# A Coordination Middleware for Wireless Sensor Networks

Manuel Díaz, Bartolomé Rubio and José M. Troya
Dpto. Lenguajes y Ciencias de la Computación.
Málaga University
29071 Málaga, SPAIN
{mdr,tolo,troya}@lcc.uma.es

## Abstract

*Middleware development in the growing and promising field of sensor networks is a major challenge in order to facilitate the programmer task and bridge the gap between the applications and the hardware. This paper presents a new middleware for wireless sensor networks based on the Coordination paradigm. The middleware is thought to support both an operational setting based on a (hierarchical) architecture of sensor regions and a coordination model rested on tuple channels. Tuple channels are FIFO structures that allow one-to-many and many-to-one communication between region sensors and the region leader in a single-hop way. The main components of the middleware architecture are described.*

## 1. Introduction

Wireless sensor networks are a new and growing class of highly dynamic, complex environments on top of which applications must be built. The complexity of designing and implementing this kind of applications makes the supply of higher-level abstractions of low-level functionality by means of middleware support necessary in order to ease the application programmer task, as it occurs in other distributed wireless systems.

Designing a middleware architecture for ad hoc sensor networks is not an easy task. It must be flexible enough in order to adapt to a dynamic environment. Sensor network applications require a minimum quality of service (QoS) and this level must be maintained over an extended period of time. The middleware may have to operate over a wide range of devices with different capabilities, e.g. laptops (plenty of memory and processing ability) and motes (extremely limited resources). Furthermore, wireless signals are prone to interference from the environment. This means the middleware must be designed to handle unpredictable message loss [8].

Much work has targeted the development of middleware trying to meet the challenges of wireless sensor networks [1] [15] [13]. Although they are designed for efficient use of this kind of systems, most of these approaches do not attempt to change the properties of the network in order to manage energy, and they are not flexible enough to support different protocol stacks or different applications' QoS requirements [11]. Moreover, they lack an appropriate abstract model that provides the application programmer with high-level abstractions of low-level concepts.

Recently, different proposals based on the Coordination paradigm have appeared [8] [5]. Coordination models and languages [3] facilitate application development by providing high-level constructs such as tuple spaces, blackboards, and channels, and have been successfully applied to a wide range of application areas: open systems [17], interactive web environments [2], scientific computation [6], just to mention some of them. Most of the Coordination paradigm based middleware approaches are thought to support an abstract model based on Linda [10], which is historically the first genuine member of the family of coordination languages and is rested on a shared memory model where data is represented by elementary data structures called tuples that are written, read and removed by processes.

This paper presents a new Coordination paradigm based middleware for wireless sensor networks supporting TC-Mote, a coordination model rested on tuple channels [7]. A *tuple channel* is a FIFO structure that allows one-to-many and many-to-one communication of data structures, represented by tuples. Our tuple channels have two characteristics that make the approach different from other channel-based coordination models. On the one hand, the get primitive, used by a consumer to obtain a tuple from a channel, does not withdraw any data from the channel from the point of view of other possible consumers and so, they can also receive the same information. On the other hand, the concept of *tuple holder* (a special kind of single-assignment variable) allows backward communication to be expressed, providing an elegant way of implicit and direct communi-

Proceedings of the 2005 Systems Communications (ICW'05)
0-7695-2422-2/05 $20.00 © 2005 IEEE

cation and reactive control.

The middleware is thought to support an operational setting based on a (hierarchical) architecture of sensor regions, each one governed by a leader with higher capabilities (power, memory, processing ability) than the rest of the region nodes (motes). The leader manages both internal and external communications. Besides the deployment of new motes, these can move inside a region or even shift region, reconfiguring the system in order to get a better performance. A region leader host owns a *tuple channel space*, which stores tuple channels used to carried out communication and synchronization between region sensors and the leader in a single-hop way. The tuple channel space interface is used to store channels in or withdraw channels from it, and to find a required one. Reactions can also be associated to tuple channel space operations.

The rest of the paper is structured as follows. To conclude this introduction, we present some related work. Section 2 shows an overview of the operational setting and the tuple channel based coordination model supported by the middleware, whose architecture is presented in Section 3. Finally, some conclusions and future work are sketched in Section 4.

### 1.1. Related Work

As stated before, different model and middleware approaches based on the Coordination paradigm are coming to the sensor network area. Two examples are Limone [8] and TinyLime [5]. The former is a lightweight coordination model and middleware to facilitate rapid application development over ad hoc networks consisting of logically mobile agents and physically mobile hosts. Each agent owns a local tuple space and an acquaintance list that defines a personalized view of remote agents within proximity. The latter extends and adapts a model and middleware called Lime [14], originally designed to incorporate mobility capabilities in a linda-like model, in order to cope with the requirements related with sensor networks.

Both Limone and TinyLime are based on the shared memory model introduced by Linda [10]. On the contrary, our approach is based on the use of channels to realize the communication and synchronization among network nodes. Several advantages, such as architectural expressiveness and data stream support in a natural and suitable way, can be obtained from the use of channels with respect to shared memory models.

There are also several proposals that are not based on the Coordination paradigm, such as the SINA [15] middleware architecture and the data dissemination paradigm called Directed Diffusion [16]. Both use an attribute-based naming scheme in order to facilitate the data-centric characteristics of sensor queries. SINA allows hierarchical clustering of sensor nodes in order to facilitate scalable operations within sensor networks. It is based on a distributed database interface to get the information from a sensor network with database-style queries. On the other hand, Directed Diffusion adopts a publish/subscribe-based API, using a data dissemination protocol family.

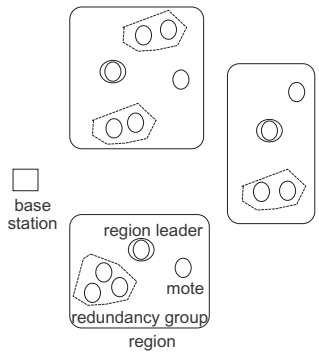


**Figure 1. operational setting.**

Our approach also use an attribute-based naming scheme due to its mentioned advantage. As SINA, our approach is also based on a hierarchical structure, but our region leaders have higher capabilities (power, memory, processing ability) than the rest of the region nodes. Moreover, the communication between region sensors and the leader is carried out in a single-hop way and by means of tuple channels, which allow one-to-many and many-to-one communication schemes. This is also a difference with respect to the Direct Diffusion paradigm, where a data dissemination algorithm is established to achieve the subscribe and publish attributes matching across a multi-hop network.

## 2. The TCMote Model

TCMote has been conceived to support the development of applications in an operational scenario where motion and single-hop communication among nodes are carried out. Our operational setting is based on an architecture of sensor regions (Fig. 1), each one governed by a leader with higher capabilities than the rest of the region nodes (motes). Due to the sheer numbers of nodes involved in a sensor network, some degree of redundancy can be expected, improving reliability and easing the self-configuration process. In our operational setting, several motes may form a redundancy group inside a region. The region monitored by a redundancy group is covered in spite of failures or sleeping periods of the group members. Motes can move inside a region or even shift region, and new ones can be deployed, reconfiguring the system in order to get a better performance.

A hierarchical structure may be achieved clustering different regions into a super-region, whose member nodes are

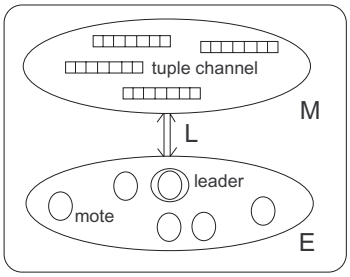


**Figure 2. Virtual Machine.**

the corresponding region leaders, one of which also acts as the leader of the super-region and the rest act as its "motes". In our operational setting, the base station can be considered as the leader of a region grouping leaders of outermost regions.

In TCMote, each region of the operational setting just described is considered as a Virtual Machine (VM) comprising the three items that a coordination model must contain (Fig. 2): E represents the entities to be coordinated (leader and motes), M is the coordination media used to coordinate these entities (tuple channel space) and L is the semantics framework the model adheres to (storing and withdrawing of channels, asynchronous sending of tuples through a channel, blocking consumption of tuples from a channel, ...).

In our model, leaders and motes are identified by means of an attribute-based data structure:

```
[attribute1 = value1, attribute2 = value2, ...]
```

For example, in a region split in four quadrants, the motes deployed on the southeast one could be identified by the following structure:

```
[identifier = mote_id, location = S-E, ...]
```

A mote motion inside a region may imply a change in some of its attributes (for example `location`). On the other hand, the motion of a mote from a region to another is modeled in TCMote by removing the mote from the corresponding source VM and creating a new mote in the target VM.

The tuple channel space is a shared data space accessed by the members of a region. It stores tuple channels used to carried out communication and synchronization between region motes and the leader in a single-hop way. A tuple channel is also specified by means of the attribute-based naming scheme just described and is a FIFO structure that allows one-to-many (from the region leader to some region motes) and many-to-one (from some region motes to the region leader) communication. Data structures communicated through channels are represented by tuples. A tuple is a sequence of fields, which may be: 1) a tuple channel identifier; 2) a value of any established data type of the host language where the model is integrated; 3) a tuple holder.

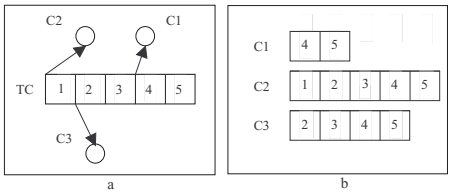


**Figure 3. Consumption behavior.**

When a tuple channel consumer obtain a tuple, this is not withdrawn from the channel from the point of view of the other consumers and so, the same information can be received by all of them. We can say that each consumer accessing a channel will have, at any time, a view of it, which may be different from the views of the rest of consumers sharing the channel. Fig. 3 shows an example describing this consumption behavior. There are 3 consumers sharing the channel TC. Fig. 3.a shows a situation where consumer C1 has consumed the first three tuples, C3 has only consumed the first one, and C2 has consumed none. Fig. 3.b represents the situation of Fig. 3.a, showing the view that each consumer has of channel TC.

Many-to-one communication scheme is useful when a continuous sending of values from sensors to the region leader is required. On the other hand, both together one-to-many and many-to-one communication schemes are appropriate to deal with typical queries in sensor networks such as "which region/area/quadrant has temperature higher than $40°C$?", "what is the average temperature in each region?". The consumption behavior allows that every region/area/quadrant can receive the same query (one-to-many). Implied motes can answer it (many-to-one) by means of a new channel whose identifier was sent inside the tuple representing the query. In addition to information communication, channels can also be used to control communication. For example, messages such as "move to northeast quadrant", "change to region i" can be sent to a mote to reconfigure the system.

Besides the communication and synchronization scheme through tuple channels described above, TCMote also provides another useful mechanism for control and information exchange based on the use of tuple holders, which are single-assignment variables. Unbound tuple holders can be sent through a channel (as a tuple field) to achieve backward communication. This allows an elegant way of implicit and direct communication. The tuple consumer is the one that will instantiate the tuple holder in order to answer some request, avoiding the use of another channel. This mechanism

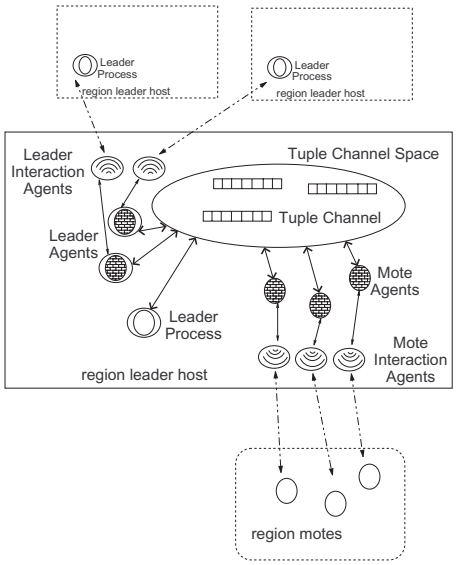


**Figure 4. Middleware architecture.**

can be useful to deal with redundancy groups, where the answer of any mote may be enough to obtain the required information about the zone controlled by the group. Moreover, a region leader could answer base station queries such as "what is the average temperature in the southeast quadrant of region i?" through a tuple holder. In addition, the use of tuple holders supports an event handling mechanism, which is suitable for sensor network applications where sensor nodes are programmed to process asynchronous events such as receiving a message or an event triggered by a timer.

## 3. The TCMote Middleware Architecture

In order to support the TCMote coordination model described in the previous section, a Virtual Machine representing a sensor region is implemented on the region leader host. As said before, a region leader has high capabilities of power, memory and processing (for example a laptop or PDA), so that is able to support the VM development. Clearly, motes (and leaders of other regions when considering a hierarchical structure) are not really physically on the leader host, but as usual, it is the middleware that takes care of creating this abstraction to simplify the programmer's life. Fig. 4 depicts the main components of the middleware architecture inside a region leader host, which are described in the following sections.

### 3.1. Tuple Channel Space

The tuple channel space (TCS) is a shared data space accessed by the leader process, mote agents and leader agents.

```
TCS Primitives:
  store(attributes)
  withdraw(search_pattern,time_out)
  find(search_pattern,time_out)
  react(operation,reaction)

Reaction Operations:
  store_r(attributes)
  withdraw_r(search_pattern)
  find_r(search_pattern)
  no_r(search_pattern)

TC Primitives:
  connect(tc_id)
  disconnect(tc_id)
  put(tc_id,tuple)
  get(tc_id,time_out)
```

**Figure 5. TCS and TC Primitives.**

All of them are described in section 3.2. The TCS interface offers the first four primitives shown in Fig. 5.

As indicated in section 2, a tuple channel is specified by means of an attribute-based naming scheme. This way, when a tuple channel is stored in the TCS its attributes are specified. For example, consider a region split in four quadrants. It could be useful to assign different channels to quadrants. Tuple channels can be provided with an attribute such as `location`. So, for tuple channels assigned to the southeast quadrant, the attribute-based data structure identifying them could be as following:

```
[identifier = tc_id, location = S-E]
```

On the other hand, when a tuple channel is desired to be removed from the TCS, an attribute-based data structure with (probably) some partially specified fields (`search_pattern`) is used as first argument of the `withdraw` primitive. For example, the following operation:

```
withdraw([identifier = ?, location = S-E])
```

will withdraw from the TCS all channels assigned to the southeast quadrant.

Consider that a mote agent (for example on behalf of a new deployed mote or a mote that has carried out a motion) comes into the southeast quadrant. It can use the `find` primitive in order to find the appropriate channel to interact with the leader process. This primitive uses as first argument a search pattern. `[identifier = ?, location = S-E]` is a good candidate in our example. This primitive is governed by a pattern matching scheme where, if several candidates are found, one of them is chosen in a non-deterministic way. The partially specified fields (`identifier` in our case) are instantiated with the appropriate values from the found channel information.

Going on with the example, consider now that there was no tuple channel assigned to the southeast quadrant when the mote agent initiated the find operation. So, the `find`

primitive will block and the mote agent will receive no channel. It is highly probable that the leader process does not want this situation occurs. This way, it can previously associate a reaction to a find operation in order to provide a new required channel. This is carried out by means of the `react` primitive, which has two arguments: the implied operation and the reaction it is desired to associate to it. The occurrence of the operation will trigger the reaction.

A reaction is defined as a conjunction of non-blocking operations, and has a success/failure transactional semantics: a successful reaction may atomically produce effects on the tuple channel space, a failed reaction yields no result at all. The operations we have initially considered as candidates to be included inside a reaction are shown in Fig. 5. The first three primitives have the same effect as the previously specified ones, but they are non-blocking primitives. So, `withdraw_r` and `find_r` succeed when the search pattern matches any data structure in the TCS, but fail otherwise. Complementary to them, `no_r` succeeds when its argument does not match any data structure in the TCS, but fails otherwise.

In our example, the leader process can associate the following reaction to a find operation:

```
react(find([identifier = ?, location = S-E]),
   (no_r([identifier = ?, location = S-E]),
  store_r([identifier = new_tc_id, location =
                    S-E])))
```

After find operation execution, the specified reaction is triggered. If no tuple channel has been found, no_r operation succeeds and store_r operation stores a new channel in the TCS. This will resume the find operation, which will provide the mote agent with the required channel.

Before an entity can send/receive information through/from a TC, it must establish a connection by means of the `connect` primitive. On the other hand, when it does not need a channel any more, it executes the `disconnect` primitive in order to disable the TC connection. Connection information will be useful to the run-time system in order to achieve an efficient channel implementation.

A producer will use the `put` primitive to send information through a channel. Each time a put operation is executed, a new tuple is added to the end of the channel.

A consumer will use the `get` primitive to receive information from a channel. Each time a get operation is executed, a new tuple from the beginning of the channel is obtained. The `get` primitive follows the consumption behavior described in section 2.

### 3.2. Coordinated Entities

As said before, motes and other region leaders (hierarchical structure) are not really physically on the leader host of a particular region. Instead, the middleware must take care of creating the corresponding abstraction in order to facilitate the model utilization and implementation. In our approach, besides the *leader process*, which is in charge of doing the region leader task, there are two other kinds of agents accessing the TCS, i.e. the *leader agent* and the *mote agent* (see Fig. 4). These three components communicate and synchronize each other by means of the tuple channel space mechanism previously described.

A leader agent staying at a region leader host `A` acts on behalf of a leader process in a region leader host `B`, which is trying to communicate with the leader process in host `A`. The interaction between the leader agent and the leader process in host `B` is carried out by means of a *leader interaction agent*. Similarly, a mote agent inside a host acts on behalf of a mote in the region where this host is. The interaction between the mote agent and the mote is achieved through a *mote interaction agent*. Both the leader interaction agents and the mote interaction agents are described in the next section.

### 3.3. Interaction Agents

We have decided to highly decouple internal communication inside a region leader host (via TCS scheme) from external communication (via message passing through the wireless network). This way, we introduce interaction agents in order to deal with the second one. This decoupling brings some advantages. A change in the way the wireless communication is carried out only affect the interaction agents. On the other hand, interaction agents may be reused in applications other than TCMote-based to provide a straightforward interface to access motes from a host.

When a mote (a new one or a mote coming from another region) is deployed on a region, a pair "mote agent/mote interaction agent" is created inside the region leader host. The mote interaction agent asynchronously interacts with the mote agent to handle request, replies, and all communication with the mote. The main task of this interaction agent is to translate high-level TCS-based information into packets understandable by the mote and vice versa. In principle, communication between the interaction agent and the mote is just message passing, but it is not as straightforward in sensor networks as in traditional ones. It is well known that motes must sleep most of the time in order to conserve energy, waking up on a regular basis to receive and process information. This way, the interaction agent must repeatedly send a single packet.

Besides information exchange, the interaction agent is also in charge of control communication carried out in order to reconfigure the system. A mote could be on top of a robot or inside a pocket of a person (consider a city pol-

lution measurement application, where people from different city quarters can be provided by a mote). A mote may be required to move inside its region or even shift region to obtain a better system performance. This way, the interaction agent must communicate with the robot or a PDA carried by a person in order to send motion requirements. From the point of view of a mote agent, a mote motion only implies an attribute change in the attribute-based data structure identifying it, or maybe its removing from the region leader host (and the removing of the corresponding interaction agent) if the mote has shifted region.

Finally, the pair "leader agent/leader interaction agent" acts in a similar way, but in this case the communication between the interaction agent and the leader process (inside another region leader host) is more straightforward due to the features of communicating parties.

## 4. Conclusions and Future Work

A middleware architecture for wireless sensor networks has been presented. The main components of the middleware architecture inside a region leader host, the tuple channel space, the coordinated entities and the interaction agents, have been described. Data-centric characteristics of sensor queries are enabled by means of an attribute-based naming scheme together with the tuple channel and tuple holder communication and synchronization mechanisms of the supported model.

We are currently developing a middleware prototype. We are using Java as both the host language (where the tuple channel model is integrated) used by the application programmer and the programming language used to implement the middleware architecture components inside the different region leader hosts (laptops and PDAs). On the other hand, nesC [9] is being used to program the necessary components inside the motes. Crossbow Micaz motes [4] running the open source operating system called TinyOS [12] are being used.

## References


[1] P. Bonnet, J. Gehrke, P. Seshadri, "Querying the Physical World", *IEEE Personal Communications*, 7(5), 2000, pp. 10–15.

[2] G. Cabri, L. Leonardi, F. Zambonelli, "MARS: a Programmable Coordination Architecture for Mobile Agents", *IEEE Internet Computing*, 4(4), 2000, pp. 26–35.

[3] N. Carriero, D. Gelernter, "Coordination Languages and their Significance", *Communications of the ACM*, 35(2), 1992, pp. 97–107.

[4] Crossbow Technology Inc. http://www.xbow.com/

[5] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A.L Murphy, G.P. Picco, "TinyLime: Bridging Mobile and Sensor Networks through Middleware", in *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communication (PerCom 2005)*, IEEE Computer Society, Kauai Island, Hawaii, 2005, pp. 61–72.

[6] M. Díaz, B. Rubio, E. Soler, J.M. Troya, "A Border-based Coordination Language for Integrating Task and Data Parallelism", *Journal of Parallel and Distributed Computing*, 62, 2002, pp. 715–740.

[7] M. Díaz, B. Rubio, J.M. Troya, "TCMote: A Tuple Channel Coordination Model for Wireless Sensor Networks", to appear in *IEEE International Conference on Pervasive Services (ICPS 2005)*, Santorini, Greece, July 2005.

[8] C-L. Fok, G-C. Rooman, G. Hackmann, "A Lightweight Coordination Middleware for Mobile Computing", in *Proceedings of the 6th International Conference on Coordination Models and Languages (COORDINATION 2004)*, LNCS vol. 2949, Pisa, Italy, 2004, pp. 135–151.

[9] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems", in *Proceedings of Programming Language Design and Implementation (PLDI 2003)*, 2003.

[10] D. Gelernter, "Generative Communication in Linda", *ACM Transactions on Programming Languages and Systems*, 7(1), 1985, pp. 80–112.

[11] W.B. Heinzelman, A.L. Murphy, H.S. Carvalho, M.A. Perillo, "Middleware to Support Sensor Network Applications", *IEEE Network*, January 2004, pp. 6–14.

[12] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, "System Architecture Directions for Network Sensors", in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, Cambridge, MA, USA, 2000, pp. 93–104.

[13] T. Liu, M. Martonosi, Impala: "A Middleware System for Managing Automomic Parallel Sensor Systems", in *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, California, USA, 2003, pp. 107–118.

[14] G.P. Picco, A.L. Murphy, G-C. Roman, "Lime: Linda Meets Mobility", in *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*, 2003, pp. 368–377.

[15] C-C .Shen, C. Srisathapornphat, C. Jaikaeo, "Sensor Information Networking Architecture and Applications", *IEEE Personal Communications*, August 2001, pp. 52–59.

[16] F. Silva, J. Heidemann, R. Govindan, D. Estrin, "Directed Diffusion", *Frontiers in Distributed Sensor Networks, S. Iyengar and R. R. Brooks, editors*, CRC Press, 2004.

[17] R. Tolksdorf, "Laura: A Service-based Coordination Language", *Science of Computer Programming*, 31(2-3), 1998, pp. 359–381.