TRANSIENTLY SHARED TUPLE SPACES FOR SENSOR NETWORKS[*]

Amy L. Murphy
University of Lugano, via G. Buffi 13, 6900 Lugano, Switzerland
amy.murphy@unisi.ch

Gian Pietro Picco
Politecnico di Milano, P.za L. da Vinci 32, 20129 Milano, Italy
picco@elet.polimi.it

Abstract In this paper we argue that the notion of *transiently shared tuple space*, originally introduced by the LIME model and middleware to support application development in mobile ad hoc networks, can be successfully applied also to wireless sensor networks (WSNs). While the two scenarios are similar, the peculiar constraints posed by the WSNs (e.g., in terms of resources and energy) require non-trivial adaptations. Here, we describe two models and systems, TinyLIME and TeenyLIME, providing transiently shared tuple spaces in two different WSN operational scenarios, and elaborate on alternate designs and opportunities.

Keywords: Tuple spaces, middleware, wireless sensor networks.

1. Introduction

Wireless sensor networks (WSNs) have emerged as a novel and rapidly evolving field, enabled by continuous technological advancements. However, the progress in hardware and communication technology has not been paralleled by breakthroughs in the programming support available to application developers. As a consequence, WSN applications are typically developed from scratch directly on the operating system layer. The

[*]The work described in this paper is partially supported by the European Union under the IST-004536 RUNES project and by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

consequence is the lack of reusable and reliable solutions and, ultimately, an unnecessary burden on the programmer.

This gap between the applications and the underlying system resources is typically filled by *middleware*. While the concept has become popular in the context of mainstream distributed computing, it is even more important in dynamic scenarios such as those defined by mobility. Here, the lack of proper abstractions leaves the programmer alone with the daunting task of dealing with the dynamic context defined by the changing physical environment, by the fluid network topology and, ultimately, by the changing set of application services available.

In our previous research, we defined a model and middleware, Lime [9], which tackles the aforementioned challenges by relying on the *tuple space* abstraction previously introduced by Linda in the context of parallel programming [6]. Tuple spaces provide a data sharing abstraction that simplifies the definition of coordination among independent, remote processes. In Lime, this notion is adapted towards mobile ad hoc networks (MANETs) by introducing *transiently shared tuple spaces* whose contents are dynamically redefined based on connectivity. Moreover, the model is complemented by reactive operations that integrate the data-centric tuple space functionality with event-centric capabilities in a unified programming abstraction.

MANETs and WSNs share many characteristics including wireless communication, a dynamic network topology, and in general the need to deal with a changing physical and computational context. At the same time, there are also significant differences, most notably in terms of the computational and energy constraints present in WSNs, which make a direct port of existing mobile approaches infeasible. It is therefore reasonable to ask whether the flexible and expressive tuple space abstraction, and in particular its adaptation to mobility provided in Lime, can serve as a stepping stone towards new models supporting application development for WSNs.

In this paper we argue that the answer to this question is positive. Indeed, we contend that the Lime notion of *transiently shared tuple space* and its combination with reactive programming already provide the fundamental concepts necessary for programming WSN applications. These arguments are concretely exemplified by illustrating variations of Lime, both in terms of its model and of the corresponding middleware support, that we devised to target different WSN scenarios.

Our first adaptation of Lime was motivated by the need to support an operational scenario where mobile operators collect data from the sensors in their immediate vicinity [3, 2]. In this scenario, the application logic resides on the (mobile) sinks—a common choice in WSN applications.

Sensors are passive data producers, and at most perform local computation on behalf of the sinks. Nevertheless, the programmer is shielded from many of these details by the TinyLIME API, which presents a unified abstract view of the system where all the nodes available (sinks and sensors) share their data through the tuple space.

Recent developments in WSNs are pushing scenarios where the application intelligence is no longer relegated to the fringes of the system (i.e., on a data sink running on a powerful node) rather it is distributed within the WSN itself. One such scenario involves both sensors and actuators, with actuators basing their decisions on the sensors around them, yielding a more efficient feedback loop w.r.t. architectures where data is funneled towards a single sink [1]. To support this scenario, we devised another adaptation of the LIME model, called TeenyLIME, where we assumed that devices are capable of independent computation. (Hereafter, we use the term *device* to encompass sensor and/or actuator nodes.) This time, the application code is not confined to the powerful sinks, rather it is deployed on the devices. Further, tuple space operations are no longer used only for data collection, rather they are exploited for coordination of the devices themselves. In a sense, TeenyLIME is literally a "port" of the LIME model onto the sensor and actuator devices, while instead TinyLIME bridges the MANET and WSN environments by means of the conceptual and programming glue provided by transiently shared tuple spaces.

This paper is organized in a progression from Linda to TeenyLIME. Each evolutionary step of the tuple space model is motivated by the challenge posed by a new operational setting, in which the lessons learned at the previous step are exploited in the reformulation of the model. The basic Linda model is described in Section 1.2, together with the adaptation to the MANET environment provided by LIME. In Section 1.3 and 1.4 we illustrate how we further extended LIME to suit the needs of the aforementioned WSN operational scenarios. Section 1.5 summarizes the findings, by highlighting analogies and relationships among the various models, identifying opportunities for integration and alternative designs, and comparing our work against the literature. Finally, Section 1.6 ends the paper with brief concluding remarks.

2. Background: Linda and LIME

In this section we provide a concise introduction to the notion of tuple space made popular by Linda, and to its adaptation to the mobile environment put forth by LIME.

Linda and Tuple Spaces. Linda [6] is a shared memory model where the data is represented by elementary data structures called *tuples* and the memory is a multiset of tuples called a *tuple space*. Each tuple is a sequence of typed fields, such as ⟨"foo", 9, 27.5⟩ and coordination among processes occurs through the writing and reading of tuples. Conceptually all processes have a handle to the tuple space and can add tuples by performing an **out**(t) operation and read and remove tuples using **rd**(p) and **in**(p) which specify a pattern, p, for the desired data. The pattern is a tuple whose fields contain either *actuals* or *formals*. Actuals are values; the fields of the previous tuple are all actuals, while the last two fields of ⟨"foo", ?integer, ?float⟩ are formals. Formals act like "wild cards", and are matched against actuals when selecting a tuple from the tuple space. For instance, the template above matches the tuple defined earlier. If multiple tuples match a template, the one returned by **in** or **rd** is selected non-deterministically.

Both **in** and **rd** are blocking, i.e., if no matching tuple is available in the tuple space the process performing the operation is suspended until a matching tuple appears. Typical extensions include a pair of primitives **inp** and **rdp**, which return **null** if no matching tuple exists in the tuple space and *bulk operations* **ing** and **rdg**, which can be used to retrieve all matching tuples at once.

Lime**: Linda in a Mobile Environment.** To support mobility, the Lime [9] model breaks up the Linda tuple space into multiple tuple spaces each permanently attached to a mobile component, and defines rules for the sharing of their content when components are able to communicate. In a sense, the static global tuple space of Linda is reshaped by Lime into one that dynamically changes according to connectivity. As shown in Figure 1, the Lime model encompasses mobile software agents and physical mobile hosts. Agents are permanently assigned an interface tuple space, ITS, which is brought along during migration. Co-located agents are considered connected. The union of all the tuple spaces, based on connectivity, yields a dynamically changing *federated tuple space*. Hereafter, we consider only stationary agents.

Access to the federated tuple space remains very similar to Linda, with each agent issuing Linda operations on its own ITS. The semantics of the operations, however, is as if they were executed over a single tuple space containing the tuples of all connected components.

Besides transient sharing, Lime adds two new notions to Linda: tuple locations and reactions. Although tuples are accessible to all connected agents, they only exist at a single point in the system, with one of the agents. When a tuple is output, it remains in the ITS of the outputting agent. Lime also allows tuples to be shipped to another agent by extending the **out** operation to include a destination. The notion of location is also used to restrict the scope of the **rd** and **in** operations, effectively issuing the operation only over the portion of the federated tuple space owned by a given agent or residing on a given host.

Reactions allow an agent to register a code fragment (a listener) to be executed asynchronously whenever a tuple matching a particular pattern is found anywhere in the federated tuple space. This feature is very useful in the highly dynamic mobile environment, where the set of connected components changes frequently. Reactions can also be restricted in scope to a particular host or agent, like queries. This ability to monitor changes across the whole system by installing reactions on the federated tuple space has been shown to be one of the most useful features of Lime, as it frees the programmer from the burden of explicitly monitoring the system configuration.

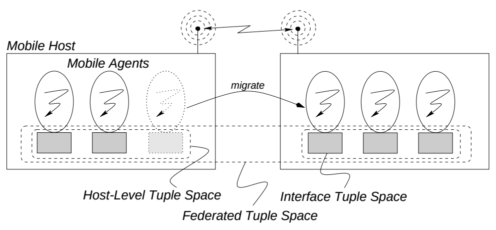

Figure 1. In Lime connected mobile hosts transiently share the tuple spaces of the agents executing on them.

3. TinyLime

TinyLime is the first adaptation of Lime's ideas to the WSN environment. As described next, its goal is essentially to extend the span of the transiently shared tuple space coordinating a group of mobile sinks to include also the data gathered by nearby sensors.

Scenario. In contrast to the typical sensor network architectures that collect data at a single centralized location, TinyLime takes an alternative approach that naturally provides contextual access, does not require multi-hop communication among sensors, and still places reasonable computation and communication demands on the sensors. The target scenario, depicted in Figure 2, assumes that sensors are distributed sparsely throughout a region and need not be able to communicate with one another. The application is deployed on a set of mobile hosts, interconnected through ad hoc wireless links—e.g., 802.11. Some hosts, e.g., the PDA, are only clients, without direct access to sensors. The others are equipped with base station hardware providing access to sensors within one hop.

Model. As in Lime, the core abstraction of TinyLime is that of a transiently shared tuple space, which in our case stores tuples containing the sensed data. However, TinyLime introduces a new component in addition to agents and hosts—the *sensor*. In the TinyLime model, a device in direct communication with some base station is represented much like an agent residing on the base station, with an associated ITS containing the set of data provided by its sensors. Each TinyLime sensor may be a device providing several types of data (e.g., temperature and light), all stored in the sensor ITS. Looking at Figure 1, it is as if on each host there were an additional agent for each device currently in range of that host. Clearly, things are quite different in practice: the sensor device is not physically on the base station, and there is no ITS deployed on it. As usual, it is the middleware that takes care of creating this abstraction to simplify the programmer's task.

The rest of the model follows naturally. For instance, operations on the federated space now span not only connected hosts and agents, but also the sensors within range of some host. Similarly to Lime, operations can be restricted to a given host. Also, the sensor identifier can be used to restrict the scope of a query or reaction to a specific sensor. For instance, in Figure 2, a client on the laptop can request a light reading from one specific sensor, from any sensor one hop from itself, or from any sensor one hop from any connected client. Clients and base stations are prohibited from modifying the data on the sensors. In other words, only reactions, **rd**, **rdp**, and **rdg** are available.

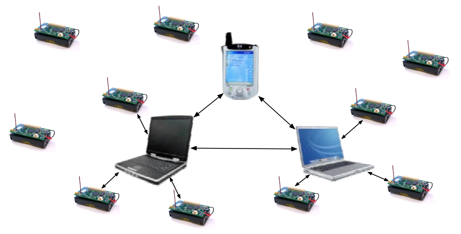

Figure 2. TinyLime operational scenario: communication occurs between base stations (laptops) and sensors, and between base stations and clients (PDAs). Clients and base stations can also coincide.

Reactions work as in Lime, modulo the changes above, and are extremely useful in this environment. Consider a situation in which a single base station agent registers a reaction to display temperature values. As the base station moves across the region, the temperature from each sensor that comes into range is automatically displayed: the programmer is spared the effort of explicitly monitoring the changing context.

Implementation. The first implementation challenge was enabling communication between base stations and sensors. Energy restrictions require sensors to sleep a majority of the time, waking up on a regular basis to listen for incoming messages. Sensors cannot receive messages while sleeping and base stations cannot keep track of the wakeup times of the sensors. However, by quantizing time in *epochs*, and ensuring each sensor is awake for a predetermined period every epoch, communication in TinyLime is enabled by having the base station repeatedly send a message until a sensor receives it and responds. With no communication errors, the delay between the beginning of transmission and the response is at most one epoch. This places the communication burden on the base stations, which have a larger energy reserve and are more easily rechargeable than the sensors.

Another design decision arose from the realization that sensor data remains valid for a period of time, after which it is no longer *fresh.* Based on this, we introduced data caching of fresh data on the base station, allowing future requests for the same data to be served directly from the cache without requiring additional, energy-consuming communication. The details can be found in [2], but here it is sufficient to note that every arriving sensor value is associated with a timestamp upon arrival at the base station. When the data is no longer fresh, it is removed and future requests must again query the sensors directly.

Another major design issue was the provision of aggregation features. WSN applications often do not simply gather raw data, rather they collect and transform it, e.g., aggregating values to find the average temperature over a time interval to reduce the impact of spurious readings. System-wide aggregation can be naturally provided at the application level using the group operation **rdg** to collect the data gathered by each base station. However, for efficiency aggregation should be pushed close to the data (i.e., on sensors), trading computation for communication. This goal requires a slight change of perspective. In our original implementation of TinyLime [3], sensors played only a *passive* role, acquiring and communicating data when requested. Instead, aggregation requires sensors to be *active*, sampling and recording data over which aggregation can later be performed. Sampling comes at the cost of both storage and computational activity to record data. In our current implementation [2], TinyLime supports aggregation by allowing the programmer to dynamically activate sampling on some or all sensors for a given number of epochs, after which sensors switch back to passive mode. Built-in aggregation modules are provided, along with mechanisms enabling programmers to supply their own. Interestingly, the same feature can be used to support any kind of active behavior on sensors, e.g., automatic periodic data reporting. This idea of generalizing the active behavior of sensors is brought to an extreme by TeenyLime, described next.

4. TeenyLime

Although TeenyLime followed TinyLime in the evolution towards WSNs, it partially represents a return to the original Lime model. The key idea behind TeenyLime is to treat the WSN devices as active components performing distributed coordination and data access.

Scenario. In contrast with TinyLime, in the target scenario of TeenyLime there is no distinction between powerful, mobile base stations and WSN devices: only the latter belong to the system. However, the devices can be heterogeneous, e.g., containing not only sensors but also actuators, as shown in Figure 3. Devices coordinate to perform the functionality of the WSN application, which is directly deployed on them, rather than on some external sink. Moreover, in TeenyLime, a device is expected to communicate directly only with neighbors in communication range.

Example applications are those where actuators collect information from neighboring sensors and perform some action based on the value of the data [1], e.g., alerting other actuators in range. For instance, a fire extinguisher can make a local decision to activate based only on readings from several sensors in their vicinity, and inform other nearby extinguishers. Alternately, the TeenyLime model can be used as a building block for different data collection strategies, e.g., where an aggregated value is computed among neighboring sources, and then made available locally or propagated to the rest of the network or to an external source with appropriate protocols.

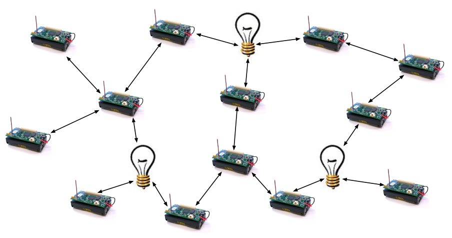
Figure 3. TeenyLime operational scenario: one hop communication among sensors (motes) and actuators (light bulbs).

Model. As in Lime and TinyLime, the core abstraction of TeenyLime is the transiently shared tuple space. However, unlike TinyLime the spaces are physically located on the devices themselves, and unlike Lime they are shared only one hop. Because each device has a different set of one hop neighbors, the shared tuple space view is different for each device. This is fundamentally in contrast to Lime in which the view of the transiently shared tuple space is composed of the tuple spaces of all connected hosts, where connectivity is assumed transitive.

Operations in TeenyLime are essentially those of Lime. Also, as opposed to TinyLime, operations to remove tuples (e.g., **inp**) are supported. Indeed, tuples no longer represent read-only data sensed by remote sensors, rather it represents application data under the direct responsibility of the devices, which therefore must retain full control over it. Reactions function as in LIME, but with a limited one hop range. This does not undermine their usefulness, as the ability to react to specific data in the neighborhood is of paramount importance. For example, the fire extinguisher example can be implemented simply by installing a reaction on nearby sensors for temperature readings. When sufficiently many high readings are received by the extinguisher, it should be activated. Reactions, like operations, can also be installed on specific (neighboring) devices, as in LIME. Interestingly, the information about which devices are currently directly reachable can be stored in a special tuple space, therefore providing a single unified abstraction for representing both the application and system context, similarly to the LimeSystem tuple space provided in LIME.

Implementation. Among our adaptations of LIME to the WSN environment, TeenyLIME is the least mature from a system perspective. Nonetheless, it serves as a clear proof-of-concept of what can be accomplished by pushing the tuple space model onto small WSN devices.

The implementation currently uses best effort communication. Depending on the device duty cycle, it is possible that a transmission (e.g., requesting a **rdp**) is not received by all neighbors. While acceptable for some applications, we are currently investigating mechanisms to synchronize devices, either at the application or MAC layer.

Another interesting aspect of the middleware is the mechanism to handle sensor readings. Currently, when a request arrives, the new reading is taken and the value returned. Instead, an alternative design enables a sensor to store in the tuple space not the actual data, but some description of it in a *capability tuple*. A query or reaction matches both the actual data and its capability tuple. Essentially, the latter acts as a placeholder for the real data, which is inserted in the tuple space only upon a request, therefore reducing the necessary storage. Also, this mechanism can be generalized to tasks more powerful than sensing, including aggregation, therefore providing a flexible and efficient mechanism for coordinating devices.

5. Discussion and Related Work

As evidenced by our presentation thus far, the unifying theme across our models is the use, as the main programming abstraction, of a transiently shared tuple space enhanced with reactive operations. This concept was originally introduced in LIME to deal with the dynamicity of the MANET environment. The main advantage it brings to this scenario is the ability to reduce the changes caused by the dynamicity of the system to indirect changes in the configuration of the tuple space. This provides the application programmer with a single unified view of both the application and system context, and frees her from the burden of explicitly and proactively monitoring the system dynamics. Reactive operations provide the complementary ability to define actions to be asynchronously triggered when a given data element is found in the overall context.

WSNs define an application scenario that is even more amenable to the model put forth by LIME. Indeed, applications intrinsically exhibit a mixture of data-centric and event-centric characteristics, as they usually need to collect and share data as well as efficiently report notifications and alarms. In TinyLIME, we adapted transiently shared tuple spaces to provide the glue between the mobile sinks and the surrounding sensors, therefore extending the span of the system context accessible through the tuple space abstraction. Instead, in TeenyLIME the LIME tuple space remains the main abstraction enabling inter-node coordination, but this time for devices that have different computational and communication constraints w.r.t. those in a MANET.

The relationship among the various models is depicted in Figure 4, showing that as we move from Linda to TeenyLIME the dynamicity of the system increases, from parallel systems to mobile and then sensor-based ones. Similarly, from TinyLIME to TeenyLIME the model accounts for "smarter" devices, which are increasingly independent from the sink, perform localized application functions, and coordinate among themselves. From LIME to TeenyLIME, the various systems are different variations revolving around the same core idea of transiently shared tuple spaces, demonstrating its expressive power and flexibility. The fact that they are currently separate systems is simply an accident caused by the pragmatic need to experiment with different implementations in a scenario that is realistic and of practical use, but narrow enough to be tackled independently. Nevertheless, our ultimate goal is to rejoin the characteristic dimensions of our systems into a single, unified programming model and middleware that seamlessly supports MANETs, WSNs, and their combination.

In pursuing this goal, we intend to explore other alternatives that are currently not captured by the spectrum shown in Figure 4. One prominent new issue is the support for multi-hop communication. In LIME, this is delegated to the underlying transport layer, and the middleware simply exploits it when available. However, in WSNs efficiency reasons often demand that this is integrated into the programming abstraction.

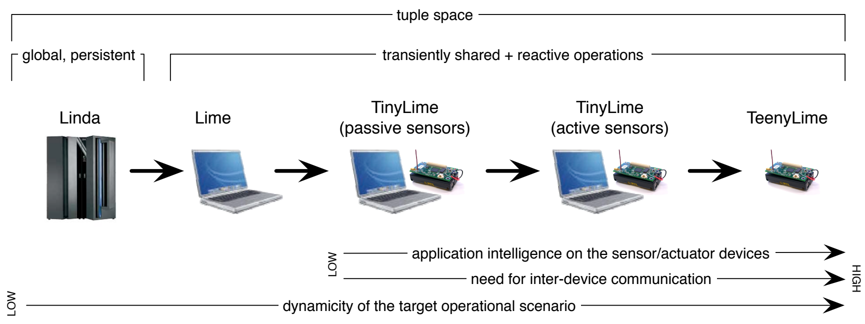
tuple space
global, persistent
transiently shared + reactive operations
Linda
Lime
TinyLime (passive sensors)
TinyLime (active sensors)
TeenyLime
LOW
application intelligence on the sensor/actuator devices
need for inter-device communication
HIGH
LOW
dynamicity of the target operational scenario

Figure 4. Tuple spaces from Linda to wireless sensor networks.

Various alternatives are possible, leading to different models and implementations. One immediately available option is to leverage off work done in our research group on *logical neighborhoods* [8]. This abstraction enables communication towards a set of devices that, unlike the physical neighborhood defined by wireless broadcast, is defined by the programmer based on characteristics of the devices (e.g., all temperature sensors with enough battery), and whose implementation accounts for efficient multi-hop routing. By integrating logical neighborhoods with our Lime derivatives we would not only readily obtain a multi-hop implementation, but also open opportunities for alternative models where data sharing occurs only among functionally related devices.

Our efforts are motivated by the increasing awareness that appropriate programming abstractions are needed to deal with the complexity of WSN application development. The work in [4] advocated early on the need for constructs that enable coordination among the WSN nodes, and to do so by privileging *localized interactions* among them. An incarnation of this concept is the Hood system [11], which provides programming constructs for interacting with the devices in the one hop neighborhood defined by physical communication range. The details of messaging, data caching, and maintaining node membership are built in the middleware: the programmer focuses on acquiring and processing the data fed by the neighboring nodes. As such, Hood relies on a many-to-one coordination paradigm. In comparison, TeenyLime provides a similar capability to access transparently the data shared in the physical neighborhood, but provides the programmer with a more general coordination interface where there is no directionality of the information flow. Moreover, TeenyLime provides both proactive and reactive operations, which enable the programmer to synchronously query for data or asynchronously receive notifications of its presence. Hood, instead, relies only on local access to cached data. Interestingly, in TeenyLime such caching can be effectively provided by using the appropriate reaction registrations.

Another strong link between the work described here and the scientific literature on WSNs exists between TinyLime and the Data Mules architecture [10]. In this work, the authors propose to exploit mobile nodes (e.g., humans, animals, vehicles) to support data collection in sparse WSNs, where alternative solutions would be technically or economically impractical. The mobile nodes (called data mules) exploit opportunistic interactions with the nodes in their proximity, buffer the data, and later make it available to some collection point. However, the work reported in [10] and similarly by other researchers focuses mostly on evaluating under which conditions and to what extent mobility is beneficial. No mention is made about how applications based on this paradigm can be designed and implemented. Interestingly, the TinyLime model provides a natural match for this scenario, with its middleware incarnation greatly simplifying application development without sacrificing efficient communication.

Finally, tuple spaces have been considered by other researchers for use in WSNs. Context Shadow [7] exploits multiple tuple spaces, each holding only locally sensed information thus providing contextual information. The application is required to explicitly connect with the tuple space of interest to retrieve information. TinyLime, with its focus on the combination of MANET and sensor networks, exploits physical locality to restrict interactions without application intervention. Instead, Agilla [5] exploits mobile agents as the main communication media, but resorts to tuple spaces for local coordination of co-located agents. Agents can also interact with remote tuple spaces, which are nonetheless distinct. In comparison, TeenyLime enables a higher level of abstraction, by enabling data sharing among agents on neighboring devices, albeit only within one hop in the current implementation.

6. Conclusions

In this paper we outlined our approach for easing the development of applications in the challenging environment of WSN. While the choice to exploit transiently shared tuple spaces was based on our successful experiences in the MANET environment, our initial experiments with TinyLime and TeenyLime show the appropriateness and versatility of the model when applied to the WSN environment.

The systems described here are implemented for the Crossbow MICA2 platform, using nesC/TinyOS. TinyLime can be downloaded from the main Lime Web site, `lime.sf.net`.

References

[1] I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: Research challenges. *Ad Hoc Networks Journal*, 2(4):351–367, October 2004.

[2] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A.L. Murphy, and G.P. Picco. Mobile data collection in sensor networks: The TinyLime Middleware. *Elsevier Pervasive and Mobile Computing Journal*, 4(1):446–469, December 2005.

[3] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A.L. Murphy, and G.P. Picco. TinyLime: Bridging Mobile and Sensor Networks through Middleware. In *Proc. of the 3rd IEEE Int. Conf. on Pervasive Computing and Communications (PerCom 2005)*, pages 61–72. IEEE Computer Society, March 2005.

[4] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: scalable coordination in sensor networks. In *Proc. of the 5th Int. Conf. on Mobile computing and networking (MobiCom)*, 1999.

[5] C.-L. Fok, G.-C. Roman, and C. Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Proc. of the 25th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 653–662, 2005.

[6] D. Gelernter. Generative communication in Linda. *ACM Computing Surveys*, 7(1):80–112, January 1985.

[7] M. Jonsson. Supporting context awareness with the context shadow infrastructure. In *Wkshp. on Affordable Wireless Services and Infrastructure*, June 2003.

[8] L. Mottola and G.P. Picco. Logical Neighborhoods: A Programming Abstraction for Wireless Sensor Networks. In *Proc. of the 2nd Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, 2006. To appear. Available at `www.elet.polimi.it/upload/picco`.

[9] A.L. Murphy, G.P. Picco, and G.-C. Roman. Lime: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2006. To appear. Available at `www.elet.polimi.it/upload/picco`.

[10] R. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: Modeling and Analysis of a Three-Tier Architecture for Sparse Sensor Networks. *Elsevier Ad Hoc Networks Journal*, 1(2–3):215–233, September 2003.

[11] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: A neighborhood abstraction for sensor networks. In *Proc. of the 2nd Int. Conf. on Mobile systems, applications, and services.* ACM Press, 2004.