Behavioral Simulations of Lattice Modular Robots with VisibleSim

Pierre Thalamy, Benoît Piranda, André Naz and Julien Bourgeois

Abstract Robotics research needs complex hardware and software that is why simulation is often view as an alternative for testing. Large scale self-reconfiguring modular robotic systems needs a scalable simulation environment which cannot be physics-based.
This paper presents *VisibleSim*, an open-source behavioral simulator for lattice-based modular robots that uses discrete-event simulation to simulate ensembles of up to millions of modules. We describe the principles behind the simulator and introduce its features and usage from a user standpoint. *VisibleSim* is built with extensibility, versatility, and flexibility in mind, can be used as a powerful visualization tool, and already has a proven track record with several modular robotic architectures.

1 Introduction

Simulation can be used for multiple purposes in robotics research, one of them being, researching software solutions for managing complex systems that are not yet producible. This is particularly relevant for self-reconfigurable modular robotic systems [1] which assemble individual modules latched to one another. The software challenge is to coordinate all the modules to achieve a common goal like in self-reconfiguration [23].

In this paper, we present *VisibleSim*[1], a framework for creating *behavioral* simulators for distributed lattice-based modular robotic systems in regular 3D environments. *VisibleSim* can be used for studying the behavior and programmability of such distributed systems, but it does not comprise physics simulation.

Pierre Thalamy · Benoît Piranda · André Naz · Julien Bourgeois
Univ. Bourgogne Franche-Comté (UBFC), University of Franche-Comté (UFC), FEMTO-ST Institute, UMR CNRS 6174, 1 Cours Leprince-Ringuet - 25200 Montbéliard, France.
e-mail: `\{benoit.piranda, julien.bourgeois\}@femto-st.fr`

[1] `https://projects.femto-st.fr/programmable-matter/visiblesim`

Each module executes the same program as all other modules, generating communications and/or events that are handled deterministically by *VisibleSim*'s discrete-event scheduler. Beyond reproducibility, this allows for scaling up in the number of simulated modules which can be greater than 32 millions. A video presentation of the simulator and its aforementioned features is available on Youtube[2].

2 Related Works

There are many general simulation frameworks dedicated to robots [11]. Some of them have been used for many different kinds of robotics projects like Player [8], associated at Stage or at Gazebo [10]. Webots [13] is another reference of a commercial open-source simulator, now a leader in this field. ARGos [17] simulator for swarm robotics, which can simulate large and heterogeneous multi-robots systems. These simulators are not particularly well-suited for the specificities of modular self-reconfigurable systems.

On the one hand, regarding hardware-specific modular robot simulators, many platforms for simulating self-reconfiguration for lattice modular robots are unnamed simulators with core features implemented using Java3D [25, 22].

On the other hand, several generic modular robot simulators have been developed, mostly physics-based and targeting chain or hybrid modular robots like Rebots [4], including models for Roombots [21], Smores [5] and Superbot [20]. Sim [24], USSR [3] including models for ATRON, Odin [12], and M-TRAN [9].

Nevertheless, there are still a few generic simulators that are designed for lattice modular robots, the type of robots targeted by *VisibleSim*. For example, SRSim [7] used with Sliding-Cube [7], Crystalline [19] and Superbot [6]. Finally, DPRSim [2] has been shown to efficiently simulate ensembles with up 20 millions of Catom modules, both in their 2D and 3D forms, by maximally leveraging the potential for multi-threading of the simulation and using computing clusters.

The common feature among all the aforementioned simulators except perhaps SRSim, is that they are all physics-based, which is useful when developing robotic designs, evolving controllers, and interacting with complex environments, but might be superfluous and prohibitively costly when researching distributed robotic control from a more fundamental, or *behavioral*, point of view. This is the kind of simulator that *VisibleSim* is thus intended to be, a framework for performing all kinds of behavioral simulations on lattice-based modular robotic systems with low environmental interactions. Furthermore, it is most similar to USSR and SRSim in its usage, being a framework for developing simulator instances rather than an actual monolithic executable software where all simulation parameters are interpreted.

3 Simulator Overview

VisibleSim is designed for researchers that have computer programming experience as it consists in a **C++ framework** for building lattice modular robot simulators

[2] Video presentation of *VisibleSim*: `https://youtu.be/N09KElCbUNk`

controlled by distributed programming. Several sample modular robot simulators are provided with the software. *VisibleSim* takes the form of an open source project under AGPLv3 license and is available on Github[3].

In *VisibleSim* lingo, the distributed program that is executed on each module during the simulation is named a *BlockCode*. It is effectively the controller of the modules and where users will describe the behavior of the robot in response to all kinds of events whether external (interactions with the world, reception of a message, etc.), or internal (interruption or timer, initialization, end of a motion, etc.).

Unlike other simulators where each robot is fitted with a number of sensor and actuator components, this distinction is not materialized in *VisibleSim*. Modules from any type of robots are however fitted with a constant number of *interfaces*, depending on the geometry of their lattice, and which can both be used for sensing connected modules (by examining whether an interface is connected) and communicating with them. In the current state of the simulator communication between modules is only natively allowed in a peer-to-peer manner between connected neighbors.

Previous work on modular robots can be classified based on the shape of the robots and the type of grid in which they are placed. Each grid has a specific number of positions adjacent to each of its cells, which determines the number of neighbors a module in that grid can communicate with (2D square or hexagonal lattices, Face Centered Cubic lattice, etc.).

VisibleSim offers different classes of modular robots across these different lattices, as shown in Figure 1.

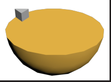
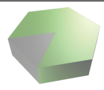
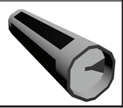
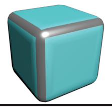
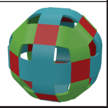
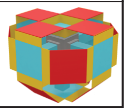
2D Nodes Square lattice (2D) 4 neighbors Motion: Slides along a neighbor or Turns around an edge. Display: Lights in color 	**Smart Blocks** Square lattice (2D) 4 neighbors Motion: Slides along a vertical border Display: Lights in color and draws numbers on the top
Hexanodes Hexagonal lattice (2D) 6 neighbors Motion: Turns around a neighbor (pivot) Display: Lights in color 	**2D Catoms** Hexagonal lattice (2D vertical) 6 neighbors Motion: Turns around a neighbor (pivot) Display: Lights in color 
Blinky Blocks Cubic (3D) 6 neighbors Display: Lights in color Sensor: tap 	**Sliding Cubes** Cubic (3D) 6 neighbors Motion: Slides along a neighbor or Turns around an edge. Display: Lights in color 
3D Catoms Face-Centered Cubic lattice (3D) 12 neighbors Motion: Turns around a neighbor (pivot) Display: Lights in color 	**Datoms** Face-Centered Cubic lattice (3D) 12 neighbors Motion: Deforms to turn around a neighbor (pivot) Display: Lights in color 

Fig. 1: Several shapes of robots proposed in *VisibleSim*.

[3] https://github.com/ProgrammableMatterProject/VisibleSim

What characterizes a modular robot in *VisibleSim* is therefore: the geometry and visual aspect of its modules; the lattice in which they belong (hence their number of possible neighbors), and a specific mode of motion. Additional components and state visualization features such as a display, speakers, or tap sensors can however be added.

4 Programming Environment and Features

4.1 User Application Demonstration

This section presents an example of a *SlidingCube* modular robot application, where a message is broadcast distributively through the robot from a leader module (identified by its identifier) to instruct modules to perform a random motion. Though this application has no practical purpose, it demonstrates concisely the structure of a user application as well as elements of its motion and communication API.

Furthermore, a visual *BlockCode* generator is available online[4], which takes a target robotic architecture and a list of messages as input and returns a code template for that setup.

Listing 1: **Sample BlockCode:** Broadcast of a message across the robot from a master module and moves upon reception

```
#include "myBlockCode.h"

void MyBlockCode::startup() {
    addMessageEventFunc(BROADCAST_MSG, bind(&myBlockCode::onBroadcastRcvd,
                        this, std::placeholders::_1,  std::placeholders::_2));
    if (module->blockId == 1) { // module #1 is the master
        this->broadcastReceived = true;
        sendMessageToAllNeighbors(new Message(BROADCAST_MSG));
    } else {
        this->broadcastReceived = false;
    }
}

void MyBlockCode::onBroadcastRcvd(shared_ptr<Message> msg,
                                  P2PNetworkInterface* sender) {
    if (not this->broadcastReceived) {
        this->broadcastReceived = true;
        // Propagate broadcast (ignoring sender interface)
        sendMessageToAllNeighbors(new Message(BROADCAST_MSG), sender);
        //  and move to first available location
        list<Cell3DPosition> dests = getPossibleDestinations();
        if (not list.empty()) initiateMotionTo(dests.front());
    }
}
```

[4] https://services-stgi.pu-pm.univ-fcomte.fr/visiblesim/generator.php

Listing 2: **Sample main file:** initiates and cleans up the simulation

```
#include <iostream>
#include "robots/slidingCubes/slidingCubesSimulator.h"
#include "robots/slidingCubes/slidingCubesBlockCode.h"
#include "myBlockCode.h"

int main(int argc, char **argv) {
    // Create simulation world and modules
    //  attach a MyBlockCode instance to each module
    createSimulator(argc, argv, MyBlockCode::buildNewBlockCode);
    // Previous call returns only once scheduler has ended
    deleteSimulator();
    return 0;
}
```

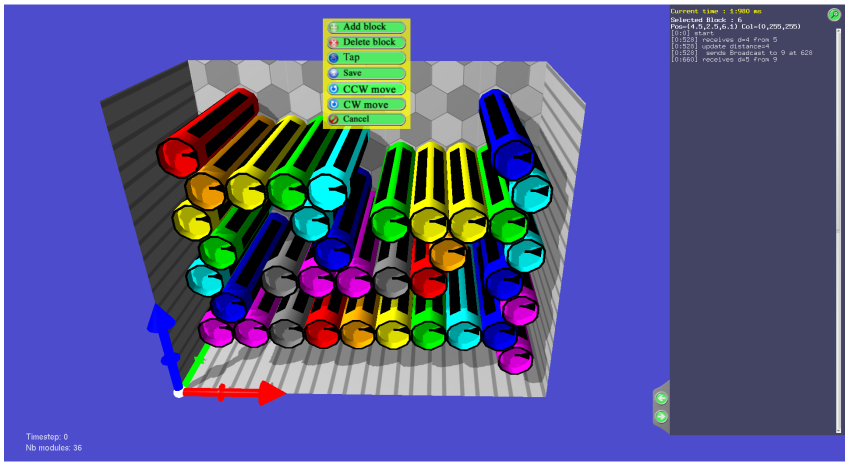
Listing 3: **Sample XML configuration file:** describes the simulated world; the modules within it; and other simulation parameters

```
<?xml version="1.0" standalone="no" ?>
<world gridSize="20,20,20" windowSize="1920,1080">
    <blockList defaultColor="128,128,128" ids="RANDOM">
        <!-- Describe individual modules -->
        <block position="3,4,2" color="127,255,43" />
        <!-- or use Constructive Solid Geometry (CSG) -->
        <csg content="union() { cube([10, 5, 5]); cube([5, 10, 5]); }"/>
    </blockList>
    <targetList> <!-- Goal shape for reconfiguration -->
        <target format="csg">
            <csg content="sphere(10)"/>
        </target>
    </targetList>
</world>
```

4.2 User Interactions

In fixed-increment time progression mode, *VisibleSim* supports pausing and resuming of the simulation (programmatically or using the keyboard), which can be used to inspect the simulated world at any given time. This is especially useful since *VisibleSim* has a built-in console that provides useful information about a number of core (messages sent or received, motions, etc.) or custom (any user-implemented event or debugging trace) events. This includes the time of the event and any other useful information that is necessary for retracing what in the chain of events that led to the current state of the simulation. Not only can the traces concerning all the modules in the system be shown at once from within the simulation window, but individual threads of events relative to a specific module can be shown by selecting the module from the GUI.

Furthermore, left-clicking a module opens a pop-up for interacting with the simulated world and the module itself. These interactions are the addition and removal of neighbors on the interfaces of a module, motion commands, or a physical event such as an accelerometer tap. Finally, the current world configuration can be exported, making *VisibleSim* both a simulation software and a sandbox for building robotic configurations.



```
Current time : 1:980 ms
Selected Block : 6
Pos=(4.5,2.5,6.1) Col=(0,255,255)
[0:0] start
[0:528] receives d=4 from 5
[0:528] update distance=4
[0:528]  sends Broadcast to 9 at 628
[0:660] receives d=5 from 9
```

```
Timestep: 0
Nb modules: 36
```

Fig. 2: Screenshot of a *VisibleSim* simulation of Catom 2D modules with the console and interaction menu open for the selected module.

4.3 Customization Hooks

VisibleSim proposes a number of customization hooks that are called at various points of the simulation and that can be used to implement custom behaviors for a given *BlockCode* application. Some of these functions provide greater flexibility to the user, others simply facilitate debugging:

- Parsing custom configuration file elements pertaining to the world or individual modules.
- Parsing custom command line arguments exclusive to this specific *BlockCode* application.
- Responding to custom keyboard events generated by the user during simulation and specific to that application.
- Drawing custom graphical elements in the OpenGL world every time it is updated.
- Drawing custom text onto the OpenGL window to keep some essential information always visible.
- A custom function that gets called on a module whenever a *VisibleSim* assertion has been triggered for that module, and that can provide critical information on its current state.

4.4 Export Tools

An important aspect of *VisibleSim* is that it is *more than a simulation tool*. Its other main purpose is to produce impactful visual results for academic research, at a low cost for the user. It does so by allowing researchers to easily export various content from their simulated world: screenshots and videos of a simulation, 3D animation data for stunning videos in Blender or similar software, Stereolithography (STL) data for the 3D printing of a configuration, etc. This thus makes *VisibleSim* a *visualization tool* as well as a simulator, supporting research at multiple levels.

Apart from the standard simulation workflow of *VisibleSim*, where a simulation is run in real time either in terminal-mode or with a graphical output, *VisibleSim* offers to export a simulation to a file on disk, so that the simulation can be later be visualized on a *replayer* in an interactive fashion. This *replayer* allows the user to slowly and repeatedly analyse events generated during the simulation, record a part of it as a video or an animation, or compare results. This feature is particularly helpful for inspecting a problematic simulation when debugging, or for viewing the graphical output of a terminal-mode simulation that would have taken too long or too much memory to be computed graphically.

5 Usage and Evaluation

In this section, we highlight a number of different modular robots and applications that have been successfully simulated using *VisibleSim* in published research. Our aim is to highlight different ways *VisibleSim* can be used and has been used. We also show that the simulation of existing hardware system can show a high level of fidelity to hardware experiments. Finally, we bring to light the current capabilities of *VisibleSim* in terms of scalability.

5.1 Use Cases

VisibleSim has become over the years the dedicated simulation and experimentation tool of the *Programmable Matter Project*. As such, not only has it been used for simulating diverse modular robotic models, but also for a wide variety of applications (distributed time synchronization, self-reconfiguration, center election, etc.). Figure 3 thus illustrates the versatility and reliability of the *VisibleSim* framework by highlighting select research work that has relied on it in recent years.

5.2 Simulation Fidelity

In addition to faithfully reproducing the functional behavior of algorithms, *VisibleSim* also accurately simulates timing. Communication and clock models can be customized and passed to *VisibleSim* in order to fit best with the simulated modular robotic platform.

After having modeled the communication system of the *Blinky Blocks* in *VisibleSim* [14], we measured the execution time of the ABC-CenterV1 algorithm [15, 14] — an algorithm for electing an approximate-center module in modular robots — on hardware *Blinky Blocks* and in simulations. Table 1 shows that the simulated execution times (average and standard-deviation) on *VisibleSim* closely match the execution time obtained experimentally on hardware *Blinky Blocks*, for small and larger configurations, and for sparse (e.g., lines), less-sparse (e.g., squares), compact

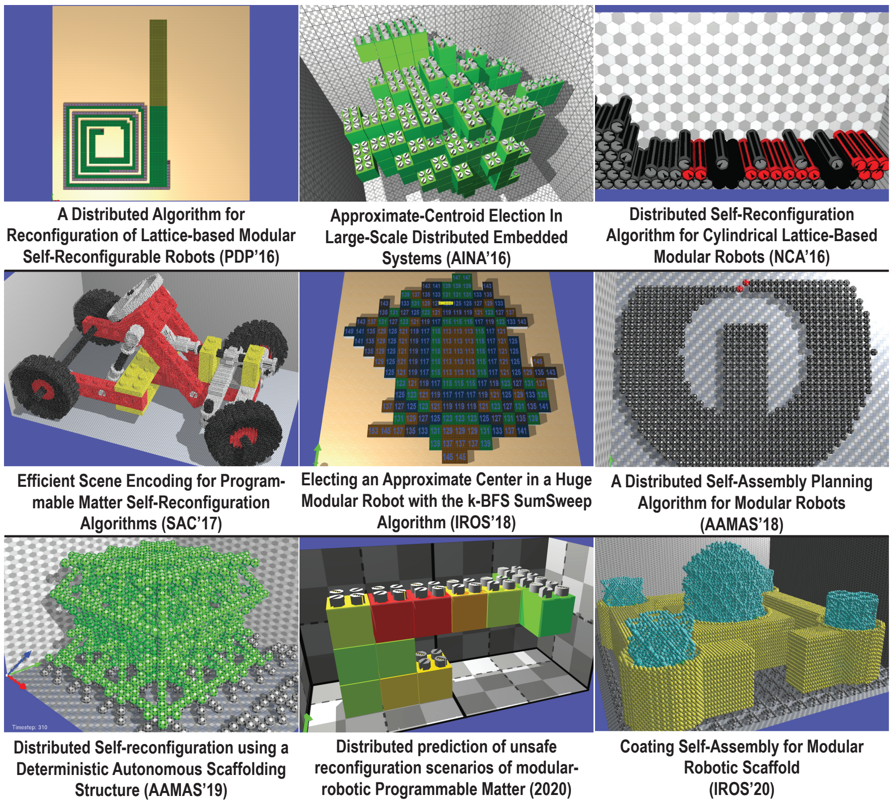

A Distributed Algorithm for Reconfiguration of Lattice-based Modular Self-Reconfigurable Robots (PDP'16)
Approximate-Centroid Election In Large-Scale Distributed Embedded Systems (AINA'16)
Distributed Self-Reconfiguration Algorithm for Cylindrical Lattice-Based Modular Robots (NCA'16)
Efficient Scene Encoding for Programmable Matter Self-Reconfiguration Algorithms (SAC'17)
Electing an Approximate Center in a Huge Modular Robot with the k-BFS SumSweep Algorithm (IROS'18)
A Distributed Self-Assembly Planning Algorithm for Modular Robots (AAMAS'18)
Distributed Self-reconfiguration using a Deterministic Autonomous Scaffolding Structure (AAMAS'19)
Distributed prediction of unsafe reconfiguration scenarios of modular-robotic Programmable Matter (2020)
Coating Self-Assembly for Modular Robotic Scaffold (IROS'20)

Fig. 3: Several results from previous work based on *VisibleSim* across several module types and tasks.

(e.g., cubes) and mixed-density configurations with compact components linked by a critical path (e.g.,the dumbbell-like shape).

We have also modeled the *Blinky Blocks* hardware clocks in *VisibleSim* and evaluated the synchronization precision of the Modular Robot Time Protocol (MRTP) [16] — a protocol for providing global time synchronization across a modular robotic system — both with hardware modules and simulations. Experiments were conducted on a doubled L-shaped system composed of 10 *Blinky Blocks* over an hour, with a synchronization period of 5 seconds. Synchronization error distribution looks Gaussian both in simulation and hardware experiment results [14]. In the hardware *Blinky Blocks* system (resp. in *VisibleSim*), MRTP has an average precision of 0.06 ms (resp. -0.11 ms) and a standard-deviation of 1.62 ms (resp. 1.40 ms).

Results obtained using *VisibleSim* show a very high fidelity to the hardware results, which indicates that *VisibleSim* is able to perform an accurate timing simulation of the algorithms.

Shape	Size (module)	Diameter (hop)	Average execution time ± standard deviation (ms)		Absolute error of the average execution time simulator vs hardware (ms) (relative error)
			Hardware	Simulator	
Line	5	4	234 ± 1	244 ± 3	10 (4.27%)
	10	9	545 ± 5	544 ± 5	1 (0.18%)
	50	49	2873 ± 23	2885 ± 17	12 (0.42%)
Square	9	4	598 ± 45	588 ± 14	10 (1.67%)
	25	8	1117 ± 30	1119 ± 27	2 (0.18%)
	49	12	1684 ± 48	1686 ± 44	2 (0.12%)
Cube	27	6	1229 ± 56	1214 ± 31	15 (1.22%)
	64	9	1927 ± 51	1941 ± 33	14 (0.73%)
Dumbbell	59	15	1262 ± 56	1252 ± 57	10 (0,79%)

Table 1: Average execution time of ABC-CenterV1 on hardware Blinky Blocks and in simulations. Statistics on the execution time were computed over 25 runs for every configuration.

5.3 *Scalability*

In order to demonstrate the scalability of the *VisibleSim* simulation framework, we have designed a stress test experiment which consists in simulating a sort of *brownian motion* of as many modules as possible, within a square grid. The underlying *BlockCode* program is quite straightforward:

- At the start, a single leader module *activates* and sends an *activation* message to all its neighbors.
- Upon reception of an *activation* message, modules turn into the *activated* state.
- *Activated* modules then alternate between a 0.5 s wait, and a random motion lasting 1 s.
- When a motion ends for a module, it sends an *activation* message to its new neighbors, if any, before starting the next wait/move cycle.
- The simulation ends when all modules are in the *activated* state.

This simple distributed program will thus propagate agitation across an entire modular robotic system, generating a massive number of messages, motions, and wait events in the process. The aim is therefore to stress the *VisibleSim* scheduler as much as possible and show that a graphical simulation is still possible with a massive robotic ensemble. Executions of this stress test program can be seen in the video mentioned in footnote 2.

We run the program with a large set of square configurations and for each of them we compute the number of messages and the number of displacements that are necessary to activate the entire robotic ensemble. For each size of configuration, the initial set of modules is made by growing a tree of modules from a regular list of seeds, ending when branches reach a cell that is already in a filled state.

Figure 4 shows the number of messages and displacements as a function of the number of robots in the several configurations. As shown in the figure, we are able

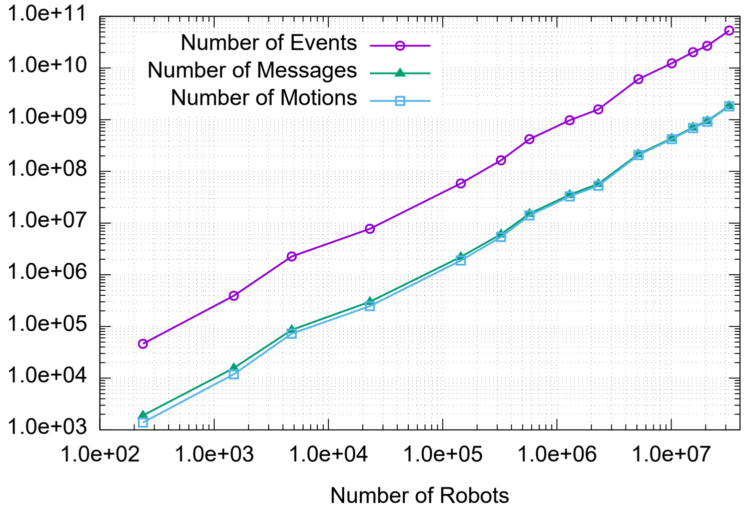


Fig. 4: Number of motions and messages simulated during the stress test experiment (Log-Log plot).

to simulate more than 32 million robots communicating and moving through the grid, which is to the best of our knowledge a new record in the field of modular robotic simulation. The first experiments, dealing with up to 3 million robots have been made on a laptop with 32 GB of RAM, and all subsequent simulations have been made on a server with 3 TB of RAM.

6 Conclusion and Future Work

In this paper, we have introduced *VisibleSim*, a C++ framework for simulating large-scale lattice-based distributed modular robotic ensembles. It differs from other modular robot simulators in its philosophy as a behavior-focused simulator, and its corresponding discrete-event-based style of scheduling. Various modular robotic designs supported by *VisibleSim* have been introduced, along with how to add new architectures by instantiating the OOP simulator framework, and implementing user applications. We have shown that it doubles as a powerful visualization software for effectively communicating research results, and that the simulator is flexible and easy to customize. Finally, we have outlined the versatility, reliability, and scalability of *VisibleSim*, by showing diverse usages of the software in published research, outlining the accuracy of simulations, and performing graphical simulations with more than a million individual modules. We therefore argue that *VisibleSim* can benefit any present of future research on the algorithmic foundation of modular robotic systems, especially since it is freely available as open source software. *VisibleSim* is an ongoing project and there are a number of features that are currently under investigation, detailed below. In its current implementation, all the scheduling tasks are performed on a single thread. While it guarantees an accurate simulation, this also limits the scalability of the software. We are thus enabling multi-thread scheduling for the simulator, which raises a number of challenges for the preservation of the integrity of the simulation flow. Moreover, with distributed algorithms being notoriously difficult to develop and debug, we are seeking to implement DPRSim-style debugging [18] to provide critical support to application development.

Acknowledgment

This work was partially supported by the ANR (ANR-16-CE33-0022-02), the French Investissements d'Avenir program, the ISITE-BFC project (ANR-15-IDEX-03), and the EIPHI Graduate School (contract ANR-17-EURE-0002).

References

1. Ahmadzadeh, H., Masehian, E., Asadpour, M.: Modular Robotic Systems: Characteristics and Applications. Journal of Intelligent & Robotic Systems **81**(3), 317–357 (2016). DOI 10.1007/s10846-015-0237-8. URL https://doi.org/10.1007/s10846-015-0237-8
2. Ashley-Rollman, M.P., Pillai, P., Goodstein, M.L.: Simulating multi-million-robot ensembles. In: 2011 IEEE International Conference on Robotics and Automation, pp. 1006–1013. IEEE, Shanghai, China (2011). DOI 10.1109/ICRA.2011.5979807. URL http://ieeexplore.ieee.org/document/5979807/
3. Christensen, D., Brandt, D., Stoy, K., Schultz, U.: A unified simulator for Self-Reconfigurable Robots. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 870–876. IEEE, Nice (2008). DOI 10.1109/IROS.2008.4650757. URL http://ieeexplore.ieee.org/document/4650757/
4. Collins, T., Shen, W.M.: ReBots: A Drag-and-drop High-Performance Simulator for Modular and Self-Reconfigurable Robots. Tech. Rep. 714, University of Southern California, Information Sciences Institute (2016)
5. Davey, J., Kwok, N., Yim, M.: Emulating self-reconfigurable robots - design of the SMORES system. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4464–4469. IEEE, Vilamoura-Algarve, Portugal (2012). DOI 10.1109/IROS.2012.6385845. URL http://ieeexplore.ieee.org/document/6385845/
6. Fitch, R., Butler, Z.: Million Module March: Scalable Locomotion for Large Self-Reconfiguring Robots. The International Journal of Robotics Research **27**(3-4), 331–343 (2008). DOI 10.1177/0278364907085097. URL https://doi.org/10.1177/0278364907085097
7. Fitch, R., Butler, Z., Rus, D.: Reconfiguration planning for heterogeneous self-reconfiguring robots. In: Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, pp. 2460–2467 (2003). DOI 10.1109/IROS.2003.1249239
8. Gerkey, B.P., Vaughan, R.T., Howard, A.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: In Proceedings of the 11th International Conference on Advanced Robotics, pp. 317–323 (2003)
9. Kamimura, A., Yoshida, E., Murata, S., Tomita, K., Kokaji, S.: A Self-Reconfigurable Modular Robot (MTRAN) – Hardware and Motion Generation Software –. In: 5th International Symposium on Distributed Autonomous Robotic Systems, p. 10 (2002)
10. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), vol. 3, pp. 2149–2154. IEEE, Sendai, Japan (2004). DOI 10.1109/IROS.2004.1389727. URL http://ieeexplore.ieee.org/document/1389727/

11. Kramer, J., Scheutz, M.: Development environments for autonomous mobile robots: A survey. Autonomous Robots **22**(2), 101–132 (2007). DOI 10.1007/s10514-006-9013-8. URL http://link.springer.com/10.1007/s10514-006-9013-8
12. Lyder, A., Garcia, R., Stoy, K.: Mechanical design of odin, an extendable heterogeneous deformable modular robot. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 883–888. IEEE, Nice (2008). DOI 10.1109/IROS.2008.4650888. URL http://ieeexplore.ieee.org/document/4650888/
13. Michel, O.: Webots: Professional Mobile Robot Simulation. Journal of Advanced Robotics Systems **1**(1), 39–42 (2004). URL http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf
14. Naz, A.: Distributed Algorithms for Large-Scale Robotic Ensembles: Centrality, Synchronization and Self-reconfiguration. PhD Thesis, FEMTO-ST Institute, Univ. Bourgogne Franche-Comté, CNRS (2017)
15. Naz, A., Piranda, B., Goldstein, S.C., Bourgeois, J.: ABC-Center: Approximate-center election in modular robots. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2951–2957. IEEE, Hamburg, Germany (2015). DOI 10.1109/IROS.2015.7353784. URL http://ieeexplore.ieee.org/document/7353784/
16. Naz, A., Piranda, B., Goldstein, S.C., Bourgeois, J.: A Time Synchronization Protocol for Modular Robots. In: 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), pp. 109–118. IEEE, Heraklion (2016). DOI 10.1109/PDP.2016.73. URL http://ieeexplore.ieee.org/document/7445320/
17. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence **6**(4), 271–295 (2012). DOI 10.1007/s11721-012-0072-5. URL http://link.springer.com/10.1007/s11721-012-0072-5
18. Rister, B.D., Campbell, J., Pillai, P., Mowry, T.C.: Integrated Debugging of Large Modular Robot Ensembles. In: Proceedings 2007 IEEE International Conference on Robotics and Automation, pp. 2227–2234. IEEE, Rome, Italy (2007). DOI 10.1109/ROBOT.2007.363651. URL http://ieeexplore.ieee.org/document/4209415/. ISSN: 1050-4729
19. Rus, D., Vona, M.: Crystalline Robots: Self-Reconfiguration with Compressible Unit Modules. Autonomous Robots **10**(1), 107–124 (2001). DOI 10.1023/A:1026504804984. URL https://doi.org/10.1023/A:1026504804984
20. Salemi, B., Moll, M., Shen, W.m.: SUPERBOT: A Deployable, Multi-Functional, and Modular Self-Reconfigurable Robotic System. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3636–3641. IEEE, Beijing, China (2006). DOI 10.1109/IROS.2006.281719. URL http://ieeexplore.ieee.org/document/4058969/
21. Spröwitz, A., Laprade, P., Bonardi, S., Mayer, M., Moeckel, R., Mudry, P.A., Ijspeert, A.J.: Roombots—Towards decentralized reconfiguration with self-reconfiguring modular robotic metamodules. In: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on (2010). DOI 10.1109/IROS.2010.5649504
22. Støy, K., Nagpal, R.: Self-Reconfiguration Using Directed Growth. In: Distributed Autonomous Robotic Systems 6, pp. 3–12 (2007). DOI 10.1007/978-4-431-35873-2_1. URL https://doi.org/10.1007/978-4-431-35873-2_1
23. Thalamy, P., Piranda, B., Bourgeois, J.: A survey of autonomous self-reconfiguration methods for robot-based programmable matter. Robotics and Autonomous Systems **120**, 103,242 (2019). DOI 10.1016/j.robot.2019.07.012. URL https://linkinghub.elsevier.com/retrieve/pii/S0921889019301459
24. Vonásek, V., Saska, M., Košnar, K., Přeučil, L.: Global motion planning for modular robots with local motion primitives. In: Robotics and Automation (ICRA), 2013 IEEE International Conference on, pp. 2465–2470. IEEE (2013)
25. Yim, M., Zhang, Y., Lamping, J., Mao, E.: Distributed Control for 3D Metamorphosis. Autonomous Robots **10**(1), 41–56 (2001). DOI 10.1023/A:1026544419097. URL https://doi.org/10.1023/A:1026544419097