Report Number: wucse-2009-2

2009

Enhanced Coordination in Sensor Networks through Flexible Service Provisioning

Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu

Heterogeneous wireless sensor networks represent a challenging programming environment. Servilla addresses this by offering a new middleware framework that provides service provisioning. Using Servilla, developers can construct platform-independent applications over a dynamic set of devices with diverse computational resources and sensors. A salient feature of Servilla is its support for dynamic discovery and binding to local and remote services, which enables flexible and energy-efficient in-network collaboration among heterogeneous devices. Furthermore, Servilla provides a modular middleware architecture that can be easily tailored for devices with a wide range of resources, allowing resource-constrained devices to provide services while leveraging the capabilities... **Read complete abstract on page 2.**

Enhanced Coordination in Sensor Networks through Flexible Service Provisioning

Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu

Complete Abstract:

Heterogeneous wireless sensor networks represent a challenging programming environment. Servilla addresses this by offering a new middleware framework that provides service provisioning. Using Servilla, developers can construct platform-independent applications over a dynamic set of devices with diverse computational resources and sensors. A salient feature of Servilla is its support for dynamic discovery and binding to local and remote services, which enables flexible and energy-efficient in-network collaboration among heterogeneous devices. Furthermore, Servilla provides a modular middleware architecture that can be easily tailored for devices with a wide range of resources, allowing resource-constrained devices to provide services while leveraging the capabilities of more powerful devices. Servilla has been implemented on TinyOS for two representative hardware platforms (Imote2 and TelosB) with drastically different amounts of resources. Microbenchmarks demonstrate the efficiency of Servilla's implementation, while an application case study on structural health monitoring demonstrates the efficacy of its coordination model for integrating heterogeneous devices.

2009-2

Enhanced Coordination in Sensor Networks through Flexible Service Provisioning

Authors: Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu

Corresponding Author: liangfok@wustl.edu

Web Page: http://mobilab.cse.wustl.edu/projects/servilla/

Abstract: Heterogeneous wireless sensor networks represent a challenging programming environment. Servilla addresses this by offering a new middleware framework that provides service provisioning. Using Servilla, developers can construct platform-independent applications over a dynamic set of devices with diverse computational resources and sensors. A salient feature of Servilla is its support for dynamic discovery and binding to local and remote services, which enables flexible and energy-efficient in-network collaboration among heterogeneous devices. Furthermore, Servilla provides a modular middleware architecture that can be easily tailored for devices with a wide range of resources, allowing resource-constrained devices to provide services while leveraging the capabilities of more powerful devices. Servilla has been implemented on TinyOS for two representative hardware platforms (Imote2 and TelosB) with drastically different amounts of resources. Microbenchmarks demonstrate the efficiency of Servilla's implementation, while an application case study on structural health monitoring demonstrates the efficacy of its coordination model for integrating heterogeneous devices.

Notes:
This is currently under submission to Coordination 2009.

Type of Report: Other

Enhanced Coordination in Sensor Networks through Flexible Service Provisioning

Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu

Dept. of Computer Science and Engineering
Washington University in St. Louis
Saint Louis, MO, 63105, USA
[liang, roman, lu]@cse.wustl.edu

Abstract. Heterogeneous wireless sensor networks represent a challenging programming environment. Servilla addresses this by offering a new middleware framework that provides service provisioning. Using Servilla, developers can construct platform-independent applications over a dynamic set of devices with diverse computational resources and sensors. A salient feature of Servilla is its support for dynamic discovery and binding to local and remote services, which enables flexible and energy-efficient in-network collaboration among heterogeneous devices. Furthermore, Servilla provides a modular middleware architecture that can be easily tailored for devices with a wide range of resources, allowing resource-constrained devices to provide services while leveraging the capabilities of more powerful devices. Servilla has been implemented on TinyOS for two representative hardware platforms (Imote2 and TelosB) with drastically different amounts of resources. Microbenchmarks demonstrate the efficiency of Servilla's implementation, while an application case study on structural health monitoring demonstrates the efficacy of its coordination model for integrating heterogeneous devices.

1 Introduction

Wireless sensor networks (WSNs) [18] are becoming increasingly heterogeneous both in terms of computational power and sensor types. This is due to two primary reasons. First, network heterogeneity allows a network to be both computationally powerful and deployed in high densities. Powerful nodes can perform complex operations like data analysis, but incurs higher cost and power consumption. Conversely, low power WSN nodes can provide higher deployment densities and increase network lifetime. By integrating both types of nodes, a heterogeneous WSN can combine the best features of each, i.e., high levels of computational power, network densities and lifetimes. Second, network heterogeneity follows from the natural evolution of WSNs. WSN nodes are embedded in the environment and remain for a long period of time. During this time, new nodes and sensors are developed and deployed, resulting in a heterogeneous WSN.

Network heterogeneity presents a formidable problem for application developers. Since the target platform is no longer well defined, the application must be platform-independent to avoid having to custom-tailor it to each platform. Yet, the application must still be able to access platform-specific capabilities like sensing and computing to

make full use of the underlying hardware. Furthermore, the application must accommodate an extremely diverse set of node capabilities and resources. These seemingly contradictory requirements complicate application development and motivates the need for a new programming model.

To address the challenges in programming heterogeneous WSNs, we developed Servilla, a novel middleware framework that supports a novel coordination model. Servilla advances coordination models for WSNs in three important ways. First, Servilla structures applications in terms of platform-independent tasks and expose platform-specific capabilities as services. This ensures that applications remain platform-independent, which is critical as WSNs become increasingly heterogeneous. It also enables applications to access resources on a node without having active processes, or agents, on the node. This reduces the system's minimum resource requirements, increasing the range of devices that can be supported. Second, Servilla provides a specialized service description language, which enable application tasks to selectively but flexibly access services that exploit the capabilities of the hardware available at a particular time and place. This allows better adaptation to network heterogeneity, and facilitates in-network collaboration between heterogeneous WSN nodes, achieving higher levels of efficiency and flexibility. Finally, Servilla provides a modularized middleware architecture and enables asymmetry in the middleware among WSN nodes. This widens the scope of hardware devices that can be integrated.

Servilla's coordination model is inspired by the concept of *Service-oriented computing* (SOC) [52], which provides loose and flexible coupling between application components. It is used on the Internet and has recently been explored in the context of WSNs. Two systems in particular are Tiny Web Services (TWS) [54] and PhyNet^{TM} [6]. TWS implements an HTTP server on each node and enables applications to invoke services using HTTP requests. PhyNet^{TM}provides a central gateway that exposes WSN capabilities as web services. Unlike these systems, Servilla uniquely takes the SOC programming model *inside* a WSN. It exploits the loose coupling between service consumers and providers to separate application-level platform-independent logic from the low-level software components that exploit platform-specific capabilities. Furthermore, by allowing application logic to execute inside a WSN, higher levels of efficiency are obtainable via in-network processing [33]. For example, in a structural health monitoring application, a low-power node may use a simple threshold-based algorithm to detect potentially damage-inducing shocks, and only activate more powerful nodes that perform the complex operations to localize damage when necessary [28]. Or, in a surveillance application, low-power nodes may sense vibrations from an intruder and activate more powerful nodes with cameras [31]. The ability to support collaboration among heterogeneous devices inside a WSN is a key feature that distinguishes this work from existing SOC middleware for WSNs.

The remainder of the paper is organized as follows. Section 2 presents Servilla's programming model. Section 3 presents Servilla's programming languages. Section 4 presents Servilla's middleware architecture and implementation. Section 5 presents an empirical evaluation on two representative sensor platforms with diverse resources. Section 6 evaluates the efficacy of Servilla by using it to implement a structural health mon-

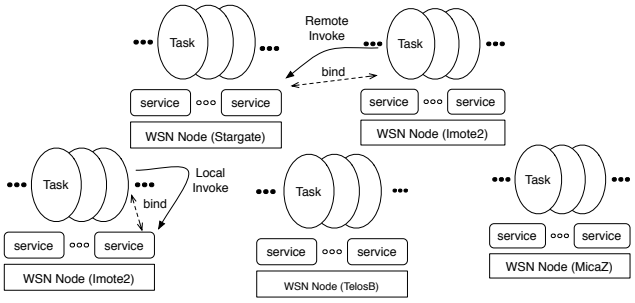


Fig. 1. Servilla targets heterogeneous and dynamic WSNs in which nodes provide services that are used by application tasks either locally or remotely. Services are platform-specific while tasks are platform-independent.

itoring application. Section 7 presents related work. The paper ends with conclusions in section 8.

2 Programming Model

An overview of a WSN using Servilla is shown in Figure 1. Servilla is meant for heterogeneous networks that integrate resource-poor nodes with resource-rich ones. Resource-poor nodes are less costly, more energy efficient, and can be deployed in greater numbers and at higher densities, enabling finer and more frequent sensing. Resource-rich nodes are more expensive but offer computational power and advanced sensing capabilities. Servilla is designed for heterogeneous WSNs with different classes of nodes; it is not meant for flat WSNs composed entirely of resource-poor nodes.

Applications are implemented as tasks that invoke services through a service provisioning framework supported by Servilla. Tasks are platform-independent application processes that contain code, state, and service specifications. To ensure platform-independence, the code cannot contain instructions that access platform-specific capabilities like sensors. Instead, these capabilities are revealed as services that are accessed through a *service provisioning framework*. The service provisioning framework takes a task's service specifications and finds services that match them. The service specifications describe both the service's interface and non-functional properties, such as the energy consumed by a service. This enables tasks, for example, to bind to the most energy efficient services.

Services expose platform-specific capabilities, are implemented natively, and can, thus, be fine-tuned for maximum efficiency. They provide a description that can be compared with a task's service specification. Services are able to maintain state, provide multiple methods, and have their own thread of control, enabling them to operate in parallel with tasks. This enables higher degrees of concurrency and efficiency. For example, in a structural health monitoring application, a service provided by a low-power node can continuously monitor an accelerometer and set a flag if the vibrations exceed a threshold. A task executing on a more powerful node can remain asleep the majority

of the time, and only periodically wake up to check this flag to determine whether there is potential damage.

For brevity, the mechanism by which tasks communicate are not shown in Figure 1 since service provisioning is the focus and main contribution of this paper. Tasks communicate via localized tuple spaces that are structured in the same manner as that in Agilla [23]. Tuple space coordination facilitates decoupled communication, allowing better adaptation to a changing network. They serve as a flexible means of communication between application processes and are orthogonal to service provisioning. While service provisioning messages could be sent using tuple spaces, they are sent in an RPC-like [19] fashion in the current implementation.

Tasks remain platform-independent by delegating all platform-specific operations to services. There are two essential steps for this to occur: *binding* and *invocation*. Binding is the process of discovering and establishing a connection to the service. Invocation is the process of accessing a service.

Service binding consists of a three-step process: discovery, matching, and selection. Discovery consists of finding services that are available to a given task. In many traditional SOC frameworks, it involves querying a centralized service registry located somewhere in the network. While this is sufficient in traditional networks, and may serve as a fall back mechanism for allowing Servilla tasks to discover distant services, it is usually not appropriate in WSNs. First, most WSN nodes operate on batteries and accessing a distant registry is not energy efficient and can unacceptably reduce network lifetime. Moreover, the spatial aspect of WSNs are relevant in that closer services are usually preferred over distant ones, e.g., if a task wants to know the temperature, it usually wants to know the ambient temperature rather than a distant location's. For these reasons, Servilla is optimized for *localized* coordination and does not rely on a centralized service registry. Instead, each node has its own registry containing only the services that it provides.

During the service discovery process, the local registry is initially checked for a match. If no match is found, the registries on single hop neighboring nodes are checked. This increases a network's flexibility by allowing tasks to run on nodes that only partially satisfy the service requirements, since missing services can be provided by neighboring nodes. Furthermore, although accessing a remote service requires wireless communication, energy efficiency can be increased overall by allowing high-power nodes to use low-power ones, enabling the high-power nodes to remain asleep a larger percentage of the time.

Service matching involves finding a service that fulfills a task's requirements. Recall that tasks include specifications that can be compared to descriptions provided by services. The matching process must be flexible since the service and tasks are usually developed separately. Yet, it must be semantically correct to ensure that the service behaves in a predictable manner. A service can be minimally described by its interface. Ideally, the names of the methods, the order, number, and types of their parameters, and even the return types should not require an exact match for service binding, since that would enable maximum flexibility. To achieve this, large amounts of meta-data must be included in the specification that describe the method names, input parameters, and return values in great detail. Unfortunately, such a specification is verbose and requires

a complex parser, both of which consume sizable computational resources that are not available on many WSN nodes. To account for this, Servilla compromises by dividing specifications into functional and non-functional properties. Functional properties include the interface and require an exact match. Nonfunctional properties describe attributes like power consumption and do not require an exact match. For example, suppose an FFT-calculating service has a non-functional attribute specifying that it is version 5. Such a service can be bound to a task that specifies it requires *at least* version 4. By enforcing an exact match between functional properties and an inexact match between non-functional ones, Servilla provides a degree of service binding flexibility while still maintaining reasonable resource requirements.

Once a matching service is found, the binding process is completed by selecting it. Selection consists of informing the task of the chosen service, and is accomplished by saving the provider's network address in the task's state. Once saved, the task is able to access the service by invoking it. Note that this address is hidden from the application developer, who is able to invoke the service based on its name, a process that is described next.

Service invocations are analogous to remote procedure calls (RPCs) [19]. The task provides the name of the service, the method to execute, and the input parameters. After the service executes, return results are given to the task. Since the task and service may be located on different nodes, the process may fail, e.g., due to message loss. To account for this, Servilla provides a mechanism that notifies a task when and why an invocation fails. This is necessary because service invocations may fail in many ways depending on whether the service is local or remote, and tasks may want to handle various error conditions differently. For example, local invocations may fail because the service is busy, in which case the task may try again later, while remote invocations may fail due to disconnection, in which case the task may want to abort.

3 Programming Language

Servilla provides two light-weight programming languages tailored to support service provisioning in WSNs. The first, *ServillaSpec*, is used to create service specifications and descriptions that enable flexible matching between tasks and services. The second, *ServillaScript*, is used to create tasks and compiled into bytecode that runs on a Virtual Machine. Services are implemented in NesC [25] on TinyOS [32] and compiled into native binary code for run-time efficiency. Each of Servilla's specialized languages are now described.

3.1 ServillaSpec

ServillaSpec is used to describe services and is needed to match services required by tasks to those provided by nodes. To support resource-constrained nodes, the service specification language must be compact and should not require an overly complex matching algorithm. As such, standard specification languages using on the Internet like WSDL [56] are avoided due to their relative verboseness and highly complex parsers.

```
NAME = fft
METHOD = fft-real
INPUT = {int dir, int numSamples, float[] data}
OUTPUT = float[]
ATTRIBUTE Version = 5.0
ATTRIBUTE MaxSamples = 5000
ATTRIBUTE Power = 10
```

Fig. 2. A specification describing a FFT service

```
1.  uses Temperature; // declare required service
2.
3.  void main() {
4.      int count = 0; float temp;
5.      bind(Temperature, 2); // bind service within 2 hops
6.      while(count++ < 10) {
7.          temp = invoke(Temperature, "get"); // invoke service
8.          send(temp);
9.      }
10.     unbind(Temperature);
11. }
```

Fig. 3. A task that invokes a temperature sensing service 10 times

ServillaSpec avoids verbose syntax and limits the types of properties that can be included in a service specification. An example is shown in Figure 2. The first line specifies the name of the service. It is followed by three-line segments each specifying the name, input parameters, and output results of a method provided by the service. The remainder of the specification is a list of attributes that specify non-functional properties of the service. They enable flexibility in matching by defining a name, relation, and value. Using attributes, a task can, for example, require a floating point FFT service that consumes *at most* 50mW. Such a specification would match a service whose description is shown in Figure 2.

By limiting the property types to be only the five shown in Figure 2 (i.e., NAME, METHOD, INPUT, OUTPUT, and ATTRIBUTE), and arranging them to always be in the same order, the specification can be greatly compressed. For example, since the service's NAME property always appears first, the property's identifier, NAME, can be omitted. Thus, the NAME property in the specification shown in Figure 2 can be compressed to just 4 bytes, "fft" followed by a null terminator. This compression saves memory and enables greater matching efficiency.

3.2 ServillaScript

ServillaScript is used to create application tasks. Its syntax is similar to other high level languages like JavaScript [22], but with key extensions for service provisioning. An example, shown Figure 3, implements an application that periodically takes the temperature and sends the reading to the base station. It declares the name of the file containing the specification of the required service on line 1, which in this case is a

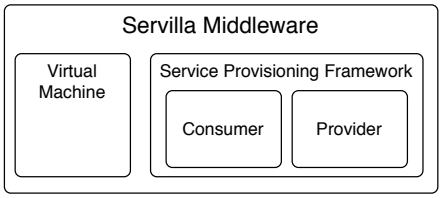


Fig. 4. Servilla's middleware consists of a virtual machine and a service provisioning framework (SPF). The SPF consists of a consumer and provider.

temperature sensing service. The task initiates the service binding process on line 5. In this case, registries within two hops are searched. The task then loops ten times invoking the service on line 7 and sending the temperature to the base station on line 8. The task ends by disconnecting from the service on line 10.

The example above illustrates how ServillaScript enables tasks to 1) indicate which services are needed, 2) initiate the service discovery process, 3) invoke services, and 4) disconnect from services. Aspects not shown for brevity include checking whether a service is bound, and, if so, how many hops away the service is located. This will allow the task to throttle how often it invokes the service based on its distance. Another aspect not shown is error handling code. If an error occurs due to a service becoming unavailable, the invocation will return an error indicating the cause, as discussed in Section 2.

4 Middleware

Servilla's middleware architecture, as shown in Figure 4, consists of a virtual machine (VM) and a service provisioning framework (SPF). The VM is responsible for executing application tasks. The SPF consists of a consumer (SPF-Consumer) that discovers and accesses services, and provider (SPF-Provider) that advertises and executes services.

A VM is used because WSN nodes contain processors that span a wide range of instruction sets. Application tasks are compiled into the VM's instruction set, which is uniform across all hardware platforms, ensuring that tasks are platform-independent. Furthermore, the VM provides the dynamic deployment of application tasks, justifying the need for dynamic service binding. The VM is based on Agilla [23] though with major extensions to support services and the SPF. Specifically, whenever a task performs an operation involving a service, the VM passes the task to the SPF-Consumer, which is described next.

4.1 SPF-Consumer

The SPF-Consumer is responsible for discovering, matching, and invoking services on behalf of tasks. As shown in Figure 5 the SPF-Consumer consists of a Service Finder, Binding Table, and Service Scheduler. The Service Finder is responsible for finding services that match a task's specifications. It first searches locally and, if no matches

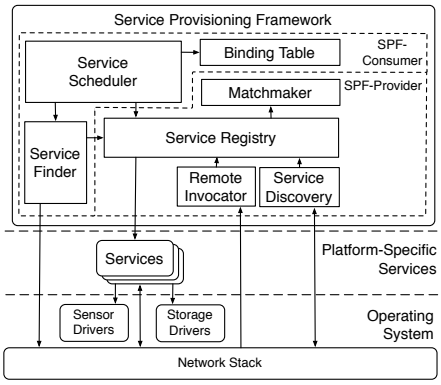


Fig. 5. The detailed architecture of the Service Provisioning Framework.

are found, searches one hop neighbors. Note that while this increases the likelihood of selecting a local service, it does not necessarily select the most energy efficient provider. If a task wanted to bind to an energy efficient provider, it can include an energy attribute in its service specification, thus enabling energy efficient service provisioning. When a provider is selected, its address is stored in the Binding Table. The Binding Table maps the task's service specification to the provider that will perform the service. It is updated when the Service Finder discovers a better provider and when a task explicitly unbinds from a service. A task can query a Binding Table to determine whether it has access to a particular service.

The Service Scheduler carries out the actual invocation. It takes the input parameters provided by the task, sends them to the provider, and waits for the results to arrive. Once the results arrive, it passes them to the task which can then resume executing. If the results do not arrive within a certain time, the Service Scheduler aborts the operation and notifies the task of the error.

4.2 SPF-Provider

The SPF-Provider is responsible for providing and executing services. Its architecture, shown in Figure 5, consists of a Service Registry, Matchmaker, Remote Invocator, and Service Discovery component. The Service Registry contains the specifications of all locally-provided services. The Matchmaker is used to determine whether a service meets the requirements of a task. When the SPF-Consumer tries to find a service, the Matchmaker is used to determine whether a matching service exists. Note that in this architecture, the task's specification must be sent from the SPF-Consumer to the SPF-Provider. This is because the Matchmaker is located on the SPF-Provider. Alternatively, the Matchmaker can be moved onto the SPF-Consumer to reduce the footprint of the SPF-Provider. However, this requires that all specifications be sent to the SPF-Consumer, a process that may incur higher communication cost.

	TelosB	Imote2
Processor	8MHz 16-bit TI MSP430	13-416MHz 32-bit Intel PXA271 XScale
Radio	IEEE 802.15.4	IEEE 802.15.4
Memory	48KB Code, 10KB Data	32MB Shared
Price	$99	$299

Table 1. WSN nodes vary widely in computational resources.

4.3 Middleware Modularity

WSNs are becoming extremely diverse with resources that differ by several orders of magnitude [53, 16]. To accommodate the wide range of resource availability, Servilla's middleware is modularized and configurable such that a node does not need to implement every module to participate in the network. For example, the middleware can be configured in the following ways:

- **VM + SPF:** The full Servilla framework.
- **VM + SPF-Consumer:** Executes tasks and provides access to remote services only.
- **SPF-Provider:** Provides services for neighboring tasks to use.

A detailed analysis of the memory consumed by each configuration is given in Section 5.1. The configuration containing only the SPF-Provider is particularly interesting because it allows resource-weak but energy efficient nodes to provide services to more powerful nodes. This can result in greater overall energy efficiency and, assuming the weak nodes are less costly and more numerous, increase sensing density while achieving greater sensing coverage.

It is important that, while Servilla allows different configurations to support heterogeneous platforms, the different configurations are *transparent* to applications tasks due to the decoupled nature of the SOC model. For example, a task need not know whether there is a local SPF-Provider. If a task requires a service, it will be coincidentally be bound to a remote provider.

4.4 Implementation

Servilla has been implemented on TinyOS 1.0 and two representative hardware platforms shown in Table 1. It is divided into two levels as shown in Figure 6: a lower level consisting of shared components and a higher level consisting of Servilla's VM and SPF. This section first discusses the lower level followed by the upper level. It ends with a discussion of Servilla's programming languages.

The shared components implement low-level mechanisms needed by most high-level components. The dynamic memory manager makes more efficient use of memory. This is important because Servilla has several components that require varying amounts of memory over time. The dynamic memory manager provides just enough memory for each higher-level component to complete their function and reclaims the memory when it is no longer needed. It is shared by most components in Servilla's middleware, maximizing the flexibility of memory allocation. To aid in debugging, Servilla provides

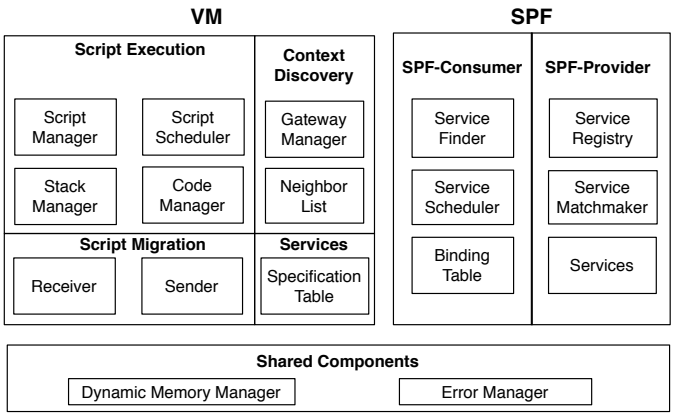


Fig. 6. Servilla's middleware components.

an error manager that detects and sends summaries of problems to the base station. The error manager is shared by all other components in Servilla's middleware.

The SPF is implemented natively using NesC and is divided into two modules, the SPF-Consumer and SPF-Provider, as shown in Figure 6. In the SPF-Consumer, the implementation of the Service Scheduler is simplified by serializing service invocations. This has the added benefit of avoiding saturating the wireless channel. To increase energy efficiency, the Service Finder first searches the local Service Repository, if one is available, before searching one-hop neighbors. In the SPF-Provider, the Service Registry is able to support up to 256 local services.

Servilla's compiler can compile ServillaScript and ServillaSpec into a compact binary format. For example, the task shown in Figure 3 is compiled into 181 bytes of code and 30 bytes of specifications, and the specification shown in Figure 2 is compiled into just 64 bytes. Both the Servilla middleware and compiler have been released as open-source software at `http://mobilab.wustl.edu/projects/servilla/`.

5 Evaluation

This section presents empirical measurement of the code size and performance overhead of Servilla on both the TelosB [53] and Imote2 [16] platforms. The efficacy of the Servilla programming model is demonstrated through an application case study in the next section.

5.1 Memory Footprint

An Imote2 has sufficient memory (32MB) to hold the entire Servilla middleware. Compiled for the Imote2, the total size of the middleware without services is a mere 318KB. This is only about 1% of the total, leaving plenty for services. In contrast, TelosB nodes only have 48KB of code memory. While TelosB does not have enough memory to hold all the components of Servilla, it can support the SPF-consumer configuration which

Node	CPU Speed	Bus Speed	Sig.	Attr. 1	Attr. 2	Attr. 3	Other	Total	Units
TelosB	8MHz	8MHz	18	14	24	29	8	92	ms
Imote2	13MHz	13MHz	1569	1421	2642	3272	784	9688	μs
Imote2	104MHz	104MHz	198	180	330	408	94	1209	μs
Imote2	208MHz	208MHz	99	89	165	204	47	604	μs
Imote2	416MHz	208MHz	71	62	113	136	31	413	μs

Table 2. Service matching latency when comparing two FFT-real service specifications

only consumes 32 KB of code memory. This capability allows TelosB to join and contribute to a WSN as service providers to more powerful nodes. As shown in previous work [26] and our case study presented in Section 6, effective integration of resource-constrained nodes and more powerful nodes can combine the advantages of pervasive low-power sensing and computational resources and enhance energy efficiency. This example shows how Servilla's modular architecture enables it to support diverse hardware platforms.

5.2 Efficiency of Service Binding

Service binding consists of three parts: discovery, matching, and selection. This study first focuses on discovery followed by matching and selection. Recall that the current implementation requires the Service Finder to query each neighbor individually for a match. This is because Servilla uses a reliable network interface that does not support wireless broadcasts. To optimize the selection, the Service Finder first searches locally before remotely. Since the latency of a local search is negligible, we evaluate the latency of a remote search.

The latency of a remote search depends on the number of neighbors, the percentage of them that provide a matching service, and the order in which they are queried. At a minimum, one neighbor will be queried. Since each additional query will proportionately increase the latency, this study evaluates only a single query. An Imote2 is used to query a TelosB to determine whether the TelosB provides a particular service. In this case, the service being queried is `FFT` and the specification is shown in Figure 2. It is compiled into 64 bytes, which must be sent from the Imote2 to the TelosB. Due to various bookkeeping variables, the size of the query message is 72 bytes, and the reply message is 16 bytes. The time between sending the query to receiving a reply is measured by toggling a general I/O pin before and after the query, and capturing the time between toggles using an oscilloscope. Averaged over 100 trials, the latency and 90% confidence interval is $245.6 \pm 1ms$. This latency is acceptable to many WSN applications. Moreover, it may be amortized over multiple invocations of the same service after it is bound to the task.

To evaluate the efficiency of service matching, the Matchmaker is used to compare two copies of `FFT`, shown in Figure 2. This incurs the worst-case latency since every property within the specification must be compared. Each experiment is repeated twenty times on both TelosB and Imote2 platforms running at all possible CPU speeds

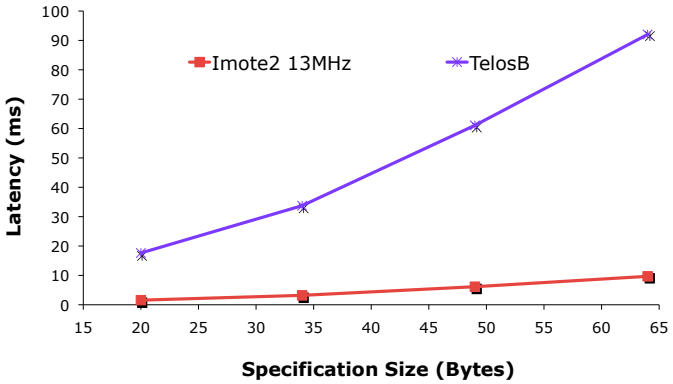


Fig. 7. The latency of comparing a specification vs. its size.

and the average latency is calculated. The results are shown in Table 2.[1] They indicate that the TelosB takes about $92ms$ to perform a match, while the Imote2 is at least ten times faster. The latencies are small compared to the execution times of certain VM instructions. Note that while service matchmaking does introduce overhead, it is usually done infrequently relative to service invocation.

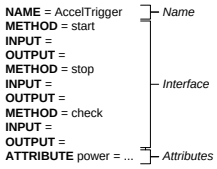
To determine how the specification's size affects matching latency, `FFT` is compared to versions of itself with one, two, and all three of its attributes removed. The matching latencies is plotted against their sizes and the results are shown in Figure 7. For brevity, only the Imote2 running at 13MHz is shown. The latency when the Imote2 is running at higher frequencies is significantly lower. The results indicate that the latency is roughly proportional to its size. It is not exactly proportional because of the additional overhead incurred with the addition of each attribute, as indicated by the "other" column in Table 2.

6 Application Case Study

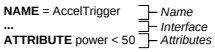
This section evaluates Servilla using an application case study, specifically one designed to localize damage in structures (e.g., a bridge). This application enables real-time evaluation of a structure's integrity, reducing manual inspection costs while increasing safety. WSNs have recently been used to successfully localize damage to experimental structures using a homogeneous network of Imote2 nodes [28]. In this case, the algorithm, called Damage Localization Assurance Criterion (DLAC), was written using NesC specifically for the Imote2. The implementation using Servilla generalizes and improves upon the original by making it platform-independent and increasing its overall energy efficiency by exploiting network heterogeneity.

The heterogeneous WSN used in this study consists of TelosB and Imote2 nodes. DLAC can only run on the Imote2 due to insufficient memory on the TelosB. However,

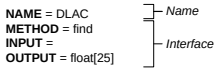
[1] The confidence intervals are negligible since the experiment runs locally and the measurements exhibit very low variance.



(a) The specification of service `AccelTrigger` provided by Imote2 and TelosB nodes. The power attribute specifies the amount of power the service consumes. It is 145mW on the Imote2, and 9mW on the TelosB.



(b) The specification of a low-power version of service `AccelTrigger`, which is provided by the application task. Its interface is omitted since it is the same as the one in Figure 8(a). A high-power version has attribute power $\geq$ 50 mW.



(c) The specification of service `DLAC` provided by Imote2 nodes.

Fig. 8. The services used by the damage localization application

Imote2 nodes consume more energy than TelosB nodes. Our new structural health monitoring application combines the advantages of both platforms by keeping the Imote2 nodes idle as much as possible, and using the TelosB nodes to monitor the ambient vibration levels. The Imote2 nodes are only activated when the TelosB nodes detect that the ambient vibration levels exceed a certain threshold, at which time they perform the DLAC algorithm. The dual-level nature of this configuration is common to other applications such as surveillance [30], and is essential for conserving energy and ensuring network longevity.

The Servilla implementation relies on two services: `AccelTrigger` and `DLAC`. Ambient vibrations are monitored by `AccelTrigger`, which sets a flag when a threshold is exceeded. Its specification is shown in Figure 8(a). The service has three methods: `start`, `stop`, and `check`. Methods `start` and `stop` control when the service monitors the local accelerometer. The status of the flag is obtained by invoking `check`. Both the Imote2 and TelosB nodes provide `AccelTrigger`. They differ in their power attribute, since the Imote2 consumes more power than the TelosB (145mW vs. 9mW).

The specification of service `DLAC` is shown in Figure 8(c). It contains a single method, `find`, that takes no parameters and returns an array of floating-point numbers that are used to localize damage to the bridge [28].

The application's task is shown in Figure 9. The first three lines specify the names of the files containing the required service specifications. The content of `AccelTriggerLP` is shown in Figure 8(b), and the content of `DLAC` is shown in Figure 8(c). Notice that `AccelTriggerLP` matches the TelosB version of the `AccelTrigger` service shown in Figure 8(a) because its power attribute is less than 50mW. `AccelTriggerHP` contains the same specification as `AccelTriggerLP` except its power attribute is ≥ 50 mW, which matches the service provided by the Imote2.

```
1.  uses AccelTiggerHP;
2.  uses AccelTiggerLP;
3.  uses DLAC;
4.
5.  void main() {
6.      bind(DLAC, 0); // bind DLAC service
7.      if(!isBound(DLAC)) exit(); // failed to bind DLAC
8.      bind(AccelTriggerLP, 1); // bind low-power AccelTrigger service
9.      if(isBound(AccelTriggerLP)) {
10.         invoke(AccelTriggerLP, "start");
11.         waitForTrigger(1);
12.     } else {
13.         bind(AccelTriggerHP);
14.         if(isBound(AccelTriggerHP)) {
15.             invoke(AccelTriggerHP, "start");
16.             waitForTrigger(0);
17.         }
18.     }
19. }
20.
21. void waitForTrigger(int useLowPower) {
22.     while(true) {
23.         int vibration;
24.         if (useLowPower)
25.             vibration = invoke(AccelTriggerLP, "check");
26.         else
27.             vibration = invoke(AccelTriggerHP, "check");
28.         if (vibration == 1) {
29.             if (useLowPower)
30.                 invoke(AccelTriggerLP, "stop");
31.             else
32.                 invoke(AccelTriggerHP, "stop");
33.             doDLAC();
34.         }
35.         sleep(1024*60*5); // sleep for 5 minutes
36.     }
37. }
38.
39. void doDLAC() {
40.     float[25] dlac_data;
41.     dlac_data = invoke(DLAC, "find");
42.     send(dlac_data); // send DLAC data to base station
43. }
```

Fig. 9. The damage localization application task

The application attempts to reduce energy consumption by preferentially binding to an `Acceltrigger` service that consumes less power. It does this by first attempting to bind using the specification within `AccelTriggerLP` on line 8, before using the specification within `AccelTriggerHP` on line 13. Once an `AccelTrigger` service is bound, the task periodically queries it to determine if the acceleration readings are above a certain threshold (lines 21-37). If it is, `DLAC` is invoked and the results are sent to the base station (lines 41-42).

To evaluate the benefit of exploiting network heterogeneity on Servilla, the task shown in Figure 9 is injected into two WSNs: a homogeneous network consisting of only Imote2 devices, and a heterogeneous network consisting of both Imote2 and TelosB devices. Since the application is written using Servilla, it is able to run on both types of networks without modification. In both cases, `DLAC` is executed by the Imote2, meaning the power consumption of performing damage localization is constant. However, the power consumption of `AccelTrigger` does vary. This is because Servilla's

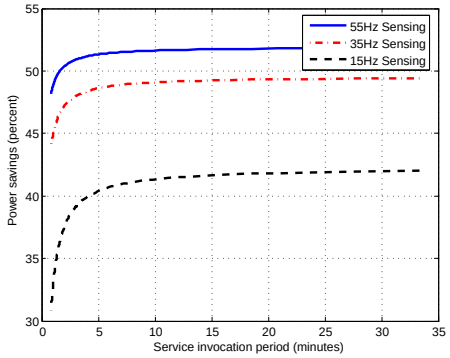


Fig. 10. Percent power savings of heterogeneous vs. homogeneous WSN.

service provisioning framework enables an application to exploit more energy-efficient services when possible in a platform-independent and declarative fashion. Specifically, if TelosB nodes are present, the service will be executed on a TelosB node since its `AccelTrigger` service consumes less power, otherwise it will be executed on the Imote2. We compare the power consumption of invoking `AccelTrigger` in different network configurations.

Since invoking `AccelTrigger` on the TelosB requires a remote invocation, the amount of energy saved depends on the invocation and sensing frequencies. If the service is invoked too often, more energy will be spent on wireless communication. Likewise, if the sensor is accessed very infrequently, the benefits of the TelosB is diminished since the nodes will remain asleep a larger percentage of the time. To determine how much energy savings is possible, an oscilloscope is used to measure the time each platform spends computing, communicating wirelessly, and sensing, in both a homogeneous and heterogeneous network. The sensing frequency is varied between 15Hz and 55Hz (the maximum sampling frequency of the TelosB), and the service invocation frequency is varied between 50 seconds to 35 minutes. The percent savings of using a heterogeneous network relative to a homogeneous network is then calculated and the results are shown in Figure 10.

The results show that the heterogeneous implementation using Servilla achieves as much as 52% power savings, and the savings increase with sensing frequency. The results also show that invoking the service too frequently will reduce the amount of power saved since doing so incurs more network overhead. There is a limit to the amount of energy that can be saved as the service invocation period increases since it approaches the difference between the sensing energy consumed by the Imote2 versus the TelosB.

This case study demonstrates how Servilla enables the development of platform-independent applications that operate over a heterogeneous WSN, and how Servilla facilitates in-network collaboration between different types of nodes to attain higher energy efficiency. Moreover, it demonstrates that Servilla enables an application to bind to a more energy-efficient service through service specification.

7 Related Work

SOC has long been used on the Internet to enable independently-developed applications to interoperate. There are many SOC systems including SLP [35], Jini [36], CORBA [51], Salutation [14], and Web Services [2]. They provide technologies that enable language-independent communication, which is essential for interoperability. Some of these include SOAP [20], RPC [19], DCOM [20], and WCF [45]. Servilla has three salient features that distinguish it from general-purpose SOC frameworks. First, it focuses on how service-provisioning language and middleware can be made lightweight. This is necessary due to the limited resources available on some WSN nodes. Second, Servilla is specifically designed for localized service binding which is a common coordination model for WSNs. Moreover, Servilla provides a modular middleware architecture that can be configured for diverse sensor network platforms with an extremely wide range of resources.

SOC is a topic of interest in the coordination community. For example, new languages have been developed that enable formal reasoning about complex service interactions and compositions [8, 1, 43, 3]. Calculi have been developed to model sessions and multi-party dynamic interactions between service users and providers [44, 13]. New ways of specifying quality-of-service requirements and achieving higher levels of reliability have been proposed [4, 10, 12, 49, 9]. SOC has even been used in non-traditional environments like mobile ad hoc networks [29]. Recently, there has been increased interest in context-aware applications [50, 24, 17]. WSNs, being embedded and able to sense the environment, are inherently context-aware. This paper takes the natural next step of applying SOC principles to WSNs. The key distinguishing feature of Servilla lies in its capability to support both resource-constrained nodes and more powerful nodes, and its light-weight language and middleware tailored for in-network coordination among sensors.

There are some efforts to port traditional SOC technologies into the WSN domain. They include Tiny Web Services [54] and Arch Rock's PhyNet™ [6]. Both optimize traditional Internet protocols to function under the severe resource constraints of WSNs. Unlike Servilla, they do not provide a mechanism for service discovery or the flexible matching between service users and consumers *within* the WSN. Instead, they focus on how to enable language-independent communication between services inside the WSN and applications outside of the WSN. Servilla is complementary to these efforts; Servilla may leverage off of these systems to expose WSN services to applications external to the WSN, while these systems may rely on Servilla to bring the full capabilities of SOC inside the WSN itself.

In addition to SOC, Servilla shares the common approach of using scripts in a WSN, though for different reasons. Some scripting systems, including Maté [39], ASVM [40], SwissQM [48], and Agilla [23], enable reprogramming. Other systems, including Melete [58] and SensorWare [11], enable multiple applications to share a WSN. All of these systems come with different scripting languages [38, 21, 27, 42, 57]. Servilla differs by focusing on challenges due to network heterogeneity and dynamics. Unlike other systems, Servilla allows scripts to remain platform-independent and dynamically find and access platform-specific services. One scripting system, DVM [7], explores the similar idea of integrating platform-independent scripts with native services. It features a dynamically

extensible virtual machine in which services can register extensions. While this enables tuning the boundary between interpreted and native code, DVM does not support flexible matching between scripts and services.

Servilla introduces the idea of a modular and configurable platform in which extremely resource-poor nodes only implement a fraction of the entire framework. This enables a hierarchy in which weak nodes serve more powerful nodes. The idea of having a hierarchy within a WSN is not new. Tenet [26] promotes this idea by creating a two-tired WSN in which the lower tier consists of resource-poor nodes that can accept tasks from higher-tier nodes. It differs from Servilla in that it does not support SOC which enables flexible discovery and binding between different nodes. SONGS [41] is an architecture for WSNs that allows users to issue queries that are automatically decomposed into graphs of services which are mapped onto actual nodes. SONG does not provide flexible service binding among heterogeneous nodes.

8 Conclusions

The increasing difficulty of developing applications for heterogeneous and dynamic WSNs demands a new coordination model. Servilla provides this by introducing a novel service provisioning model that enables application components to communicate in a dynamic network, while service provisioning enables applications to be platform-independent while still able to access platform-specific services. A specialized service description language is introduced that enables flexible matching between applications and services, which may reside on different nodes. Servilla provides a modular middleware architecture to enable resource-poor nodes to contribute services, facilitating in-network collaboration among a wide range of devices. The efficiency of Servilla's implementation is established via microbenchmarks on two representative classes of hardware platforms. The effectiveness of Servilla's programming model is demonstrated by a structural health monitoring application case study.

Acknowledgment

This work is funded by the National Science Foundation under grants CNS-0520220, CNS-0627126, and CNS-0708460.

References

1. ABREU, J., AND FIADEIRO, J. L. A coordination model for service-oriented interactions. In Lea and Zavattaro [37], pp. 1–16.
2. ALONSO, G., CASATI, F., KUNO, H., AND MACHIRAJU, V. *Web Services*. Springer, 2003.
3. ANKOLEKAR, A., HUCH, F., AND SYCARA, K. P. Concurrent semantics for the web services specification language daml-s. In Arbab and Talcott [5], pp. 14–21.
4. ARBAB, F., CHOTHIA, T., MENG, S., AND MOON, Y.-J. Component connectors with qos guarantees. In Murphy and Vitek [46], pp. 286–304.
5. ARBAB, F., AND TALCOTT, C. L., Eds. *Coordination Models and Languages, 5th International Conference, COORDINATION 2002, YORK, UK, April 8-11, 2002, Proceedings* (2002), vol. 2315 of *Lecture Notes in Computer Science*, Springer.

6. ARCH ROCK. Arch Rock PhyNet™. http://www.archrock.com/product/.
7. BALANI, R., HAN, C.-C., RENGASWAMY, R. K., TSIGKOGIANNIS, I., AND SRIVASTAVA, M. Multi-level software reconfiguration for sensor networks. In *EMSOFT'06* (New York, NY, USA, 2006), ACM Press, pp. 112–121.
8. BETTINI, L., NICOLA, R. D., AND LORETI, M. Implementing session centered calculi. In Lea and Zavattaro [37], pp. 17–32.
9. BOCCHI, L., CIANCARINI, P., AND ROSSI, D. Transactional aspects in semantic based discovery of services. In Jacquet and Picco [34], pp. 283–297.
10. BOCCHI, L., AND LUCCHI, R. Atomic commit and negotiation in service oriented computing. In Ciancarini and Wiklicky [15], pp. 16–27.
11. BOULIS, A., HAN, C.-C., AND SRIVASTAVA, M. Design and implementation of a framework for efficient and programmable sensor networks. In *MobiSys'03* (May 2003), USENIX, pp. 187–200.
12. BRAVETTI, M., AND ZAVATTARO, G. A theory for strong service compliance. In Murphy and Vitek [46], pp. 96–112.
13. BRUNI, R., LANESE, I., MELGRATTI, H. C., AND TUOSTO, E. Multiparty sessions in soc. In Lea and Zavattaro [37], pp. 67–82.
14. CHAKRABORTY, D., AND CHEN, H. Service discovery in the future for mobile commerce. *Crossroads 7*, 2 (2000), 18–24.
15. CIANCARINI, P., AND WIKLICKY, H., Eds. *Coordination Models and Languages, 8th International Conference, COORDINATION 2006, Bologna, Italy, June 14-16, 2006, Proceedings* (2006), vol. 4038 of *Lecture Notes in Computer Science*, Springer.
16. CROSSBOW TECHNOLOGIES. Imote2 datasheet. http://tinyurl.com/5jrw85.
17. CUBO, J., SALAÜN, G., CÁMARA, J., CANAL, C., AND PIMENTEL, E. Context-based adaptation of component behavioural interfaces. In Murphy and Vitek [46], pp. 305–323.
18. CULLER, D., ESTRIN, D., AND SRIVASTAVA, M. Overview of sensor networks. *IEEE Computer 37*, 8 (2004), 41–49.
19. DAVE MARSHALL. Remote procedure calls (rpc). http://www.cs.cf.ac.uk/Dave/C/node33.html.
20. DAVIS, A., AND ZHANG, D. A comparative study of soap and dcom. *J. Syst. Softw. 76*, 2 (2005), 157–169.
21. DUNKELS, A. A low-overhead script language for tiny networked embedded systems. Tech. Rep. T2006:15, Swedish Institute of Computer Science, Sept. 2006.
22. FLANAGAN, D. *JavaScript: The Definitive Guide, 4th Ed.* O'REILLY, Inc., 2001.
23. FOK, C.-L., ROMAN, G.-C., AND LU, C. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *ICDCS'05* (June 2005), IEEE, pp. 653–662.
24. FREY, D., AND ROMAN, G.-C. Context-aware publish subscribe in mobile ad hoc networks. In Murphy and Vitek [46], pp. 37–55.
25. GAY, D., LEVIS, P., VON BEHREN, R., WELSH, M., BREWER, E., AND CULLER, D. The nesc language: A holistic approach to networked embedded systems. In *PLDI'03* (New York, NY, USA, 2003), ACM, pp. 1–11.
26. GNAWALI, O., JANG, K.-Y., PAEK, J., VIEIRA, M., GOVINDAN, R., GREENSTEIN, B., JOKI, A., ESTRIN, D., AND KOHLER, E. The tenet architecture for tiered sensor networks. In *SenSys'06* (New York, NY, USA, 2006), ACM Press, pp. 153–166.
27. GREENSTEIN, B., KOHLER, E., AND ESTRIN, D. A sensor network application construction kit (snack). In *SenSys'04* (New York, NY, USA, 2004), ACM, pp. 69–80.
28. HACKMANN, G., SUN, F., CASTANEDA, N., LU, C., AND DYKE, S. A holistic approach to decentralized structural damage localization using wireless sensor networks. In *RTSS'08* (11 2008), IEEE.

29. HANDOREAN, R., AND ROMAN, G.-C. Service provision in ad hoc networks. In Arbab and Talcott [5], pp. 207–219.
30. HE, T., KRISHNAMURTHY, S., LUO, L., YAN, T., GU, L., STOLERU, R., ZHOU, G., CAO, Q., VICAIRE, P., STANKOVIC, J. A., ABDELZAHER, T. F., HUI, J., AND KROGH, B. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Trans. Sen. Netw. 2*, 1 (2006), 1–38.
31. HE, T., KRISHNAMURTHY, S., STANKOVIC, J. A., ABDELZAHER, T., LUO, L., STOLERU, R., YAN, T., GU, L., ZHOU, G., HUI, J., AND KROGH, B. Vigilnet:an integrated sensor network system for energy-efficient surveillance. *ACM Transactions on Sensor Networks (under submission)* (2004).
32. HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., AND PISTER, K. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems* (2000), pp. 93–104.
33. INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking* (New York, NY, USA, 2000), ACM, pp. 56–67.
34. JACQUET, J.-M., AND PICCO, G. P., Eds. *Coordination Models and Languages, 7th International Conference, COORDINATION 2005, Namur, Belgium, April 20-23, 2005, Proceedings* (2005), vol. 3454 of *Lecture Notes in Computer Science*, Springer.
35. KEMPF, J., AND PIERRE, P. S. *Service location protocol for enterprise networks: implementing and deploying a dynamic service finder*. John Wiley & Sons, Inc., New York, NY, USA, 1999.
36. KUMARAN, I., AND KUMARAN, S. I. *Jini Technology: An Overview*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
37. LEA, D., AND ZAVATTARO, G., Eds. *Coordination Models and Languages, 10th International Conference, COORDINATION 2008, Oslo, Norway, June 4-6, 2008. Proceedings* (2008), vol. 5052 of *Lecture Notes in Computer Science*, Springer.
38. LEVIS, P. The TinyScript Manual. http://tinyurl.com/57kycj, July 2004.
39. LEVIS, P., AND CULLER, D. Maté: a tiny virtual machine for sensor networks. In *ASPLOS'02* (New York, NY, USA, 2002), ACM Press, pp. 85–95.
40. LEVIS, P., GAY, D., AND CULLER, D. Active sensor networks. In *NSDI'05* (May 2005).
41. LIU, J., AND ZHAO, F. Towards semantic services for sensor-rich information systems. In *2nd Int. Conf. on Broadband Networks* (2005), pp. 44–51.
42. MADDEN, S., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev. 36*, SI (2002), 131–146.
43. MAZZARA, M., AND GOVONI, S. A case study of web services orchestration. In Jacquet and Picco [34], pp. 1–16.
44. MEZZINA, L. G. How to infer finite session types in a calculus of services and sessions. In Lea and Zavattaro [37], pp. 216–231.
45. MICROSOFT. Windows communication foundation. `http://msdn2.microsoft.com/en-us/library/ms735119.aspx`.
46. MURPHY, A. L., AND VITEK, J., Eds. *Coordination Models and Languages, 9th International Conference, COORDINATION 2007, Paphos, Cyprus, June 6-8, 2007, Proceedings* (2007), vol. 4467 of *Lecture Notes in Computer Science*, Springer.
47. MURTY, R., GOSAIN, A., TIERNEY, M., BRODY, A., FAHAD, A., BERS, J., AND WELSH, M. Citysense: A vision for an urban-scale wireless networking testbed. Tech. Rep. 13-07, Harvard University, 2007.
48. MLLER, R., ALONSO, G., AND KOSSMANN, D. A virtual machine for sensor networks. In *EuroSys 2007* (March 2007).

49. NORES, M. L., DUQUE, J. G., AND ARIAS, J. J. P. Managing ad-hoc networks through the formal specification of service requirements. In Ciancarini and Wiklicky [15], pp. 164–178.
50. NÚÑEZ, A., AND NOYÉ, J. An event-based coordination model for context-aware applications. In Lea and Zavattaro [37], pp. 232–248.
51. OBJECT MANAGEMENT GROUP. Corba basics. `http://www.omg.org/gettingstarted/corbafaq.htm`.
52. PAPAZOGLOU, M. P., TRAVERSO, P., DUSTDAR, S., AND LEYMANN, F. Service-oriented computing: State of the art and research challenges. *Computer 40*, 11 (2007), 38–45.
53. POLASTRE, J., SZEWCZYK, R., AND CULLER, D. Telos: enabling ultra-low power wireless research. In *IPSN'05* (Piscataway, NJ, USA, 2005), IEEE Press, p. 48.
54. PRIYANTHA, N., KANSAL, A., GORACZKO, M., AND ZHAO, F. Design and implementation of an evolutionary sensor network. In *SenSys'08* (New York, NY, USA, 2008), ACM.
55. STREELINE. Parking management. `http://www.streetlinenetworks.com`.
56. W3C. Web services description language (wsdl). `http://www.w3.org/TR/wsdl`.
57. YAO, Y., AND GEHRKE, J. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec. 31*, 3 (2002), 9–18.
58. YU, Y., RITTLE, L. J., BHANDARI, V., AND LEBRUN, J. B. Supporting concurrent applications in wireless sensor networks. In *SenSys'06* (New York, NY, USA, 2006), ACM Press, pp. 139–152.