COMiS : Component Oriented Middleware for Sensor Networks

D. Janakiram, R. Venkateswarlu and S.Nitin
Distributed and Object Oriented Systems Lab,
Department of Computer Science and Engineering IIT Madras,
Chennai, India 600036.
{djram, rv, nitin}@cs.iitm.ernet.in

Abstract— **Recent advances in wireless technology have enabled the rapid development of wireless sensor networks. Such networks, consisting of ten to thousands of randomly deployed nodes collaborating to achieve a goal, are used in a variety of applications. However, due to extreme resource constraints and lack of suitable programming abstractions, programming sensor networks becomes a tedious process. This coupled with the event driven nature of the applications necessitates different programming paradigms for the middleware design. The literature survey shows that existing middleware are application specific. This is due to peculiar characteristics of sensor networks(such as, nodes are prone to failures, restricted resources, etc). The generic middleware is required to develop a wide variety of applications in an efficient manner. So there is a strong need for generic middleware rather than application specific. This paper presents a generic middleware known as COMiS(Component Oriented Middleware for Sensor Networks).**

The middleware(COMiS) is developed in terms of components, to satisfy the resource constraints such as memory and power. The components of middleware may be different at different nodes based on the functionality of the sensor node. So components are loaded into memory based on the application semantics. Middleware also provides services such as discovering k components, discovering components within distance d, registration, component updation, power management, etc.

This paper discusses the design and implementation of COMiS middleware. This paper also presents the comparison of existing middleware with respect to programming ease, modularity, adaptability and code size.

Keywords: Programming Languages, Middleware, Paradigm, Components, Styles, Sensor Networks, Programming Abstractions

I. Introduction

Wireless sensor networks randomly deploy tens to thousands of sensor nodes. Each sensor node has a separate sensing, processing, storage and communication unit. The position of sensor nodes need not be predetermined. This allows random deployment in inaccessible terrains or disaster relief operations.

The characteristics of sensor networks[1] are :

- It contains several thousands of nodes.
- Sensor nodes are prone to failures.
- The topology changes very frequently due to node failures.
- Sensor nodes have constrained resources.
- Sensor nodes mainly use a broadcast communication paradigm.

Challenges in Sensor Networks

- **Restricted Resources :** Sensor network has constrained resources such as energy, computing power, memory and bandwidth.
- **Dynamic Networks :** Due to node mobility, environmental obstructions, restricted resources, etc, the the sensor networks exhibit a highly dynamic network topology.
- **Scalability :** The sensor network should scale from ten to thousands or millions of sensor nodes. This needs automatic-configuration, maintenance, upgrading of individual devices.
- **Integrating with Real World :** Sensor networks can be used to monitor real world phenomena. Hence, identifying time and location in sensor networks is crucial [2].
- **Uncertainty in Sensor Readings :** Signals detected at physical sensors have uncertainty due to limitations of the sensor, and they may contain

environmental noise.

Middleware components have to be light weight to fit the constraint such as restricted resources(limited memory, energy and band width). In order to meet the scale of deployment challenge, the middleware should support mechanisms for self-configuration and self-maintenance of collections of sensor nodes.

Distributed systems [3], [4], [5] including the CORBA Component Model [6], Java RMI, EJB(enterprise java beans) and Microsoft's COM [7]. These components are specified by the interfaces they provide or use. The component models are more heavy weight in terms of resources(because memory and power are constrained). So component models(heavy weight) are not used in sensor networks.

Much work has been done to the development of new protocols that promote efficient utilization of resources such as bandwidth, power and memory. Middleware provides interfaces to the application developer, and hides the details of communication protocols, in order to ease the developing of application.

Most of the middleware(except mate, sensorware, etc) assumed that algorithms are hard-coded into memory of the sensor node. After few months, if we want to change the behaviour of sensor network then it is not feasible manually, to load the algorithm into memory of each sensor node, because usually sensor network contains ten to thousands of nodes. COMiS supports the reprogramming, that is, dynamically changing the behaviour of sensor network.

Some of the middleware approaches viewed the sensor network as a distributed database, in which every site is a sensor node. These middleware does not support the reprogramming. Even though database approach is used for developing the applications, but its expressiveness is limited. COMiS is aimed at achieving the distributed programming. The existing middleware are totally platform dependent andor application specific.

The remainder of the paper is organized as follows. Section II explains the overview of our project. Background work is explained in section III. Section IV explains the architecture and components of COMiS, section V discusses the implementation details of COMiS. Section VI presents the survey of related work, and section VII concludes this paper with an out look on future directions.

II. Overview of the TinyMaCLaS Project

TinyMaCLaS project is aimed at achieving high degree of flexibility and programming accross sensor nodes(distributed programming). Flexibility means, ease of developing applications. Distributed Compositional Language(DCL) is a part of TinyMaCLaS project. We have implemented a "Distributed Compositional Language" [8] for wireless sensor networks that represents a new programming paradigm for application developers. This language simplifies the process of implementing applications in sensor networks. The emphasis, in this language is on styles. The styles specify the interaction of components. We have shown, how styles such as Event style, Pipe style and Group Communication style, can be used in sensor networks. The novel idea in DCL is, its ability to capture the interactions among components, which reside in different sensor nodes. That is, the application developer is not aware of the low level communication mechanism. The goal of DCL is, to provide ease of developing applications in sensor networks.

Component Oriented Middleware for Sensor Networks(COMiS) is also part of TinyMaCLaS project. The middleware is developed in terms of components, to satisfy the resource constraints such as memory and power. The components of middleware may be different at different nodes based on the functionality of the sensor node. So components are loaded into memory based on the application semantics. Middleware also provides services such as discovering k components, discovering components within distance d, registration, component updation, power management, etc.

III. Background Work

The component interactions is the key issue in Distributed Composition Language. These components are basically small software entities. This means, the design of the component is simple. Software components are black-box abstractions. Every component provides a set of services and may also need a set of required services. Components communicate with each other through well defined ports(methods).

A. Runtime Environment

The Runtime Environment mainly consists of

- Symbol-Table
- Glue-Component

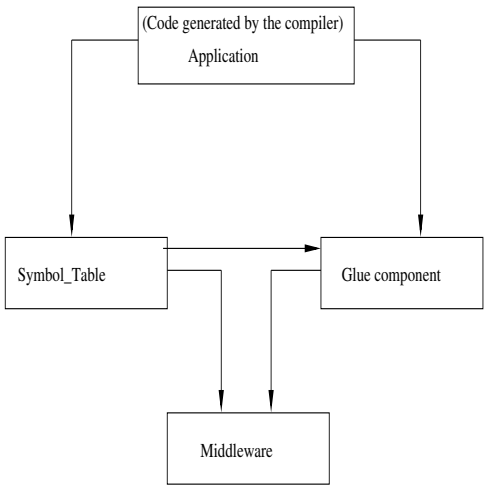

Fig. 1. Runtime Environment

- Middleware

Symbol-Table: This stores all the symbols defined in the script(program). It also stores the mapping of the symbols associated with the components. So, whenever a symbol is referred then its associated component name is returned. The referred components are stored for future use.

Glue-Component: The Glue-Component provides the glue code required in the language to implement and support different styles. The implementation of the semantics of the styles are done in the Glue-Component. Glue-Component also refers to the Symbol-Table to resolve the symbols. This is used both at compile time and runtime. At run time the Glue-Component provides different interfaces for each style where the component interactions are implemented.

Middleware : Glue-component uses the interfaces provided by middleware. Component Oriented Middleware(COMiS) is used with this language. The middleware is developed in terms of components, to satisfy the resource constraints such as memory and power.

B. Implementation Details :

We have implemented DCL(Distributed Compositional Language), which occupies 40KB of memory. It is some what similar to a scripting language. We have used Lex and Yacc tools to implement this language. The output of the compiler is a C file. The C file is compiled by the C compiler. So finally we get the binary file, which is executed in the sensor node. This language uses a middleware called COMiS (Component Oriented Middleware for Sensor Networks), which occupies 56KB of memory. Components are developed according to the application semantics.

C. How to specify styles?

This section explains how to declare components and how to specify the styles.

def Component com1 = login

In the above declaration, def and Component are keywords of language, where as login is a component. When parser encounters a def keyword, it understands that the statement is a declaration of component. The com1 is a variable associated with login component.

run: com1.transfer() → com2.verify()

In the above declaration, assume com1, com2 are two components. → arrow is called operator or connector and run is a keyword. The above style specifies the interaction of components, component2 is receiving data from component1. **Event Style:** It is truly asynchronous in notifying the generation of the events [9]. The sensor nodes are listening (subscribing) for an event. Some other nodes in the network may be listening for same event. The event generation could be anywhere in the network. The syntax of the event style [10] is
event:eventType ? listenerObjA.

def Component temp-event=Temperature

def Component observer=Observer

run:temp-event ? (Then) observer

Here, event and observer are components. The interaction between components is event style. If event(e.g temperature is greater than 70^0C) occurs then notify the observer. We can use ? or Then as an operator.

IV. Component Oriented Middleware for Sensor Networks

A. Architecture and Runtime Environment of COMiS

Figure 2 shows the System Model. The middleware resides between the operating system and user applica-

tions. The upper layer contains application components distributed across different nodes in the WSN. These applications use various strategies to achieve a common task of gathering environment information and routing it to a centralized base station. The lower layer contains the footprint of the middleware present in the node. This may vary from node to node depending on the application running on it.

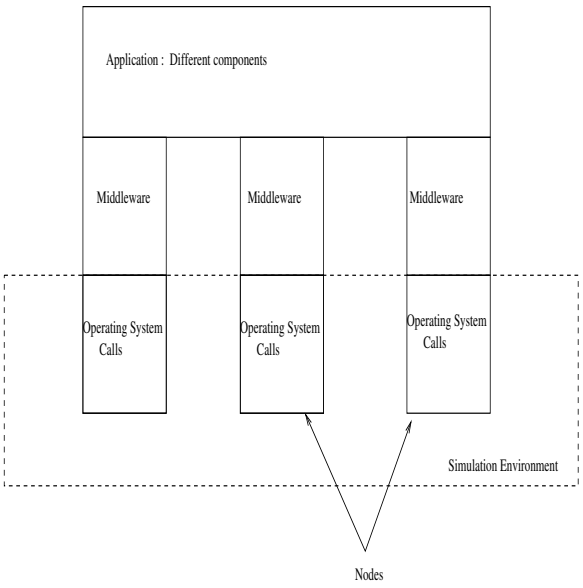

Fig. 2. Framework of the System Model

Figure 3 shows the run time environment of an application running at a node. The basic middleware layer contains six components: *listener, discovery, send, register, update* and *power management*.

TABLE I

MIDDLEWARE PACKET HEADER

Field	Purpose
DEST_TYPE	Defines the type of destination.
MSG_TYPE	The type of message, different message types are shown in Table II
SOURCE_ADDR	The nodeID(IP Address) of node sending packet.
DEST_ADDR	The nodeID(IP Address) of destination node. "255.255.255.255" indicates BROADCAST.
MSG_ID	The count of the number of the messages sent from source node.
TTL	Time To Live. Number of hops, after which the packet becomes invalid(applicable only for BROADCAST).
K	If the destination type is first K nodes.
D	If the destination type is all nodes within D hops.
SOURCE_COMP	Sending Component.
DEST_COMP	Destination Component.
DATA	The data being transmitted depending on the packet type.

TABLE II

MIDDLEWARE MESSAGE TYPES

Field	Message Type
DISCOVER	The packet is a discovery packet. DEST_COMP is the component to be discovered.
DISCOVERED	The packet returns the nodeID of discovered component SOURCE_COMP.
EVENT_NOTIFY	The packet notifies of a particular event within th network.
DATA_PKT	The packet is solely to exchange DATA.
CONTROL	The packet is used for transmitting control information in the network, such as node up/node down.

B. Components of COMiS

Packet Headers The different fields of the packet header are shown in table I.

1) Listener: The Communication Components provide communication primitives for component interaction. These components use the lower-most components *m*essage(such as send and receive) and *t*imer provided by OS. The listener component is responsible for receiving all incoming packets. Its method *listen* is initialized as a separate thread and continuously receives messages. On receiving a packet it may perform depending on the type of the packet:

1) If the message is intended to itself, it adds the message to the buffer.
2) If the message is of type *DISCOVER*, then it checks whether the component is present locally, and if yes returns a *DISCOVERED* message.
3) If the packet is broadcast, then *listen* forwards the packet to its neighbors until *TTL* = 0.

The method *retrieveMsg* retrieves a message of the specified type from the received message buffer. Thus although at the Operating System Layer the receive is of synchronous nature, the middleware provides a non-blocking receive of messages.

2) Register: Register component is responsible for registering a component locally. It makes an addition into the *localRepository*, and henceforth the component will be available for discovery.

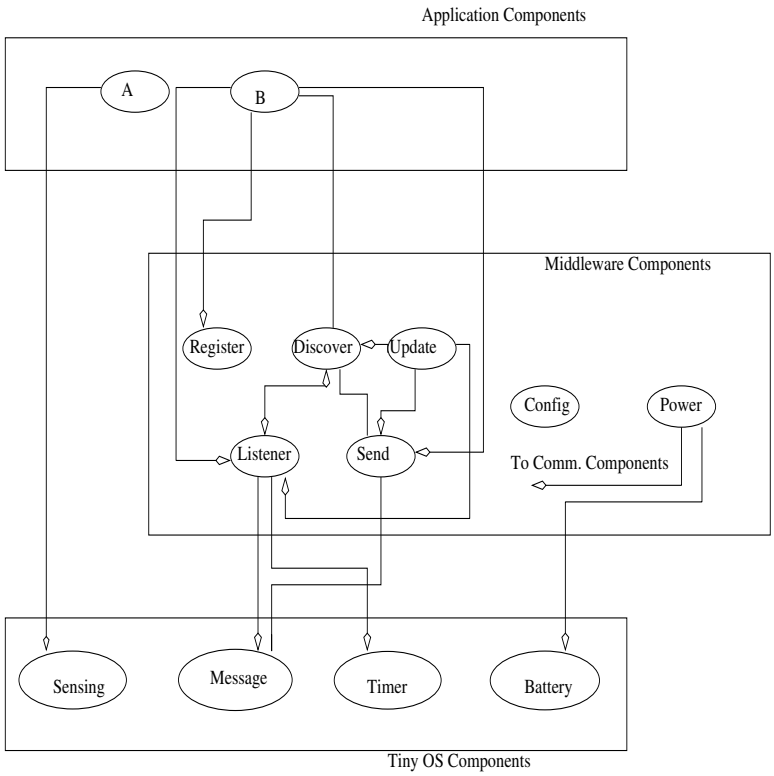

Fig. 3. The Run Time Environment

3) Discovery: Remote components registered locally can be discovered using the discovery component. Discovery runs a simple broad-cast based algorithm locating components across the network. Discovery provides a wide range of look up semantics. This is needed because different nodes may contain the same component. In some applications it may be necessary to send messages to a subset of the nodes, which contain a certain component. After discovery a unique *connectionID* is returned which can be used for further reference to the components that have been discovered.

4) Send: Once remote components are discovered, a unique connection ID is returned. The send component is used to send data to all components part of this connection.

5) Update: The update component is used for component deployment and updation. Compiled Binaries of Components are deployed into the network and installed at nodes where updation/deployment are to be performed. First a connection is established with a subset of nodes in the network using the discovery component. Then using the update method components are deployed into the network. On receiving an update at a node, the listener thread checks the version number of the arrived component against the installed component. If the arrived component's version number is greater than the installed component, the binary component is installed and a relinking of the modules are performed.

The advantage of the component oriented middleware architecture with respect to component updates is that software updates can be performed on middleware components as well. Moreover this eliminates the need for manually updating the middleware at all nodes. Thus by allowing making changes to middleware, a certain amount of adaptivity is incorporated in the system.

6) Power: The middleware extends the power routines provided by the OS.

V. Implementation Details of COMiS

A. Data Structures And Global Variables

This section discusses the data structures and global variables used in the implementation.

Neighbor List: This list is maintained by the OS. It contains the *nodeIDs* of all nodes within radio range of the node. This information is crucial for implementing the *broadcast* protocol at the OS level.

Local Component Repository: All components present at a node available for remote reference is part of this data structure. An entry in the *localRepository* is of the form *(componentName, moduleName, version)*. Thus all remote reference to the component is made through the *localRepository*. Updations to local components are also made after comparing the version number of the existing component's version number with the incoming components version number.

Remote Component Repository: Discovered components are entered into the *remoteComponentRepository*. An entry in this data structure is of the form *(connectionID, componentName, DISCOVERY_TYPE, listOfNodeIDs)*. *ConnectionID* is a unique reference to the discovered components and is used by *send* and *update*. *ListofNodeIDs* is a list which maintains the set of nodes part of this connection. Thus if a *send* is called on *connectionID*, the message is sent to all nodes part of this list.

Message Buffer: The *listener* stores incoming messages into the *msgBuffer*, and thus can be retrieved by components in a non-blocking manner using *retrieveMsg*.

Number Of Local Components: Keeps track of the number of components locally present on a node.

Number Of Remote Components: Keeps track of the number of remote mappings maintained.

Number Of Packets Sent: Maintains the number of packets sent by the middleware. The allowed message types are shown in Table II.

B. Algorithms for Implementing the Middleware

This subsection explains the main algorithms employed in implementing the different middleware components.

1) Discovery: Discover(componentName, discoveryType):

All components within D hops:

```
1. Set Timeout based on D
2. Initialize and Broadcast
      Discovery Packet.
3. Make new Entry in the
  Remote Component Repository.
4. Until(TimeOut)
 4.1 Retrieve message of type DISCOVERED
  with SOURCE_COMP componentName.
 4.2 Add the source address to the list
   of node IDs of connection
5. Return Connection ID.
```

At remote Node on receiving a discovery Packet

```
1.If DEST_COMP locally present, then
  send DISCOVERED with SOURCE_COMP
  componentName to SOURCE_ADDR of pkt.
2. If TTL of packet > 0
 2.1 TTL = TTL -1
 2.2 Broadcast packet to all Neighbours.
```

2) Update: Update(componentName, moduleName, version)

At the node sending update

```
1. Discover component and version across all
   nodes in the network - connectionID.
2. Remove nodes from connectionID which have
   version number >= current version number
3. Compile the module corresponding
   to the component to be updated.
4. Pack the component binary in a message
   and send to each node in connectionID.
```

At the remote Node

```
1.Receive New Component Packet.
2.If new component version number
   > existing version number
  2.1 Install new binary component
      in place of old.
  2.2 Relink the object file.
  2.3 Restart the application.
```

C. Initialization

The *initialize* method is called by an application before using middleware routines. It is also called when an update is performed to a middleware component. The functions performed by initialized are as follows:

```
1.Initializes the operating system
 components such as Sensor and Message.
2.Registers middleware components
   in the Local Repoitory.
3.Initializes all global variables
   and data structures.
4.Starts the listener thread.
```

D. Performance Analysis

Table IV shows the individual middleware component sizes:

TABLE IV
MIDDLEWARE CODE SIZE

Component	Code Size(in bytes)
Listener	8000
Register	5100
Send	7500
Discover	12000
Update	6400

Table III shows the comparison of different middleware based on the size, flexibility, adaptability, modularity and ease of programming. The comparison is made against the Distributed compositional language applications running on the Component Oriented Middleware(COMiS).

VI. RELATED WORK

TinyDB [11] approach treats the sensor network as a virtual database. Each sensor node contains TinyDB(tiny database) and this database is queried by using SQL like query language called Tiny SQL. This approach is very much useful for data aggregation related applications, but it can not efficiently handle the collaborative applications such as target tracking, measuring temperature of a small portion in a region, etc. But where as COMiS middleware can efficiently handle the collaborative applications [8].

Cougar [12] and SQTL [13] are viewed sensor networks as a distributed database, in which each site is a sensor node. These approaches does not support the reprogramming and can not efficiently handle the complex applications. But where as COMiS supports the reprogramming.

Mate [14] is a virtual machine with a purpose of reprogramming the sensor network. Reprogramming means changing the behaviour of the sensor network on the fly. Mate is used for deploying the component throughout sensor network, but it has limited set of instructions, so expressiveness is too limited when compared to the COMiS.

MiLAN is aimed at allowing applications to provide a specified QoS level to applications. MiLAN [15] is not suited for networks deployed in inaccessible regions, but where as COMiS can be used because of reprogramming.

Impala [16] is used in dynamic sensor networks, where node failures and network disconnectivity are frequent. However complex sensing applications such as target tracking, etc are difficult to implement using Impala. But the COMiS can easily handle the complex applications.

SensorWare [17] is suitable for most of the applications but it is a heavy weight for sensor node(mote). So COMiS is not heavy weight because of light-weight components.

A data-centric service middleware in sensor networks called DSWare[18].Data caching(more caching) and storage is not possible in a sensor node and parsing overhead for SQL like statements.

Enviro Track is an object−based distributed middleware system that raises the level of programming abstraction by providing a convenient and powerful interface to the application developer[19]. But this middleware is a heavy weight.

VII. CONCLUSIONS AND FUTURE DIRECTIONS

We believe that COMiS(Component Oriented Middleare for Sensor Networks) is well suited for sensor networks, in order to satisfy the resource constraints. We also believe that TinyMaCLaS can enable a wealth of new sensor based services. All existing middleware are specific to a particular set of applications. So there is a need for generic middleware, to develop a wide range of applications. There is a trade-off between expressiveness of middleware(middleware is not an application specific) and performance. This is because, as the middleware becomes more general then the performance degrades. There is a strong need for a programming abstraction that simplifies application development while still maintaining flexibility. DataMining is one dimension to look at the middleware. COMiS supports for heterogeneous sensor networks and collaborative applications.

REFERENCES

[1] I.F.Akyildiz, W.Su, Y.Sankarasubramaniam, and E.Cayirci, "Wireless sensor networks: a survey," in *Computer Networks*, pp. (38)393–422, 2002.

[2] K.Romer, "Programming paradigms and middleware for sensor networks," in *GI/ITG Workshop on sensor networks, Karlsruhe, Germany*, pp. pp. 49–54, February, 2004.

[3] A. Herbert, "An ansa overview ieee network," pp. 18–23, 1994.

[4] ISO/IEC, International, Standard, and 10746-3, "Odp reference model architecture," 1995.

[5] S. Microsystems, "Enterprise java beans," in *Available at http://www.sun.com.*

[6] "Object management group, corba component model," in *Available at http://www.omg.org*.

[7] "Ole2 programmer's reference," in *Volume one. Microsoft Press*, 1994.

[8] D. Janakiram and R. Venkateswarlu, "A distributed compositional language for wireless sensor networks," in *IEEE Conference on Enabling Technologies for Smart Appliances (ETSA)*, 2005.

[9] A. Vijay, Srinivas, D. Janakiram, R. Koti, and A. U. Kumar, "Realizing large scale distributed event style interactions," in *the Proceedings of the ECOOP Workshop on communication abstraction for Distributed Systems*, 2004.

[10] A. Uday.Kumar, "Design and implementation of distributed object composition language," in *M.Tech Thesis, IIT Madras*, 2004.

[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," in *Proc. the 5th OSD*, December 2002.

[12] Yong.Yao and Johannes.Gehrke, "The cougar approach to in-network query processing in sensor networks," in *ACM Sigmod Record*, p. 31(3), September 2002.

[13] Chaiporn, Jaikaeo, Chavalit, Srisathapornphat, and C.-C. Shen, "Querying and tasking in sensor networks," in *SPIE's 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Control (Digitization of the Battlespace V), Orlando, Florida*, April, 2000.

[14] P. Levis and D. Culler, "Mate: A tine virtual machine for sensor networks," in *Proceedings of the tenth International Conference on Architectural Support for Programming Languages and Operating Systems ,ACM press, New York*, pp. 85–95.

[15] Wendi, B.Heinzelman, A. L.Murphy, Hervaldo, S.Carvallo, and M. A. Perillo, "Middleware to support sensor network applications," in *IEEE Network Mag.*, pp. 18(1):6–14,, 2004.

[16] T.Liu and M.Martonosi, "Impala:a middleware system for managing automatic,parellel and sensor systems," in *ACM SIGPLAN Symposium on Principles and Practice of Parellel Programming(PPoPP)*, 2003.

[17] A. Boulis, C. Han, and M. B. Srivastava, "Design and implementation of a framework for programmable and efficient sensor networks," in *MobiSys 2003, San Francisco, USA*.

[18] S.Li, S.Son, and J.Stankovic, "Event detection services using data service middleware in distributed sensor networks," in *Proceedings of the 2nd Internnational Workshop on Information Processing in Sensor Networks*, 2003.

[19] T.Abdelzaher, B.Blum, Q.Cao, Y.Chen, D.Evans, J.George, S.George, L.Gu, S.Krishnamurthy, L.Luo, S.Son, J.A.Stankovic, R.Stoleru, and A.Wood, "Envirotrack: Towards an environmental computing paradigm for distributed sensor networks," in *ICDCS 2004, Tokyo, Japan*, 2004.

TABLE III

COMPARISON OF THE DIFFERENT MIDDLEWARE

Middleware	Size	Applications	Adaptability	Modularity	Programming Ease
TinyDB	70K	Data Gathering/Monitoring	NA	NA	high
SensorWare	NA	Data Gathering, Tracking	low	low	medium
Impala	NA	Monitoring, Tracking	OS level	high	medium
COMiS	56K(total)	Monitoring, Control Apps, Tracking	Application level	high	high
Mate	27K	Tracking, Monitoring, Control Apps	OS level	high	low