Towards A Flexible Global Sensing Infrastructure

Chien-Liang Fok, Gruia-Catalin Roman and Chenyang Lu
Washington University in St. Louis
{liang, roman, lu}@wustl.edu

Abstract

Wireless sensor networks can potentially become larger, more prevalent, and interconnected via the Internet, forming a global sensing infrastructure that serves many users. In order to realize this and maximize its utility, it is necessary to develop a software architecture that enables new components to be integrated, multiple applications to share the same sensing substrate, and the integration of sensor and IP networks. We propose one such architecture that encodes applications as malleable, platform-independent, high-level mobile scripts. Multiple users are supported by allowing these scripts to execute concurrently, while flexibility is achieved by having them invoke platform-specific services. Services provide reusable functional capabilities that may vary across platforms and environmental conditions. They are platform-specific and may be discovered and redeployed on-demand at runtime. Service provision allows applications to exploit whatever computational capabilities are available, and new services to be added in response to changing application needs, resource availability, or environmental conditions. Our approach allows applications to function in diverse settings by employing dynamic rebinding of mobile scripts to different services as they execute over extended intervals across different types of networks.

1 Introduction

Wireless sensor networks (WSNs) consist of numerous miniature wireless sensors embedded in the environment. While they have been successfully used in the past, their full potential has not been exploited for a variety of reasons. One major reason is the lack of a flexible software architecture that enables the integration of new technology, concurrent application execution, and application flexibility. Existing software for WSNs tends to be vertically integrated consisting of custom components that are not reusable, making it hard for new applications to be developed. As WSNs become larger, more heterogeneous, and integrated with the Internet, they will form a complex and dynamic *global sensing infrastructure* (GSI) that renders vertical integration untenable. The GSI will demand new software architectures that can serve multiple transient users simultaneously, and unifies a heterogeneous and dynamic mix of networks with a wide range of capabilities. This paper presents a new software architecture that provides this using mobile scripts that are *reactive*, *service-centered*, and *platform independent*. It provides an integrated programming environment that simplifies application development while still enabling the network to evolve and applications to adapt.

The field of WSNs has quickly grown and matured over the past few years. Much progress has been made addressing problems associated with meager resources (e.g., limited battery power, unreliable wireless networking, and node failures), and the nature of WSN applications (e.g., software autonomy and collaborative sensing). However, these individual solutions often cannot be easily integrated, hindering progress. Furthermore, new concerns are emerging due to the changing nature of these networks from being small-scale, homogeneous, and application-specific WSNs to being large-scale, heterogeneous, and general-purpose GSIs. The heterogeneity of the GSI requires a standard communication interface. This is partially provided by the Sensor Network Architecture (SNA) [5]. SNA offers a common link-level communication and discovery protocol for WSNs, but does not address interface issues between sensor and IP networks. We propose to build on SNA by providing a cross-platform architecture that covers both sensor and IP networks. The general-purpose and shared nature of future GSIs requires networks to be re-programmable and able to support multiple applications concurrently. The highly diverse and dynamic set of devices within a GSI requires applications to adapt to their execution context. We propose to develop a novel *reactive service-centered programming* model for such an environment.

As more WSNs are deployed for longer intervals, our environment will become saturated with multiple overlapping WSNs forming a *heterogeneous* and *continuously evolving* GSI. This heterogeneity is due to continuous improvements in embedded devices and the current trend towards integrating micro-servers within the network. Micro-servers help overcome the resource constraints of typical WSN nodes and connect to the Internet allowing transient Internet users to share the GSI. The sensors, micro-servers, and Internet collectively form the GSI.

Two example applications supported by a GSI are global supply chain monitoring and disaster scenario coordination. In global supply chain monitoring, wireless sensors are attached to each product that is loaded into cargo containers with micro-servers that interface between the WSN and Internet. The resulting system can serve many users simultaneously. For example, product owners can track their products, shippers can track their containers, carriers can monitor their vessel capacities, and security personnel can monitor the sensors to ensure the products are not tampered with. Disaster scenarios result in emergency situations that require a tremendous amount of coordination. Unfortunately, in many disasters, the fixed infrastructure has been destroyed, pre-

venting the use of cell phones and the Internet. In such a situation, network flexibility is critical. For example, if an office building catches fire, and the building's fire detectors, lights, and thermostats are part of the GSI, they could be quickly reprogrammed to guide people out of the building. Also, the building's security system could be reprogrammed to coordinate with the fire detectors and thermostats to direct first responders to people in need. The ability to commandeer these nodes to perform tasks they were not originally intended to perform requires dynamic reprogramming. The fact that the building's network is helping to evacuate people and direct rescuers at the same time demonstrates the need for an architecture that supports multiple concurrent applications. As the full capabilities of a GSI are better understood, the public will demand more and better applications creating a significant gap between societal needs and our ability to develop software that meets these needs. To counter this gap, a new software architecture must be developed.

There are several systems related to integrating WSNs with IP networks and forming a GSI. They include Tenet [10], IrisNet [9], and SensorWeb [19]. Tenet is an architecture that integrates WSNs with an IP network. Tenet moves most of a user's application onto the IP network where it is more reliable and easier to program, leaving only simple but well understood protocols to run within the WSN. Tenet enhances network flexibility by using *tasks*, which are sent by the IP nodes onto specific WSN nodes. Once installed, tasks can no longer propagate. Both IrisNet and SensorWeb focus on how to manage the data produced by a global sensor network. Specifically, SensorWeb provides a web portal that plots real-time sensor data on a map and provides query primitives for accessing this data, while IrisNet uses distributed databases to store sensor readings.

Our GSI architecture differs from these existing systems by focusing on providing a unified framework that spans both WSNs and IP networks. It does this by providing mobile scripts that can adapt to changes in the environment by autonomously moving to a different node, fetching a different set of services, or otherwise changing their behavior. This offers an additional level of adaptibility and flexibility, and is complimentary to IrisNet and SensorWeb (i.e., IrisNet's distributed database may be a service, and SensorWeb's portal may be an interface to the GSI). Our architecture, however, contrasts with Tenet by pushing more functionality into the WSN. Consider the burning building example. Using Tenet, the fire detector that senses the fire will have to first notify its master node on the IP network before a route discovery task can be deployed. Our proposed architecture differs by allowing a mobile script on the fire detector to fetch a route discovery service in the neighborhood, reducing overhead while increasing its responsiveness to environmental events.

Our proposed architecture addresses four problems that need to be resolved in order to build and fully exploit a GSI. The central problem is the difficulty in programming applications. The architecture must provide a new programming model and middleware to ease application development. Second, existing WSN protocols and services are not easily integrated due to inconsistent assumptions about the underlying hardware and software platform. This prevents code reuse, hindering progress by preventing new technologies from being integrated. The new architecture must ensure that current and future WSN subsystems interoperate and be easily integrated. It must also present a unified interface to these components allowing users to write a single platform-independent application that is able to run across a heterogeneous GSI. The third problem our architecture addresses is the need to integrate WSNs with the Internet, and to support multiple GSI-spanning applications simultaneously. Finally, our architecture provides a mechanism for applications to autonomously adapt to both an evolving platform and a changing environment.

Our solution structures applications as malleable, platform-independent, high-level mobile scripts that discover and invoke platform-specific services. Scripts will facilitate meta-level programming, be designed to maintain the structural and functional integrity of the application, and will enable both application restructuring and migration. Services provide reusable functional capabilities. In general, they will be written using native code, discovered at runtime, and distributed strategically within the sensor network or on servers accessible via the Internet. Service provision will enable an application to use computational resources that are best suited for a specific device or network class, at that particular point in time and location, and in response to changing application needs or environmental conditions. Our approach will allow applications to function in diverse settings by employing dynamic rebinding of programmer-specified capabilities to different services as they execute over extended periods of time within the same network or across multiple networks.

The remainder of this paper is organized as follows. Section 2 describes the proposed system architecture. Section 3 discusses the challenges of building such a system. Section 4 describes our approach. The paper ends with conclusions in Section 5.

2 System Architecture

Future GSIs will be large scale, heterogeneous, and have long-term deployments. They will consist of a wide variety of devices with vastly different capabilities, and span both WSNs and the Internet. GSIs form a powerful platform that enables applications to exploit a wide range of sensing data ranging from local data obtained from tiny embedded sensors to global data obtained from satellites. To maximize the system's utility, its software architecture must be able to support multiple concurrent applications owned by different users that may span the entire GSI, and be able to efficiently integrate new functionality as they are developed and needed. In this section, we first discuss the physical architecture of the GSI followed by the software architecture.

2.0.1 Physical Architecture

The physical architecture consists of the sensor nodes, micro-servers, and the Internet. An overview of it is shown in Figure 1. The figure shows multiple sensor networks connected to the Internet which contains databases for storing applications and data, and user terminals for accessing the system. Not shown are the sensing resources available on

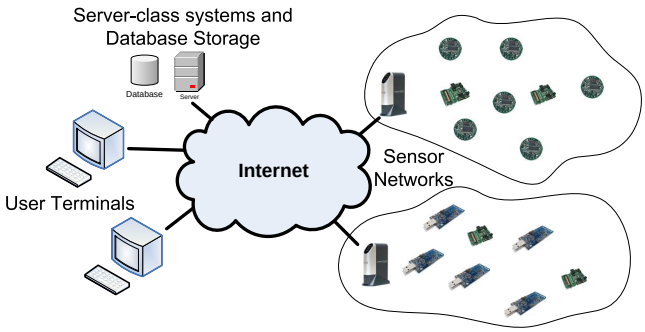

Figure 1. The physical architecture of the network

the Internet (e.g, data from weather satellites, seismographs, street sensors, web cams, etc.) that will also be part of the GSI. The physical architecture consists of two basic types of networks: the resource-constrained WSNs, and the Internet.

Each WSN may be deployed and administered independently. They primarily consist of tiny embedded sensors, but also contain micro-servers capable of more computationally-intensive tasks and interfacing with the Internet. They may range in scale, consist of different hardware, and use different wireless technologies. Multiple sensor networks may overlap. Since the system has long-term deployment and must support multiple transient users, it is important to allow the network to physically evolve. As new sensors are developed, integrating them within an existing GSI should be straightforward. A changing environment may require new sensor networks with different devices to be deployed and existing ones to be dismantled. Continuous exposure to the elements will cause sensor nodes to fail, requiring new nodes to be deployed. Some sensor nodes may be mobile, e.g., mounted on a car or person. All of this results in a highly dynamic physical architecture.

The Internet provides a relatively reliable, static, and resource-rich environment through which users can interact with the GSI. It has servers that can process sensor data, and databases for storing applications and data. The Internet also allows application components located in one WSN to communicate with components located in other WSNs or on the Internet itself. In some cases, an application may wish to move to another WSN due to the mobility of a physical object of interest. For example, in the supply chain monitoring application, a container may be moved from a ship onto a train, meaning the application tracking it may need to jump from the ship's WSN onto the train's WSN. The Internet provides the physical means for this application migration to occur.

2.0.2 Software Architecture

The software architecture consists of user-defined mobile scripts, services, and an execution engine. Applications are written as a collection of mobile scripts. These scripts make references to services that are dynamically bound at runtime. The execution engine interprets the scripts and binds their service references to the components that implement them. If a required service is not locally available, the execution engine finds a component that implements it and either fetches and locally executes it or remotely executes it. If a particular service is not found, the execution engine may have to compose multiple components together to achieve the desired service. By dynamically binding services to the mobile scripts, a user's application is able to adapt to changes in its context, and new services can be efficiently integrated. By executing multiple mobile scripts simultaneously, the network is able to support concurrent applications enhancing the GSI's utility.

Mobile scripts are written in a high-level language like TinyScript [1], can be dynamically installed, and are able to migrate across and execute on all nodes in the GSI. This seamless integration of sensor and IP networks simplifies application development but results in the mobile script experiencing large changes in resource availability. The dynamic binding of services allow the script to adapt to these changes. In order minimize execution and migration overhead and allow these scripts to fit on highly constrained nodes, they must be lightweight. In our proposed architecture, they consist of a sequence of calls to high-level services and a minimal amount of control code. However, some policy must be in place for handling situations where a mobile script attempts to migrate onto a node that does not have enough resources to support it. This policy may be application-specific or network wide and dynamic or static, but is enforced by the execution engine. Mobile scripts are the most dynamic components of the GSI. They may quickly come and go based on the presence of users and changes in the environment.

The execution engine plays a key role in our architecture because it provides a foundation for mobile scripts. It must be designed to span many generations of WSNs so that an application need not be re-implemented each time a new WSN or Internet technology is developed. This is achieved by including a minimal amount of functionality within the execution engine itself, and exporting as much as possible into services. Functions likely to be included in the execution engine include the script scheduler and quality of service enforcer, service discovery and binding, inter-script communication and coordination, and script migration. The execution engine may also provide mechanisms for composing and decomposing scripts and services to achieve greater flexibility (e.g., enabling them to fit on even more highly resource-constrained devices). The execution engine should be relatively static. It can be occasionally updated through code reversioning [12, 14], but should otherwise provide a stable foundation for mobile scripts.

Services implement reusable functionality and are dynamically bound to scripts. Unlike services in a traditional distributed computing, the services in our proposed architecture are not the same throughout the network. Instead, they are tailored to the unique and current characteristics of their execution environment. For example, a useful service for structural health monitoring applications is the fast Fourier transform (FFT). In a resource-deprived sensor node, the FFT service may only partially compute the FFT locally, exporting the majority of computation to a more powerful device on the Internet. On the other hand, the FFT service on an Internet PC may perform the entire FFT locally. Lo-

cation is a another context-sensitive service where scripts on nodes outdoors use GPS while those indoors use an indoor localization system [18, 13, 15, 4]. Another example is routing where a script on the Internet will use OSPF while one running in a WSN may use MintRoute [21]. The details of which service is used is hidden from the script, allowing it to be platform-independent. Services are strategically deployed across the GSI to enable the execution engine to efficiently exploit them when they are needed. Services may be dynamically installed and uninstalled, or remotely executed. They are designed to be updated more frequently than the execution engine, but less than that of scripts. They should typically be implemented in native code to minimize overhead.

3 Challenges

There are many challenges involved in developing a GSI. This section describes some of these challenges.

Application Concurrency. While supporting multiple concurrent WSN applications has been investigated in the past [22], they did not provide quality of service provisions for each application. This is problematic when the applications conflict with each other, and is especially true in networks with extremely limited resources (e.g., one application uses up so much memory and bandwidth that another cannot run). A mechanism must be developed that guarantees each application a minimum quality of service. Policies may include traditional ones like admission control and priority levels, but may also include more radical ones that are specific to our architecture like script reallocation (i.e., moving a script to a neighboring node with more resources), or script morphing (i.e., changing the service bound to a script to reduce its resource utilization).

Service Description. The reliance on service provision entails developing new, compact, and flexible service specifications. The spatiotemporal existence of the applications is likely to foster the introduction of time and space concepts into the scripting language with multiple notions of space coexisting in a single application. For example, space may correspond to a geographic area, a network, hop count region, or nodes with certain application-defined attributes. Also, since GSIs are dynamic, details of the local computing infrastructure may have to be exposed to the script.

Scripts must describe the types of services they require, and the services must describe what they provide. A service-description language must be created to ensure the requirements specified by the script can be matched to the services that implement them. This language must be extensible and expressive enough to allow automated service composition to satisfy the requirements set forth by the scripts. For example, in the disaster scenario, the script may require a service that finds a safe exit route. Since such a service is application-specific and is unlikely to be provided natively, the middleware must compose several other services together to achieve this service, or the user must design the scripts to make use of whatever services are available. The the service description language's representation will likely have to be heterogeneous. On the Internet, it may be implemented in WSDL, while within a sensor network, it may be implemented using primitive data types. This suggests the need for a meta-specification language that is used by the script programmer, but is decomposed into a network-specific representation when the script enters the network.

Service Discovery. The limited resources on a typical WSN node prevents them from holding a copy of every service. Distributed service repositories must be created, and a service discovery protocol must be provided. When a service is required, the execution engine must find the code that implements it using a service discovery protocol, and bind it to the script. The binding can either be done using a remote procedure call, or the code can be downloaded and executed on the local node. One interesting question is whether the services should be bound to a script or node. The answer will likely depend on the type of service. Some services like a simple data aggregation algorithm (e.g., average) may be bound directly to a script and, hence, moved with the script as it migrates. This may not always be desirable since it increases migration overhead and may result in inefficient code memory usage when the destination already has a copy of the service. Other services like sending a message or reading a sensor are inherently tied to hardware on a particular node and hence should not be bound to a script. Regardless, once the service is no longer needed, it must be disposable. This should ideally be automatically handled by the execution engine.

Device heterogeneity. Some services like `sense` are inherently network-specific. This is problematic because a script may attempt to execute it on a node that does not have the sensor and, thus, cannot implement the service. New mechanisms must be developed to either prevent such a scenario, or to gracefully handle them. The scenario can be prevented by analyzing the services required by a script and preventing it from migrating to a node that does not provide all of the required services. The situation can be handled more gracefully by trying to get the sensor reading from a nearby node that has the sensor, and including a confidence level with the sensor reading.

Inconsistent Resource Availability. Another problem arises from the fact that scripts are mobile and the devices in a GSI differ widely in resource availability. Specifically, it is possible that a script may attempt to migrate onto a node that is unable to hold it due to lack of memory. In such circumstances, there are several policies that can be enforced. The simplest is to abort. This is undesirable because it may make the application less reliable and complicate application development. A more elegant approach is to have the script automatically leave part of itself behind, or be able to temporarily split into multiple fragments until it moves onto a node with enough memory. A script may also drop unnecessary services, drop code that it will never execute again, or de-allocate data memory that is no longer needed. If a script is only briefly entering a more resource constrained network, the execution engine may create a proxy script that enters the network in place of the original script, and returns the result to the original script when done. New protocols need to be developed to provide such functionalities.

Basic service set. A key challenge in our architecture is determining the appropriate boundary between high-level platform-independent scripts and low-level platform-specific

services. Services should be general to support many applications, and yet be high-level to simplify application development. Determining a basic set of services at the appropriate level of abstraction is essential. This set of services include service discovery and binding, communication, script migration, and script morphing.

Coordination. Since applications are expected to be structured in terms of multiple mobile scripts, new coordination and restructuring mechanisms are needed. Scripts belonging to different applications will often have to share the same resources. Allowing them to coordinate their resource utilization is essential, especially if some applications have higher priorities than others. Since the scripts are mobile and volatile, a decoupled style of communication like those provided by shared memory and tuple spaces [8] is preferable. New group communication schemes and coordination models are needed to ensure the scripts belonging to the same application can effectively cooperate, and scripts belonging to different applications do not conflict.

Event distribution A mobile script must be able to react to changes in its context. This requires an event generation and distribution mechanism. The wide range of event types, sources, priorities, and usage patterns is likely to lead to a complex and heterogeneous event generation and distribution system. Some events may be highly localized while others may be distributed across significant distances; some events may be generated only on demand through a subscription system, others may be distributed to all applications within some logical or geographic scope, and others may be governed by application-specific rules.

4 Approach

Creating a GSI will require integrating a variety of Internet and WSN systems. In general, they can be divided into those that provide service discovery, and those that enable script mobility and concurrency. On the IP network, the execution engine can be implemented by combining standard Web Service technology [3] with a mobile agent middleware like LIME [16] or Limone [6]. Web services provides languages [17] for describing reusable services, and components for registering and discovering them [20]. Mobile agents provide strong script mobility where execution state is maintained across migrations. This can potentially simplify applications since a script need not restart each time it migrates (it prevents the script from having to manually transfer its state and having an initial case statement that determines where the agent should start executing upon arrival at the destination). Within sensor networks, a standard platform like SNA [5] can be used to provide a common set of components and services, and a framework to integrate them. It can be integrated with a mobile agent technology for WSNs like Agilla [7] to implement the GSI execution engine. The overall GSI system is similar to the Arch Rock Primer Pack [2], in that it abstracts WSNs into web services that can be programmed using standard programming languages. However, it goes one step further by providing web service-like functionality to scripts *within* a WSN, and also offers the additional flexibility and utility gained through script mobility and concurrent script execution.

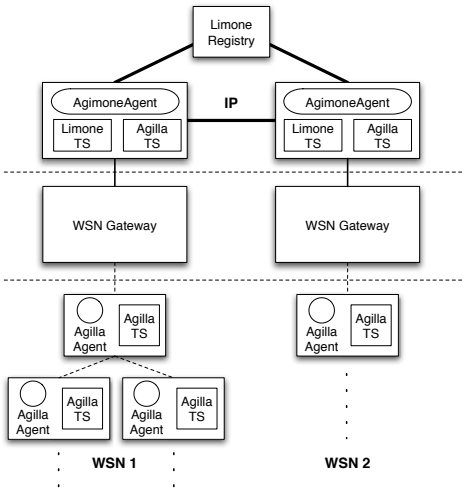

Figure 2. The Agimone system architecture

Our proposed GSI architecture draws upon previous experience developing Agimone [11], the first middleware supporting mobile agent migrations between sensor networks via the Internet. Agimone's system architecture is shown in Figure 2. It integrates Agilla and Limone. Both middleware systems provide mobile agents as the basic units of execution, and tuple space-based coordination [8]. A mobile agent is an autonomous unit of execution that contains its own state, and is able to migrate across nodes while maintaining its state. A user implements a sensor network application within one or more Agilla agents. Using Agimone, these mobile agents are able to discover other sensor networks and migrate to them. This is useful when the agents are unable to finish their tasks in one sensor network but can in another, or when the agents' task requires them to gather data from multiple sensor networks. For example, in the supply chain monitoring scenario, mobile agents may need to traverse the Internet and multiple WSNs located on different ships looking for containers containing a particular item.

While Agimone demonstrates the feasibility and usefulness of integrating WSNs with the Internet, it does not truly integrate sensor and IP networks since it does not allow Agilla agents to migrate onto the Internet; it only allows them to pass through the Internet on their way to another WSN. Also, it does not allow the agent to morph, meaning it may not be able to migrate onto networks with fewer resources. GSI goes beyond Agimone by providing a framework that covers both WSNs and the Internet. It allows the user to write platform-independent scripts that react to changes in its environment. It is a service oriented architecture where mobile scripts can incorporate services tailored to the particular capabilities of the local network, and can drop them when they are no longer needed or can no longer function in the current environment. This will enable new applications that are more flexible and comprehensive than presently possible.

5 Conclusion

WSNs consisting of thousands of nodes are growing increasingly sophisticated and diverse. As more of them are deployed and connected to the Internet, they form a powerful global sensing infrastructure (GSI) that can offer sensing capabilities far greater than what a single WSN can offer. In order to accelerate GSI formation, a new software architecture must be developed that enables independently developed software components to be easily integrated, multi-application support, and application flexibility. We propose a software architecture that provides this via high-level platform-independent mobile scripts that are dynamically bound to low-level platform-dependent services. Applications are created as a collection of mobile scripts. Concurrent applications are supported by allowing multiple scripts to execute on a node. Sensor and IP networks are unified by allowing scripts to migrate seamlessly between the two networks. Flexibility is achieved through script mobility and dynamically binding platform-specific services to the scripts. New components can be integrated in the form of additional services. Applications can be incrementally upgraded to use these new services by replacing their scripts. While there are many challenges, such an architecture is needed to reconcile the disparity between the capabilities of existing WSN software architectures, and the demands of future GSI applications.

Acknowledgment

This research is supported by the NSF under NOSS contract CNS-0520220.

6 References

[1] `http://www.cs.berkeley.edu/~pal/mate-web/files/tinyscript-manual.pdf`.

[2] Arch rock primer pack. `http://www.archrock.com/downloads/datasheet/primerpack_datasheet.pdf`.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer-Verlag, October 2003.

[4] N. Bulusu, J. Heidemann, and D. Estrin. Gps-less low cost outdoor localization for very small devices. Technical Report 00-729, USC, April 2000.

[5] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao. Towards a sensor network architecture: Lowering the waistline. In *Tenth Workshop on Hot Topics in Operating Systems (HotOS X)*, June 12 2005.

[6] C.-L. Fok, G.-C. Roman, and G. Hackmann. A Lightweight Coordination Middleware for Mobile Computing. In *Proc. of Coordination'04*, pages 135–151, February 2004.

[7] C.-L. Fok, G.-C. Roman, and C. Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Proc. of ICDCS'05*, pages 653–662. IEEE, June 2005.

[8] D. Gelernter. Generative Communication in Linda. *ACM Trans. on Programming Languages and Systems*, 7(1):80–112, January 1985.

[9] P. Gibbons, B. Carp, Y. Ke, S. Nath, and S. Seshan. Irisnet: An architecture for a worldwide sensor web. *IEEE Pervasive Computing*, pages 22–33, October-December 2003.

[10] O. Gnawali, K.-Y. Jang, J. Paek, M. Vieira, R. Govindan, B. Greenstein, A. Joki, D. Estrin, and E. Kohler. The tenet architecture for tiered sensor networks. In *Proc. of SenSys '06*, pages 153–166, New York, NY, USA, 2006. ACM Press.

[11] G. Hackmann, C.-L. Fok, G.-C. Roman, and C. Lu. Agimone: Middleware support for seamless integration of sensor and IP networks. In *Lecture Notes in Computer Science*, volume 4026, pages 101–118, 2006.

[12] J. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proc. of SenSys'04*, pages 81–94. ACM Press, 2004.

[13] Y. Kwon, K. Mechitov, S. Sundresh, W. Kim, and G. Agha. Resilient localization for sensor networks in outdoor environments. In *Proc. of ICDCS '05*, pages 643–652, 2005.

[14] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. NSDI'04*. USENIX, 2004.

[15] D. Moore, J. Leonard, D. Rus, and S. Teller. Robust distributed network localization with noisy range measurements. In *Proc. of Sensys'04*, November 2004.

[16] A. L. Murphy, G. P. Picco, and G.-C. Roman. LIME: A Middleware for Physical and Logical Mobility. In *Proc. of ICDCS'01*, pages 524–533, April 2001.

[17] E. Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley, 1st edition, May 2002.

[18] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Mobile Computing and Networking*, pages 32–43, 2000.

[19] A. Santanche, S. Nath, J. Liu, B. Priyantha, and F. Zhao. Senseweb: Browsing the physical world in real time. In *Proc. of IPSN'06*, April 2006.

[20] UDDI-Organization. Uddi technical white paper. http://www.uddi.org/pubs/Iru UDDI Technical White Paper.pdf, 2000.

[21] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *Proc. of SenSys'03*, pages 14–27, 2003.

[22] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun. Supporting concurrent applications in wireless sensor networks. In *Proc. of SenSys '06*, pages 139–152, New York, NY, USA, 2006. ACM Press.