An Architecture for Programming and Managing Sensor and Actuator Networks in Enterprise Environment

Han Chen[1], Young-ri Choi[2], and Paul B. Chou[1]

[1]IBM Thomas J. Watson Research Center, 19 Skyline Dr, Hawthorne, NY 10532, U.S.A.
{chenhan,pchou}@us.ibm.com

[2]Department of Computer Sciences, The University of Texas at Austin, U.S.A.
yrchoi@cs.utexas.edu

***Abstract*— In this paper, we argue that there is the need for a comprehensive approach to the programming and managing sensor and actuator applications in enterprises. We then propose the design of such an architecture. It includes a network infrastructure model, an application programming model, a distributed runtime, and tools for complete lifecycle management of the network and the applications. Based on the current proof-of-concept implementation we also discuss a number of technical challenges.**

I. Introduction

Miniature sensors are being increasingly widely deployed, as their cost and power consumption continue to fall. Today, sensors network technology is used in a multitude of applications, such as industrial automation, environment monitoring, asset management, intrusion detection, smart living space and healthcare, etc [10], [14], [21], [18], [4], [5], [19].

Increasingly, sensor networks are augmented with additional actuators to further enhance their functionality. Consider a smart office building as an example. In such a building, a network of tiny sensors is deployed to measure ambient light, temperature, humidity, and human presence, etc. In addition, computerized actuators such as light switches, dimmers, HVAC blowers and dampers are also integrated into the network. The companys goal is to provide customizable comfort zones for individual office users while creating maximum energy conservation. To accomplish this, the network is connected to the back-end enterprise information system where the user preferences and company policy data are stored.

To fully realize the value of the sensor and actuator network, the gap between technology and business goals needs to be bridged. Unlike a pure data gathering sensor network, programming such a multimodal sensor and actuator network entails a lot more than a multi-hop routing algorithm to send data from sensors to the gateway. Furthermore, managing such a network is an equally important task.

In this paper, we propose an architecture that covers all the major aspects in programming and managing sensor and actuator networks in an enterprise environment. The rest of the paper is organized as follows. Section II articulates the design principles and describes the proposed architecture in detail. Section III reports the status of a prototype implementation and discusses the technical challenges. Some related works are discuess in Section IV. Finally, Section V summarizes this paper.

II. Programming and Managing Sensor and Actuator Networks

While prior works on sensor network programming model provide many important low-level abstractions, e.g., routing, discovery, naming, etc, integrating sensors and actuators into entreprise environment has its own unique characteristics and presents additional challenges.

- Sensors do not have to rely solely on ad-hoc communication—a portion of the network can take advantage of the existing wired or wireless networks for connectivity.
- Sensor networks are more likely to be deployed in a planned way; and they will gradually evolve as business needs change.
- Applications on the sensor networks are likely to change, independent of the networks, over time as well.
- Sensor networks need to work with existing computing systems.
- The labor cost of deploying and managing the sensor networks must be low enough to realize the benefit.

A. Design Principles

Based the above observations, we postulate the following design principles, which will later guide the design of the architecture.

1) Virtualization of infrastructure: The goal of virtualization is to separate application logic from topology. The word virtualization is fairly overloaded. Here it means several things. First, the entire network of devices should be viewed and managed as a whole, i.e., a single management domain. This allows application logic to be agnostic of the execution location. Second, the devices themselves should be virtualized. For example, the building manager can create an application that turns off lights in the corridor after midnight. A virtual ”corridor lights” device is used in the application, and it controls all the physical corridor lights. This enables application logic to be expressed in invariant, abstract terms against an evolving topology. Finally, an application container should provide a shielded execution environment for individual applications. This will minimize unforeseen and unwanted behaviors caused by interaction among applications. It can also be the mechanism to implement controlled access rights to devices in a typical enterprise environment.

2) Separation of Concerns: As with any other large-scale enterprise software systems, a sensor and actuator network can grow to be very complex over time. The task of development and administration for such a network can become intractable without a well architected plan. A common practice in software engineering is to separate the concerns in a complex system by defining clear roles and responsibilities. Although this is a less articulated point in current sensor network systems, we believe it is important to the ultimate success of sensor and actuator applications in an enterprise setting.

3) Enabling Enterprise Integration: Customers employ sensor and actuator networks to derive business value by improving existing business processes or enabling entirely new processes. In many cases, it requires integrating the sensor network with existing enterprise networks and business systems. For example, for retailers to reap the benefit of RFID technology fully they need to integrate the readers and other sensors with various systems, such as, ERP, SCM,

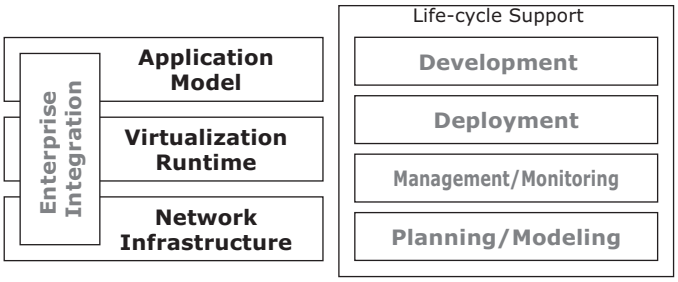

Fig. 1. Architecture for Programming and Managing Sensor and Actuator Applications

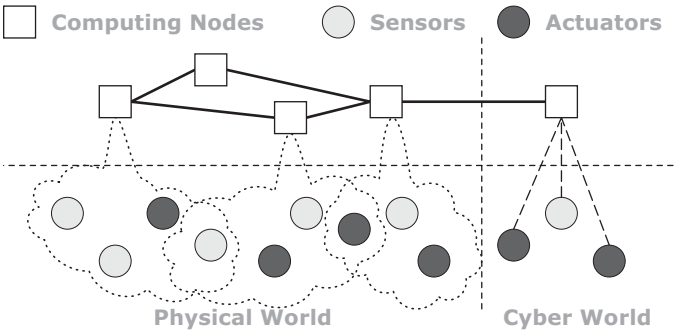

Fig. 2. Network Infrastructure Model

WMS, etc. The architecture should provide the necessary platform and adapters for applications to integrate wiht legacy systems easily.

4) Complete Life-cycle Support: Last by not the least, the architecture should include both tooling and runtime to support the end-to-end life cycle management of both the network and the applications, namely, 1) the modeling of the infrastructure, 2) the creation of the applications, 3) the deployment of the applications, and 4) the monitoring and management of both the network and the applications.

B. Architecture

Based on the aforementioned design principles, the architecture should clearly separate the application logic from the network infrastructure, while providing tools for different roles involved in the life cycle of the system. Figure 1 gives a high level outline of the main components in this architecture, each of which is discussed in detail.

1) Network topology model: We consider a network to include sensors and actuators, both physical and virtual (see discussion in Section II-B.4), and a number of computing nodes capable of performing non-trivial application logic. The computing nodes act as a bridge to link the physical devices to the back-end systems. Nowadays, with embedded processor on a sensor or actuator device being powerful enough, they can be qualified as computing nodes as well. Here we distinguish a computing node from a sensor based on the following characteristics. A computing node has enough resource to run a standard, modern OS, such as J2ME. A computing node could have reliable communication links such as Ethernet or WiFi, whereas sensors relies on ad-hoc peer-to-peer links, such as Bluetooth or Zigbee. A computing node can be addressed and managed easily.

Due to the differences outlined above, we organize the network into two distinct tiers, as shown in Figure 1. The lower tier consists of the devices and the upper tier consists of the computing nodes. We posit the following two connectivity requirements. First, from any device there is a route to reach at least one of the computing nodes (gateway), for example, a multi-hop ad-hoc wireless links or a directly serial connection. Second, the computing nodes are fully connected, via either point-to-point links or existing network infrastructure.

We believe that **a substantial amount of application logic should reside in the sensor and actuator network**, instead of being pushed all the way to the back-end servers. First, data cleansing and fusion logic can be deployed in the network to reduce network traffic; this is especially helpful when the nodes are battery powered and communicate via wireless links. Second, in sensor and actuator networks, local control logics can be performed in the network without involving the back-end server, which reduces latency and improved response time. Third, in some environments, network connections among nodes may be intermittent, dispersing application logic in the network allows the system to continue functioning in the event of network disconnection.

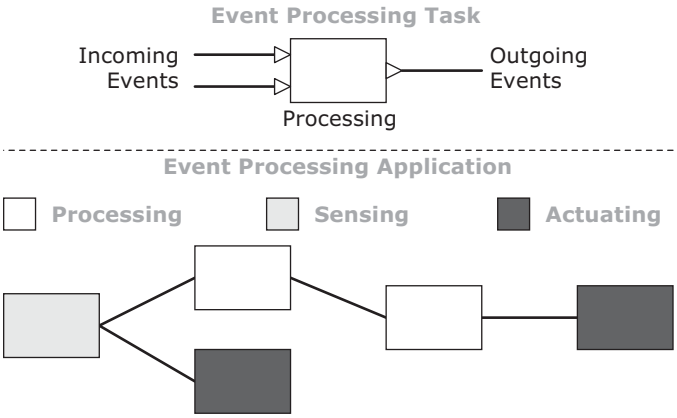

Fig. 3. Sensor and Actuator Application Programming Model

Finally, this is feasible due to the increasingly more powerful and more energy efficient computing platforms.

Although application logic can be executed on an ad-hoc, distributed sensor network, the potential benefit does not justify the cost of reduced reliability and manageability, especially in an enterprise enrivonment, where these factors have a great impact on the total cost of ownership of such a technology. Therefore, we believe that, except for very simple tasks, such as filtering or data conversion, **application logic should be excecuted on the computing nodes**. This view is in general agreement with that proposed in [7].

2) Application programming model: Unlike traditional procedural programming languages, where an application executes a series of commands until it reaches the end of the flow, most sensor and actuator applications are event-driven. Such an application is typically modeled as a network of tasks, as shown in Figure 3. Tasks can be broadly classified into three types, sensing task, actuating task, and processing task. A sensing task generates output events. For example, a temperature sensor is modeled as a sensing task, which generates temperature reading at some rate. An actuating task receives input events and trigger certain actions in the physical (or cyber) world. A processing task reacts to incoming events; after processing these events, it can generate new output events.

The programming model should include the following

- **Task programming interface** This includes the interfaces and supporting base classes used by the developer to create new tasks.
- **Composition syntax** This defines how the primitive tasks are wired together to form an application. It can be either an

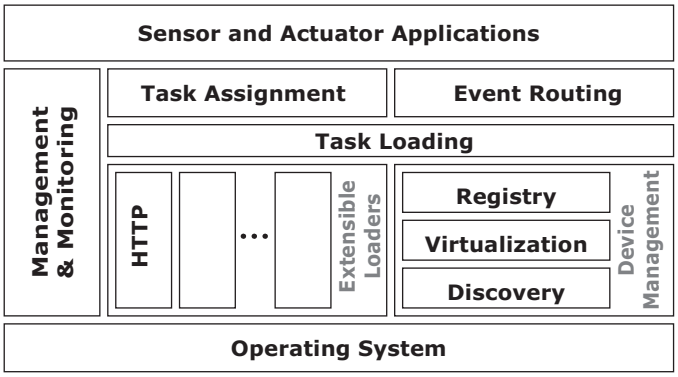

Fig. 4. Runtime Software Stack on Computing Nodes

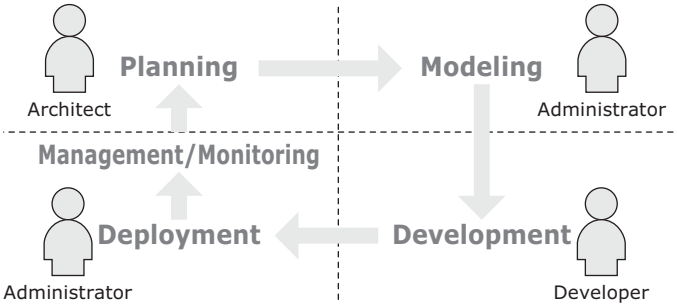

Fig. 5. Life-cycle of a Sensor and Actuator Network

interpretation engine that orchestrates the event flows specified in a flow language or a set of base classes used by developers or code generators (see Section II-B.5) to create application code.
- **Composition semantics** This portion is an integral part of the composition engine or base class. The reason we separate semantics from the syntax is that the wiring syntax is generic while the semantics of the composition can vary from implementation to implementation. For example, the event passing among tasks can be completely asynchronous or synchronized according to some global execution order.

3) Runtime support: To separate the concerns of administrator and developer, we propose an application model that is logically independent of the network topology. A runtime layer on the computing nodes is needed to support the execution of the application logic. Figure 4 shows the software stack of a typical runtime. The following modules are essential to the runtime.

- **Device discovery/registry** The module discovers and maintains a list of nearby sensors and actuators that are reachable. The communication link can be wireless or wired.
- **Task loading** This module is responsible for dynamically loading application tasks from external repositories. This is key to the ability of application management.
- **Event routing** This module allows an application task to send messages to or receive messages from other tasks or devices. This is essential in maintaining the logical independence of application logic from network topology, because application logic does not need to know where it is executed.
- **Task assignment** This module maps the tasks in an application to the computing nodes according to certain criteria. With the message routing support, task assignment does not affect the functionality of the applications deployed on the network. However, it can be used to improve the quality of the applications, for example, to balance the energy consumption of nodes, to improve the responsiveness of the application, etc.
- **Management and monitoring** This module is responsible for managing the applications deployed on the computing nodes. It also actively monitors the applications and the computing node itself and generates monitoring events, which can be used for the external management and monitoring applications.

4) Integration support: As we have argued before, integration with existing enterprise systems is an important aspect of the deployment of sensor technology. To make the process easy, integration support must be built into the system, instead of being applied as an afterthought. In the proposed architecture, integration is modeled as tasks in applications, e.g., writing to a queue, making a web service invocation, reading from a database table, etc. We propose the following three levels of support.

- **Modeling** We view the integration task (queues, DBs) as virtual sensors and actuators in the cyberspace. Thus, they will be modeled in the network topology and treated just as other physical devices.
- **Integration platform** Typically the integration resources are only accessible from a server (for example, J2EE) node. In order to support seamless application deployment, we need to include the server nodes in the sensor and actuator network infrastructure as the integration point, see Figure 2. To accomplish this, the runtime software stack need to work on these integration points.
- **Integration tasks** We provide a set of pre-built integration tasks so that solution developers can create solutions quickly. Developers can create additional new integration adapters if necessary, by following the application programming model.

5) Management and tooling: A good development API is just a portion of a successful enterprise architecture. To complete the picture, it needs to cover the entire life cycle of the network and the applications and provide associated tooling support. We break down the life cycle into several distinct phases and discuss the roles involved in each phase and the tools needed.

- **Topology planning** In this phase the planner decides what types of and how many devices and computing nodes are needed in order to fulfill a given business objective. The tools needed for this phase are a topology designer and a performance simulator.
- **Topology modeling** In this phase the modeler captures the topology of the network into some persistent storage, for example, a directory server. Whenever the topology changes the model should be updated. The tool for this phase is a topology modeler.
- **Application development** In this phase the developer creates either primitive processing tasks or composition sensor and actuator applications. The tools needed include the target language development tool (C/C++, Java, .NET, etc), a graphical composition editor, a code generator for the composite application.
- **Application deployment** In this phase the deployer uses the application deployment tool to install applications. The applications could come from local or remote file systems or from a code repository specified by a URL.
- **Application management** In this phase the administrator uses an admin console to manage all deployed applications on the network. Management tasks include starting/stopping an application, setting the application's virtual event sources and sinks, etc.
- **Topology and application monitoring** In this phase the admin-

istrator uses the admin console to monitor the operational status of the entire network and applications. Performance related data such as processor load, energy consumption, latency, etc, may also be collected, which may trigger appropriate runtime response, for instance, dynamic task re-assignment, or necessitates another iteration of topology planning when the load exceeds the capacity of the system.

III. Prototype Status and Technical Challenges

We are creating a prototype system according to the architecture. Here we report the status of the work in progress and discuss some of the main technical challenges that are of particular research interest.

A. Proof-of-Concept Prototype

The proof-of-concept prototype system is based on Java programming language and Eclipse tooling environment. The current system consists of the following components:

- Java classes for creating primitive event processing tasks.
- Java classes for creating composite event processing flows.
- Syntax and sementics for a graphical task composition language.
- Eclipse-based graphical composition tool for creating composite event processing applications.
- Eclipse-based code generation facility that generates Java code from the flow language.
- Eclipse-based task/application publication and reuse mechanism.
- Java (J2ME/J2SE) based distributed runtime that executes the event processing application on a set of computing node. It provides event routing and location transparency, but currently does not support dynamical task assignment to runtime nodes.
- Eclipse-based admin console that manages the entire sensor and actuator network. It allows the administrator to perform tasks including topology modeling, application installation/uninstallation, starting/stopping applications, binding event sources/sinks in applications to devices in the topology, statically assigning tasks in applications to computing nodes, etc.

B. Examples Use Cases

We have implemented or considered several applications of the architecture in different industries.

1) RFID for Supply Chain: In supply chain optimization, we use RFID reader, motion sensor, light stacks, etc to automate and augment the existing operation processes. We have implemented use cases such as, dock door receiving and print-verify-ship.

2) Smart building: In smart office building, we use various environmental sensors to control the lighting, HVAC, and other office devices to accommodate diverse user preferences while enforcing certain building-wide policies, such as security and energy conservation. We are implementing use cases to demonstrate the architecture.

3) Asset management: There is increasing need from various industries to actively monitor and manage their assets, both equipment and human, in order to improve safety, security, or operational efficiency. We are in the process of designing the use cases for these applications.

C. Technical Challenges and Future Work

Our current prototype demonstrates the concept of an all encompassing architecture for integrating sensors and actuators into enterprise. There are still some challenging issues to be addressed in order to realize its value to the fullest extent. We consider them as our future work in this area.

1) Dynamic Task Assignment: The architecture presents the image of a single, virtualized, distributed runtime across the computing node. When the event processing applications are executed on such a runtime, the runtime needs to assign the tasks to the underlying runtimes. The mapping from tasks to runtimes falls into the general task assignment problem. Although task assignment does not affect the functionality of the event processing application, that is, given the correct runtime event routing implementation, any task assignment will allow the applications to function correctly, it does affect the quality (or non-functional) aspects of the applications, for example, end-to-end latency, event throughput, energy consumption, etc.

Generally speaking, there is an incentive to maximize the performance of a given network topology to reduce the need for unnecessary infrastructure upgrades and thus achieve better return of investment. This motivates us to consider a task assignment algorithm that can dynamically react to factors such as current system load, network throughput, energy reserve, incoming event rate, etc and adjust the assignment of tasks to the runtimes according to an objective function to achieve an optimization goal.

To create such an algorithm, we need to consider the following design questions:

- Whether it should be performed by a centralized scheduler, by all runtimes in a completely peer-to-peer fashion, or some combination of both.
- What is the right adjustment granularity, spatially and temporally.
- How to balance the overhead in the dynamic assignment process with any potential saving it can achieve.
- What design patterns should be used in programming the processing tasks so as to minimize the cost of migrating and/or replicating them across the runtimes.

2) Topology management: In a full deployment of sensors and actuators across an enterprise, there is likely to be a large amount of devices in the topology. This poses a question: how should these devices to visualized, bound to applications, managed, and monitored?

As we argued in Section II-A, device virtualization is partially the answer. However, this is not trivial to implement right. We need to consider virtualization at many dimensions, as shown in the following list.

- **By type:** all lights, all fluorescent lights.
- **By location:** lights in office 102, RFID readers in the receiving department.
- **By function:** warning device (siren and/or red light)
- **By ownership/authorization:** all Mr. Smith's devices
- **By composition:** receiving portal (made of an RFID reader, two buttons, a lightstack, two motion sensors)

This list is by no means exhaustive—when new applications are created for new industries, new ways of virtualizing the devices will crop up. Therefore, we believe that there is the need for an extensive device virtualization framework along with the support tools.

3) Performance modeling: An successful IT architecture should allow it to grow according to the business demand—under-capacity results in poor performance and loss of productivity while over-capacity can cause unnecessary investment and ultimately lower ROI. Accurate performance modeling is key to finding the sweetspot.

We believe that a comprehensive performance modeling tool can accomplish this goal. It should allow the administrator to perform the following tasks

- **Simulation**, that is, to estimate the performance metrics of the system given its topology and applications installed.
- **Validation**, that is, to determine whether the system can achieve the desired performance.
- **Planning**, that is, given performance goals, to recommend a system that can achieve the gaols.

The simulation capability is the core of the performance modeling tool. Simulation can be performed with characteristics collected from instrumentation and profiling or created with probablistic models.

IV. Related Work

Techniques for in-network aggregation in sensor networks have been studied in [12], [8], [9], where raw sensing data is aggregated and processed and so only aggregated results are forwarded to a base station. More general models for sensor network applications have been proposed [2], [15], [11], [12]. In general, the purpose of these models is to hide application programmers from low-level details such as routing, group management, and resource management in a sensor network. These models assume that an application is processed by sensors within the sensor network.

A tiered architecture for sensor networks, called Tenet, were proposed in [7]. The authors of Tenet argued that a tiered architecture consisted of sensors and master nodes has advantages to develop and deploy new applications for sensor networks.

Distributed stream processing systems have been studied to process stream-based continuous queries efficiently [6], [1]. In these systems, an application is expressed as queries.

Many task mapping algorithms have been studied for distributed computing systems. In static mapping algorithms where the mapping is computed at compile [3], it is assumed that the execution and communication costs of tasks are priori known. Dynamic mapping algorithms [13] recompute the mapping of tasks that have not been executed yet at runtime, using the execution and communication cost of tasks measured at runtime. These algorithms target traditional procedural applications. They need to be augmented to support the dynamic nature of event-driven applications.

There have been earlier efforts for task mapping within a single cluster in sensor networks [20], [17] or in ad hoc networks [16]. These algorithms focus on energy constraints of sensors or computing nodes in ad hoc networks, but assume a fixed single application for mapping without considering dynamic applications.

V. Conclusion

Sensors and actuators are increasingly being deployed in enterprise environments to achieve various business objectives. We believe that a comprehensive approach is needed for the programming and managing the sensor and actuator network and propose an architecture for it. We have implemented a proof-of-concept prototype and are actively working to address additional technial challenges.

References

[1] D. J. Abadi and et al. The Design of the Borealis Stream Processing Engine. In *Proceedings of the 2nd Biennial Conference on Innovative Database Systems (CIDR'05)*, 2003.

[2] T. Abdelzaher and et al. EnvioTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, Tokyo, Japan, March 2004.

[3] A. H. Alhusaini, V. K. Prasanna, and C. S. Raghavendra. A unified resource scheduling framework for heterogeneous computing environments. In *HCW '99: Proceedings of the Eighth Heterogeneous Computing Workshop*, page 156, Washington, DC, USA, 1999. IEEE Computer Society.

[4] A. Arora and et al. A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. *Computer Networks (Elsevier), Special Issue on Military Communications Systems and Technologies*, 46(5):605–634, December 2004.

[5] A. Arora and et al. ExScal: Elements of an Extreme Scale Wireless Sensor Network. In *Proceedings of 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2005)*, 2005.

[6] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable Distributed Stream Processing. In *Proceedings of the First Biennial Conference on Innovative Database Systems (CIDR'03)*, 2003.

[7] R. Govindan and et al. Tenet: An architecture for tiered embedded networks. *CENS Technical Report 56*, 2005.

[8] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the Symposium on Operating Systems Principles*, pages 146–159, Chateau Lake Louise, Banff, Alberta, Canada, October 2001. ACM.

[9] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 457, Washington, DC, USA, 2002. IEEE Computer Society.

[10] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 64–75, New York, NY, USA, 2005. ACM Press.

[11] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao. State-Centric Programming for Sensor-Actuator Network Systems. *Pervasive Computing*, pages 50–62, October-December 2003.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(Winter):131–146, 2002.

[13] M. Maheswaran and H. J. Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *HCW '98: Proceedings of the Seventh Heterogeneous Computing Workshop*, page 57, Washington, DC, USA, 1998. IEEE Computer Society.

[14] A. Mainwaring, J. Polastre, R. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, September 2002.

[15] R. Newton and M. Welsh. Region Streams: Functional Macroprogramming for Sensor Networks. In *International Workshop on Data Management for Sensor Networks, DMSN (VLDB 2004)*, 2004.

[16] S. Shivle and et al. Static Mapping of Subtasks in a Heterogeneous Ad Hoc Grid Environment. In *Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, 2004.

[17] Y. Tian, E. Ekici, and F. Ozguner. Energy-Constrained Task Mapping and Scheduling in Wireless Sensor Networks. In *Proceedings of the First IEEE International Workshop on Resource Provisioning and Management in Sensor Networks (RPMSN05)*, 2005.

[18] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A Macroscope in the Redwoods. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys 2005)*, November 2005.

[19] G. Virone, A. Wood, L. Selavo, Q. Cao, L. Fang, T. D. an, Z. He, and J. Stankovic. An Assisted Living Oriented Information System Based on a Residentia l Wireless Sensor Network. In *Proceedings of Transdisciplinary Conference on Distributed Diagno sis and Home Healthcare (D2H2)*, 2006.

[20] Y. Yu and V. K. Prasanna. Energy-balanced task allocation for collaborative processing in networked embedded systems. In *LCTES '03: Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, pages 265–274, New York, NY, USA, 2003. ACM Press.

[21] P. Zhang, C. Sadler, S. Lyon, and M. Martonosi. Hardware Design Experiences in ZebraNet. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, November 2004.