel algorithm design *vs.* quick parallel implementation. In *Proceedings of European Workshop on OpenMP (EWOMP)*, pages 145–150, Edinburgh, Scotland, UK, September 2000.