A Spreadsheet Approach to Programming and Managing Sensor Networks

Alec Woo, Siddharth Seth, and Feng Zhao
Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
{awoo, t-sseth, zhao}@microsoft.com

Abstract

We present a spreadsheet approach to simplifying the process of managing sensor networks and addressing the "in-situ" nature of sensor-net programming. An Excel spreadsheet prototype has been built to demonstrate the idea. This spreadsheet environment provides users who are already familiar with spreadsheet a very convenient and powerful tool for data programming and analysis. We discuss the architecture of this prototype and present our experience in implementing the tool and using it to build two different classes of sensor-net applications.

1. Introduction

Today, one of the hurdles in deploying a sensor network is the difficulty of programming and running the entire system. It involves programming and managing the sensors, programming the gateways, interpreting and processing the data streams, and setting up the servers for data archival. The truth is that even experts in the field find this entire process difficult and troublesome.

The underlying challenges arise from the inherent distributed nature of sensor networks that cross multiple tiers of computing: sensors, gateways, and servers. In addition, the unpredictability of the physical phenomenon being monitored often requires much iteration of data analysis and algorithmic or processing changes. This "in-situ" property is unique to sensor network since deployment specific constraints can impact the entire programming design and process; the deterministic behavior generally assumed in traditional programming is rare here.

Recent research efforts in simplifying sensor programming have led to a few interesting macro-programming frameworks [MMGM04, RNAMW05]. The general approach of these works focuses on designing a high-level language that can abstract away programming individual sensor nodes and low-level system details. These abstractions build up the important foundation and representation for defining programming specifications that aim to span multiple nodes. The high-level description of logic is very useful for programming and managing sensor systems composed of many nodes. However, sensor-net programming involves hurdles beyond building richer abstractions. The "in-situ" nature of sensor network requires a joint real-time data analysis, programming, and processing environment. System management support that allows user to reconfigure and program the entire sensor system is also important. Therefore, a sensor network programming environment requires three concurrent components: data analysis, data programming and processing, and run-time system management and reconfiguration support.

The contribution of this work is to allow users to achieve these three requirements using the familiar and widely used spreadsheet environment. This environment builds upon the abstractions and run-time support developed by the macro-programming effort and the sensor-net tools provided by [TinyOS]. The ultimate goal is to provide a simple user experience for sensor-net programming. Although processing is currently performed centrally at the spreadsheet in our prototype, it can be distributed in the future for scalability reasons but requires an underlying run-time support.

In this paper, we implemented the prototype with Excel over a service-oriented abstraction architecture provided by [JLFZ05]. Such a service-oriented abstraction is not an essential requirement for the success of the prototype. We used it because of our familiarity with it, and its flexibility and ease of integration with Excel.

The roadmap of the paper is as follows. Section 2 discusses how spreadsheet fits within the multi-tier architecture typically found in sensor network deployments. Section 3 walks through a high-level description of the implementation of our prototype and how it addresses our programming and management goals. Section 4 presents two application scenarios showing how our prototype helps building sensor-net applications. Section 5 evaluates this spreadsheet approach to programming and building sensor-net applications from our experience. We discuss related work in Section 6, and we conclude and address future work in Section 7.

2. The Role of Spreadsheet

Figure 1 shows the architectural overview of a typical sensor network deployment.

Figure 1: A multi-tier sensor-net architecture

The lowest-tier of computing at the bottom of Figure 1 consists of sensors deployed over a field. The sensors self-organize into a single or multiple-hop network streaming data towards the gateways, the next tier of computing. The gateway nodes may either be wireless or wired, but typically will have higher link bandwidth and reliability. They can be programmed to further process the data streams and or even operate as delegates for querying the sensor tier. The gateway, as its name suggests, also acts as an entry point to the Internet for the sensor nodes. The data streams from the gateways can be aggregated and archived for further processing by more powerful servers or databases at the next tier. This last tier of computing at the top of Figure 1 supports an Internet-scale sensor system. Managing and programming each of the tier, especially the sensor-tier, is difficult, not to mention integrating them together to form a complete system. Therefore, the design space crossing these different tiers is large and many of the issues are out of the scope of this paper. In this section, we focus on identifying the roles of spreadsheet relative to Figure 1.

Spreadsheet can play the roles of simplifying data analysis, data programming and processing, and run-time system management and reconfiguration for the users. Since it is already a familiar programming interface for scientific and business purpose, applying the same spreadsheet programming model for processing incoming data streams in real time would be very useful for everyday computer users who do not know or want to learn sensor network programming. The same built-in statistical functions and data visualization tools in a spreadsheet can be reused for analyzing data streams in the sensor-net context.

Another advantage of using the spreadsheet model is to reuse the tabular interface for binding spatial deployment information with incoming data streams. This is particularly useful for system management and reconfiguration. For example, one can overlay the deployment map and bind the spreadsheet cells to the sensor node positions on the map. This allows users to both manage the node and its data stream like manipulating individual cell on a spreadsheet. For example, changing the values in the cell of a node can automatically configure or even reprogram the node.

This integration of spatial deployment information with data analysis, data programming and system management directly addresses the “in-situ” nature of sensor networks. Spreadsheet naturally positions itself between the user and the gateway and sensor tier as shown in Figure 1. Furthermore, the ease of integration of spreadsheet with web servers or database at the highest-tier makes spreadsheet a very convenient integration tool. To preserve the distributed nature of the system as shown in Figure 1, data processing at the spreadsheet can fall back to other nodes, the gateways, or even the sensor-net tiers. However, this goes beyond the current scope of this paper.

3. System Implementation

The implementation of the spreadsheet prototype has two parts. The first involves augmenting Excel for sensor network management. The second involves making Excel as a server to receive incoming data streams. We discuss the details of these two parts in this section.

3.1 Network Management

We augment Excel with its built-in VBA scripting capability to provide a user interface that eases the process of programming (or downloading code to) the sensor tier. The first step involves discovery of deployed sensor nodes. If nodes are connected as a USB test-bed, discovery simply means calling the relevant TinyOS tools, such as "motelist", to query for the sensor nodes. If nodes are configured on an Ethernet network, discovery often means using a lookup table which can be a list in Excel. If nodes are deployed outdoors, we can rely on tools, such as Nucleus [GTDC05], for node discovery and display it in Excel.

Once node discovery is done, each sensor is represented as a cell. Users select the cells to represent motes for programming. All TinyOS applications in the *apps* directory are shown as a list in Excel for users to select. Figure 2 shows the screen shot of such an interface.

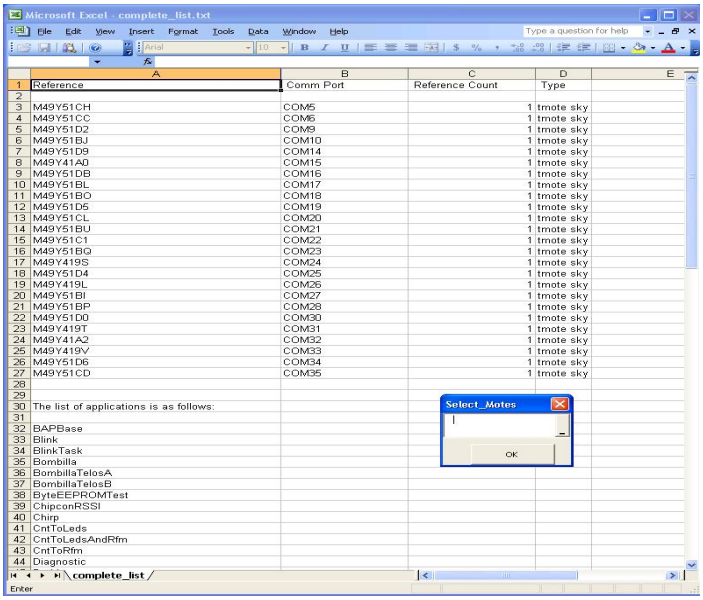

Figure 2: Management interface

User also has a choice of programming methods: over USB, Ethernet, or wireless with Deluge [JHDC04]. Deluge has a limitation that all images of the node must run the same program image. As a result, it is not possible to partition the network with different programs using Deluge.

More sophisticated system management can also be done in Excel with Nucleus since Nucleus exposes the necessary attributes for users to query and modify. These attributes can be shown in Excel in XML format for users to manipulate as shown in the next section. However, integration with Nucleus is not implemented yet.

3.2 Sensor Data Stream Processing

We use the latest version of Visual Studio 2005 Beta 2 with .NET 2.0 to augment Excel to act as an agent to query the gateway tier for incoming sensor data streams. This new version of Visual Studio integrates well with Excel development. The gateway tier runs the service-oriented architecture provided by SONGS. We use Tmote as the sensor tier platform and run the latest TinyOS 1.x distribution.

SONGS provides a service abstraction over the gateway and sensor tiers. An extended XML language called MSTML is used by SONGS for defining service composition over the gateway and the sensor tier. In our current implementation, Excel uses a given user-defined MSTML file to task the two tiers. In the future, the construction of the service composition can also be done within Excel.

SONGS uses XML as the basic gluing logic and structure definition for service composition and data stream objects. That is, SONGS gateways have services to automatically convert data streams from the lower-tier, platform-dependent sensor data format into platform-independent XML streams. On reception, Excel converts these incoming XML streams into standard spreadsheet lists. The layout and categorization of the list can either be user-defined or inferred automatically from the XML structure. The later approach is attractive but requires user guidance to filter out unneeded information within the XML structure. A wizard should play such a role to assist users in the future.

Our current implementation uses a manual mapping of the XML structure to the list layout in Excel. Once data streams are mapped onto the Excel lists, processing on the data is the same as usual spreadsheet programming. Section 4 shows how we built two application scenarios within this framework.

3.3 Database Server

While spreadsheet provides a data programming and visualization environment to the users, the archival of the data requires a database, especially when data streams are continuous for a long period of time.

Our prototype can store each data stream and its corresponding processed values into the database. To ease the integration effort between Excel and the database, we use the latest beta version of SQL Server 2005 because it natively supports XML data types. With this XML capability, XML streams can be stored directly in the database without knowing the packet structure a priori. XML query processing is thus possible over both the raw data streams and the processed data streams from the spreadsheet. We use XML as the key of simplifying the integration process for the different tiers as we believe that future databases and data streams will widely support or exploit XML.

With a database, the spreadsheet becomes a cache for the most recent data streams. This cache size is especially limited if processing is done centrally. In our current implementation, Excel lists of the data and processing streams have a limited size defined by the user. That is, each list signifies a fixed time window of the most recent data, with the current sample being at the bottom of the list. The stale samples, already stored in the database, will be automatically purged by Excel as the lists are bounded by their maximum length. The choice of this size is a balance on the amount of history that needs to maintain for processing versus scalable performance as the amount of streaming increases.

4. Application Scenarios

We apply our prototype to two different classes of sensor-net applications. One involves event detection for the presence of a vehicle in a parking lot and the other is a typical environmental data collection application within an office building.

4.1 Vehicle Event Detection

The first application involves detecting vehicles and counting them within a parking lot infrastructure. We use the same deployment setup as discussed in [JLFZ05]. Infrared break-beam sensors and their reflectors are deployed in the parking lot. Whenever the line of sight between a sensor and its reflector is physically blocked by an obstacle, such as a vehicle, the sensor will report a low-level binary detection event to the gateway. A vehicle passing over these sensors will generate a stream of events, which are time-stamped by the gateway and sent to Excel. Figure 3 shows the screenshot of the data streams in Excel.

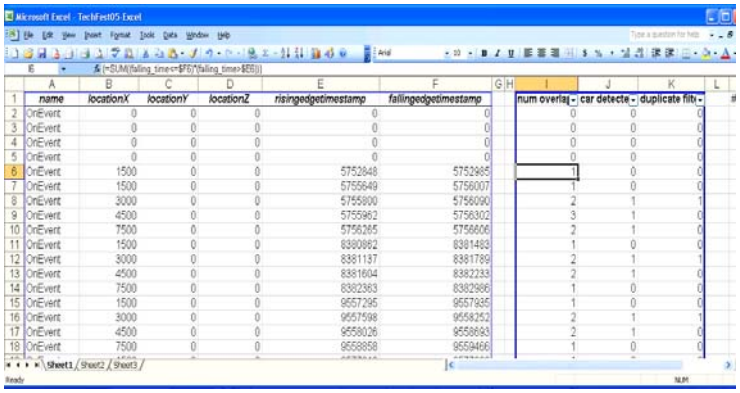

Figure 3: Vehicle detection processing

The list on the left shows the raw incoming data streams of the break-beam sensors from the gateways. These sensor data streams are represented as XML objects as shown below:

```
<?xml version="1.0" encoding="utf-8" ?>
<microserver name="ebox">
      <event name = "On Event"
        locationX = "0"
        locationY = "0"
        locationZ = "0"
        risingedgetimestamp = "10000"
        fallingedgetimestamp ="10300"/>
</microserver>
```

In this case, an “On Event” means that the break-beam sensor has an event detected. The location of the sensor and the corresponding timestamps of the rising and falling edge of the event are included in the XML object. These XML objects form the raw data streams and they are mapped to the list on the left as suggested by the list column headers in Figure 3.

In our deployment, sensors are strategically placed along the roadside such that a typical vehicle would block at least two of the sensors during crossing. One simple mechanism in detecting a vehicle is to count if two or more sensor events within a given physical location overlap in time. The list on the right with a thick border in Figure 3 shows how users can specify such processing using conventional spreadsheet array formula.

For each raw data detection event, we calculate the number of events that overlap with it in time using the following Excel array formula: *{=SUM((falling_time<=$F6) *(falling_time>$E6))}*. In this case, the current row is 6. Therefore, F6 represents the falling-edge timestamp and E6 represents the rising-edge timestamp of the current event. "falling_time" is the name of column F defined by the user a priori. It signifies the falling-edge timestamp of all the events in the worksheet. The action of the formula is to sum up the number of events in the list that has the falling-edge timestamps within the rising-edge and falling-edge of the current event at row 6. This is an array formula, as denoted by {}, because it has to process the entire "falling_time" (Fth) column. Expanding the formula to include location information in Figure 3 can filter out events that are not spatially correlated. As the raw data list grows, the processing list grows as well with the relative cell addressing, such as F6 and E6, in the formula adapting automatically.

The next column defines a threshold for vehicle detection; if the number of overlapping detections equals or exceeds a threshold of 2, a vehicle is detected. This logic is translated to the following Excel expression in cell J6 in Figure 3: *=IF(num_overlaps>=2, 1,0)*. This is not an array formula since we are not processing the entire "num_overlaps" (Ith) column. "num_overlaps" refers to the cell in column (I) on the same row numbered 6.

These two simple examples demonstrate how users can use the built-in Excel functions and expressions to derive more high-level statistics, such as vehicle speed and traffic flow information within the parking lot. The naming convention in Excel can be confusing when writing formulas since it depends on the context of the formula and we hope future Excel can improve on its naming scheme.

Notice that incoming raw data streams do not follow a temporal order in Figure 3. However, the amount of buffer (list size), which determines how far back the spreadsheet can go in time, is adequate to absorb these issues for our simple application. However, users need to explicitly process the unordered streams if tight temporal processing is required.

4.2 Environmental Data Collection

Our second application involves collecting environmental statistics within a typical office building. The deployment consists of 9 sensor nodes that are capable of sensing ambient light, photo-synthetic light, temperature, and humidity. These sensor nodes are deployed across seven offices and a hardware laboratory. We use Excel to maintain the spatial information of the deployment as shown in Figure 4.

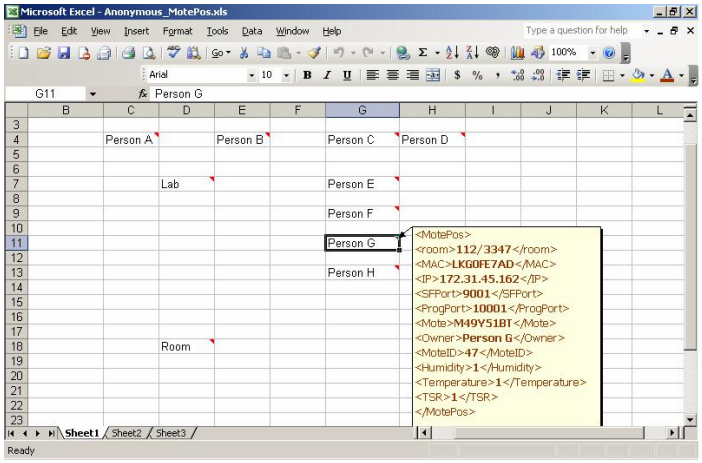

Figure 4: Deployment specific information

Deployment specific information of each node can be described using XML and stored as cell values in the spreadsheet. The information may include node ID, types of sensors on the node, and location of the node. Depending on how the sensor program is created, such information can directly be mapped to Nucleus in TinyOS or abstracted by a SONGS service for management and reconfiguration purposes. For example, the following shows the XML descriptions for one of our deployed node.

```
<MotePos>
  <HOST>LKG0FE7AD</HOST>
  <IP>172.31.45.162</IP>
  <SFPort>9001</SFPort>
  <ProgPort>10001</ProgPort>
```

```
  <MoteSerial>M49Y51BT</MoteSerial>
  <Owner>Person G </Owner>
  <room>112/3347</room>
  <MoteID>47</MoteID>
  <Humidity>1</Humidity>
  <Temperature>1</Temperature>
  <TSR>1</TSR>
</MotePos>
```

The XML elements, including HOST, IP, SFPort, ProgPort, and MoteSerial, describe the test-bed information about the mote. The rest of the elements specifically describe the deployment information and the services provided by the node. The deployment information is expected to be defined by the users while the node services are expected to be exposed by the run-time support such as Nucleus. For example, a user can simply turn on/off the humidity data stream from the sensor by changing the binary value of the XML Humidity node. Such action will automatically update the node's internal state to reflect the changes. This feature will be supported by our prototype once we integrate it with Nucleus.

To visualize and process the sensor data streams with Excel, users can select a subset or all of the sensors discovered and shown on the spreadsheet. The users can simply select the cells corresponding to the nodes to subscribe for their data streams. The selected data streams will be shown automatically on a different worksheet. The following shows a sample XML object from a sensor data stream.

```
<?xml version="1.0" encoding="utf-16"?>
<OscopeMsgxmlns:xsd="http://www.w3.org/
2001/XMLSchema"xmlns:xsi="http://www.w3
.org/2001/XMLSchema-instance">
  <sourceMoteID>1</sourceMoteID>
  <lastSampleNumber>6140
    </lastSampleNumber>
  <channel>0</channel>
  <data1>986</data1>
  <data2>985</data2>
  <data3>984</data3>
  <data4>985</data4>
  <data5>985</data5>
  <data6>984</data6>
  <data7>985</data7>
  <data8>985</data8>
  <data9>985</data9>
  <data10>984</data10>
</OscopeMsg>
```

This XML object represents a packet from the TinyOS Oscilloscope application. The SONGS architecture has a generic data collection service that automatically transforms the application payload of TinyOS packets into XML objects. Excel can automatically infer the schema of these objects and display them on the spreadsheet. However, this is often not desired because users may want to have the control to layout and select important fields from the XML data streams. Our current prototype manually filters for the interesting part of the data streams and lays them out on the spreadsheet from left to right, with user-defined processing lists interleaving between them. Figure 5 shows an example of the filtered real-time data streams from three sensor nodes in Figure 4, with the most recent samples always located at the bottom of each list. In this case, each processing list simply converts the unit of raw temperature data from the list on its left into Celsius. In the future, we plan to use a wizard to guide users to control such a process.

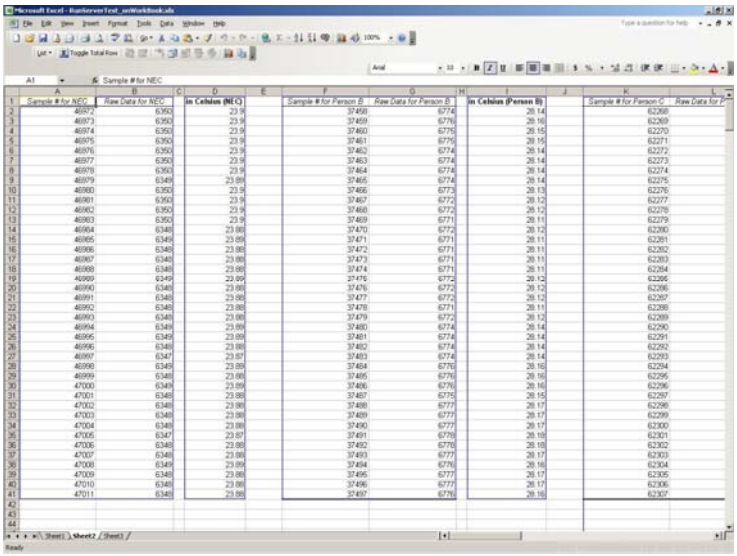

Figure 5: Data streams from the selected sensor nodes

Users can utilize Excel's built-in plotting ability to graph real-time statistics of the data streams similar to the Oscilloscope tool from the TinyOS distribution. In addition, users can reuse the deployment specific information layout in Figure 5 to visualize the spatial distribution of the data. For example, Figure 6 shows a very simple spatial contour graph of the temperature variations among the different locations. The color spectrum range is adjusted dynamically, depending on real-time data as shown in cells L3 and L4. Clicking on the node will

display a callout box showing the average temperature of that location.

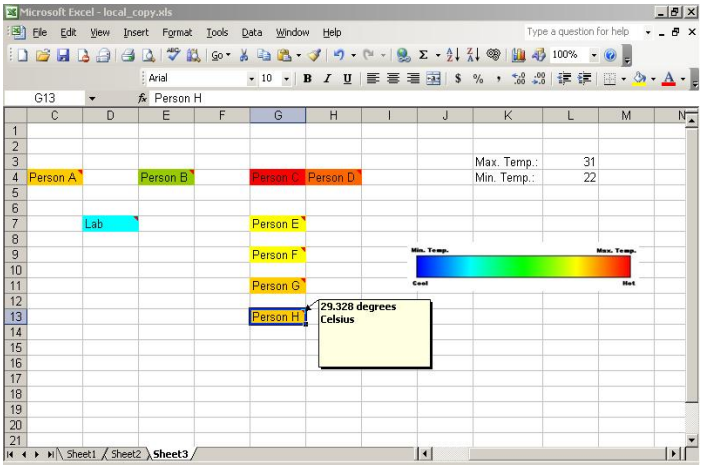

Figure 6: Real-time spatial visualization of data distribution

5. Discussion

Our prototype allows us to gain a lot of first-hand experience in evaluating this spreadsheet approach to sensor networks. We discuss our programming and implementation experience in this section.

5.1 Programming Experience

For developers who are used to imperative languages for programming, spreadsheet programming may feel restrictive in the beginning. It is easy to arrive with an early judgment that "it can't be done in a spreadsheet". This is because spreadsheet may provide less flexibility with its simple expression language and its requirement of the data to be layout in a tabular format. For sophisticated data processing, which we haven't evaluated with our two application scenarios, spreadsheet programming may be cumbersome. For example, list-to-list processing primitives, which are very common for data streams, are not supported by Excel. Data naming is also weak as the reference of the list's name depends on the context of the formula. While VBA can be used to provide a rich functional language programming platform, it breaks the spreadsheet programming model. Non-VBA functional programming on Excel is possible as demonstrated by the work in [SJMBAB03], but it is still an early research prototype. All in all, at the current stage, we have not explored using spreadsheet for programming complex processes.

However, the very point of a spreadsheet is to provide a simple programming platform for non-computer scientists, who will be the main users of sensor networks. Although our application scenarios are simple, the demonstrated processing is fundamental and common to many potential sensor-net applications. Furthermore, the formula expression language in Excel is fairly general and supports a rich set of statistical functions for scientific studies. Finally, the programming logic and the associated buffer usage are quite explicit on a spreadsheet. This can be advantageous since such information is useful in delegating the processing to the same or lower tiers.

5.2 Implementation Experience

We found that the built-in VBA capability provides a simple way for users to customize user interfaces and automate data processing for each application, which is important since sensor network is often application-specific.

Our design choice of relying on XML for integrating the different pieces of the system together is convenient as it allows us to decouple the implementation platform details among the individual pieces and thus, simplify the process of integration.

We found that updating the Excel's display with the data streams has a high latency overhead. To improve efficiency, we chose to update the spreadsheet with batches of data stream samples rather than per each packet arrival. This approach helps to increase Excel's data service rate to catch up with the incoming rate of the data streams. The size of the batch is defined by the user and it depends on the scale of the network and the expected latency in observing and processing the samples.

The amount of statistical history needs to be maintained by Excel is governed by the number of nodes, the sampling rate, and the required time-window to perform the statistical analysis. Excel has a limitation as to how much data it can maintain before reaching the point of thrashing. This needs to be experimentally measured in the future.

6. Related Work

The idea of using the tabular or spreadsheet interface for managing and processing data is not new; it is used in everyday business, scientific, and industrial settings. It is this reason why we advocate the spreadsheet approach to managing and programming sensor networks. Our contribution is in developing a prototype that demonstrates the potential of this approach for "in-situ" sensor data management. Several other research works have taken a similar path to simplify sensor data processing and visualization.

TinyDB [TinyDB] provides the tabular interface for users to visualize collected sensor data. It presents a SQL query interface for data processing and provides a primitive tool for data visualization. Its main goal is to perform in-network processing in the sensor tier in response to a given SQL query.

Moteview from Crossbow [Xbow] is a sensor network management and data collection tool. It supports visualization of the physical deployment and maps it with the corresponding sensory data. The tool is oriented towards visualization of sensor data and network statistics such as network topology. It does not present a programming interface for users to process the incoming data streams.

Similar to ours, the work in [HBMM05] is exploring the use of a spreadsheet interface to help scientists to visualize data and perform some limited processing. The main idea is to provide a pivot-table interface so that users can visualize the data relationships from different perspectives. Instead of using Excel, they build a complete spreadsheet tool from scratch. The work also proposes to let users write simple expressions to define event triggers that will eventually be compiled down to the sensor-tier acting as low-level filters.

The Nucleus management console project from Berkeley [MM05] provides a spreadsheet web interface for showing system information of each sensor node provided it has the TinyOS Nucleus run-time installed. The system information can represent networking statistics or even the internal states of each node. The console provides a tabular interface for users to view and alter such information, thus, performing system-wide management and reconfiguration support.

7. Future Work and Conclusion

We have successfully built a spreadsheet prototype in Excel to address three important issues of sensor-net "in-situ" programming, which include real-time data programming, data analysis, and system management and reconfiguration. We built two different classes of applications to demonstrate the feasibility of the spreadsheet approach and presented our experience.

A systematic performance and robustness study should be done to stress test our prototype in order to understand scalability issues. We plan to do such evaluation using our test-bed and grow the scale incrementally. This allows us to explore the crossover point in offered load between centralized and distributed stream processing with Excel. Beside performance and stability, we should conduct usability study of our prototype to guide us on the interface design. We also plan to build other applications to drive the further development of the tool.

We only explored some of the data analysis, programming and system management issues with our current prototype. A few interesting and important directions remain to be explored. They include user-interface design on manipulating many data streams, new ways of data and network visualization, identification of useful statistical functions, and process delegation and service composition for the gateways and sensor-tier. We plan to do these by releasing our prototype as a tool to the research community in steps and get feedback along the way.

References

[GTDC05] Gilman Tolle, David Culler. *Design of an Application-Cooperative Management System for Wireless Sensor Networks.* Second European Workshop on Wireless Sensor Networks (EWSN), Istanbul, Turkey, January 31 - February 2, 2005.

[HBMM05] James Horey, Patrick Bridges, Arthur Maccabe, and Angela Mielke. *Work-in-Progress: The Design of a Spreadsheet Interface, IPSN* (2005)

[JHDC04] Jonathan W. Hui and David Culler. *The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. The 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, 2004

[JLFZ05] Jie Liu and Feng Zhao, *Towards Semantic Services for Sensor-Rich Information Systems*. The 2nd IEEE/CreateNet International Workshop on Broadband Advanced Sensor Networks (Basenets 2005), Boston, MA, October 2005..

[MM05] Matt Massie, *TinyOS Nucleus Management Console*. Berkeley CENTS Retreat, May 27th, 2005.http://www.cs.berkeley.edu/~massie/talks/nucleus/NucleusInternals.ppt

[MWGM04] Matt Welsh and Geoff Mainland. *Programming Sensor Networks Using Abstract Regions.* In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04), March 2004.

[RNAMW05] Ryan Newton, Arvind, and Matt Welsh. *Building up to Macroprogramming: An Intermediate Language for Sensor Networks*. In Proceedings of the Fourth International Conference on Information Processing in Sensor Networks (IPSN'05), April 2005.

[SJMBAB03] Simon Peyton Jones, Margaret Burnett, Alan Blackwell. *A user-centred approach to functions in Excel*. Proc International Conference on Functional Programming, Uppsala, Sept 2003 (ICFP'03), pp 165-176, 12 pages.

[TinyOS] http://www.tinyos.net

[TinyDB] http://telegraph.cs.berkeley.edu/tinydb

[Xbow] http://www.xbow.com