Sensor networks and self-reconfigurable robots

Schultz, Ulrik Pagh; Støy, Kasper; Dvinge, Nicolai; Christensen, David Johan

Published in:
Proc. of Workshop on Building Software for Sensor Networks at OOPSLA 06

Publication date:
2006

Sensor Networks and Self-Reconfigurable Robots

BSSN'06 Position Paper

Ulrik P. Schultz, Kasper Støy, Nicolai Dvinge and David Christensen
Maersk Institute
University of Southern Denmark

Abstract

A self-reconfigurable robot is a robotic device that can change its own shape. Self-reconfigurable robots are commonly built from multiple identical modules that can manipulate each other to change the shape of the robot. The robot can also perform tasks such as locomotion without changing shape. We observe that the challenges in programming self-reconfigurable robots are similar to the challenges in programming sensor networks: a large number of low-end embedded systems connected using ad-hoc networking need to gather information using sensors and to communicate information to other systems in the network. However, there are significant differences as well, notably a self-reconfigurable robot must typically function autonomously and the network topology will evolve continuously as the robot changes its shape. Software for self-reconfigurable robots, beyond controlling shape-change and locomotion, remains largely unexplored, but we believe experiences from sensor networks to be applicable within this domain. Moreover, we present an initial design of a middleware for self-reconfigurable robots and relate it to similar solutions for sensor networks.

1 Introduction

A self-reconfigurable robot is a robot that can change its own shape. Self-reconfigurable robots are built from multiple identical modules that can manipulate each other to change the shape of the robot. The robot can also perform tasks such as locomotion without changing shape [9, 11, 16, 2, 20, 12]. Changing the physical shape of a robot allows it to adapt to its environment, for example by changing from a car configuration (best suited for flat terrain) to a snake configuration suitable for other kinds of terrain. Programming self-reconfigurable robots is complicated by the need to (at least partially) distribute control across the modules that constitute the robot and furthermore to coordinate the actions of these modules. Algorithms for controlling the overall shape and locomotion of the robot have been investigated (e.g. [3, 18]), but the issue of providing a high-level programming platform for developing controllers remains largely unexplored. Moreover, constraints on the physical size and power consumption of each module limits the available processing power of each module.

A sensor network is a distributed, embedded system dedicated to gathering sensor information and conveying this information to a central unit. Network connections are typically wireless and ad-hoc, and price, size, and power constraints impose severe resource limitations on the individual sensor nodes and their network communication. Research in providing a higher-level programming platform for sensor networks includes dedicated programming languages [7], mobile agents [5], and domain-specific languages [8]. Moreover, there is an extensive body of work in network protocols for ad-hoc networks, applicable within the field of wireless sensor networks; in this paper we do not consider the issue of network protocols.

We observe that the issues in programming self-reconfigurable robots are similar to the issues in programming sensor networks: a large number of low-end embedded systems connected using ad-hoc networking need to gather information using sensors and to convey this information to the appropriate party. Moreover, dynamic software updates are in both cases critical to enable evolution of the system after it has been deployed. Nevertheless, there are significant differences as well, notably that a self-reconfigurable robot must typically function autonomously and that the network topology will evolve continuously as the robot changes its shape. Furthermore, the modules of a self-reconfigurable robot need to coordinate their actions carefully to avoid modules disconnecting from the robot.

In this paper we explore the potential for synergy between sensor networks and self-reconfigurable robots, with a focus on software development. We observe that apart from the algorithms controlling change of shape and locomotion, software development for self-reconfigurable robots remains largely unexplored. We believe existing techniques for building software for sensor networks to be applicable within the domain of self-reconfigurable robots. Moreover, we present an initial design of a middleware for communication in self-reconfigurable robots, and relate this design to similar solutions for sensor networks.

Example: ATRON production system

This paper is based on the ATRON modular, reconfigurable robot [9]. An ATRON robot is constructed from a number of identical, spherical modules that each can attach to, manipulate, and communicate with their immediate neighbors; see Section 2 for a more detailed presentation of the ATRON system. As a concrete

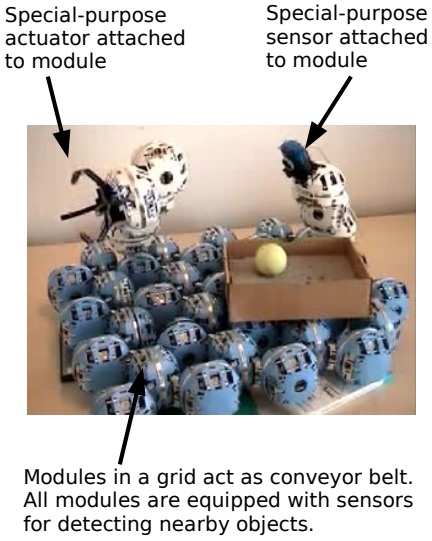


Figure 1. Production system using a modular, self-reconfigurable robot

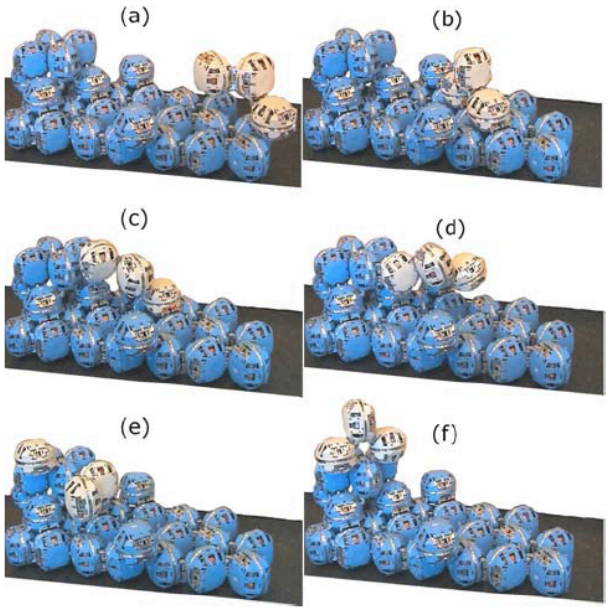


Figure 2. The ATRON self-reconfigurable robot. A group of three ATRON modules move across the robot using a sequence of rotations, connects and disconnects.

example, consider the robot shown in Figure 1, which is an early prototype of a small production system. The "floor" is built from ATRON modules that can function as conveyor belts. A special-purpose gripper and a camera have been fitted onto the end of the two arms, in principle allowing them to extract the ball from the box. In practice, nor the hardware or the software support such an action (in the current set-up the modules simply move randomly). In this paper we will look at the software challenges.

All ATRON modules are equipped numerous IR sensors allowing them to sense nearby objects. In the production system example, all modules can collect information about their immediate surroundings, for example allowing them to detect presence and dimensions of the box (the camera would however be needed to extract the ball from the box). Aggregating this sensor information and using it to control the overall behavior of the robot is one of the key challenges in programming the ATRON system. Moreover, coordinating the movement of the individual modules is a key challenge; the robot could for example transform part of the floor into an extra arm, or change the shape of the floor to accommodate objects with specific physical dimensions.

2 The ATRON Self-Reconfigurable Robot

The ATRON self-reconfigurable robot is a 3D lattice-type robot [9]. Figure 2 shows an example shape change in the ATRON robot. Each module has one degree of freedom. A ATRON module is spherical, is composed of two hemisphere, and can actively rotate the two hemispheres relative to each other. A module may connect to neighbor modules using its four actuated male and four passive female connectors. The connectors are positioned at 90 degree intervals on each hemisphere. Eight infrared ports, one below each connector, are used by the modules to communicate with neighboring modules and sense distance to nearby obstacles or modules. A module weighs 0.850kg and has a diameter of 110mm. Currently 100 hardware prototypes of the ATRON modules exist. Motion constraints on the modules affect their ability to self-reconfigure. The single rotational degree of freedom of a module makes its ability to move very limited: in fact a module is unable to move by itself. The help of another module is always needed to achieve movement. All modules must also always stay connected to prevent modules from being disconnected from the robot. They must avoid collisions and respect their limited actuator strength: one module can lift two others against gravity.

Other examples of self-reconfigurable robots include the M-TRAN and the SuperBot self-reconfigurable robots [11, 16]. These robots are similar from a software point of view, but differ in mechanical design e.g. degrees of freedom per module, physical shape, and connector design. This means that algorithms controlling change of shape and locomotion often will be robot specific, however general software principles are more easily transferred.

3 Programming Self-Reconfigurable Robots

3.1 Considerations

The primary challenges in programming modular, self-reconfigurable robots are controlling transformations (and thus movements in general) and propagation of information between individual modules. The issue of controlling transformations is an issue of coordinating the movements of the individual modules to produce the required overall effect on the robot. Numerous control algorithms with different characteristics have been investigated [18, 3], although the applicability such algorithms inadvertently are tied to the specific characteristics of the robot. The coordination can be local between modules, but may in general require coordination between different logical parts of the robot. For this reason, we believe inter-module communication to be an underlying, critical issue in programming such robots. In the specific case of the ATRON system, the communications library provides reliable communication of (short) messages

sent on specific IR ports: transmitting data on a specific port simply causes this data to be received on the corresponding port on the neighboring module, if present.[1] The communication in an ATRON robot is thus strictly neighbor-to-neighbor, with the neighbor relation evolving dynamically as the robot changes shape, which gives a strong analogy to sensor networks.

As a first step towards a middleware for ATRON, we are interested in providing a minimal abstraction over the basic communication mechanism in the ATRON modules. Rather than working in terms of raw sequences of bytes, we wish to provide a higher-level interface similar to remote method invocation. Moreover, rather than addressing other modules in terms of a specific IR port, which is dependent on the three-dimensional rotation of the module, we wish to provide a position-independent means of addressing neighboring modules. A key observation here is that shape and activity are connected in the sense that a module which is in a specific position will often be used for a specific purpose. Thus, it makes sense to address modules in terms of their functionality ("left wheel on this axle") rather than their specific position ("attached to connector at IR transmitter 5"). For now, the mapping between functionality and position is performed manually, but we are working on providing a declarative language for describing ATRON configurations, which should allow us to compute these mappings automatically.

3.2 Design

Our design is inspired by the physical characteristics of the ATRON modules as well as a combination of object-oriented programming and behavior-based control, specifically role-based control of modular robots [19]. The behavior of the robot at any given time is driven by a combination of sensor inputs and internally generated events. Roles allow modules to interpret sensors and events in a specific way, thus differentiating the behavior of the module according to the concrete needs of the robot.

From a conceptual programming perspective, we see each module as an object that can send and respond to events.[2] Reception of an event by a module causes the corresponding operation (defined on the module) to be executed; we note that there is not currently any automatic propagation of events to other modules, and that it is an open question how to best automate this propagation while conserving resources. Indeed, an event does not contain any routing information, since the physical juxtaposition of modules determines the receiver of the message. The set of events that a module can respond to at a given time is determined by the active role(s) of the module. For example, a module that plays the role of a wheel can respond to events such as "move forward" whereas the same module responds to a different set of messages when it plays the role of the tail of a snake. This basic form of message passing corresponds to a very simple form of one-way remote method invocation as known from distributed programming.

A role is a collection of named operations (similarly to interfaces in e.g. Java) but with various limitations such as the lack of a return value and only primitive types are allowed as parameters. A role may inherit from one or more super-roles, meaning that the set of operations contained in the super-roles are inherited by the role. Unlike an interface, a role declaration can as a convenience contain simple rules for delegating messages to other modules. For example, a module playing the role of an axle in a car may delegate specific messages to the wheels that are attached to it. All modules run the same program and hence can assume any role at any time; the decision of when to change roles is currently implemented manually by the programmer, but we envision role changes being prompted by transformations to the shape of the robot. The active role thus defines the reactive behavior, corresponding to the principle of role-based control as known from modular robots [19]. Active behavior would typically result from reactive behavior to time-based events. The question of timing thus becomes important; we have yet to address this issue, but imagine synchronizing locally between neighboring modules as a means of obtaining a globally (within the robot) consistent behavior.

We conceptually use the whole-part object-oriented design pattern as a means of aggregating modules [1]. Specifically, a connected set of individual modules ("parts") that all implement the same role can by coordinating their actions provide a "whole" implementing this role. For example, the modules that are used to build a car can all implement the role "moveable" which allows them to be addressed as a "moveable entity." An arm could thus be mounted on and control either a car or a walker without any changes to its programming, raising "programming to an interface" to the physical level.

3.3 Example

As a concrete example, consider the ATRON car shown in Figure 3, left. Two sets of wheels (ATRON modules with rubber rings providing traction) are mounted on ATRON modules playing the role of an axle; the two axles are joined by a single module playing the role of "connector." Communication in this robot can be performed based on the role declarations shown in Figure 3, right (this is a pretty-printed version of the real declaration, which is in XML and contains information on how the modules are physically connected). When a `move` event is delivered to the "connector" it forwards it to the two axles, which again forward it to the wheels. The action performed by the wheel is implemented separately (see below). Similarly for the `turn` event.

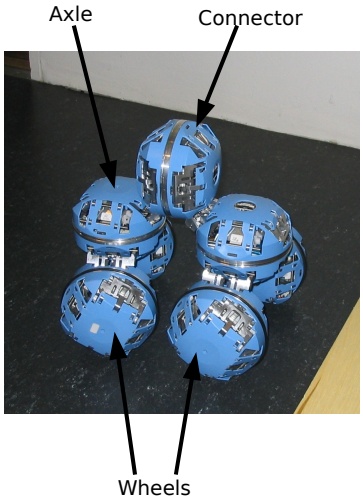
The roles thus provide a means of denoting the behavior of each module as it is used for a specific purpose in the robot. Moreover, the roles provide a simple and light-weight way of (albeit manually) routing events through the topology of the robot.

3.4 Implementation

The role specification language serves as an interface definition language for distributed communication between ATRON modules. This language can be compiled either to Java, for use with the ATRON simulator, or to C, for execution directly on the modules. At the time of writing, only the Java backend has been completed whereas the C backend is under construction. The Java backend simply generates an implementation of the proxy and state design patterns [6]. From the point of view of C, a role will compile to a skeleton that invokes C functions written by the programmer and to a proxy represented as a collection of C functions that can be used to send messages to other modules implementing this interface.

[1] Due to hardware problems, the physical communication is not currently reliable since it may erroneously transmit packets to a non-neighboring port.

[2] We could also see each module as a component, which may be more appropriate in the sense that a module can implement a fairly rich functionality. Nevertheless, we believe objects to be more appropriate as a metaphor since they have an active behavior and an internal state.




```
role Connector<Axle frontAxle,Axle rearAxle> {
  move(value) -> frontAxle.move(value),
                 rearAxle.move(value);
  turn(value) -> fronAxle.turn(value/-2),
                 rearAxle.turn(value/2);
}
role Axle<Wheel leftWheel,Wheel rightWheel> {
  move(value) -> leftWheel.move(value),
                 rightWheel.move(-value);
  turn(value) -> self;
}
role Wheel {
  move(value) -> self
}
```

Figure 3. Anatomy of a car: hardware and software configurations

As an alternative to programming the control logic in C, we are currently working on a compiler for an embedded version of Java. This compiler is based on the same principles as the JEPES system (which was shown to compile embedded Java programs to devices with only 4K ROM and 0.5K RAM) [14], but using Soot for whole-program optimization and with only a single back-end generating C code for improved portability. We expect that the Java programmer will be able to define roles using Java interfaces, but this issue has yet to be explored.

4 Synthesis: Robots and Sensor Networks

4.1 Sensor networks into robots

NesC is a small C-like language designed for event-driven programming [7]. It has a lightweight component model and a library dedicated to sensor network functionality. The language and component model are designed to enable whole-program analysis (for safety, e.g., from race conditions) and optimization (for size reduction). We believe nesC would be appropriate for programming ATRON systems using behavior-based control, where the behavior of the robot can be seen as an immediate response to sensor inputs, whereas a more traditional AI approaches based on a model would not fit well into the programming model. The nesC library functionality is centered around hardware access, data acquisition, and radio communication using the TinyOS network stack. If ATRON modules were equipped with a radio (see below) we expect that similar components would be useful, however additional components capturing features specific to self-reconfigurable robots would of course be required.

SNACK is a domain-specific language for generating sensor network applications from components written in nesC [8]. The basic language abstractions allow components to be configured and connected to form complete applications. The SNACK component library provides similar functionality to nesC but is structured differently to facilitate application generation. As was the case for nesC, given components that implement appropriate robot-specific functionality, SNACK should support building behavior-based controllers for ATRON modules.

Agilla is a mobile agent platform for sensor networks [5]. Here, the agent platform provides sensor network functionality, and agents are written in a simple script language that contains basic operations for gathering and processing data. To use Agilla with ATRON, we would need to extend the agent language with robot-specific operations, such as controlling and monitoring the actuators. Such an extended language would for example enable programming the robot using techniques from multi-agent systems. Nevertheless, since the robot is normally required to operate autonomously, the language would need to be powerful enough to express all the processing needed to control the robot based on its sensor inputs. For these reasons, we expect that the resulting complexity of the mobile agents and the platform would be unfeasible for implementation on small, embedded systems. Alternatively, each ATRON module could be considered an embodied agent in a multi-agent system, which fits perfectly with the idea of behavior-based programming.

Communication in the ATRON modules is currently hampered by the difficulty of working precisely with infrared signals. As an alternative to relying primarily on infrared signals, radio-based communication as used in wireless sensor networks is currently under considerations. Infrared communication would thus be used solely as a means of detected neighboring modules. Concretely, we are considered using the “Sun SPOT” hardware platform from Sun Microsystems [10]. This platform would facilitate experiments with software and potentially also enable the software to evolve as the robot is reconfigured to a different shape.

4.2 Robots into sensor networks

Modern, behavior-based robot control emphasizes autonomy and simple reactive behaviors based on sensor inputs, and can thus serve as an inspiration for achieving a higher degree of locality and autonomy in sensor networks [4]. The specific context of modular, self-reconfigurable robots further emphasizes this trend: here, each module contains limited resources but must nonetheless coordinate its actions with all other modules. When programming self-reconfigurable robots, sensor input from different modules needs to be aggregated, resulting in complex relations between sensors from different modules. Conversely, the short physical distance between modules makes wireless communication easier than in traditional wireless sensor networks since all modules are within radio distance of each other. Nevertheless, the requirements on conserving

power and bandwidth still prompts us to limit communication.

The *Robomote* is a concrete example of combining robotics and sensor networks [17]. Here, a Berkeley mote was equipped with wheels, enabling physical movement. This is however an example of a multi-robot system, where robots move individually and hence are not required to coordinate their actions, unlike modular self-reconfigurable robots where there is a tight coupling between each module. Moreover, the principal focus was on gathering sensor data, there were no experiments on e.g. using these data in real-time to control how the robots manipulate other physical objects.

Research on software for self-reconfigurable robot may also lay the foundation for more interactive sensor networks. In addition to being a distributed data collection and monitoring network, the self-reconfigurable robot can also physically interact with and manipulate its environment. This opens new challenges such as how to propagate sensor information in order for the robot or sensor network to make a timely response and how to coordinate actions across the robot and sensor network.

4.3 Synthesis

Our initial design of a middleware for modular, self-reconfigurable robots uses the concept of roles to structure behavior and allow it to vary according to the usage context of the module. We believe that complex sensor networks could benefit from a similar approach, for example if the same sensor can be used for different purposes. We consider the combination of nesC and TinyOS to a very interesting option for a future version of ATRON extended with wireless communication. Nevertheless, we are also experimenting with using Java to program the ATRON modules, so a combination of nesC and Java would also be an option: object instances could perhaps take the place of nesC modules if combined with a predictable optimization process based on program specialization [15, 13]. In general, we believe modular, self-reconfigurable robots to be an interesting challenge for existing techniques from sensor networks, due to the need for autonomously combining information from many parts of the network. Our goal is to enable easy programming of complex systems such as the production system setup shown in the beginning of the paper.

Acknowledgements

The ideas on ATRON programming described in Section 3 and the concept of ATRON modules as embodied agents in a multi-agent system (as mentioned in Section 4.1) are being developed in collaboration Yves Demezeau.

5 References

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley, 1996.

[2] A. Castano and P. Will. Autonomous and self-sufficient conro modules for reconfigurable robots. In *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 155–164, Knoxville, Texas, USA, 2000.

[3] D.J. Christensen and K. Støy. Selecting a meta-module to shape-change the ATRON self-reconfigurable robot. In *Proceedings of IEEE International Conference on Robotics and Automations (ICRA)*, pages 2532–2538, Orlando, USA, May 2006.

[4] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, January 2002.

[5] C.-L. Fok, G.-C. Roman, and C. Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'05)*, pages 653–662, Columbus, Ohio, June 2005.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[7] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03)*, San Diego, California, June 2003.

[8] B. Greenstein, E. Kohler, and D. Estrin. A sensor network application construction kit (SNACK). In *Proceedings of the 2nd international Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, USA, November 2004.

[9] M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund. Modular ATRON: Modules for a self-reconfigurable robot. In *Proceedings of IEEE/RSJ International Conference on Robots and Systems (IROS)*, pages 2068–2073, Sendai, Japan, September 2004.

[10] Sun Microsystems. Project Sun SPOT. `http://www.sunspotworld.com`.

[11] S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji. Hardware design of modular robotic system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2210–2217, Takamatsu, Japan, 2000.

[12] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Journal of Autonomous Robots*, 10(1):107–124, 2001.

[13] U.P. Schultz. A unification of inheritance and automatic program specialization. In *Proceedings of the 2004 Conference on Generative Programming and Component-Based Software Engineering (GPCE'04)*, volume 3268 of *Lecture Notes in Computer Science*, pages 244–265, Vancouver, Canada, October 2004.

[14] U.P. Schultz, K. Burgaard, F.G. Christensen, and J.L. Knudsen. Compiling Java for low-end embedded systems. In *Proceedings of the 2003 ACM SIGPLAN conference on Languages, compilers, and tools for embedded systems*, pages 42–50, San Diego, California, USA, 2003.

[15] U.P. Schultz, J.L. Lawall, and C. Consel. Automatic program specialization for Java. *TOPLAS*, 25:452–499, July 2003.

[16] W.-M. Shen, M. Krivokon, H. Chiu, J. Everist, M. Rubenstein, and J. Venkatesh. Multimode locomotion via superbot robots. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 2552–2557, Orlando, FL, 2006.

[17] G.T. Sibley, M.H. Rahimi, and G.S. Sukhatme. Robomote: A tiny mobile robot platform for large-scale sensor networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2002)*, 2002.

[18] K. Støy. How to construct dense objects with self-reconfigurable robots. In *Proceedings of European Robotics Symposium (EUROS)*, pages 27–37, Palermo, Italy, May 2006.

[19] K. Støy, W.-M. Shen, and P. Will. Implementing configuration dependent gaits in a self-reconfigurable robot. In *Proceedings of the 2003 IEEE international conference on robotics and automation (ICRA'03)*, pages 3828–3833, Tai-Pei, Taiwan, September 2003.

[20] M. Yim, D. Duff, and K. Roufas. Polybot: A modular reconfigurable robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 514–520, San Francisco, CA, USA, 2000.