1

Shared Wireless Sensor Networks as Enablers for a Context Management System in Smart Cities

Wireless sensor networks (WSNs) are commonly used as a sensing infrastructure for smart city applications. A WSN is easy to use and can cover a wide area at low costs because of its wireless communication capability. The sensor nodes constituting a WSN are usually equipped with one or more sensor devices and can be used for different measurement purposes by reprogramming them. If WSNs could be shared by different smart city applications, they could be even more valuable enablers for smart cities. However, it is not easy to share WSNs. A shared WSN needs to support different kinds of measurement tasks at the same time and be able to accept new tasks at runtime. Even in a traditional closed WSN, its software should be carefully developed to satisfy certain quality requirements despite the severe resource constraints affecting the individual programmable sensor nodes (the sensor nodes of WSNs usually have quite limited resources, e.g. small batteries, low-spec CPU and narrow bandwidth). This issue is much harder to resolve in the case of a shared WSN. To satisfy the quality requirements of different applications, a WSN should be configured carefully according to specifications of the tasks, their quality requirements, and the environment, and should adapt its configuration in response to changes in the environment and the applications. A shared WSN should support various measurements, manage tasks at runtime and adapt to changes in the environment to reduce unnecessary consumption of resources. To develop such a shared WSN, we propose a middleware support for the network. In this chapter, we describe the architecture of our XAC middleware and the issues relevant to the shared WSN from the viewpoints of the task-description language, runtime task management and self-adaptation.

1.1. Introduction

In the smart cities of the future, many context-aware applications will support the citizen's activities by proactively controlling the various devices used therein.

Chapter written by Kenji TEI.

Context-aware applications will recognize the current context of the city they are monitoring and actuate devices to amend their status. A key service in smart cities will be context management systems, which estimate context of cities and provide it to applications.

A context management system should be able to collect and update various types of content required by context-aware applications and should be able to be used easily in various environments. Here, a wireless sensor network (WSN) will be a key infrastructure in context management systems. A WSN is a wireless *ad hoc* network consisting of tiny computers equipped with sensors and wireless communication devices. It continuously records and produces data by measuring the environment via sensor nodes. It can produce one or more kinds of data, because its nodes are equipped with one or more kinds of sensors, and it can be programed to alter or switch tasks between the different sensor devices used for monitoring. Moreover, it can be easily used because it does not require any communication cables. Its nodes communicate with each other via wireless links and transmit the measured sensor data via multi-hop communications.

These features of the WSN are quite important for context management systems in smart cities. First, its "easy-to-deploy" feature is suitable for the smart cities. Sensor nodes are usually used in outdoor spaces, but it is not easy to connect sensor nodes using cables because of monetary and legal constraints. Second, the "reprogrammable" feature is suitable because a smart city usually hosts many applications that require different kinds of sensor data. A context management system should carefully balance the demands of these applications and the resource consumption of the sensor nodes. This can be realized by reprogramming a WSN. Therefore, the WSN is a key enabler for a context management system in a smart city.

A shared WSN for smart cities should:

1) support various kinds of measurements;

2) manage tasks at runtime;

3) adapt to changes in the environment to reduce unnecessary resource consumption.

A shared WSN is used by many context-aware applications, which require different kinds of sensor data and different levels of accuracy. Therefore, it should be able to handle various measurements to produce one or more kinds of sensor data required by these applications with a level of accuracy. Moreover, applications using a WSN appear and disappear at runtime. Therefore, the WSN should be able to add or remove tasks at runtime without having to stop and start. Finally, a WSN should be able to adapt its behavior in response to changes in the environment. A WSN has

severe resource limitations because each node in a WSN has CPU, memory, bandwidth and battery restrictions, and resources must be saved to increase the number of tasks that it can handle and to prolong its lifetime. Therefore, the WSN needs to automatically adapt to changes in the environment to reduce unnecessary resource consumption, that is to say without human intervention.

To develop such a shared WSN, middleware supports are needed. This chapter describes an example of middleware for a shared WSN, called XAC middleware. In addition, we discuss the research issues related to shared WSNs and the techniques used in XAC middleware.

1.2. Background

WSN software development is not easy because it requires programmers to have an in-depth knowledge of various fields, such as analysis of sensor data, distributed programing in wireless *ad hoc* networks and optimization of embedded systems. This section presents examples of types of WSN software to identify the issues concerning shared WSNs.

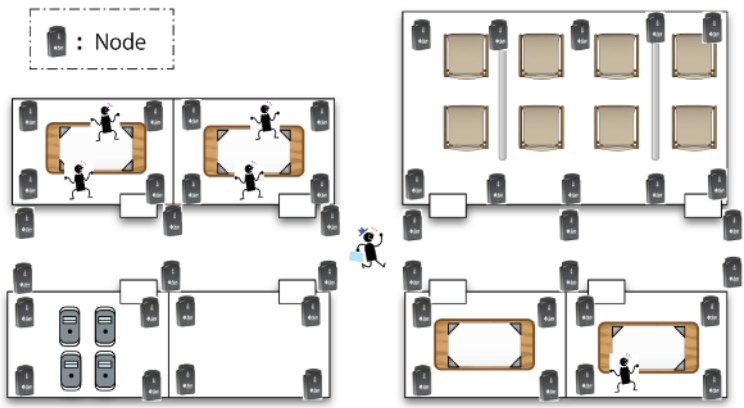


Figure 1.1. *A smart environment*

Let us first define our example– a smart environment is set up in an office building, where context-aware applications are introduced to optimize everyday business tasks. Consider the environment illustrated in Figure 1.1. Sensor nodes are used throughout rooms, corridors and stairwells to enable monitoring and to establish the current context of the building. Data sensed by the nodes are transferred via multi-hop communication to a central server (called the *base station* from here on).

Table 1.1 shows four scenarios, S1, S2, S3 and S4, envisioning context-aware applications in this example environment. In scenario S1, an application maintains the temperature levels in the conference rooms according to the preferences of the people in the room. The application in S2 determines the occupancy of conference rooms on the basis of the presence of people in the room and the reservation data of the room. Scenario S3 involves tracking applications that continuously monitor the current locations of staff inside the building, whereas the application in S4 detects suspicious intruders.

Scenarios	Application name	Operational tasks	Environmental information	Accuracy requirement
S1	Temperature management	Adjust room temperature according to preferences of people in the room	Temperature in the room	Within 2°C of actual value
S2	Meeting-room management	Maintain occupancy of conference rooms based on their current occupancy and reservations	Presence of people in conference rooms	Determine correct room occupancy with 99% accuracy
S3	Staff-tracking management	Determines the current locations of staffs	Location of staff	Within 1 m range in a public space or room
S4	Intruder detection	Determines suspicious intruders for instance by raising an alarm if people remain near an access lock for long periods without authenticating	Presence and position of people in certain locations	Within 2 m range

Table 1.1. *Examples of context-aware application scenarios*

Each application requires environmental context information related to its own operational tasks. S1 requires temperature data, S2 requires data on the presence of staff in each room, S3 requires data on the location of each member of the staff and S4 requires the location information of people in designated areas.

As we can see, each scenario possesses different non-functional requirements in terms of accuracy. Generally speaking, sensor data include a certain level of sensor error. A well-known way to improve accuracy is to aggregate sensor data coming from neighboring sensor nodes. For instance, knowing only the rough locations of staff (like the room in which a person is currently located) is enough for S3. In this

case, the low accuracy requirements can probably be satisfied with sensor data from just one or two nodes. On the other hand, S4 requires the specific positions of staff to accurately track them within 2 m. This in turn entails gathering more sensor data than in S3.

Although many sensors are required from the viewpoint of accuracy, resource usage in a WSN should be kept as low as possible to prolong the lifetime of the network. Software developers should take into account the severe resource limitations of nodes in terms of CPU power, memory, communication bandwidth and so on when creating a WSN. In particular, the battery is a precious resource. For example, on average the battery of the commonly used Crossbow Mica2 node will deplete in just 7 days by reading its temperature sensor value and sending transmissions every second [SHN 04]. Even though the lifetimes of sensor nodes are gradually increasing as a result of hardware improvements, energy consumption is still an important issue in WSNs. Load concentrations on specific nodes will drain batteries quickly, which are then hard to recharge during runtime.

As such, to extend the network lifetime, it becomes necessary to use various optimization methods to extend each node's operational time, for instance, by aggregating sensor readings before transmitting them, by adjusting the sensing frequency to meet certain accuracy requirements or by duty-cycling node operation. To yield optimal results, these methods also need to comply with the requirements of multiple applications.

1.3. XAC middleware

XAC middleware is a middleware for a shared WSN. Its main features are as follows:

– WSN as a multi-modal sensor: XAC middleware uses the fact that a WSN is a multi-modal sensor. It hides the low-level details of the WSN from context-aware applications. A WSN can therefore be seen as a single sensor covering a large area by these applications. XAC middleware also provides a way to use the WSN for different measurement purposes. One or more applications can use the WSN at the same time.

– Runtime management: XAC middleware allows context-aware applications to register or unregister their measurement tasks during runtime.

– Self-adaptation: XAC middleware monitors changes in the WSN and adapts configurations in response to these to reduce unnecessary resource consumption and to maintain the required level of accuracy.

1.3.1. *Architecture of XAC middleware*

XAC middleware is implemented using SunSPOT, a commercial sensor node provided by Oracle. A WSN application with SunSPOT consists of base-station-side software running on a Java VM, and sensor-side software running on a Squawk VM in each node. Thus, the middleware consists of base-station-side middleware and sensor-side middleware.

The base-station-side middleware runs on the base station. It provides interfaces to context-aware applications to register or unregister their measurement tasks. In response to requests from the context-aware application, it either inputs software components relevant to a measurement task into the sensor side or removes them. The inputted components are used on sensor nodes designated by the deployment policy, and they report sensor data to base-station-side middleware. The base-station-side middleware provides the sensor data to the context-aware applications or initiates events by analyzing the sensor data in accordance with the specifications defined by each application. It also monitors the sensors and changes the configuration if it decides it is necessary to do so.

The sensor-side middleware runs on each sensor node. It manages the components inputted by the base-station-side middleware. A group-based approach is used to model a measurement task. A group consists of a master and slaves. The slaves are responsible for measuring designated sensor data and reporting it to the master. The master is responsible for aggregating the data and reporting it to the base station.

Figure 1.7 shows the architecture of the XAC middleware consisting of the base-station-side (XAC Middleware-B) and the sensor-side (XAC Middleware-S).

XAC Middleware-B consists of three components: UserInterface, EventManager and GroupManager. The programmer uses the UserInterface component to register and delete tasks and event handlers, and to obtain the results of the task. The event handler is managed by the EventManager component, and the task is managed by the GroupManager component. The GroupManager component generates Group components corresponding to the task and uses and activates them on the sensor nodes. Each Group component iniates an event when its measurement data satisfies a certain condition and sends it to the EventManager component through the GroupManager. Then, the EventManager component calls the handler corresponding to the event.

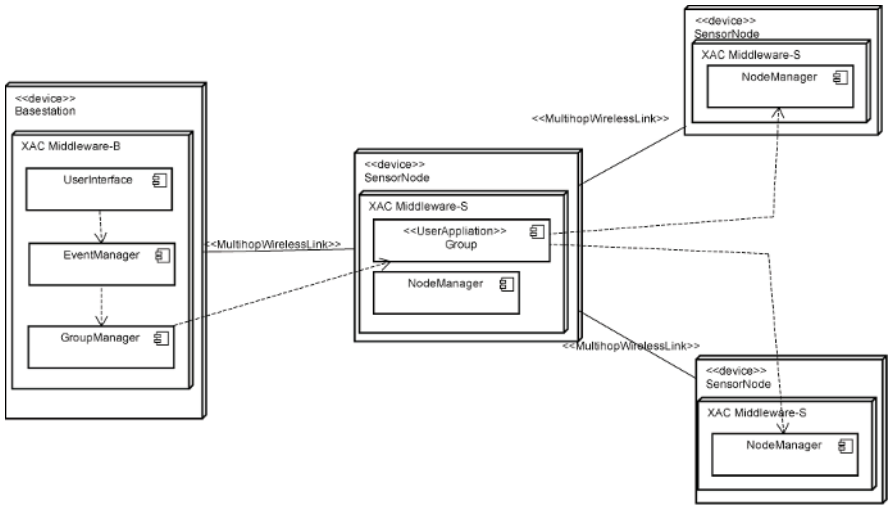


Figure 1.2. *Architecture of prototype implementation*

XAC Middleware-S consists of Group and NodeManager components. The Group has deployment conditions of itself and its measurement tasks. Group components are generated by the GroupManager component of XAC Middleware-B and used on nodes that satisfy its deployment conditions. The Group component finds and selects nodes to satisfy the deployment condition of its measurement task and makes measurement requests of the NodeManager components of those nodes. The Group component collects the measurement results from the NodeManager component and aggregates them. If the aggregated data satisfy the conditions, the Group component fires an event and sends it to the GroupManager component.

The following sections outline the outstanding issues of shared WSN middleware, the existing work on resolving them and the XAC middleware solution from the point of view of the task-description language, runtime task management and self-adaptation.

1.4. Task-description language

Each piece of middleware has its own task-description language. A task-description language is a domain-specific language that specifies the behavior of a WSN for capturing the current context of target phenomena. It provides an abstract view of the WSN to programmers and thus constitutes different levels of programmability.

1.4.1. *Existing solutions*

According to the level of abstraction, existing task-description languages can be classified into data-, group- and node-level languages.

1.4.1.1. *Data-level languages*

Data-level task-description languages, such as those of *TinyDB* [MAD 05], *Cougar* [YAO 02], *TinyLIME* [CUR 05] and *TeenyLIME* [COS 07], allow programmers to describe what kind of data they require and how these data are supposed to be processed to produce a context. These languages each take different approaches to abstraction. For example, *TinyDB* and Cougar abstract the WSN as a relational database and provide SQL-like languages, whereas *TinyLIME* and *TeeneyLIME* abstract the WSN as a tuple space and provide tuple-query languages. Figure 1.2 shows a sample task description written in the language provided by *TinyDB*. The description produces the average of the temperature value measured by all the sensors on the fourth floor of our example environment (section 1.2).

Data-level languages focus not on how to measure data, but on what to measure. Therefore, the concrete behaviors of nodes in the WSN are managed by the middleware and remain transparent to programmers. There are many ways to derive the required data processing functionality. The middleware may support a set of behaviors selected from a potentially adequate functionality. For example, to handle the task description shown in Figure 1.2, *TinyDB* uses a tree-based network topology to route the measured data (i.e. the temperature) and aggregates the data (i.e. to form the average) along the routing tree. However, the programmer cannot change this concrete routing behavior to an alternate one for the purpose of optimization.

```
SELECT AVG(temp)
FROM sensors
WHERE floor = 4
SAMPLE PERIOD 5S
```

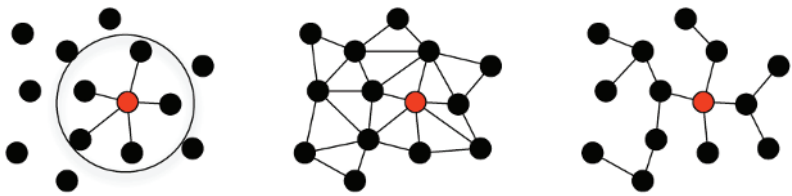
Figure 1.3. *A task description in TinyDB*

1.4.1.2. *Group-level languages*

Group-level task-description languages, such as those provided by *EnviroTrack* [ABD 04], *Hood* [WHI 04], *Abstract Region* [WEL 04], *Generic Role Assignment* [FRA 05] and *DFuse* [KUM 03], allow the programmer to describe macro-level behaviors for a group of nodes to achieve the desired data processing functionality in a WSN. These languages require a programmer to describe definitions and behaviors of a node group. The programmer usually defines the conditions of the

nodes to form a group. *EnviroTrack*, *Hood* and *Abstract Region* form a group consisting of neighboring nodes. A programmer can define which nodes constitute the neighboring ones by using the number of hops from the node closest to the target in the case of *EnviroTrack* or by using physical distances in the case of *Hood* and *Abstract Region*. On the other hand, the programmer may influence the node selection by using node properties such as the battery level, equipped sensor types or bandwidth in the case of *Generic Role Assignment* and *DFuse*.

Moreover, the programmer can define the macro-behaviors of a group by assigning roles to the nodes in a group. For example, the roles used in *EnviroTrack* are classified into two types – the role called *member* measures data and the role called *leader* aggregates or fuses data retrieved from member nodes through a cluster-based network topology. The languages provided by *Hood*, *Abstract Region*, *Generic Role Assignment* and *Dfuse* allow the programmer to describe the network topology between the roles. For example, *Abstract Regions* allow the programmer to select a *geographic*, *planar mesh* or *spanning tree* topology (Figure 1.3). Furthermore, *Dfuse* allows the programmer to define a topology as a data flow graph.



a) Geographic b) Planar mesh c) Spanning tree

Figure 1.4. *Topologies of an Abstract Region*

Compared with data-level languages, group-level languages provide the programmer with a more concrete view of the WSN. Therefore, a task written in a group-level language can be optimized more effectively but this in turn requires detailed knowledge about the node behaviors in the WSN.

1.4.1.3. *Node-level languages*

Middleware such as *Squawk* [SIM 05], *Agilla* [FOK 05], *SensorWare* [BOU 07] and *ActorNet* [KWO 06] provide node-level-description languages. Node-level languages allow a programmer to specify the behavior of a task running on a single node.

Squawk provides the Java programing language to define the behavior of a node. *Agilla*, *SensorWare* and *ActorNet* provide mobile agent-based languages to describe a task that can migrate from one node to another.

Node-level languages provide a more concrete view of WSNs to the programmer than group-level languages do. Since, the behaviors of each node are programmed directly, a task written in a node-level language can be more thoroughly optimized than one written in a group-level language. However, the burden of the programmer increases at the same time. A programmer using a node-level language must have knowledge about distributed programming, since he/she has to implement each task and support, in addition to the main functionality, different aspects like routing, topology management and the synchronization of the nodes.

1.4.2. *XAC middleware solutions*

As indicated above, the existing task-description languages can be classified into data-level, group-level and node-level languages. Each class has different levels of abstraction. Data-level languages provide the most abstract view of a WSN, and node-level languages provide the most concrete view. The level of abstraction constitutes a tradeoff between the description cost and the room for optimization. A higher level of abstraction reduces the description cost, but also reduces the room for optimization at the same time. The adequate level of optimization also depends on the non-functional requirements of the tasks.

The existing solutions each provide only one language with a fixed level of abstraction. For example, *TinyDB* only supports spanning-tree network topologies and does not allow a programmer to use other network topologies such as planar mesh topologies or cluster topologies. From the viewpoint of resource consumption, the selection of an adequate topology depends on the characteristics of the task to be implemented, such as the geographical space to be covered, the type of data to be measured and so on. Therefore, the programmer has to carefully choose an adequate network topology to optimize resource consumption. Group-level or data-level languages allow for this, but data-level languages hide the network topology from the programmer.

XAC middleware provides languages with multiple levels of abstraction. It provides data-level, group-level and node-level languages so that the programmer can choose an adequate level of abstraction. Here, an adequate level of abstraction may depend on the knowledge of the programmer and on the non-functional requirements of his tasks, such as resource consumption, response time or accuracy of the results. Moreover, we are trying to find a proper model transformation for task development (Figure 1.4). A task described at an abstract level should be transformable into a concrete one. A task at the data level may be transformed into a corresponding task at the group level that achieves the data processing functionality described in the data level task with the basic protocols necessary for group management; and a task at the group level may be transformed into a corresponding one at the node level that achieves the macro-behaviors described at the group level.

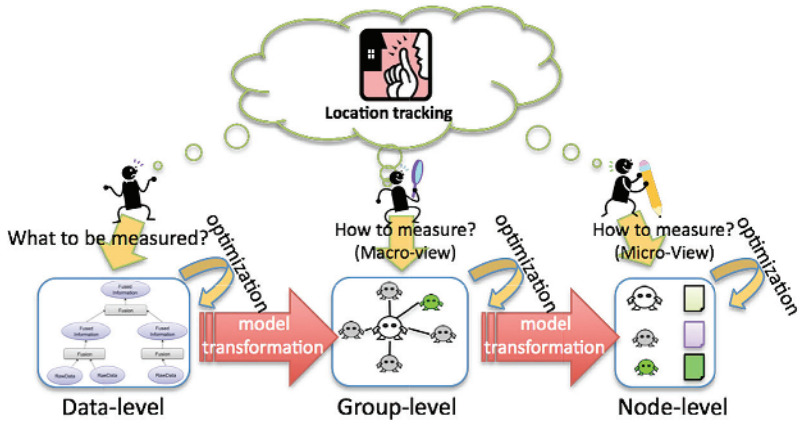


Figure 1.5. *Model-driven development for task descriptions*

A transformed model should have enough information to be executed in the WSN. If needed, the programmer should be able to optimize behaviors at arbitrary levels to fulfill non-functional requirements. We should be able to make suitable optimizations for data processing at the data level, as well as make optimizations of the topology, routing and in-network aggregation in a group at the group level and optimizations of duty-cycling or network device management at the node-level, to achieve the desired accuracy levels, and to decrease resource consumption. For example, consider scenarios S2 and S4 in Table 1.1. Both S2 and S4 require the locations of people near the target location, but S4 requires a higher accuracy than S2. From the viewpoint of data processing, simple localization in which sensors detect a radio signal from beacons that people carry would be enough for S2, but a more sophisticated localization, such as one based on RSSI (radio signal strength indication) that sensors measure, would be required for S4. If such an RSSI-based method is applied at the data processing level, accuracy and resource consumption depend on group-level behaviors such as the number of nodes that measure the RSSI, the routing topology in a group and the in-network aggregation techniques.

The approach presented here is to apply model-driven development to tasks in the WSN. Here, we analyzed the descriptive capabilities of existing languages and constructed a reference model for each level of abstraction. Moreover, we made a catalog of optimization patterns at each level, and one of the transformation patterns from abstract levels to more concrete levels, to support optimization and manual model transformation [TEI 07]. We also constructed transformation rules and development processes according to model transformation and optimization patterns and devised verification methods to ensure consistency between models at each

level to guarantee that the behaviors written in higher-level descriptions would still to be valid after subsequent optimizations at lower levels [TEI 14].

1.5. Runtime task management

Sensing tasks have to be assigned to specific nodes for execution, and such resource allocations have to be managed. A rather general approach taken in the early days of wireless sensor networks was to assign tasks to certain nodes by physically connecting each node to a base station and using specific tasks thereby setting up code modules on the nodes. This made changing tasks or assigning new tasks after using the sensor nodes a tedious exercise requiring physical retrieval of the nodes in question. Nowadays, nodes are reprogrammed by assigning tasks dynamically over a wireless communication channel.

1.5.1. *Existing solutions*

Reprogramming techniques can be divided into two distinct approaches:

1) Those managed by the base station: the base station centrally manages tasks and assigns them to the nodes.

2) Those managed by the tasks: the tasks themselves decide which node they use to and move between nodes autonomously[1].

1.5.1.1. *Task deployment management by the base station*

When the base station is used to manage task deployment, it becomes necessary to decide whether tasks should be disseminated throughout the whole network or only sent to specific nodes. For instance, in scenario S1 of Table 1.1, the tasks need only be assigned to nodes inside rooms that need their temperature managed, whereas in S3, the sensing tasks must be used across all the sensor nodes throughout the building. Assuming that changes in, for instance, the sensing conditions and task settings make updating of the tasks of S1 and S3 necessary, we need to consider the range of the affected nodes, namely only for specific nodes as in the case of S1 or in all nodes as in S3.

One existing solution for managing tasks at the base station by specifying a certain range of nodes for updates is Deluge [HUI 04]. Deluge allows the user to optionally define specific node IDs to limit the dissemination of tasks throughout the

1 Tasks can be deployed from the base station instead of moving from one node to another, but they make the deployment decisions on their own in either case.

network. To cope with the energy and communication restraints of wireless sensor nodes, Deluge works to improve efficiency as follows:

– congestion in areas of high node density is avoided by adjusting transmission intervals dynamically;

– asymmetric links are handled by selectively using stable nodes;

– broadcast storms between nodes are avoided by introducing the concept of communication rounds.

Deluge sends the complete code necessary for executing a task from the base station to the nodes. This method is inefficient in so far as it induces high transfer costs that drain the node batteries. To address this issue, other solutions make preliminary deployment of code bases that are shared among different tasks, so that only task-specific updates need to be transferred in case a change becomes necessary. A representative example of this approach is Mate [LEV 02]. Mate provides a virtual machine (VM) on each node that can execute a task-specific lightweight code. Its VM is based on a byte code interpreter that uses 1 byte per byte code. As such, a script consisting of 100 lines amounts only to 100 bytes to be transferred. Mate does not allow the user to specify certain deployment ranges, but the recent revision of Trickle [LEV 04] provides this functionality.

1.5.1.2. *Self-adaptive task deployment management*

In the case that tasks manage themselves autonomously but are used from the base station, the base station is always responsible for distributing tasks to the nodes. In this approach, if it is impossible to preliminarily deploy all tasks to all nodes, changes in the locations in which to execute certain tasks make it necessary to reuse the tasks from the base station. For example, in the previously described scenario S4 in Table 1.1, if the goods to be monitored are moved, the tasks have to be reused to the nodes in the vicinity of the goods' new location every time. This results in a heavy traffic load, especially on nodes around the base station. If the tasks can manage and also reuse themselves by moving autonomously, that burden can be alleviated[2].

To address this problem, the research community has come up with a number of solutions. Representative examples are *Agilla* [FOK 05], *ActorNet* [KWO 06] and *SensorWare* [BOU 03]. *Agilla* extends Mate's VM and gives tasks the ability to move throughout the network. Tasks in *Agilla* are based on a 1–2 byte ISA

2 If the distance between base station and the node where a task is supposed to be redeployed is longer than the distance between the current deployment node and the new deployment location, the cost of moving a task from the current node to the new node is less on average than redeploying the task from the base station.

(instruction set architecture) and thus provide lightweight task codes similar to Mate. *Agilla* can also conduct intra-task communication based on a distributed tuple space [GEL 85].

ActorNet introduces the actor model to wireless sensor networks and executes each actor on its scheme interpreter. It also provides a virtual memory space, scheduling of I/O to the scheme interpreter and garbage collection functionality.

SensorWare is similar to *Agilla* and *ActorNet* in terms of making tasks manage and move themselves. However, it assumes that the deployment platform is an iPAQ and requires about 1 MB of storage and about 128 KB of memory for execution. Thus, its platform specs (several hundred MHz of CPU power and 1 MB of storage) are not in line with those of typical resource-constrained nodes of a wireless sensor networks.

1.5.2. *XAC middleware solutions*

We assume that the wireless sensor network is a shared infrastructure that can be used by any users. Such a network is unlike the traditional ones when it comes to adding, changing and deleting tasks[3].

Furthermore, as described in scenario S4 of Table 1.1, tasks may change their execution location. Consequently, we use task-based autonomous deployment management as discussed in the previous section. The existing solutions allow tasks to be comprised of multiple components running on different nodes. For instance, in the case of scenario S1 of Table 1.1 (temperature management), the task of measuring temperature can be assigned to a certain number of components running on specific nodes inside a room. However, the existing solutions do not provide the relocation of dynamic reconstruction for multiple components. We hence had to come up with our own solutions to provide these functionalities.

1.5.2.1. *Relocation of multiple components*

Our relocation mechanism, called generative dynamic deployment (GDD) of multiple components [SUE 09], provides its functionality through middleware. GDD consists of an architecture to relocate multiple components and a novel relocation method. Past solutions did not offer any architecture for usable components, and thus, the reliability of the relocation process itself was not addressed. Generally speaking, though, if components are to relocate, they need to communicate among each other to coordinate that task, and as a result, the reliability of the relocation task

3 Wireless sensor networks are not capable of preliminary deployment of all tasks for users who suffer from packet loss [SUE 09].

declines with the number of communications. To address this problem, the architecture of GDD comprises three component types (Master, Slave-S and Slave-M), and the Master components delete the Slave-S and Slave-M components only before they relocate. Once they have completed their relocation, they reconstruct the other components on demand.

Figure 1.6 shows an example scenario of relocation based on GDD. The Master, Slave-S and Slave-M components have already been used (1) (not shown) and are about to relocate. The procedure is as follows: (2) the Master component deletes Slave-S and Slave-M components, if relocation becomes necessary; (3) the Master component moves to the new target location; (4) the Master component, upon relocating, reconstructs Slave-S and Slave-M components and uses them on nearby nodes.

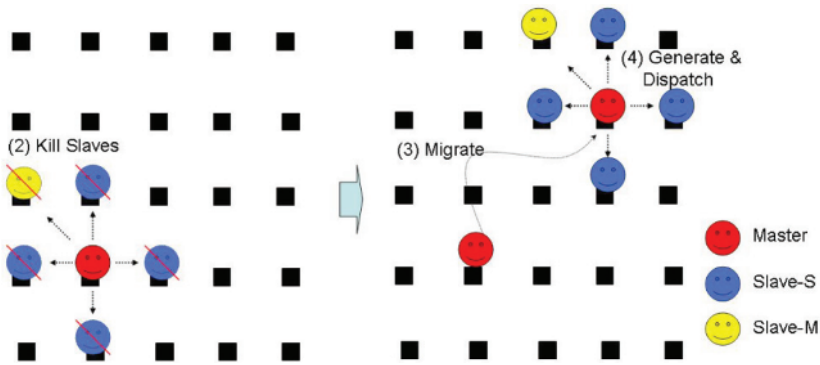


Figure 1.6. *Example scenario based on GDD*

By repeating this process, task execution can be continued at arbitrary locations inside the network. By using GDD, the applicability of tasks comprised of multiple components is increased.

1.5.2.2. *Dynamic components reconstruction*

Dynamic component reconstruction as proposed by Platon *et al.* [PLA 08] also uses tasks as multiple components and adapts the redeployment to the state of the nodes through its Ragilla middleware. In the other existing solutions, the depletion of node batteries or malfunctions of nodes causes tasks comprised of components running on such nodes to fail at fulfilling their requirements. Ragilla, on the other hand, allows the user to define relocation conditions for components (in terms of the number of components, geographical areas, battery level) that control the relocation and reconstruction process in a way that maintains the operational conditions required by each component.

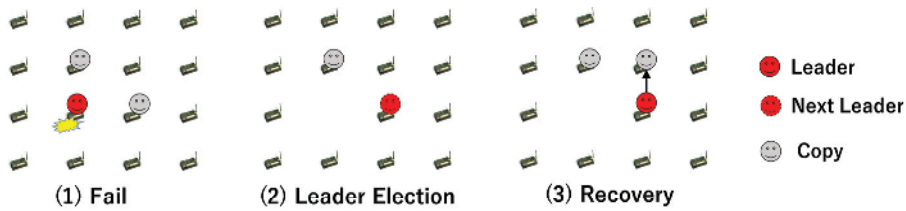


Figure 1.7. *Example scenario based on Ragilla*

For instance, if a hardware failure occurs as illustrated in Figure 1.6, the following steps make it possible to reconstruct the component dynamically at a node within a specified geographical range and continue its sensing task:

1) Because of a hardware failure, a leader component ceases operation.

2) The middleware detects the hardware failure and selects the next leader out of a number of candidate nodes.

3) The middleware also maintains a copy of the leader component and uses it to the selected new node.

By using this method, it becomes possible to improve task execution by reconstructing tasks while maintaining the user's requirements.

Current efforts are not limited to GDD and Ragilla, but they all aim to provide a relocation functionality that adapts dynamically to network conditions, for instance by modeling the role distribution of tasks comprised of multiple components in more detail and making proper deployment decisions based on such component roles.

1.6. Self-adaptation

The situation inside a WSN node, such as the available calculation resource and the battery level, changes over time. Moreover, the situation outside a node, such as the position and velocity of the target and the communication environment, may change as well. In response to these situational changes, the programmer can issue instructions to optimize the behavior of a task. However, if the programmer has to give instructions every time the situation changes, his or her workload would grow too large. To avoid this problem, the middleware should be able to adapt itself to such changes.

1.6.1. *Existing solutions*

1.6.1.1. *Self-adaptability to change inside the node*

Because the processing resource of the node is scarce, it is rather easy for too many tasks to be executed at the same time on the same node. If individual tasks use the nodes of their choice, they would compete for the resources on the nodes and there would come a point when the tasks could not be executed. As described in the section on runtime task management, one task may have to use numerous nodes to, say, improve the quality of an observation. In particular, this means that tasks executed on shared WSNs would frequently compete for the limited resources of the nodes.

There is a certain amount of research dealing with resource competition [HEI 04, DUN 06]. In order to avoid resource competition, each task should make a decision on which nodes to use, and the scheduling of the sensors devoted to the various tasks has to be properly managed.

1.6.1.2. *Self-adaptability to change outside the node*

The task should select the nodes on which it is to run according to the situation. In a pervasive environment, the surroundings of those nodes, such as the presence and number of nearby nodes and the communication links between them, would change dynamically. Moreover, the nodes' surroundings could be affected by other objects. To ensure that the quality level demanded by the task can be met, it becomes necessary to adapt the algorithm and the parameters used by the task in response to the changing situation.

Existing research responds to a changing situation by adjusting the parameters of the communication algorithm [SOH 00, YE 02, SHN 04]. For instance, when a node cannot communicate with another node on the routing tree, these methods restructure the tree so that it can communicate with another node.

1.6.2. *XAC middleware solutions*

1.6.2.1. *Self-adaptive task management*

In the shared WSN, one or more measurement tasks can be used and performed simultaneously, and these tasks can have different accuracy requirements and different priorities. For instance, S2 in Table 1.1 should be used in all rooms, whereas S1 and S3 should only be used in rooms with people in them. The task deployment management should satisfy these requirements as much as possible and sometimes should decide to migrate tasks with lower priorities to other nodes to obey resource constraints. The appropriateness of deployment depends on the

current status of the network and/or the current status of the phenomena to be measured.

We devised a self-adaptive algorithm to select the nodes on which to use components for measurements [NAK 08]. The algorithm selects the minimum number of nodes that can satisfy the accuracy requirements of the task, in accordance with the location of the target to be measured and the locations of the sensor nodes. We also devised a self-adaptive algorithm to manage the components in a node [ISH 06, BOU 11]. The component management algorithm evaluates the utility and constraints of the used components and changes the optimal deployment. We formalized this deployment as an integer linear programing problem and used simulated annealing to find semi-optimal deployment within a realistic amount of time.

1.6.2.2. *Self-adaptive communication management*

We also devised a communication algorithm that changes in response to the environment. When the environment around the node changes depending on factors like the movement of the object, the reliability of communication can be improved by selecting the appropriate algorithm for the environment. For instance, in the task of reporting the present positions of employees, reports are sent from nodes in the various rooms they normally work in. However, the reports are sent frequently from the same node when employees gather for a meeting in one room. When the report frequency fluctuates, the middleware can switch the network protocol, such as the routing or MAC protocol, to a more suitable one to reduce communication costs.

Moreover, we developed a self-adaptive algorithm to tune the parameters of the communication algorithm. Communication traffic can be reduced by aggregating data in the group when a result is sent back from the node group. This has prompted research on selecting appropriate nodes and methods of aggregating data. Thanks to this research, we can efficiently perform data aggregation and communication when the routing algorithm changes.

When the environment around the node changes, these can be changed to more appropriate ones for the circumstances, and the quality of the task can be improved.

1.7. Discussion

XAC middleware is useful in three ways to a context management system in a smart city. First, the multi-level task-description language allows users to specify its measurement task without having to know the low level details of a WSN. The users

of the context management system are usually interested in sensor data obtainable from the network but may not be experts on wireless *ad hoc* networks. Therefore, we think that our language is suitable for these users. Second, the runtime task management enables ones to add or remove measurements task at runtime. In a smart city, applications will be added or removed at runtime; therefore, the context management system should change their measurement tasks in response to these changes. Finally, the self-adaptation feature reduces unnecessary consumption of resources without the need for human intervention. This feature will prolong the network lifetime and reduce the cost of managing WSNs.

1.8. Conclusion

We described XAC middleware that enables programmers to use a WSN as a smart multi-modal sensor, as part of a context management system for smart cities. It provides different kinds of sensor data to different smart city applications at the same time. A middleware approach is effective at reducing costs associated with developing measurement tasks for a WSN. In this chapter, we overviewed the research issues related to a middleware for a shared WSN, from the viewpoint of task-description language, runtime task management and self-adaptation. We believe that the shared WSN will be a key enabler for smart cities.

1.9 Bibliography

[ABD 04] ABDELZAHER T., BLUM B. *et al.*, "Envirotrack: towards an environmental computing paradigm for distributed sensor networks", *24th International Conference on Distributed Computing Systems (ICDCS)*, pp. 582–5894, 2004.

[BOU 03] BOULIS A., HAN C.-C., SRIVASTAVA M.B., "Design and implementation of a framework for efficient and programmable sensor networks", *1st International Conference on Mobile Systems, Applications and Services (MobiSys)*, pp. 187–200, 2003.

[BOU 07] BOULIS A., HAN C.-C., SHEA R. *et al.*, "Sensorware: programming sensor networks beyond code update and querying", *Pervasive Mobile Computing*, vol. 3, no. 4, pp. 386–412, 2007.

[BOU 11] BOURDENAS T., TEI K., HONIDEN S. *et al.*, "Autonomic role and mission allocation framework for wireless sensor networks", *5th IEEE International Conference on Self-adaptive and Self-Organizing Systems (SASO'11)*, pp. 61–70, 2011.

[COS 07] COSTA P., MOTTOLA L., MURPHY A.L. *et al.*, "Programming wireless sensor networks with the teenylime middleware", *ACM/IFIP/USENIX 2007 International Conference on Middleware (Middleware)*, pp. 429–449, 2007.

[CUR 05] CURINO C., GIANI M., GIORGETTA M. *et al.*, "TinyLime: bridging mobile and sensor networks through middleware", *3rd IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 61–72, March 2005.

[DUN 06] DUNKELS A., FINNE N., ERIKSSON J. *et al.*, "Run-time dynamic linking for reprogramming wireless sensor networks", *4th International Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 15–28, 2006.

[FOK 05] FOK C.-L., ROMAN G.-C., LU C., "Rapid development and flexible deployment of adaptive wireless sensor network applications", *25th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 653–662, 2005.

[FRA 05] FRANK C., ROMER K., "Algorithms for generic role assignment in wireless sensor networks", *3rd International Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 230–242, 2005.

[GEL 85] GELERNTER D., "Generative communication in Linda", *ACM Transaction on Programming Language and Systems*, vol. 7, no. 1, pp. 80–112, 1985.

[HEI 04] HEINZELMAN W.B., MURPHY A.L., CARVALHO H.S. *et al.*, "Middleware to support sensor network applications", *IEEE Network*, vol. 18, no. 1, pp. 6–14, 2004.

[HUI 04] HUI J.W., CULLER D., "The dynamic behavior of a data dissemination protocol for network programming at scale", *2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 81–94, 2004.

[ISH 06] ISHIGURO M., TEI K., FUKAZAWA Y. *et al.*, "A sensor middleware for lightweight relocatable sensing programs", *International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA)*, p. 195, 2006.

[KUM 03] KUMAR R., WOLENETZ M., AGARWALLA B. *et al.*, "Dfuse: a framework for distributed data fusion", *1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 114–125, 2003.

[KWO 06] KWON Y., SUNDRESH S., MECHITOV K. *et al.*, "Actornet: an actor platform for wireless sensor networks", *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1297–1300, 2006.

[LEV 02] LEVIS P., CULLER D., "MatÅLe: a tiny virtual machine for sensor networks", *10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 85–95, 2002.

[LEV 04] LEVIS P., PATEL N., CULLER D. *et al.*, "Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks", *1st Conference on Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 15–28, 2004.

[MAD 05] MADDEN S.R., FRANKLIN M.J., HELLERSTEIN J.M. *et al.*, "Tinydb: an acquisitional query processing system for sensor networks", *ACM Transaction on Database Systems*, vol. 30, no. 1, pp. 122–173, 2005.

[NAK 08] NAKAMURA Y., TEI K., FUKAZAWA Y. *et al.*, "Region-based sensor selection for wireless sensor networks", *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*, pp. 326–331, 2008.

[PLA 08] PLATON E., SUENAGA S., YOSHIOKA N. *et al.*, "Transparent application lifetime management in wireless sensor networks", *Demo Track of the 10th International Conference on Ubiquitous Computing (Ubicomp)*, 2008.

[SHN 04] SHNAYDER V., HEMPSTEAD M., CHEN B.-R. *et al.*, "Simulating the power consumption of large-scale sensor network applications", *2nd International Conference on Embedded Networked Sensor Systems*, pp. 188–200, 2004.

[SIM 05] SIMON D., CIFUENTES C., "The squawk virtual machine: Java on the bare metal", *20th Annual ACM Sigplan Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pp. 150–151, 2005.

[SOH 00] SOHRABI K., GAO J., AILAWADHI V. *et al.*, "Protocols for self-organization of a wireless sensor network", Personal Communication IEEE, 2000.

[SUE 09] SUENAGA S., YOSHIOKA N., HONIDEN S., "Generative dynamic deployment of multiple components in wireless sensor networks", *6th International Conference on Wireless On-demand Network Systems and Services (WONS)*, pp. 197–204, 2009.

[TEI 07] TEI K., FUKAZAWA Y., HONIDEN S., "Applying design patterns to wireless sensor network programming", *1st International Workshop on Wireless Mesh and ad hoc Networks (WiMAN) in Conjunction with ICCCN*, pp. 1099–1104, 2007.

[TEI 14] TEI K., RYO S., FUKAZAWA Y. *et al.*, "Model-driven-development-based stepwise software development process for wireless sensor networks", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 4, pp. 675–687, 2014.

[WEL 04] WELSH M., MAINLAND G., "Programming sensor networks using abstract regions", *1st Conference on Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 29–42, 2004.

[WHI 04] WHITEHOUSE K., SHARP C., BREWER E. *et al.*, "Hood: a neighborhood abstraction for sensor networks", *2nd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pp. 99–110, 2004.

[YAO 02] YAO Y., GEHRKE J., "The cougar approach to in-network query processing in sensor networks", ACM *SIGMOD Record*, vol. 31, no. 3, pp. 9–18, 2002.

[YE 02] YE W., HEIDEMANN J., ESTRIN D., "An energy-efficient mac protocol for wireless sensor networks", *21st Conference on Computer Communications (INFOCOM)*, pp. 1567–1576, 2002.