MDL: A Module Description Language for Chained Heterogeneous Modular Robots

A. Brunete, M. Hernando, E. Gambao, J. E. Torres, A. Castro-González

Abstract— **This paper presents the new concept of a description language for modular robots called module description language (MDL). A specific implementation of this concept has been designed and tested to describe the capabilities of modules of a chained heterogeneous robot (both from the point of view of movements and tasks it can perform). Thanks to MDL each module is able to report dynamically what is able to do (capabilities like rotate, extend, push forward, measure temperature or distance) to other modules or to a central control, and it is also possible to set up new actions for the whole robot, like combined movements. The description of current capabilities of modules allows the robot to react to failures at runtime.**

I. INTRODUCTION

This article describes the concept of a Module Description Language for Chained Heterogenous Modular Robots. Heterogeneous modular robots are robots composed by different types of modules. They are called n-modular [13], being n the number of different modules. On the other hand, homogeneous modular robots are robots composed by one single type of modules. "Chained" means that modules are connected in a row, as opposed to lattice robots, in which modules can be connected in a lattice.

Regarding the types of modules robots are composed of, most modular designs are homogeneous [7][14][10][15][9][3], at least in a locomotion sense (Polypod [12] and I-Cubes[11] have two types of modules, but one of them is passive, its function is mainly to carry the power supply, Molecule [5][4] have two different modules but performing the same type of movement). There is a lack of heterogeneous drive module combination. Thus, there is no clear state of the art regarding heterogeneous modular robots.

The robot proposed in our research [2] is composed by different types of modules. Heterogenous modular robots have several advantages: robots can be more compact, cheaper to design (expensive actuators and sensors are only used where needed), can perform specific tasks, just to say a few. For this robot an specific architecture has been designed (that will be reviewed in section II). One very important part of this architecture is MDL, which this article is dedicated to.

A. Brunete and Álvaro Castro-González are with the RoboticsLab at the Carlos III University of Madrid, 28911, Leganés, Madrid, Spain abrunete@ing.uc3m.es; acgonzal@ing.uc3m.es

M. Hernando, E. Gambao and J. E. Torres are with the Centre for Automation and Robotics (CAR) UPM-CSIC José Gutiérrez Abascal 2. 28006 Madrid. Spain. miguel.hernando@upm.es; ernesto.gambao@upm.es; joseemilio.torres@upm.es

A. Brunete has developed part of his work at the CeDInt (Research Centre for Smart Buildings and Energy Efficiency), Universidad Politécnica de Madrid. abrunete@cedint.upm.es

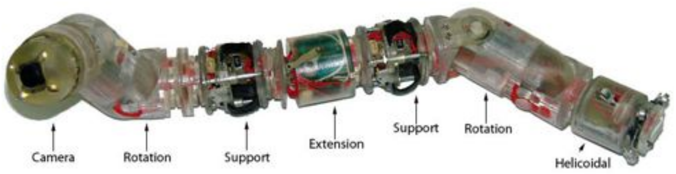

Fig. 1. Prototypes already developed

Since not all modules are the same, there is a need to find a way in which modules can communicate its capabilities to other modules, to a central control or to an operator (modules may be damaged or they may lose some functionalities at any time). And that is what MDL is designed for, as it will be explained in section III.

The idea is to set a procedure for module capabilities description so it is possible to develop new modules that are able to easily cooperate with the modules already developed.

MDL is essential to infer functions or skills for the whole robot from module features, thanks to rules and inference engines previously defined. MDL is very useful to develop new behaviors by combining module skills.

MDL has been designed as a part in the control architecture of the Microtub microrobot, but it can be used in any chained modular robot or even extended to lattice robots.

Finally, in section IV some of examples of use will be explained.

II. PREVIOUS WORK - SET UP

A. Robot description

MICROTUB is a semi-autonomous multi-configurable micro-robot for small-diameter pipe inspection and maintenance. It has been designed to explore pipes with a camera to detect breakages, holes, leaks and any type of defect. This micro-robot is composed of different modules, each of which performs a different task. Thus, multi-configurability is an essential characteristic that allows these modules to be easily interchanged depending on the task, without the need for reprogramming the micro-robot.

The different types of module developed (fig. 1) will be briefly described next, but more information can be found in [2] and [1]. The diameter of each module is 27 mm. The thickness of some parts is less than 1 mm. Table I shows a chart comparing the dimensions of all the modules.

The micro-robot is heterogeneous and modular, meaning that it is composed of different types of active (they are able to move) and passive (they have to be acted on) modules.

978-1-4577-2137-3/11/$26.00 © 2011 IEEE

To assemble them together, a common interface has been built. This interface allows for the mechanical and electrical connection between modules. The electrical bus is composed of eight wires:

- Power (5v) and ground
- I^2C communication: data and clock
- Two synchronism lines (in and out) for low-level communication between adjacent modules
- Two auxiliary lines for general purposes (for example, to transmit the video signal from the camera).

Each module includes an electronic control board which performs the following tasks:

- Control of actuators
- Communication via I^2C, or with adjacent modules
- Manage several types of sensors
- Auto-protection and adaptable motion
- Self-orientation detection
- Low-level embedded control

Microtub modules are:

1) *Camera/Contact module:* This module plays two roles. First, as a camera, is used for environment information acquisition, to detect holes, breakages or cracks in the pipes. Second, as a contact sensor, it is able to detect if the microrobot is facing an obstacle.

The module is provided with a CMOS B&W camera which allows to visualize the inner part of the pipe and with three contact sensors which allow to detect obstacles.

2) *Rotation module:* The rotation module is a two degrees of freedom module that allows rotations in the horizontal and vertical planes. A set of these modules put together can perform an undulatory movement (snake-like) that makes the robot go forward. It is composed by two commercial mini-servomotors.

3) *Inchworm modules:* Two modules have been developed to perform inchworm (or worm-like) movements: an extension module and a support module. The inchworm mode of locomotion allows the robot to maneuver in small spaces. Another advantage of this kind of motion is that the robot manages to maintain a firm grip on the surface at all times. The support module is used to fix the microrobot to the pipe, so this module does not move. And the extension module is used to extend the robot (make it go forward), and to turn right and left.

4) *Helicoidal module:* The helicoidal module was designed to be a fast drive module able to push other modules. It is composed of two parts: a body and a rotating head.

When the head turns, it goes forward in a helicoidal movement (helped by the distribution of the wheel making a 15 degrees angle with the vertical) that pulls the body of the microrobot forward. The wheels of the body help to keep the module centered in the pipe and avoid the turn of the body.

B. Control architecture description

For the proposed microrobot a semi-distributed control has been chosen with a central control (CC) that takes decisions for the whole robot and an embedded behavior-based control in every module, capable to react in real time to unpredicted events (figure 2). There is also an interpreter acting between the central control and the behaviors, the *heterogeneous agent*.The heterogeneous agents of all modules form the heterogeneous layer. It is called a middle layer because it acts between the CC (highest level layer) and the onboard control (lowest level layer). Thus, control is divided in (figure 3) as follows:

- Central Control (CC): It could be a PC or one of the modules. Nowadays it is a PC. In the future it will be one of the modules in order to make the robot autonomous. It includes the High Control Layer, that collects information from the modules, process it, and sends information (on the situation and state of the robot) and commands (with objectives) back to the modules. It also helps modules to take decisions and to coordinate them. It is also in charge of planning. It is composed of several parts, amongst which it is an inference engine and a behavior-based control.
- Onboard Control: it is embedded in each module and it is based on behaviors. It includes the layers:
  - Heterogeneous (middle) Layer: agent that translates commands coming from the CC into specific module commands. For example, it translates the command ”extend” into movements of the servomotors.
  - Low Control Layer: composed of behaviors. It allows modules to react in real time (for example to sense external and internal stimuli, as overheating, unreachable positions, adapt to the pipe shape, etc.) and to perform tasks that don’t need the CC (movements, communication with adjacent modules, simple tasks, etc.).

MDL has a very important role in the architecture, because it is the way modules can tell the CC what movements and tasks they are able to do, both at configuration and run time.

C. Simulator description

A simulation environment has been developed to provide an efficient scenario for prototype testing and for verification

TABLE I
MODULES CHARACTERISTICS

Module	Length[mm]	Diameter[mm]	Weight[g]
Camera	25	27	6,5
Support	27	27	12,5
Extension	30	27	16
Rotation	64	27	27
Helicoidal	28	27	15

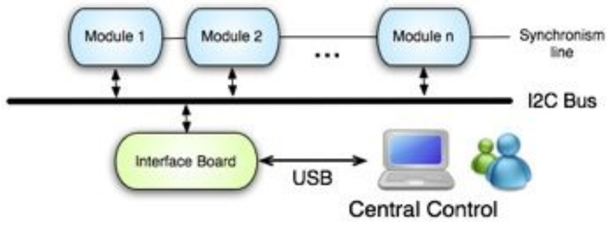

Fig. 2. Control Scheme

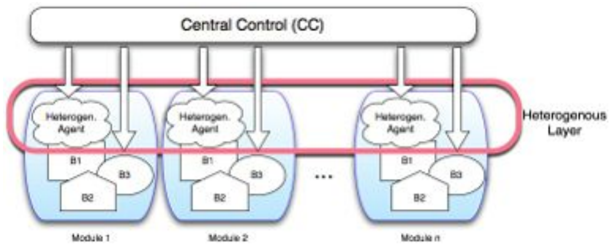

Fig. 3. Control Layers

of control algorithms, hardware design, and exploration of system deployment scenarios.

It can also be used to verify the feasibility of system behaviors using realistic morphology, body mass and torque specifications for servos.

The simulator is built upon an existing open source implementation of rigid body dynamics, the Open Dynamics Engine (ODE), an open-source physics simulation API, which allows to perform on-line simulation of rigid body dynamics, and to define a wide variety of experimental environments and actuated models. Based on ODE, a whole simulation system has been developed from scratch, including mechanical features of modules, DOF, movement ranges, servomotors, communications, processing units, etc.

In addition to simulating the dynamic behavior of the robot, the simulator is able to emulate electronics, control, communications and processing system. Thus, protective behaviors such as preventing overheating of the motors can be tested. In the same way, the fact that each simulated microcontroller has its own thread assigned in the simulator has facilitated the implementation of synchronization mechanisms between modules.

Although there are several built modules, due to the limitation in the number of modules and some of their functionalities, most of the experiments regarding MDL have been developed in the simulator. The simulator has been previously validated through several experiments including servomotor, friction, speed and locomotion tests.

III. MODULE DESCRIPTION LANGUAGE (MDL)

MDL is a language created to describe the capabilities of one module to the CC and other modules.

MDL is essential for inferring functions or skills for the entire robot from the module features through rules and inference engines. With MDL, it is possible to create units (groups of modules) that are able to perform more complex tasks.

MDL is useful for developing new behaviors by combining module skills. It is based on a series of indicators that describe the tasks that the module can perform and a range of values that indicate the level of performance for each indicator.

It is important to note that module MDL indicators are dynamic and that they may vary during the development of a task. Thus, MDL indicators can malfunction or even stop working in the module. For example, the servomotor of a module may become stuck and may be able to turn only a percentage of its nominal range of motion. When the module detects this, it can communicate the issue to the robot via MDL commands.

TABLE II
MDL INDICATORS

Position in structure	Acronym	Name
1	Ext	Extend/Contract
2	Sup	Support
3	Push_Pipe	Push in pipe
4	Push_Flat	Push in open air
5	RotX	Rotate in its x axis
6	RotY	Rotate in its y axis
7	RotZ	Rotate in its z axis
8	Att	Attach / Detach to / from other modules
9	Sense_Front	Sense proximity front
10	Sense_Back	Sense proximity backwards
11	Sense_Side	Sense proximity lateral
12	Sense_Temp	Sense temperature
13	Sense_Humi	Sense humidity
14	Sense_Grav	Sense gravity
15	Grab	Grab
16	Drill	Drill
17	PS	Power supply

A. *Indicators*

Indicators are presented in Table II. Extend/Contract refers to the capability of a module to increase or decrease its length. By contrast, support refers to the capability of a module to become fixed to the pipe.

Push in pipe indicates that the module can go forwards by itself inside a pipe, whereas Push in open air refers to large spaces (including large-diameter pipes).

Rotate in its x/y/z axis means it has a DOF along that axis.

Attach and Detach to/from other modules is designed for self-reconfigurable modules with active links (SMAs or electromechanical latches)[6] [8].

Sense proximity front/backwards/lateral refers to any sensor that may detect obstacles. Sense temperature/humidity refers to the capacity for measuring temperature/humidity. Sense gravity indicates that it has accelerometers.

Grab indicates that it is able to grab objects. Drill indicates that it is able to make a hole.

Power supply indicates that it has a power supply to share.

B. *Values*

Each indicator is associated to a value that shows the level in which the module can perform such a task (III). This value is divided in four levels, from 0 to 3 (from no competence to good competence).

C. *Packaging*

The values corresponding to each module are packed into a single structure, an array of values from 0 to 3. For example, for the rotation module it would be:

```
MDL (Rot_mod) = [00003300000003000]
```

TABLE III
MDL VALUES

Value	Indicator
0	no competence for that skill
1	little competence
2	medium competence
3	good competence

and the helicoidal module:

```
MDL (Heli_mod) = [00310000000000000]
```

D. Communication protocol

Every time each module is demanded about its capabilities, it will send this array corresponding to the tasks that it can or cannot do. But in order to obtain the whole robot configuration and capabilities, a more complex sequence has been implemented (it is interesting to point out that the synchronism lines are needed to obtain the relative position of each module along the robot):

- The CC sends a MDS message (MDL starts) to all modules.
- All modules activate their synchronism lines.
- The one which is the first (it knows that it is the first because its S_{in} synchronism line is down) replies with a PC1 (Position in chain first) message including the MDL parameters (as shown previously).
- The first module puts the S_{out} synchronism line down, so the second module knows it goes next (because now its S_{in} synchronism line is down).
- The second module sends a PC1 message message and puts its S_{in} synchronism line down, so the first module knows it has finished.
- The CC keeps collecting all the messages.
- It goes the same way for all modules.
- When it is the turn of the last module (it knows it is the last because its S_{out} synchronism line is down) it sends its a a PCL (Position in chain last) message also with its MDL parameters.
- The CC send a MDF message (MDL finishes) , so the last module knows it has finished.

Then, if a rotation module is next to some other modules that can "rotate" in the same axis as it does, they can form a unit that moves as a snake. If a extension module is preceded and followed by modules that have the ability to "expand/contract", they can form a unit that moves as a worm.

E. Inference engine

The capabilities of the whole microrobot are the consequence of the combination of the capabilities of all modules and its position in the chain. It is not the same having a extension module in between two support modules, that having the extension module at the side of two support modules in a row. In the first case the chain could perform an inchworm movement while in the second one it is not possible.

To know what are the capabilities of the microrobot, a set of rules has been implemented. This rules can be extended either by writing new rules when new features appear or by developing new rules by learning.

In a general way, rules can be described as:

$$\sum sequential(MDL) + \sum adjacent(MDL) + \sum anywhere(MDL) => \sum robot(MDL)$$

For example, the rotation module does not have extension/contraction capabilities, but a unit composed of three rotation modules together do have that feature:

$$ADJACENT(Rot\_mod + Rot\_mod + Rot\_mod) + ANYWHERE(Open\_air) => Extension/Contraction\ (grade\ 3)$$

$$ADJACENT(Rot\_mod + Rot\_mod + Rot\_mod) + ANYWHERE(Pipe) => Extension/Contraction\ (grade\ 1)$$

To explain it clearly, let us suppose that a chain is composed by three module with the following MDL structures:

```
MDL(module 1) = [00003203001003000]
MDL(module 2) = [00003302130003000]
MDL(module 3) = [00003000200003111]
```

Each MDL structure is merged with each of the indicators masks to know if the module has that specific capability. For example, the mask for indicator RotX is [00001000000000000]. Merging each of the modules' MDL with the masks, gives [00003000000000000].

Afterwards, capabilities are inserted in the rules, and those which are fullfilled are activated. In this case the activated rules are:

$SEQUENTIAL(00003000000000000, 00003000000000000, 00003000000000000) + ANYWHERE(Open\_air) => 30000000000000000$

$SEQUENTIAL(00003000000000000, 00003000000000000, 00003000000000000) + ANYWHERE(Pipe) => 10000000000000000$

Thus, new capabilities are obtained, which can fill, in its turn, new rules to obtain newer capabilities.

Through these rules, the CC can deduce or infer the capabilities of a robot. It goes through all of the rules and selects those that are fulfilled. Then, the procedure is repeated by incorporating previously obtained conclusions. This procedure continues until there are no new rules fulfilled in a cycle.

The CC can also deduce or infer which modules are needed for a specific task. For example, if a robot needs to split into two, it can decide the optimal point at which to split such that each part of the robot keeps the necessary modules to accomplish the task to be executed.

F. Offline algorithms

A way to train the inference engine and to develop new rules are GA based offline algorithms. One of the use cases

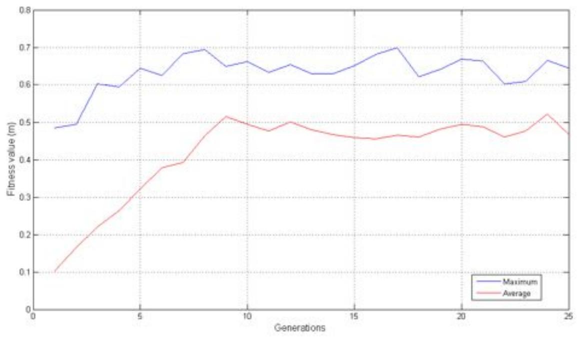

Fig. 4. Results of GA algorithm in a pipe with an elbow (configuration demand)

of the Microtub microrobot is the “configuration demand”, in which, for a specific mission, the CC selects the best modules to use and its position. For different tasks and scenarios (elbows, bifurcations, undulated terrain, etc.) the algorithm combines all types of modules to choose the optimal configuration and chain position.

As an example, the following experiment was designed to find the optimum configuration of a robot composed of a touch module and 8 rotation and helicoidal modules. The touch module should be in first place, and the other 8 modules could be either rotation or helicoidal.

In the experiment, starting from a population of 40 individuals randomly selected, the best individual was $THRHRHRHH$ where “T” stands for Touch, “H” for Helicoidal and “R” for Rotation.

Figure 4 shows the evolution of the fitness value with each generation. The highest value of the fitness function occurs at the generation 17, corresponding to the configuration with the maximum number of helicoidal modules that is able to negotiate the elbow: helicoidal modules placed between the rotation modules in order to turn. At the end of the microrobot it is possible to have two consecutive helicoidal modules since they can turn together with the previous rotation module.

IV. EXAMPLES OF USE

In this section some examples of movements performed by the heterogeneous modular robots are presented to prove the efficiency of MDL. Without its use it wouldn’t be possible to perform the combination of movements.

In the experiments, it will be mentioned the use of “passive” modules, meaning modules without drive capabilities. Passive modules will be represented by “battery” or “traveler” modules.

A. Rotation plus Extension Modules

Rotation modules can be used as support modules. In figure 5 a unit composed by three rotation modules can be used as a support module. It can expand and contract as support module. The configuration is three rotation modules (support) + extension + three rotation modules (support).

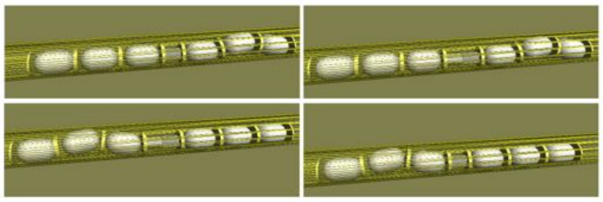

Fig. 5. Inchworm gait performed by rotation and extension modules

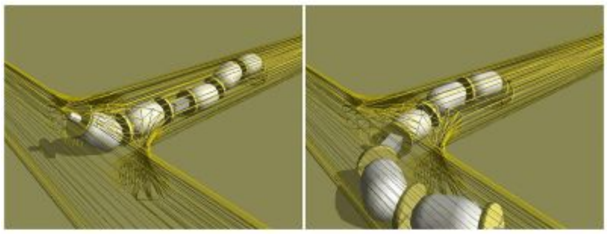

Fig. 6. Example of extension modules in snake-like configurations

$ANYWHERE(Rot\_X + Rot\_X + Rot\_X) => Sup$

$SEQUENTIAL(Sup + Ext + Sup) => Inchworm\ Gait$

Also the extension module can be used in a snake-like configuration, as shown in figure 6.

B. Rotation plus Support plus Extension plus Helicoidal Modules

By combining several types of modules, several types of gaits work together: snake-like, worm-like and helicoidal. Each of them fits better for each situation in pipes or open air.

Figure 7 shows an example of the camera/touch, rotation, helicoidal, extension and support modules working together and performing simultaneously vertical sinusoidal, helicoidal and worm-like movements.

In this case the following MDL commands are used: Sup and Ext in the right position to make inchworm movements, several Rot_Z modules in a row to make

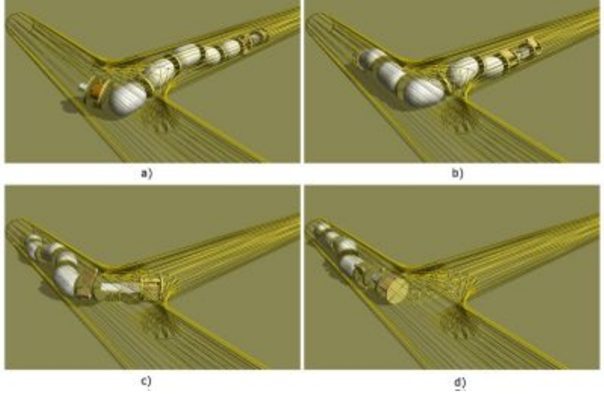

Fig. 7. Example of heterogeneous configuration

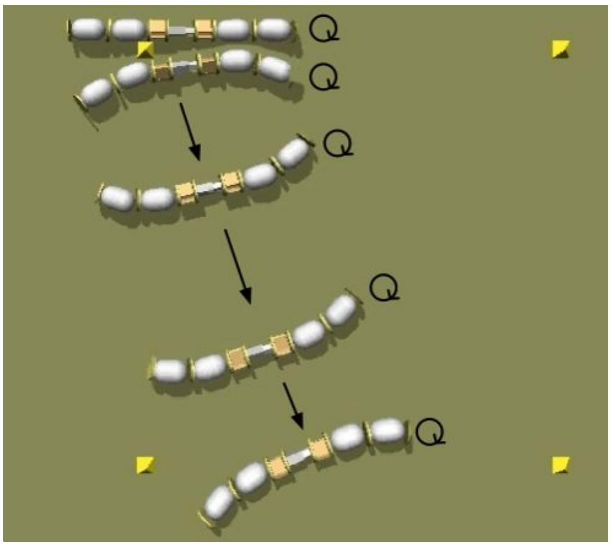
Fig. 8. Example of rolling movement in heterogeneous configuration: Rotation and inchworm modules

snake-like movements, and Push_Pipe anywhere for the helicoidal drive.

The MDL pseudocode is:
$\sum SEQUENTIAL(Rot\_X) + \sum ADJACENT(Sup + Ext) + \sum ANYWHERE(Push\_Pipe) => Heterogeneous\ Gait$

C. Rotation plus Inchworm Unit plus Rotation Modules

This configuration allows the microrobot to perform an inchworm movement inside pipes, and snake-like movements (making use of the rotational degree of freedom of the extension module) in the open air.

In figure 8 it is shown a chain composed by two rotation, support, extension, support and two rotation modules performing a rolling gait. This is very important because it proves that the locomotion gaits of the homogeneous snake-like configurations can be used with other module in between them.

$\sum ANYWHERE(Rot\_X + \sum Rot\_Y) => Rolling\ Gait$

V. CONCLUSIONS AND FUTURE WORKS

In this article the new concept of Module Description Language (MDL) has been presented. It is a language that has been developed to allow modules of a robotic system to transmit their capabilities. This information can be used by other modules or by a central control to process it and choose the best configuration and parameters for the microrobot. This is specially relevant to give the system the ability of recovering from permanent damages or temporal lose of some of its functionalities.

MDL also allows modules to be grouped in units to have different capabilities. Units can in turn be grouped in super-units to have newer capabilities, and so on.

MDL has been tested in the modular chained heterogenous microrobot Microtub and some examples of its use has been shown.

Future work will focused in the development of rules and algorithms that make used of the MDL parameters to develop new locomotion gaits.

VI. ACKNOWLEDGEMENTS

The authors would like to thank the contributions of Laura Santos and the Dpto. de Ing. Mecánica y Fabricación, Division de Ingeniería de Fabricación and Taller de Prototipado Rápido. E.T.S.I.Industriales, UPM.

REFERENCES

[1] A. Brunete, M. Hernando, and E. Gambao. Modular multiconfigurable architecture for low diameter pipe inspection microrobots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, May 2005.

[2] A. Brunete, J.E. Torres, M. Hernando, and E. Gambao. A proposal for a multi-drive heterogeneous modular pipe-inspection micro-robots. *International Journal of Information Acquisition (IJIA)*, 5:111–126, 2008.

[3] M.W. Jorgensen, E.H. Ostergaard, and H.H. Lund. Modular atron: Modules for a self-reconfigurable robot. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Japan, 2004.

[4] Keith Kotay and Daniela Rus. Efficient locomotion for a self-reconfiguring robot. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, Barcelona, Spain, 2005.

[5] Keith Kotay, Daniela Rus, Marsette Vona, and Craig McGray. The self-reconfiguring robotic molecule. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 424–431, 1998.

[6] H. Kurokawa, A. Kamimura, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata. M-tran ii: metamorphosis from a four-legged walker to a caterpillar. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003*, volume 3, pages 2454 – 2459, 2003.

[7] H. Kurokawa, K. Tomita, A. Kamimura, E. Yoshida, S. Kokaji, and S. Murata. Distributed self-reconfiguration control of modular robot m-tran. In *IEEE International Conference on Mechatronics and Automation*, volume 1, pages 254 – 259, Jul 2005.

[8] Satoshi Murata and Haruhsa Kurokawa. Self-reconfigurable robots. *Robotics &amp; Automation Magazine, IEEE*, 14(1):71–78, 2007.

[9] Wei-Min Shen, Maks Krivokon, Harris Chiu, Jacob Everist, Michael Rubenstein, and Jagadesh Venkatesh. Multimode locomotion via superbot reconfigurable robots. *Auton. Robots*, 20(2):165–177, 2006.

[10] Wei-Min Shen, Yimin Lu, and Peter Will. Hormone-based control for self-reconfigurable robots. In *Proceedings of the International Conference on Autonomous Agents*, Barcelona, Spain, 2000.

[11] C. Unsal and P. K. Khosla. Mechatronic design of a modular self-reconfigurable robotics system. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1742–1747, 2000.

[12] Mark Yim. New locomotion gaits. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2508–2514, 1994.

[13] Mark Yim, David Duff, and Kimon Roufas. Evolution of polybot: A modular reconfigurable robot. In *COE/Super-Mechano-Systems Workshop*, Tokyo, Japan, 2001.

[14] Ying Zhang, Kimon D. Roufas, and Mark Yim. Software architecture for modular self-reconfigurable robots. In *Proceedings of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2001.

[15] V. Zykov, E. Mytilinaios, B. Adams, and H. Lipson. Self-reproducing machines. *Nature*, 435:163–164, 2005.