The Assemble and Animate Control Framework for Modular Reconfigurable Robots

David Johan Christensen, Ulrik Pagh Schultz, and Mikael Moghadam

Abstract— **This paper describes the "Assemble and Animate" (ASE) control framework. The objective of ASE is to provide a flexible and extendable control framework, which facilitates rapid development and deployment of modular reconfigurable robots. ASE includes a simple event-driven application framework, a library of common control and adaptation strategies, and a module abstraction layer which allows ASE to be cross-compiled for a number of different modular robotic platforms and easily ported to new platforms. In this paper we describe the design of ASE and present example applications utilizing ASE for planetary contingency, adaptive locomotion, self-reconfiguration, and tangible behavior-based programming.**

I. Introduction

Reconfigurable robots consist of robotic modules which can be assembled or can self-reconfigure into numerous robot configurations [6], [24], [21]. Developing a control program for a modular robotic application is particular complex due to its distributed nature and the many potential sources of errors and limitations related to modules hardware, inter-module communication, and embedded software. ASE is an open-source[1] control framework designed to facilitate this development with the following design objectives:

Flexible: The polymorphic nature of modular robots makes their mechatronics flexible, but this is generally not true for the control which is often designed on a case-by-case basis. ASE is designed to be flexible so that it can be used and easily adapted to control very different robots in different application scenarios.

Extendable: Modular robotics is an active research area and new platforms and control strategies are continuously proposed. ASE is therefore designed to be simple to extend with new control strategies and simple to port to new modular robotic platforms.

Limited Resources: Most modular robot are controlled by small microcontrollers with limited computation and memory resources. ASE is designed to fit and run on microcontrollers having few kilobytes of RAM and at clock speeds of only a few MHz.

Code Reuse: In modular robotics research a control strategy must often be reimplemented for evaluation on new platforms or applications. ASE supports code reuse both between different applications and between different modular platforms (physical or simulated) by utilizing a module abstraction layer.

In the rest of this paper we first review related work in Section II, then provide an overview of the design and implementation status of ASE in Section III, and finally describe four example applications created utilizing the ASE framework in Section V.

D. J. Christensen and M. Moghadam is with the Center for Playware, Department of Electrical Engineering, Technical University of Denmark, DK-2800 Lyngby, Denmark {djchr, mikm}@elektro.dtu.dk

U. P. Shultz is with the Modular Robotics Lab, The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, DK-5230 Odense, Denmark ups@mmmi.sdu.dk

[1]Available from https://github.com/mimog/Assemble-and-Animate

II. Related Work

Numerous robotic development environment, frameworks, and middleware have been developed for different areas of robotics such as mobile robots and manipulators [8], [14]. In several aspects the ASE framework is similar to such systems since it provides an event-driven publish-subscribe infrastructure to facilitate responsive interaction between otherwise decoupled components (e.g. similar to ROS [16]). Further, unlike most such systems ASE requires no advanced OS features and can fits and runs on resource constrained embedded devices such as most modular robots (e.g. unlike Player [22] which requires a TCP stack and threads). Also, ASE provides hardware abstraction for typical modular robots equipped with simple neighbor-to-neighbor communication and a library of common algorithms for distributed coordination, adaptation, and control of modular robots. Finally, ASE is much more limited than systems such as ROS, Player/Stage, and Webots [11] since it does not provide support for multiple programming languages with higher-level abstractions, visualization/debugging tools, integrated simulation environments, or powerful algorithms for conventional monolithic robots such as vision, planning, SLAM, etc. In short ASE is designed as a minimalistic embedded distributed control framework for modular robots, while other systems are more powerful and generic they might not currently be a good match for such autonomous resource constrained modular robots.

Several domain specific languages (DSL) have been proposed for self-reconfigurable modular robots. DynaRole is a DSL which introduces abstractions for module roles and structure and compiles to virtual machine byte code for control diffusion [17], [18]. Meld [1] and LDP [5] are two programming language made primarily to program large ensembles of self-reconfigurable Catom modules. Meld also compiles to TinyOS nesC applications. ASE does not impose a new programming language but does therefore also not provide the same level of abstraction as these DSLs. Instead, ASE is written in C (C99) to maximize portability and integration with existing and new platforms.

TinyOS [11] is an event-driven operating system designed for sensor networks, a similar operating system is not yet available for modular robots, several existing systems therefore use TinyOS [7], [15], [2]. ASE can run on top of TinyOS as it has been done for several versions of the ATRON.

Simulators can facilitate control development and enable scale and optimization experiments not feasible on a physical platform. Often a custom simulator is created each time a new modular robot is developed. However, several simulators are able to simulate several different systems. The open-source Unified Simulator for Self-Reconfigurable Robots (USSR) is an physics-based simulator designed to simulate a number of different platforms [3]. Also the commercial Webots simulator by Cyberbotics Ltd [23] is able to simulate several existing modular robotic systems and can be extended to simulate new systems without modifying its source code. Both of these simulators are based on the Open Dynamic Engine (ODE) for collision detection and rigid body dynamics [19]. ASE applications have previously controlled modular robots in both of these simulators.

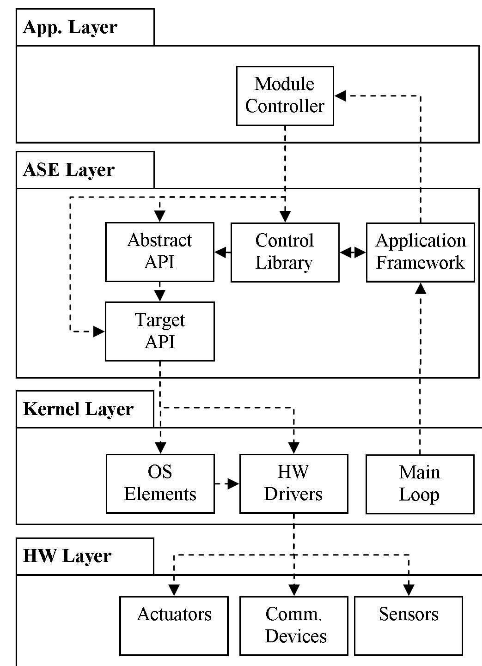

Fig. 1. Layered architecture of a modular robot using the ASE framework.

III. The ASE Control Framework

A. Overview of Architecture

The objective of ASE is to provide a flexible and extendable control framework which enables rapid development and deployment of modular reconfigurable robots. To fulfill this objective, the ASE control framework is designed with an layered architecture (illustrated in Figure 1). The architecture separates the “Hardware Layer” from the “Kernel Layer” which might be everything from a full OS, simple firmware, to a physics-based simulator. ASE interfaces to the “Kernel Layer” through a “Target API” component which is the only part of the “ASE Layer” that contains target specific code. The “Application Layer” contains a “Module Controller” which can be target independent if it does not use the specific “Target API”. The “ASE Layer” itself consist of the following parts: 1) Target Module API, 2) Abstract Module API, 3) Application Framework and 4) Control Library. Below we explain the design of each of these parts and explain how to utilize ASE and how to port it to new platforms.

B. Target Module API

The Target Module API is used by the Abstract Module API and the Module Controller to provide access to the module’s actuators, sensors, and communication devices by utilizing the kernel layer. In order to port the ASE framework to a new modular platform a Target Module API must therefore be implemented. As an example, the following functions are part of the Target API for the ATRON system:

```
long atronApi_getMsTime();
int atronApi_sendMessage(...);
void atronApi_rotateToDegree(float rad);
void atronApi_connect(int channelID);
int atronApi_getBatteryLevel();
...
```

In most cases module controllers developed in ASE will use the Target API for application specific hardware access. Therefore, to maintain application portability between the physical and simulated robots the Target API must be identical for both the physical and simulated target and the application should only access the “Kernel Layer” through the Target API.

C. Abstract Module API

The Abstract Module API uses the Target Module API to implement a number of functions required by the Control Library and Application Framework in a target independent fashion. For example the following functions are part of the Abstract Module API:

```
long getLocalMsTime();
int getHardwareID();
int getNumberOfNeighborChannels();
int sendMessageToNeighbor(char* msg, ...);
int setActuatorPosition(float pos, int actID);
float getActuatorPosition(int actID);
...
```

As a general rule, to keep the interface generic, every actuator, sensor and communication channel is indexed (0, 1, 2, ...) and return values for sensors and actuators positions are scaled from 0.0-1.0.

D. Application Framework

The Application Framework provides an event-driven intra-module publish-subscribe infrastructure for both the Module Controller as well as for the Control Library. The Framework imposes an initialization phase followed by a runtime phase. Here we will describe the Framework through a couple of small controller examples.

The first step in the initialization phase is for the Kernel Layer to initialize ASE by invoking `ase_init(...)`. This in turns invokes the Module Controller's `controller_init()`, which will potentially initialize parts of the ASE Control Library or subscribe for a number of Events. The following is a fully functional module controller which will subscribe to an `ACT_EVENT` and therefore receive a callback periodically which is used to drive a sine-based control of actuator indexed 0:

```
#include <math.h>
#include <ase/Infrastructure.h>
#include <ase/targets/AbstractModuleApi.h>

void ctrl_act(char* topic, Event_t* event) {
  float pos = (1 + sin(getLocalTime()))/4.0f;
  setActuatorPosition(pos, 0);
}

void controller_init() {
  EventManager_subscribe(ACT_EVENT, ctrl_act);
}
```

In the runtime phase the Kernel Layer periodically evokes `ase_act()` which will cause the Framework to publish an `ACT_EVENT`. Generally, events are used throughout ASE both in the Application Framework and Control Library as a way to decouple components and ensure responsiveness. With respect to inter-module data communication, ASE imposes a specific 3 byte message header:

Type	Label	Data
1 byte	2 bytes	N bytes

The message header consists of a Type and a Label field. The Type field enables ASE to route a message (currently only intra-module) to the specific strategy type which is subscribing for that message type. The Label field allows several strategies of the same type but with different labels to communicate directly. Usually a particular communication stack provided by the target platform requires another message format, and then ASE messages are simply wrapped inside such a message at the Target API. The following example illustrates the use of timers and how a controller can subscribe to a particular message type `LED_MSG`:

```
#define LED_MSG     128
Timer_t* timer;
void timer_fired(int id) {  //evoked every 200 ms
   char msg[4] = {LED_MSG, 0, 0, atronApi_getLeds()};
   sendMessageToAllNeighbors(msg, sizeof(msg));
}

void handle_msg(Msg_t* msg) {
   atronApi_setLeds(msg->message[3]+1);
}

void controller_init() {
   MsgManager_subscribe(LED_MSG, handle_msg);
   timer = TimerManager_createPeriodicTimer(
                            200, 0, timer_fired);
}
```

This ATRON controller creates a periodic timer to send messages contaning its LEDs state to neighbor modules. The LEDs will count if the modules are able to communicate. The same mechanisms for message routing and timers are used by the strategies in the Control Library.

E. Control Library

The Control Library part of ASE provides a number of strategies which are self-contained, self-driven, and target-independent code that can be combined into specific module controllers to produce the desired robot behavior. The different categories are:

Control: Numerous ways to control modular robots have been proposed. ASE provides generic implementations of some of the most common.

Adaptation: Modular robots are more than conventional robots required to be adaptive due to their metamorphic nature. Therefore, ASE provides a number of machine learning algorithms to help implement adaptive applications in ASE.

Communication: ASE provides a number of distributed communication strategies to simplify coordination of the different modules within a robot.

Behaviors: Several control strategies and machine learning strategies have been combined into behaviors which are included in ASE.

Arbitration: Mechanisms to arbitrate several active behaviors are included in ASE.

Tools: Various tools and utilities, which are used by ASE itself, or can be used directly by the application.

In order to simplify the controller development the strategies are self-contained and self-driven when appropriate. The self-driven behavior is implemented through the use of special components which manage a particular strategy. The controller will initialize and start a strategy, which will then run independently by using the event-driven infrastructure provided by the ASE applications framework. The controller can modify the strategy's runtime behavior through its API or the strategy may publish an event or use a call back function to interact with the module controller. Further, to make a strategy self-contained its data is encapsulated in a single data structure which allows the controller to concurrently utilize several strategies of the same type. As an example consider the following simple controller:

```
Gradient_t *g0;
void handler(char* topic, Event_t* event)  {
  Gradient_t* g = (Gradient_t*)event->val_prt;
  ase_printf("Gradient=%i\n", Gradient_getValue(g));
}

void controller_init() {
  EventManager_subscribe(
             GRADIENT_UPDATED_EVENT, handler);
  g0 = GradientManager_createGradient(99, 1.0f);
  GradientManager_startGradient(g0);
  if(getHardwareID()==3) {
    Gradient_setSeed(g0, true, 5);
  }
}
```

This controller creates a gradient with label 99 which state is transmitted to the neighbor modules at a rate of 1.0 Hz. The gradient on module with id = 3 is set to seed with a value of 5 so that its neighbors will get gradient value 4, and so forth. Each time a gradient value is changed an event is triggered causing the handler function to print out the current gradient value.

Module	Simulator	Kernel	MCU
ATRON [15]	USSR	TinyOS	Atmega128
ATRON-FPGA [2]	USSR	TinyOS	MicroBlaze
LocoKit [10]	-	USB2Dynamixel	Remote control
PlayTiles [12]	-	Custom Firmware	Atmega1280
Bioloid	-	CM510 Firmware	Atmega2561
Roombots [20]	Webots	-	PC only
Odin [13]	USSR	-	PC only
M-TRAN [9]	USSR	-	PC only

TABLE I

MODULAR ROBOTICS TARGETS INCLUDED IN ASE

IV. CURRENT IMPLEMENTATION STATUS

A. Implemented Targets

The currently available modular robotic targets in ASE are listed in Table I. In addition ASE has been used on several not-yet-public platforms. The most resource constrained of these microcontrollers is the Atmega128, which has 4KB SRAM, 128 KB Flash and runs at 16 MHz, however, we expect no issues with running ASE on even more resource constrained systems. ASE has been made to run as a standalone kernel with the hardware drivers implemented directly in the Module Target API or on top of TinyOS which provides communication stacks, drivers, etc. In addition to the physical platforms ASE has been used to control modular robots both in the USSR and the Webots simulators.

B. Implemented Control

The Control Library is partly developed to support the authors own research in modular robotics control. The included strategies are therefore somewhat biased. However, even if an application requires different strategies than those included, the ASE framework will support the development of event-driven applications which are easily ported between simulated and physical platforms. The strategies currently implemented in ASE include:

Control: Distributed strategies include average consensus, artificial gradients, distributed state machines, and central-pattern generators. Centralized strategies include artificial neural networks and record/playback control.

Adaptation: Machine learning techniques to enable adaptation includes random search, simultaneous perturbation stochastic approximation, genetic algorithms, particle swarm optimization, k-nearest-neighbors algorithm, and distributed reinforcement learning.

Communication: Strategies to enable remote control, information broadcast and gossip as well as shared timers and state sharing are included.

Behaviors: Generic behaviors include locomotion based on central pattern generators and adaptive CPG gaits. Self-reconfiguration behaviors are currently included exclusively for ATRON (based on DynaRole scripting).

Arbitration: Subsumption architecture.

Tools: Timers, protothreads, as well as several domain specific data structures.

V. EXAMPLE APPLICATIONS

This section describes four different example applications which are using the ASE control framework to achieve different objectives. The purpose here is to illustrate the diversity of ASE and the advantages of utilizing a multi-target framework.

A. Planetary Contingency Challenge

An ASE application was developed and utilized by the winning ATRON team at the Planetary Contingency Challenge at ICRA 2010. In this competition a robotic solution to an unforeseen problem must quickly be developed and deployed using only available materials. See Figure 2.

To address this challenge a user interface called "modular commander" was developed for flexible remote control from a laptop. The laptop communicated wireless with the ATRON robot by utilizing an XBee dongle. The ATRON-FPGA modules main processor is a softcore MicroBlaze running on an FPGA. The softcore was running TinyOS and the Target API was implemented directly in nesC utilizing TinyOS to control the actuators, communication etc. Before the competition started each ATRON-FPGA module was programmed with identical ASE controllers developed to provide flexible and reliable remote control and monitoring of the state of the module. A special RC strategy part of the ASE control library would enable the controller to subscribe to specific events triggered by received command messages. Command messages sent from the PC could be broadcasted or addressed to specific module IDs. The RC strategy would handle the potential routing of messages between modules and invoke handlers part of the controller when the module received a new command. The control handler would then implement the appropriate response by controlling the leds, center actuator, or connectors. A timer would periodically trigger a message to be sent back containing the status (communication event counts, battery level, motor position, etc.) for monitoring on the laptop.

B. Distributed Online Learning

To study morphology independent adaptive locomotion of modular robots we developed an ASE application utilizing central pattern generators and a stochastic optimization approach (SPSA). A CPG network of coupled oscillators would control the gait of a robot. Each DOF would be controlled by a single oscillator part of the CPG network. Based on velocity feedback the oscillators parameters (amplitude, phase, and offset) would be optimized online. The control application is fully distributed utilizing only neighbor to neighbor communication with each module computing its local subset of the CPG network and optimizing its own parameters based on SPSA.

The ASE framework enabled us to develop the application largely independent on the particular robotic platform. We therefore evaluated the strategy on two different platforms by only modifying the top-level platform specific part of the application. Most of the active code could be used on both platforms. The control strategy was first evaluated on simulated Roombots robots [4] and later on the physical LocoKit robot. The Roombots was simulated with the Webots simulator which can be programmed in C which allows

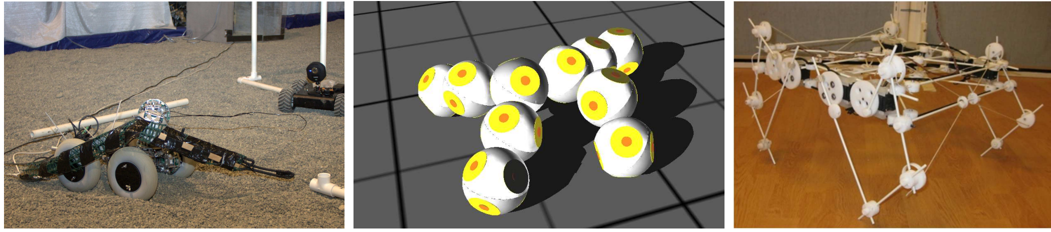

Fig. 2. (Left) ATRON robot equipped with several ad hoc tools for solving a problem during the 2010 ICRA Planetary Contingency Challenge. (Center) and (Right) Roombot simulated in Webots simulator and physical LocoKit robot, both controlled with the same adaptive strategy implemented in the ASE control framework.

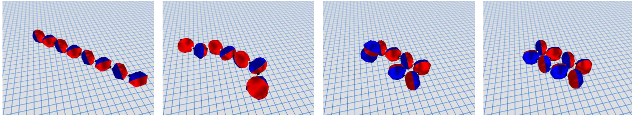

Fig. 3. ATRON self-reconfiguration sequence controlled with DynaRole script compiled to ASE application.

us to directly implement the Target API by utilizing the Webots API. The physical LocoKit quadruped was mounted on a boom which made the robot move in a circle. The rotation of the boom was measured by an encoder read by a microcontroller and sent using ZigBee to a PC. The PC was running the adaptive locomotion ASE controller which used the velocity feedback to optimize the open parameters of the coupled oscillators comprising the central pattern generator. The Target API for the LocoKit would utilize a USB to RS485 communication converter to remote control the robots Dynamixel actuators. We observed from these experiments that the same control strategy with only minor parameter adjustments was able to optimize the gait of both types of robots. The two robots are show in Figure 2.

C. DynaRole - Scripting ATRON Self-Reconfiguration

DynaRole is a domain specific language which amongst other things provide the possibility to script ATRON self-reconfiguration sequences [18]. Such scripts compile to distributed control programs which use a distributed state machine to coordinate the sequence of self-reconfiguration steps. This scripting approach is much simpler that having to write and debug the distributed control program by hand. We have therefore made a backend for DynaRole which compiles to ASE control strategies that can be utilized by a controller. In this way ASE can for example provide the locomotion control for several ATRON morphologies while the DynaRole scripts will enable the robot to self-reconfigure between the different morphologies. ASE enables a DynaRole script to first be debugged in the USSR simulator before it is cross-compiled and downloaded to the physical ATRON modules. As an example the following DynaRole script allows self-reconfiguration from an eight-module ATRON snake to a standard ATRON configuration called “Shift-8”, and moreover automatically provides the reverse sequence needed to reconfigure the robot back to the snake shape:

```
role M1 extends Module { require (self.id==1); }
role M2 extends Module { require (self.id==2); }
...
sequence snake2shift8 {
  M2.rotateFromToBy(0,180,false,150) |
  M3.rotateFromToBy(0,180,false,150) |
  M6.rotateFromToBy(0,180,true,150)  |
  M7.rotateFromToBy(0,180,true,150);
  M1.Connector[4].extend() |
  M1.Connector[6].extend() |
  M5.Connector[4].extend();
}
sequence shift82snake = reverse snake2shift8;
...
```

Figure 3 shows the corresponding self-reconfiguration sequence in the USSR simulator. Note how DynaRole allows both parallel and sequential actions using syntax of ‘|’ and ‘;’ respectively.

D. Playte - Programming with Behaviors

This ASE application was made to explore physical programming of modular robots. The objective was to create a very simple interface for kids to play with the behavior of modular robotic creatures that they potentially could build themselves.

To explore this objective we designed a tangible interface called Playte. Playte allows a user to control a robot with small behavior programs represented by special Lego behavior bricks containing an resistor for identification (see Figure 4). The Playte has ten slots where the user can place behavior bricks. The first row of five slots selects which behaviors are active in the robot. The last row of five slots allows the user to delete a behavior, record a behavior, train a behavior, and combine several behaviors into a new behavior using training. Training is performed with a gamepad controlling the robots actuators. The state of the Playte is repeatedly read by a microcontroller which sends the brick IDs through a PC to the ASE application running on the robot.

The ASE application provides the required functionality of the robot to allow it to be controlled by the state of the Playte and the gamepad. A number of simple behaviors are implemented and a subsumption architecture provides the arbitration between several active behaviors. Training is implemented by utilizing a k-nearest neighbor algorithm (k-NN). When a behavior brick is trained the system samples training data as a mapping from the robot’s sensor input to the motor output controlled with the gamepad. When the trained behavior brick is made active on the Playte the system uses the k-NN algorithm to find the appropriate

Fig. 4. (Left) The Playte: an tangible interface for behaviors-based robot control. (Center) Different predefined and trainable behavior-bricks. (Right) Example Bioloid robots that can be controlled from the Playte.

motor output in the given sensory situation. Similarly, for combining behaviors the behaviors are selected using number buttons on the gamepad (1-5) and a k-NN is trained to map the current sensory state to a selection of behaviors.

VI. Conclusion and Future Work

In this paper we presented the “Assemble and Animate” control framework for modular reconfigurable robots. ASE can, by utilizing an Abstract Module API, be cross-compiled for multiple different targets and new targets can be added simply by providing a Target Module API. To enable rapid development, ASE includes a Control Library containing common communication, control, and adaptation strategies. Further, an Application Framework provides an event-driven infrastructure to the application. Finally, ASE is designed to run on resource constrained modular robots. We have described the design of ASE and presented several applications on planetary contingency, adaptive locomotion, self-reconfiguration, and tangible behavior-based programming to illustrate its range of use. For future work, we plan to continue to maintain, extend, and improve the ASE control framework as part of our ongoing research on modular robotics control.

VII. Acknowledgements

The “Assemble and Animate” project was funded by the Danish Council for Independent Research.

References

[1] M. P. Ashley-Rollman, P. Lee, S. C. Goldstein, P. Pillai, and J. D. Campbell. A language for large ensembles of independently executing nodes. In *Proceedings of the 25th International Conference on Logic Programming*, ICLP ’09, pages 265–280, Berlin, Heidelberg, 2009. Springer-Verlag.

[2] D. Brandt, J. C. Larsen, D. J. Christensen, R. F. G. Mendoza, D. Shaikh, U. P. Schultz, and K. Stoy. Flexible, fpga-based electronics for modular robots. In *Proceedings of the IROS’08 Workshop on Self-Reconfigurable Robots & Systems and Applications*, Nice, France, September 22 2008.

[3] D. J. Christensen, U. P. Schultz, D. Brandt, and K. Stoy. A unified simulator for self-reconfigurable robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 870–876, 2008.

[4] D. J. Christensen, A. Sproewitz, and A. J. Ijspeert. Distributed online learning of central pattern generators in modular robots. In *Proceedings of the 11th International Conference on Simulation of Adaptive Behavior (SAB2010)*, Paris, France, August 2010.

[5] M. De Rosa, S. C. Goldstein, P. Lee, J. D. Campbell, and P. Pillai. Programming modular robots with locally distributed predicates. In *Proceedings of the IEEE International Conference on Robotics and Automation ICRA ’08*, 2008.

[6] T. Fukuda and S. Nakagawa. Dynamically reconfigurable robotic system. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA’88)*, pages 1581–1586, 1988.

[7] B. T. Kirby, M. Ashley-Rollman, and S. C. 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, NY, USA, 2011. ACM.

[8] J. Kramer and M. Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22(2):101–132, 2007.

[9] H. Kurokawa, K. Tomita, K. Kamimura, S. Kokaji, T. Hasuo, and S. Murata. Distributed self-reconfiguration of M-TRAN III modular robotic system. *International Journal of Robotics Research*, 27(3-4):373–386, 2008.

[10] J. C. Larsen, R. F. M. Garcia, and K. Stoy. Increased versatility of modular robots through layered heterogeneity. In *Proceedings of the ICRA Workshop on Modular Robots, State of the Art*, pages 24–29, Anchorage, Alaska, May 2010.

[11] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In *Ambient Intelligence*. Springer Verlag, 2004.

[12] H. H. Lund. Modular interactive tiles for rehabilitation: evidence and effect. In *Proceedings of the 10th WSEAS international conference on Applied computer science*, ACS’10, pages 520–525, Stevens Point, Wisconsin, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS).

[13] A. Lyder, R. F. M. Garcia, and K. Stoy. Mechanical design of odin, an extendable heterogeneous deformable modular robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS’2008)*, pages 883–888, Nice, France, September 2008.

[14] N. Mohamed, J. Al-Jaroodi, and I. Jawhar. Middleware for robotics: A survey. In *IEEE Conference on Robotics, Automation and Mechatronics*, pages 736–742. IEEE, 2008.

[15] E. H. Østergaard, K. Kassow, R. Beck, and H. H. Lund. Design of the ATRON lattice-based self-reconfigurable robot. *Autonomous Robots*, 21:165–183, 2006.

[16] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[17] U. P. Schultz, D. J. Christensen, and K. Stoy. A domain-specific language for programming self-reconfigurable robots. In *Workshop on Automatic Program Generation for Embedded Systems (APGES)*, pages 28–36, October 2007.

[18] U.P. Schultz, M. Bordignon, and K. Stoy. Robust and reversible execution of self-reconfiguration sequences. *Robotica*, 29:35–57, 2011.

[19] R. Smith. Open dynamics engine. www.ode.org, 2005.

[20] A. Sproewitz, A. Billard, P. Dillenbourg, and A. J. Ijspeert. Roombots-Mechanical design of Self-Reconfiguring modular robots for adaptive furniture. In *2009 IEEE International Conference on Robotics and Automation*, pages 4259—4264, Kobe, Japan, 2009.

[21] K. Stoy, D. Brandt, and D. J. Christensen. *Self-Reconfigurable Robots: An Introduction*. Intelligent Robotics and Autonomous Agents series. The MIT Press, 2010.

[22] R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2):189–208, 2008.

[23] Webots. http://www.cyberbotics.com. Commercial Mobile Robot Simulation Software.

[24] M. Yim, W-M Shen, B Salemi, Daniela Rus, M. Moll, H Lipson, and E Klavins. Modular self-reconfigurable robot systems: Challenges and opportunities for the future. *IEEE Robotics & Automation Magazine*, 14(1):43–52, March 2007.