Multirobot Systems and Physical Data Structures — Papers from the AAAI 2011 Spring Symposium (SS-11-07)

Towards Robot Systems Architecture

Keith J. O'Hara
Computer Science Program
Bard College
Annanadale-on-Hudson, NY 12504
kohara@bard.edu

Abstract

Just as special purpose computers and mainframes grew into the general purpose personal computers we use everyday, special purpose industrial robots are evolving into more general purpose personal robots. As robots become more capable and universal, their applications are less well-defined or even unknown at design time. We will have to design robots for classes of tasks rather than specific applications. Having guidelines for how to best organize, interface, and implement robot systems and reason about trade-offs, as we do in computer architecture, will become crucial for success. In this paper we introduce and adapt some useful notions and principles from computer architecture to robot systems architecture. We argue that notions such as locality of reference, balanced architectures, and boundedness (in terms of IO, memory, and CPU) can be leveraged in robot systems design, and in particular, in the design of distributed robot systems.

Keith J. O'Hara

Introduction

Like computer architects, robot designers must address multiple, possibly competing, requirements by balancing trade-offs in terms of processing, memory, communication, and energy to satisfy design objectives. For example, architects might strive to minimize the energy use or cost of a memory subsystem, or maximize the reliability or availability of a storage system. However, unlike computer architects, robot designers have the additional dimensions of sensing and actuation to consider.

Robots live in the real world, sensing and effecting external physical phenomena. This leads to a key consideration that is sometimes lost in traditional computing system design. Where is the robot computing system located in physical space? This consideration amplifies the role distribution plays in robot system design. The allocation and organization of the sensing, computing, actuation, energy, communication resources throughout physical space is at the core of distributed robot systems architecture. The physical distribution of resources let's us exploit the locality of the particular task in a similar manner as the design of a memory subsystem let's us exploit the locality of a computational problem. Robot architects currently lack the design guidelines, organizing principles, rules of thumb, and tools that computer architects rely upon. This paper takes a step in this direction, by analyzing the roles of heterogeneity and distribution in robot systems architecture.

Robot systems are increasingly built as large distributed systems. Robot systems are built in a distributed fashion for both essential and incidental reasons. This leads us to our first way of classifying the role of distribution in robot systems.

- **Distribution is Essential** – The fact the robot system has multiple networked components is paramount in its design. For example, consider a team of unmanned aerial vehicles providing situational awareness. The multiplicity of vehicles is necessary to satisfy the performance and fault tolerance objectives.
- **Distribution is Incidental** – The fact that the robot software system is distributed across multiple machines is only an implementation detail and often an afterthought. Typically, the systems are distributed to allow off-loading of computation for practical performance reasons or to provide a remote user interface during development.

The use of distribution in robot systems is not novel – in fact, just the opposite, it is ubiquitous. Practically all real, fielded, robot systems are built as distributed systems; however, the distributed systems aspects of robot systems are often seen as ancillary and overlooked as "implementation details" rather than being a fundamental problem. Opportunities are lost by considering distribution too late in the development cycle. The first argument this paper puts forward is that all robot systems are inevitably going to be distributed systems, and the earlier this is taken into consideration the better.

In both the essential and incidental scenarios, distribution let's us exploit the locality of the robotic task by placing the hardware resources close to where the robotic computation needs to take place. Moreover, by using a collection of specialized platforms in a coordinated fashion, we can further exploit the locality of a task. Typical approaches that use a collection of identical platforms for parallel speed-up and fault tolerance through redundancy do not exploit this fact.

This leads us to our second way of classifying distributed robot systems, by their composition. Is the robot system homogeneous or heterogeneous? Are all the hardware platforms identical, or near identical, or are the resources allocated in an asymmetric manner?

- **Homogeneous Composition** – The entities in the distributed robot systems are largely identical. Many multi-robot systems rely on multiple homogeneous platforms to perform tasks in a parallel or fault-tolerant fashion. The notion that we can achieve reliable, sophisticated performance from a multitude of unreliable, simple platforms is at work here.
- **Heterogeneous Composition** – The entities in the distributed robot system have differences in their hardware. For example, different robots might have different sensing or computational resources. Many systems that rely on distribution incidentally are highly heterogeneous. For example, a system might perform computational off-loading for practical performance reasons or to provide a remote user interface.

Just as we strive to achieve reliable performance from a multitude of unreliable, simple homogeneous platforms, we can also aim to achieve generality from a multitude of specialized platforms. The second argument this paper offers is that robot systems can be constructed from a collection of specialized robots that are both sufficiently general to solve the task and sufficiently specialized to exploit the task's locality.

Robot Systems Architecture

Just as special purpose computers and mainframes grew into the general purpose personal computers we use everyday, special purpose industrial robots are evolving into more general purpose *personal robots*. Robot systems have been proposed for a wide variety of tasks, from canonical robot tasks such as autonomous navigation to providing a motivating context for education. As robots become more capable and universal, their applications are less well-defined or even unknown at design time. We will have to design robots for classes of tasks rather than specific applications. Having guidelines for how to best organize, interface, and implement robot systems and reason about trade-offs, as we do in computer architecture (Hennessy and Patterson 2003), will become crucial for success.

This paper takes a systems architecture approach to the design of robot systems, and in particular, understanding the use of distributed robot systems to achieve design objectives. The IEEE Standard 1471-2000[1] defines *Systems Architecture* as "the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution." Likewise, we define *Robot Systems Architecture* as "the fundamental organization of a robot system, embodied in its computation, communication, sensing, and actuation resources, their relationships to each other and the environment, and the principles governing its design and evolution."

It's quite natural in robot architecture to decompose a task into loosely coupled subtasks in service of more effective design, development, maintenance, and reuse, but it stops at the level of software. This approach goes beyond "Robot Software Architecture" which typically assumes the hardware necessary for the task is available and the details of the hardware are abstracted away. A robot software architecture (like subsumption (Brooks 1986), RCS (Albus et al. 1992), Aura (Arkin and Balch 1997)) gives the roboticist a set of principles, guidelines, and tools to compose a robot's software system. Robot systems architecture puts the hardware and software on equal footing in terms of emphasis and abstraction. We are concerned with the organization of hardware resources, and to some degree the environment, since the interface to the physical world is such a crucial part of any robot system, and particularly in exploiting the locality of the particular robot task. Modern advances in wireless communication have enabled a new level of composability in terms of hardware, allowing us to repurpose hardware analogously to how we reprogram software.

Task and Architecture

Traditionally, robots have been designed with very a clear task in mind, for example, robot arms assembling automobiles. This is in contrast to personal computers that are designed for a wide variety of applications. Because personal computers can be reprogrammed to perform almost any task, they are designed in a way to accommodate many different applications. Similarly, as robot systems become more general purpose, designers will not have the luxury of a single specific task objective. Instead, robots will be reprogrammed and reconfigured for their particular task.

For some tasks, like robots exploring Mars, it's plausible to design a robot system from start to finish for a specific task or handful of tasks. However, for more commercial systems, it will be more economical to mass produce more general purpose robots that later can be reprogrammed and reconfigured. Therefore, although we might be able to design a robot system from beginning to end, and in fact, maybe design it optimally in some sense, there are few situations were this is economical. Therefore, we need to identify classes of applications we expect the robot system to address and design accordingly. Two concepts, boundedness and locality, from computer systems design are useful for describing tasks from an architectural point of view. Both of these ways of thinking about computational processes abstractly, can be leveraged in the design of robot computer systems.

Although both of these notions are abstract they aren't theoretical, a real, concrete program must be involved. Moreover, the locality of a program, or whether its IO-bound, is not only determined by the computational process, but also by the architecture. So while these concepts are useful for describing tasks generally, they are not independent of the underlying architecture.

[1] Recommended Practice for Architectural Description of Software-Intensive Systems

The Locality Principle

In computer architecture (Hennessy and Patterson 2003), the concept of "locality of reference" is crucial in the design and organization of a computer system's memory, informing things as diverse as the design of software virtual memory systems and web caches, and the hardware organization of cache memories (Denning 2005). The locality of a computational process characterizes its memory access behavior. If a computational process has a high level of spatial locality then memory accesses are local in terms of space, i.e. nearby memory locations are more likely to be accessed in the future. Likewise, if a computational process is temporally local then the same memory locations will be accessed often. Things like web caches exploit the temporal locality of web browsing behavior, i.e. storing web pages that are likely to be accessed again. Pre-fetching or read-ahead buffering are mechanisms to exploit spatial locality. Similar concepts can be used to describe robot processes and in the design of robot computing systems.

We define the concept of "physical locality" to be the environmental access behavior of a robot process. If a robot process has a high level of spatial locality then nearby locations are more likely to be accessed in the future. Likewise, if a robot process is temporally local then the same locations will be accessed often. For instance, if a robot task is temporally local, but not spatially local – we need to monitor distant locations frequently – then a pervasive sensor network is well suited for the problem. Reciprocally, when the robot process is spatially local, but not temporally local – we visit environmental regions in a "sequential" manner, but rarely visit the same point twice – a mobile robot is well suited for the problem. Therefore, the locality of a robot process can help us decide what kind of architecture is more suitable.

Access Mechanism

Consider two classes of robot computing systems: mobile robots and sensor networks. Both types of systems can be used to solve many of the same robot computing problems (e.g. environmental monitoring), but there also exist applications for which each is particularly suited. Both mobile robots and sensor networks provide access to large-scale phenomena – mobile robots via mobility, and sensor networks via pervasiveness. Thus, we term both mobility and pervasiveness as "access mechanisms".

A large class of robot computing tasks that are suitable for both mobile robots and sensor networks fall under the heading of "monitoring". For instance, consider tasks such as forest fire detection, military reconnaissance, security patrolling, and urban mapping. In these tasks, we want to collect sensor data from a large area and distill it down to the information of interest. Any of the 4-D's of robot tasks: *dirty, dangerous, difficult, and dull* may make robotic monitoring useful.

Whether to use, and how to use, mobility or pervasiveness as the access mechanism is one of the most interesting robot architectural trade-offs. We term it the "access tradeoff". One mechanism may be more suited for the task, or some mixture of both. After all, mobile robots can benefit from pervasiveness, likewise, sensor networks from mobility. Rather than just building a fully mobile sensor network or a fully pervasive robot swarm to exploit these benefits, we can take a more nuanced approach.

Boundedness

In addition to describing a computational process by its access pattern, its locality, we can also note what percentage of time, energy, or money the task devotes to computing, accessing memory, sensing, or effecting the environment. The boundedness of a task is the limiting factor. For instance, the performance (measured in time til completion) of a memory-bound task is limited by primarily by memory access times.

The notion of *boundedness* from computer systems design is a useful concept for thinking about robot tasks. Different computational processes might spend a majority of their time processing, doing input or output, or accessing memory; a computational process can be CPU-bound, IO-bound, or memory-bound. If a process is memory-bound then we can identify the memory-subsystem as the bottleneck. If we plan to improve the system, we are wise invest on improving that bottleneck rather than the other areas.

We can apply the same idea to robot systems architecture. A robot process might be bound by its sensor bandwidth, the amount of CPU processing it has, or by the speed of its wheels. A robot that is performing a batch mapping task might be limited by its laser scanning bandwidth. This notion also helps us identify, and correct, bottlenecks in robot system architecture. This idea enables us to group different tasks according to the how they are bounded.

Case Studies

We have designed and implemented three robot systems that exploit heterogeneity and distribution in novel ways.

Gnats The Gnats robot system (O'Hara, Walker, and Balch 2008) used heterogeneity and distribution to exploit the spatial locality of the path planning, coordination and foraging task. A technique was developed for performing path planning, coverage, and foraging using a system of heterogeneous robots. The locality of the path planning and coverage tasks was exploited by using a minimal, immobile sensor network, reducing the mobility and coordination resources needed by the mobile robots. Moreover, the resources necessary for the mobile robots (e.g. specific sensors or effectors) application are contained on the mobile robots lowering costs and providing flexibility. Empirical evidence showed the utility of the technique in tasks such as navigation and foraging. Experiments included simulation and some of the largest robot experiments in this domain.

AutoPower The AutoPower system (O'Hara et al. 2006) used heterogeneity and distribution to effectively manage energy in robot teams. A system-level model for characterizing the energy behavior of distributed robot software systems was developed and applied to a multi-robot search and rescue mission. Distributed computing mechanisms were leveraged not only to speed-up computation, but to prolong

the lifetime of the team. Experimental results using emulation of real robot software was used and the lifetime of the system was extended by 57%.

IPRE We have developed a distributed robot system for computing education. (Balch et al. 2008) The distributed robot system uses a standard laptop for computation as well as providing some extra input and output modalities, such as a joystick, text-to-speech, and on-screen graphics. The laptop commands a mobile robot over a bluetooth wireless link. The mobile robot (the scribbler) is augmented with a circuit board (the Fluke) that provides more on-board computation, communication, and sensing. This particular allocation of resources was decided in order to satisfy our pedagogical goals, thus we looked to maximize robustness, ease-of-use and expressiveness, while minimizing cost.

Balanced Architectures

Although Gene Amdahl is typically known for his law (Amdahl 1967) of diminishing returns concerning parallel computation, his rule of thumb (also known as his other law) (Gray and Shenoy 2000) is concerned with the architecture of computer systems. It says that for a computer to be useful, for every instruction per second of computation, the computer must also have one byte of memory, and one bit per second of IO. His rule of thumb provides a scaling relationship between the constituent parts of a computer system, and also the notion of a balanced computer system. Similar questions can be asked about robot computing systems. For instance, for a household mobile robot, what is a useful proportion of computing to memory, to sensing, to actuation? Although, a full answer to this question is beyond the scope of this paper, we do point out that a "balanced" robot architecture does not have to mean equally balanced, but rather just some desired proportioning of resources.

Amdahl's rule of thumb gives us a new way to compare and reason about different robot systems: how are the systems' resources allocated? Is the robot equally-balanced, with its resources equally allocated across sensing, computing, and effecting or is the robot rather asymmetric, for example, with sensing dominating? Some tasks, might not require symmetry, in fact they might benefit from specialized, asymmetric architectures. However, one could argue that for general purpose robots intended to provide a wide variety of applications, much like general purpose computers, a balanced architecture is beneficial. Generally, striking a balance in terms of architecture can help alleviate architectural bottlenecks.

We can arrive at a balanced robot system in a variety of ways. One possibility is for an individual robot to be balanced from its initial design. Another possibility is to achieve balance through composition. By connecting a collection of specialized platforms, a more general system results. The idea of using distribution to achieve a *balanced* system architecture is a powerful idea. For example, in designing a robot for education we arrived at a balanced, general-purpose platform from a collection of specialized platforms. This particular platform which is composed of a mobile base (the scribbler), a bluetooth sensor board (the fluke), and a laptop. The laptop is very computation centric, the scribbler mostly focused on mobility (and thus actuation), and the fluke on sensing. The ternary diagram in Figure 1. visualizes how the individual specialized platforms allocated their resources in terms of cost to the sensing, computing and effecting subsystems.

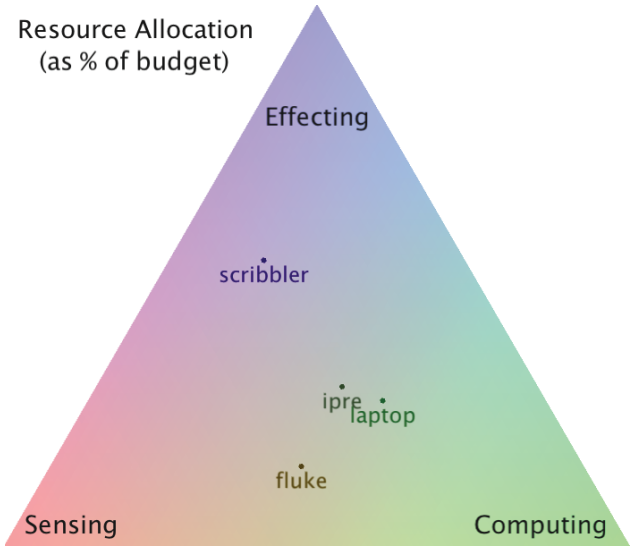

Figure 1: The resource allocation of the individual specialized platforms of the IPRE robot system.

Robot Design Constellations

The use of ternary diagrams to reflect on an architecture's balance and to compare different robot designs is useful in understanding robot systems; however, it only allows us to compare along three dimensions. In Figure 1, we chose to compare the robot platforms in terms of the budget allocated toward computing, sensing, and effecting. To move beyond only three dimensions, we have adopted the star (or radar) chart (J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey 1983) for describing robot system architectures. As noted by Chambers et al. star charts are particularly good for comparisons, the authors remark, "one of the main purposes of such multi-code schemes (like star charts) is to obtain a symbol with a distinctive shape for each observation, so that a viewer can look for pairs or groups of symbols with similar shapes, or individual observations that are very different from the rest."

We use the term "robot design constellation" to describe the use of star charts to visualize a collection of connected robot platforms. In Figure 2 we visualize the IPRE educational robot system as a design constellation. Again, this diagram clearly illustrates the platform's "balance", and how each of the constituent platforms differ. Figure 3 depicts the Gnats and AutoPower systems similarly

Design constellations not only support comparison between platforms, but also allow architects to interactively explore design trade-offs. In describing star charts, Chambers et al. suggest that "It can be helpful to cut the symbols

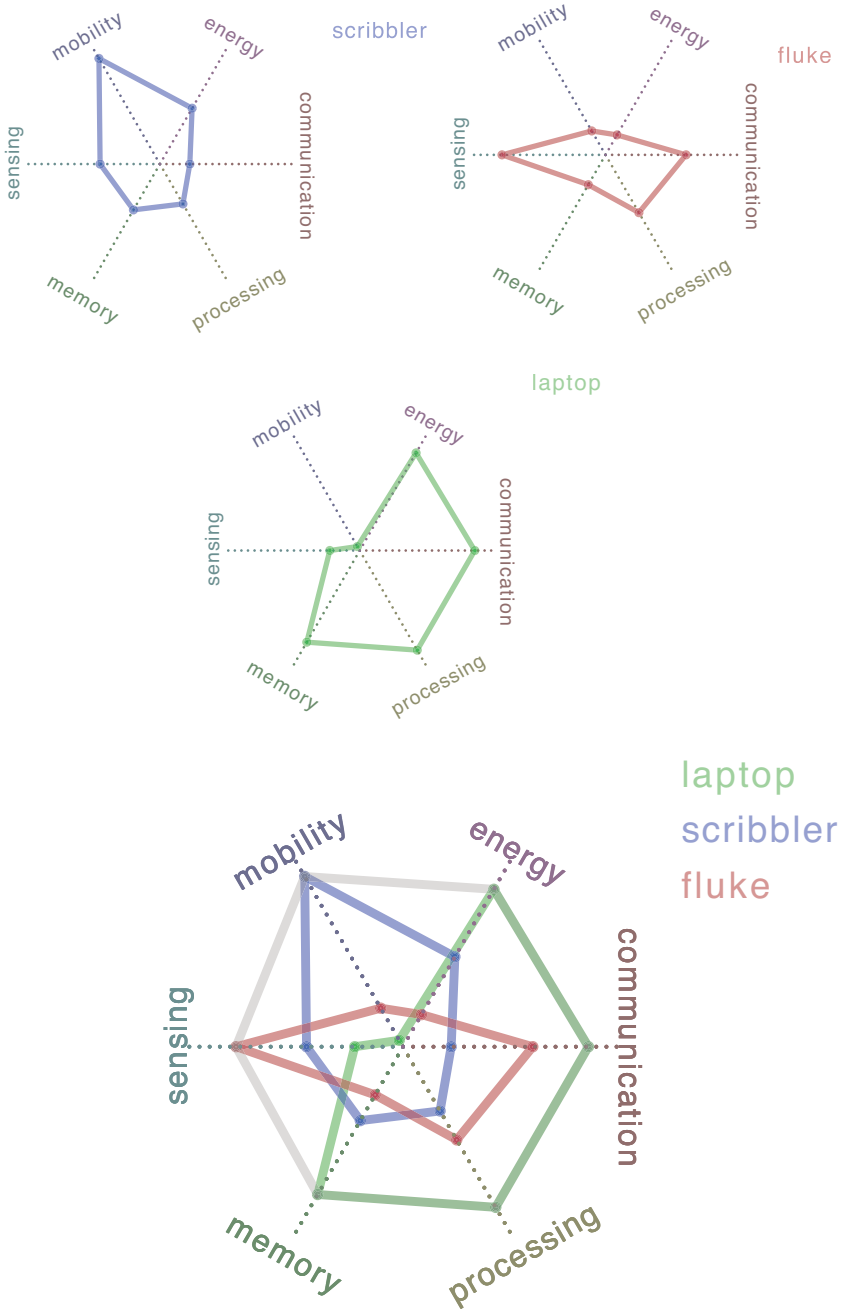

Figure 2: Design constellation of the IPRE robot system for education.

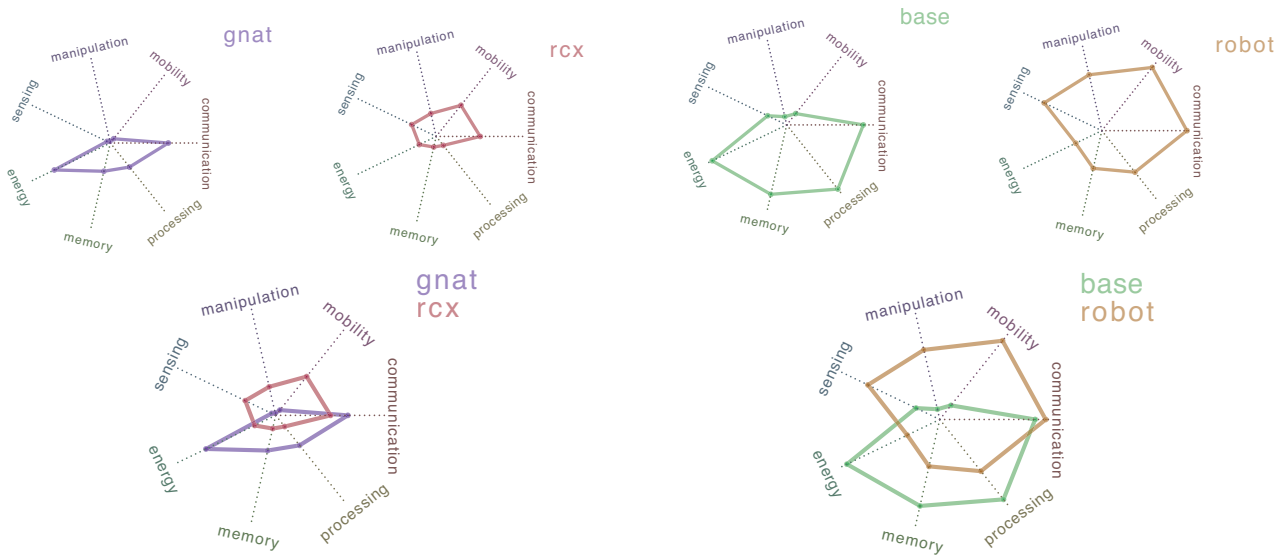

Figure 3: Design constellations of the Gnats and Autopower systems.

apart (along with their labels) onto separate slips of paper, and to slide them around on the table grouping them in interesting ways." We created a program to construct these star charts and *slide them around* as the authors suggest to support comparisons. A Processing sketch was implemented to allow architects to interact with the constellation in order to explore trade-offs. The user can add (or subtract) resources to (or from) a particular dimension and see how the overall platform is impacted.

Conclusion

This paper argued that a perspective based on robot systems architecture, enables and elucidates novel trade-offs, not obvious in traditional approaches to the design and implementation of robot systems. Distributed robot systems architecture combines methods, techniques, principles, and problems from computer architecture, advanced software systems, and robotics. As evidence for the utility of this perspective, we highlighted three distributed robot architectures for tasks as varied as energy-aware search and rescue, computing education, and multi-robot foraging. Each of these systems uses a heterogeneous collection of robot platforms in a novel manner and helps us elicit important properties of distributed robot system architectures in general. By architecture we are not only including the organization and guiding principles of the software, but also the interface, organization, and implementation of the underlying hardware. In each case study, the software, as well as the computation, communication, sensing, and actuation resources are organized to satisfy design objectives in terms of performance, energy, or cost. We also introduced robot design constellations as a way for robot architects to interactively explore design trade-offs and compare platforms.

References

Albus, J. S.; Pape, C. L.; Robinson, I. N.; cker Chiueh, T.; McAulay, A. D.; Pao, Y.-H.; and Takefuji, Y. 1992. RCS: A reference model architecture for intelligent control. *Computer* 25(5):56–79.

Amdahl, G. M. 1967. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference*, 483–485.

Arkin, R., and Balch, T. 1997. Aura: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence* 9:175–188.

Balch, T.; Summet, J.; Blank, D.; Kumar, D.; Guzdial, M.; O'Hara, K.; Walker, D.; Sweat, M.; Gupta, C.; Tansley, S.; Jackson, J.; Gupta, M.; Muhammad, M.; Prashad, S.; Eilbert, N.; and Gavin, A. 2008. Designing personal robots for education: Hardware, software, and curriculum. *IEEE Pervasive Computing.* 7(2):5–9.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* 2:14–23.

Denning, P. J. 2005. The locality principle. *Commun. ACM* 48(7):19–24.

Gray, J., and Shenoy, P. 2000. Rules of thumb in data engineering. In *ICDE '00: Proceedings of the 16th International Conference on Data Engineering*.

Hennessy, J. L., and Patterson, D. A. 2003. *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. 1983. *Graphical Methods for Data Analysis*. New York: Chapman and Hall.

O'Hara, K. J.; Nathuji, R.; Raj, H.; Schwan, K.; and Balch, T. 2006. Autopower: Toward energy-aware software systems for distributed mobile robots. In *IEEE International Conference on Robotics and Automation*.

O'Hara, K.; Walker, D.; and Balch, T. 2008. Physical path planning using a pervasive embedded network. *IEEE Transactions on Robotics,* 24(3):741–746.