Data-Centric Programming Environment for Cooperative Applications in WSN

Shunsuke Mori*‡, Takaaki Umedu* †,
Akihito Hiromori* †, Hirozumi Yamaguchi* † and Teruo Higashino* ‡
*Graduate School of Information Science and Technology, Osaka University
†Japan Science Technology and Agency, CREST
‡Research Fellow of the Japan Society for the Promotion of Science

***Abstract*—**

Due to recent evolution of MEMS technology, wireless sensor nodes will be computing nodes having more storage space and processing power, and wireless sensor networks (WSNs) will be more capable of processing complex, cooperative tasks just in time, without relying on cloud servers which may cause delay and consume WSN bandwidth. However, low level implementation of cooperative applications onto individual sensor nodes needs a lot of human efforts, and has a considerable gap with high-level requirement given by application developers. To fill the gap, we propose a support environment for WSN applications development. We design a language to describe high-level specification of cooperative applications on WSN and provide an algorithm that translates the given high-level specification into program codes of sensor nodes. We have shown some example descriptions of high-level specifications and have evaluated the quality of generated programs through several experiments.

I. Introduction

Sensing real-world phenomena, activities of people and environments will be more significant for next generation affluent and ubiquitous life and society. Wireless Sensor Network (WSN) is considered as a key platform for such a purpose. However, wireless sensor nodes should be more intelligent and cooperative to reduce traffic volume and delay to scale as cyber-physical computation becomes more essential and amount of sensor readings accordingly becomes larger. On the other hand, each node should act to accomplish collaborative behavior in fully decentralized, heterogeneous environment for data collection and delivery, where sensor readings are processed and routed among sensor nodes to enable local and collaborative event processing. However, the implementation of such collaboration requires the designers to make enormous effort since writing codes of collaborative sensor nodes in a node-centric way, while keeping the global, data-centric behavior in mind, is extremely a hard task.

In this paper, we propose a methodology to support design and development of collaborative WSN applications. The approach provides a language to specify the high-level behavior of applications without referring to the real deployment of sensor nodes, and an algorithm to automatically translate the given application specification into a platform-dependent program code of each sensor node. We provide a set of event sensing and communication primitives to achieve the given specification in WSN.

The application behavior may include time, location and network-based constraints (conditions) on event occurrences and their processing, and the description is independent of the physical placement of sensor nodes. We provide a concept that hides the details of wireless sensor network configuration, communication and processing inside the network but all the event occurrences are visible to the virtual node. In this architecture, the specification is given as a program on this node specifying pre- and post-conditions of events which are carried out by collaborative nodes in WSN. The translation algorithm automates design and implementation of complex cooperation protocols from this developer-friendly form of behavior specifications.

II. Related Work

Several approaches for WSN systems have been presented so far that support entire process of design and development [1], [2], [3]. Woehrle et al. [1] have proposed a procedure for systematic and strategic testing of target applications to verify robustness and reliability of the applications. Liu et.al [2] have proposed a method to break a given single program down into several pieces that are executed by multiple nodes in ad-hoc networks. MacroLab [3] can also derive distributed codes from a given single program, and the developers can concentrate on designing policies to collect sensor readings and manipulate them. However, the above approaches do not provide the concept of design support for cooperative event processing with time-, location- or network-dependent conditions.

In this context, the most relevant approache with ours is Ref. [4]. It proposes a set based programming approach where requirement is given by a set of nodes, a set of sensor values and so on. However, the most significant difference is Ref. [4] basically adopts a node-centric view of programming, while we allow a node-independent approach where a specification can be fully-independent of nodes and networks including neighbors and sink nodes (our scheme allows higher abstraction in other words). Cooperation among nodes to implement such a specification is more complex and challenging, *e.g.* cluster heads should appropriately be chosen to collect and process sensor data if necessary, networks should dynamically be built and sensor data should be routed efficiently in a fully decentralized environment. We believe this is the first approach to consider such highly-abstracted specifications and provide cooperative, cost-effective solutions to achieve the given requirement in WSN.

978-3-901882-50-0

III. Approach Overview

A. Outline

Our method can derive a program code of each node on WSNs for a given specification that describes actions to be taken by a group of nodes and conditions to be examined before the actions. Developers can easily describe applications by specifying such conditions that should be checked by cooperation of nodes.

We show a simple but essential example in Fig. 1 where a group of nodes that satisfy the following conditions is defined with predicates (the excerpt of them are shown in Table I) as the first group to detect a fire; (1) all the nodes in the group have detected temperatures higher than $40°C$, (2) they are located in a circle with 100m diameter and (3) the average temperature by 30 or more nodes in the group is higher than 50 $°C$. In addition, in order to alert the approach of fire, the second group of nodes (*EstimatedFireSpot*) is defined. *EstimatedFireSpot* has the similar center with the previous group (*DetectedFireSpot*) but its radius is five times larger than the maximum distance (diameter) between nodes in *DetectedFireSpot* group. The nodes in *EstimatedFireSpot* group execute its action defined with functions (the excerpt of them are shown in Table II) to warn people to escape from the fire . In this way, conditions on geometry, sensing data values and their manipulations can be written in our specification.

However, such a specification is not easy to implement since checking conditions and executing actions need cooperative operations among nodes. For example, in order to check a condition on sensing data, (i) a node group with a leader node needs to be organized, (ii) the sensing data needs to be collected onto the leader node, and (iii) it needs to be checked if the condition is met or not. The action should be executed if the condition is satisfied, or the group is dismissed.

Thus, our method can automatically derive the program code of cooperative nodes. This hides the details of node behavior, which are often complex, from the developers. Therefore they can concentrate on application logic.

```
nodegroup DetectedFireSpot
  condition:
      TestEach(temperature, ">40")
      && InFloatCircle(100)
      && AverageSelect(temperature, 30)>50
  action:
      centroid = GetCentroid()
      diameter = GetDiameter()

nodegroup EstimatedFireSpot
  condition:
      InGeoCircle(DetectedFireSpot.centroid,
                  5·DetectedFireSpot.diameter)
  action:
      ExecuteEach("ActivateAlert()");
```

Fig. 1. Specification of Fire Detection and Alert System

B. Code Derivation for WSN nodes

We discuss about the code derivation for WSN application from given specifications. Fig. 2 shows the architecture of sensing application which our method supports to development. It is composed of sensor nodes which have positioning systems

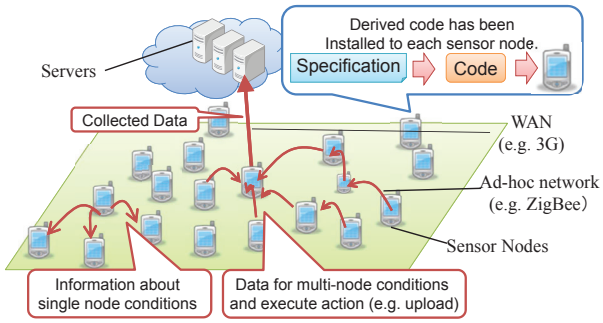

Fig. 2. An Architecture of Sensing Application Developed by Proposed Method

(e.g. GPS), ad-hoc communication facilities (like Zigbee or Bluetooth), and WAN devices (e.g. 3G). Thus, each node can get its location information and communicate with neighbor nodes and servers. Our method generates node programs and developers install them to these sensor nodes.

For a given specification described by a set of nodes with pre-conditions and post-actions, we classify the predicates that constitute the condition into two categories, single-node predicates and multi-node predicates. An example of single-node predicate is *TestEach* that checks if variable on each node satisfies a given condition (see *TestEach*(temperature, "$>$ 40") in Fig. 1). Meanwhile, both *InFloatCircle(100)* and *AverageSelect*(temperature, 30)$>$50 are multi-node predicates since they cannot be examined by single nodes. For example, *InFloatCircle(100)* needs distance calculation for every pair of nodes, meaning that it can be checked only when a group of nodes is given.

Considering this fact, we take the following strategy; Firstly, we let each node periodically check single-node predicates, and let the node be a potential constitute of the group if it satisfies the conditions. If a node becomes a potential constitute of the group, it establishes a link with neighboring potential constitutes of the same group if any. This is done by periodic neighbor discovery messages by each potential constitute. These nodes finally form a tree with a *leader node*. The choice of a leader node is simply done by determining link direction (parent-child relation), and the root node can be the leader node. Then the data values to check the multi-node predicates are collected to the leader node, and the node checks if all the multi-node predicates are satisfied or not. If true, those nodes take actions as specified. Moreover, we allow to describe conditions of groups that depend on some other groups. For example, the second group (*EstimatedFireSpot*) in Fig. 1) is such a group that refers to the "center" and "diameter" of *DetectedFireSpot* as a part of its conditions. In this case, the centroid of coordinates of nodes in *DetectedFireSpot* and the diameter between nodes in it has been calculated by the leader node of *DetectedFireSpot* to prepare for creation of *EstimatedFireSpot*, and the information is broadcast to potential constitutes of *EstimatedFireSpot* (in this case, all the nodes). Our method derives codes according to this strategy.

Therefore, our method generates a programming code that corresponds to each following step of the sequence according to the given specification: (i) periodic sensing from sensors, (ii) periodic evaluation of single-node predicates, (iii) tree construction (potential group generation) and leader election, (iv) data collection on the tree, (v) evaluation of multi-node

TABLE I. PREDICATES FOR CONDITION PART (EXCERPT)

Type	Predinate	Description	Examined by
General	TestEach(v, exp)	true *iff* variable v satisfies exp at every node	Single
Location	InGeoCircle(c,r)	true *iff* all the nodes in the group are within the circle centered at c with radius r	Single
Topology	InFloatCircle(d)	true *iff* all the nodes in the group are within a circle with diameter d	Multi
Topology	Size(min, max)	true *iff* the number of nodes in the group is in [min, max]	Multi

TABLE II. FUNCTIONS FOR VALUES AND ACTIONS (EXCERPT)

Function	Description
Average(v)	Calculate the average of variables p among all the nodes in the group
GetCentroid()	Calculate the centroid of the coordinates of nodes in the group
GetDiameter()	Calculate the maximum distance between nodes in the group
Sleep(t)	sleep in t
UploadData(d, c)	Let exactly one node in the group upload d through network interface c

predicates and (vi) execution of actions, to check if conditions are satisfied or not, and to execute actions if satisfied. To derive program codes, our method extracts predicates and parameters from given specifications, chooses modules corresponding to the extracted factors from existing code modules, and embeds them into appropriate parts of a skeleton code, which is composed of blocks for each of the steps.

IV. PERFORMANCE EVALUATION

We first demonstrate the benefit from our method in terms of developers design effort-saving. This is done by comparison of a given specification and the derived code in terms of simplicity and readability. Then we measure the communication performance in order to show that automatic derivation algorithm can derive a reasonable code.

A. Application Examples and Lines of Code Comparison

We consider two applications. The first one is simple and similar with the fire detection and alert system in Figs.1, but it can present applicability of our method to various applications. It is a noise detection system (Fig.3) where environmental noise is monitored by sensor nodes. If a sensor node detects a certain noise, then the sensor nodes in the surroundings are organized to calculate the average noise level and upload it. Each node has a facility to upload data to base station, but we would like to limit the number of nodes to upload data to only one node in the group since duplication of reports means waste of computation and communication resources. There are two groups called *Initiator* and *SensorGroup* which represent the first-detector of noise over 80db , and the group of sensor nodes in its surrounding area, respectively. `AREA_RADIUS` is a system parameter of the target area radius (constant).

Another example is a crowd estimation system. In a theme park or huge exhibition space, we assume each visitor is given a battery-operated dedicated information terminal. We simply call it *node*. The objective of using such a device is to exploit location-based guidance or navigation and to obtain crowd information (*e.g.* how each attraction is crowded in a theme park in real-time). Each node broadcasts its position to neighboring nodes, and some nodes collect neighbor positions, generate people crowd information (*i.e.* perform crowd

```
nodegroup Initiator
  condition:
      TestEach(noiseLevel, ">80db")
      && Size(1,1)
  action:
      centroid = GetCentroid()

nodegroup SensorGroup
  condition:
      InGeoCircle(Initiator.centroid, AREA_RADIUS)
  action:
      UploadData(Average(noiseLevel),BS);
```

Fig. 3. An Example Specification of Noise Detection Application

```
nodegroup CellCrowdEstimator
  condition:
      TestEach(neighborCounter, ">5")
      && ( InGeoRectangle(c(0,0), c(1,1))
             || InGeoRectangle(c(1,0), c(1,2))
             ...
             || InGeoRectangle(c(m-1,n-1),c(m,n)) )
      && Size(10, INFINITY);
  action:
      UploadData(EstimateDensity(neighborCounter),
                          3G_INTERFACE)
```

Fig. 4. An Example Specification of Crowd Estimation System

TABLE III. LINES OF CODES (LOCS) COMPARISON

Applications	Specifications	Derived Codes
Noise Detection	18	547
Crowd Estimation	17	443

estimation) and report them via 3G networks. The specification is given in Fig. 4. *CellCrowdEstimator* is a group of nodes that estimate the density of a crowded cell. We assume a region is divided into square cells, and (i, j)-th cell is specified by *InGeoRectangle* predicate with a pair of left–bottom coordinate $c(i, j)$ and right-up coordinate $c(i+1, j+1)$. In the condition of *CellCrowdEstimator*, a group of nodes where (i) each node detected more than 5 neighbors, (ii) all the nodes exist in a same cell, and (iii) the number of nodes in the group is at least 10, is organized, and one node in the group is selected to calculate the estimated density of the cell from the information sent by the group member, and report it via 3G network.

The node programs are generated from these two applications' specifications, and are assessed in comparison with the original specifications in terms of the lines of codes (LoCs) and abstraction levels. Table III shows LoCs of the specifications and derived codes. The program derivation is executed by extracting parameters and conditions from specifications, choosing primitives for each collaboration process, and embedding them into a skeleton code which contains only minimal processes for sensing. Since the derived codes need a lot of implementation level descriptions, they essentially need much more lines than the specifications.

B. Performance Analysis of Systems based on Derived Codes

We have conducted simulation experiments to observe that the automated program derivation performs well. We have used the Scenargie network simulator [5] version 1.4 where IEEE802.11g was used in the MAC and PHY layers of the ad-hoc communications. By assuming small Tx power in wireless sensor networks, the ad-hoc communication range r was about 40 m. We have targeted the noise detection application and the

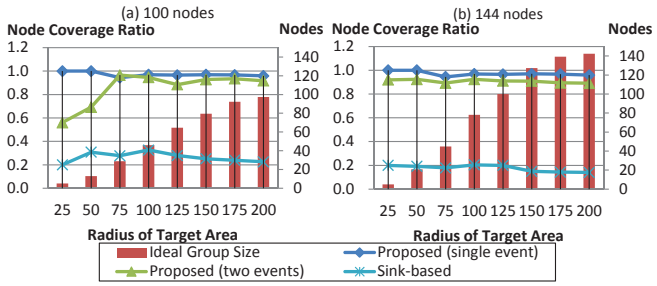

Fig. 5. Node Coverage Ratio

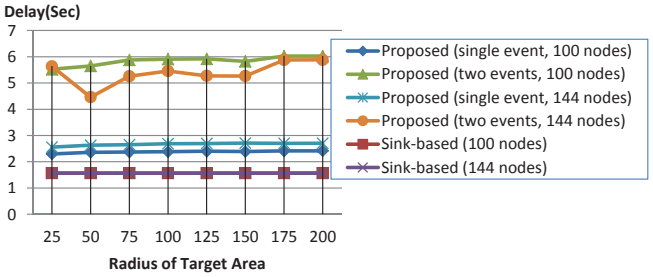

Fig. 6. Data Collection Delay

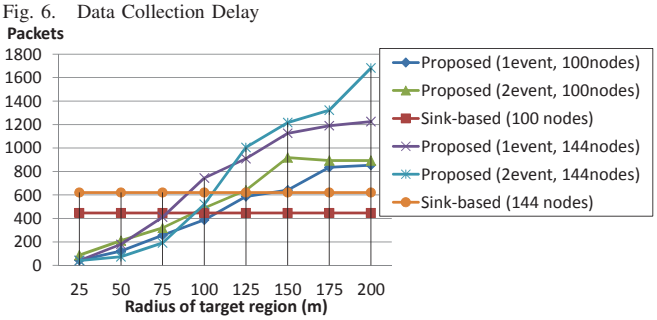

Fig. 7. The number of L2 packets

simulation was performed for 60 seconds. The size of the area was $250m \times 250m$ and nodes are deployed uniformly (grid-based deployment),

To present that the derived program codes can achieve reasonable performance levels, we have evaluated *Node coverage ratio*, *Data collection delay*, and *Number of packets in network* under different radiuses of target area. Node coverage ratio is the ratio of the number of actually-found nodes in the simulation to the number of nodes to be found according to the specification and node deployment. In other words, it shows the "completeness" of data collection. Data collection delay is the time duration from the first detection of over 80db noise occurrence to the completion of data collection process. Number of packets in network, which is the total number of data and control packets in the network layer.

For reference purpose, we have also measured those metrics by a sink-initiated data collection and computation called *sink-based collection* where a sink node broadcasts a data request packet to all the nodes in the field by a simple flooding mechanism, and each receiving node replies to this packet by sending data back to the sink. In order to verify the performance in various environments, we have prepared the following four scenarios where the number of nodes and/or the number of big noise events is different: 100 nodes (10×10 nodes) with 1 event at 20 seconds and 2 events at both 20 and 23 seconds, and 144 nodes (12×12 nodes) with 1 event and 2 events at the same timing.

Fig.5 (a) and Fig.5 (b) respectively show the node coverage ratios with 100 and 144 nodes. In these graphs, the expected number of nodes in the group is also shown as bars. We can see that the ratios are very close to 1.0 in all the cases. We note that sink-based collection achieves very low ratio (0.2 in average). This is due to unreliable message delivery back to sink nodes where these messages concentrated on a few nodes around sink nodes and some of them have been lost.

Fig.6 shows the data collection delay. They are not affected by the circle radius that determines the group size. As seen, the sink-based protocol could achieve the shortest delay, but this is mainly due to (very) low node coverage ratios (most packets were not delivered to sink nodes). On the other hand, we can observe that our algorithm could achieve reasonable trade-off between the node coverage ratio and delay.

Finally, Fig. 7 shows the number of packets observed in the network layer. The number of packets grows as the radius of circles becomes larger, but the growing tread is linear in any case. From this fact, we can say that our group-based local data collection and processing works preventing the growth of traffics to data collection toward a single point, which is often located far from the event occurrence place.

V. Conclusion

In this paper, we have proposed a support methodology for cooperative wireless sensor network application development. We have designed a language to describe high-level specification of such applications and have provided an algorithm to translate a given high-level specification into program codes for wireless sensor nodes. Our contribution is that we focus on cooperative applications in WSNs and design a methodology to implement given applications in a fully-distributed way, assuming computing and communication capabilities of intelligent sensor nodes.

Our ongoing work includes to apply the proposed method to participatory sensing systems on smartphones. In those platforms, we need to consider mobility, neighbor discovery and security issues keeping the architecture limitation in mind.

References

[1] M. Woehrle, C. Plessl, J. Beutel, and L. Thiele, "Increasing the reliability of wireless sensor networks with a distributed testing framework," in *Proceedings of the 4th workshop on Embedded networked sensors*, ser. EmNets '07. New York, NY, USA: ACM, 2007, pp. 93–97.

[2] H. Liu, T. Roeder, K. Walsh, R. Barr, and E. G. Sirer, "Design and implementation of a single system image operating system for ad hoc networks," in *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, ser. MobiSys '05. New York, NY, USA: ACM, 2005, pp. 149–162.

[3] T. W. Hnat, T. I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse, "Macrolab: a vector-based macroprogramming framework for cyber-physical systems," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ser. SenSys '08. New York, NY, USA: ACM, 2008, pp. 225–238.

[4] M. Hossain, A. Alim Al Islam, M. Kulkarni, and V. Raghunathan, "μsetl: A set based programming abstraction for wireless sensor networks," in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, april 2011, pp. 354 –365.

[5] Space-Time Engineering, "Scenargie base simulator," http://www.spacetime-eng.com/.