Visualizing Wireless Sensor Networks: An Experience Report

Mark Miyashita, Mikhail Nesterenko*, Romil D. Shah and Adnan Vora
Computer Science Department
Kent State University
Kent, OH, 44242
{mmiyashi, mikhail, rshah, avora}@cs.kent.edu

Abstract

Wireless sensor networks is a novel computing architecture with a variety of potential applications. Due to close interaction with the environment, large scale, limited resources of individual sensor nodes and other specifics, visualizing the operation of such a network with a graphical user interface (GUI) is important. In this paper we describe our experiences in designing a GUI in a succession of three projects for sensor networks: Pursuer-Evader, Line in the Sand and Extreme Scaling. Even though these projects were rather different in purpose and scale, we approached each new project as an opportunity to develop the more sophisticated visualization software. We reused old code as much as possible while adding and enhancing software features as necessary. This approach proved to be successful as all our GUIs were deployed as required. We discuss each project requirements and GUI architecture. Then we report on the design process and feedback we received when each GUI was tested and used.

Keywords: visualization of wireless sensor network applications, graphical user interfaces, wireless sensor networks.

*This research was supported in part by DARPA contract OSU-RF#F33615-01-C-1901 and by NSF CAREER Award 0347485

1 Introduction

Wireless sensor networks is an emergent computing platform. A large number of sensors spread throughout an area of interest enables such a network to collect detailed information about the environment. A wireless sensor network can be quickly deployed and then operated without human intervention for an extended period of time. These features make sensor networks suitable for a number of tasks which traditional computing architectures could not adequately address: habitat monitoring, indoor climate control, enemy troop movement surveillance, medical diagnostics, logistical support in natural disaster areas, etc.

Some potential applications of sensor networks require massive (up to 100,000) deployment of sensor nodes. To make such deployments economically viable, each individual sensor node must be rather inexpensive. Hence, a sensor node has numerous resource constraints: it is limited in memory, computing power, bandwidth and energy resources. Moreover, a sensor node does not have extensive visual aids such as external displays or monitors to determine its internal state; at most, it has a few LEDs. Due to the large scale of deployments and the limited resources of individual sensor nodes, obtaining an adequate picture of the state of the network is difficult. Thus, a graphical user interface (GUI) that presents the network's state and allows the user to interact effectively with the sensor network becomes an indis-

pensable component of a sensor network application.

A GUI for a sensor network has two main categories of users: programmers and operators. The GUI requirements for each category are somewhat different. The programmers require the GUI for testing, debugging and maintenance of the application. They need to be able to reconstruct the state of the network and the contents of the internal data structures of sensor nodes (e.g. routing topology, message propagation patterns, readings of individual sensors) in real time. Operators, on the other hand, require a convenient and effective way to control the sensor network, extract information from it and relate it to the events taking place in the real world (e.g. target movements or fire incidents). Hence, the operators need close correspondence between the events of interest happening in the real world and their visual representation as well as convenient access to the state information of various parts of the network.

Unlike a conventional program for computer networks which may be generic in nature, wireless sensor network applications are designed for a particular task. Thus, given the importance of the GUI for the success of the application, the GUI has to be tailored to the demands of the application. However, GUI for sensor networks is a complex piece of software. Thus, designing it from scratch for each application is expensive. In this paper, we describe an alternative approach to GUI development. We used a succession of wireless applications for which we had to build a GUI, to develop the features of the GUI incrementally. For each successive project, rather than starting from scratch, we refocused the GUI from previous projects to the new project and added features and functionality as needed.

A number of circumstances contributed to the success of our efforts. Sensor network applications, although disparate, tend to have some common GUI requirements: presentation of area of interest and locations of sensor nodes on it, detailed information about each node, magnification on demand, network topology and controlled playback capabilities. This allowed us to keep a large portion of GUI code as we moved from project to project. The projects with which we were involved grew in scale and complexity. Thus, our GUI development was largely incremental. Close cooperation with the project leaders allowed us to discuss specifications, receive feedback and try out preliminary versions of the GUI as we developed it. As we participated in the successive projects, we were able to observe the use of our GUI and conceive additional features and improvements to it.

Sensor node architecture, development history and paper organization. We developed the GUI for a number of applications. For all these applications we used Berkeley's prototype networked sensors [16] called *motes*. These sensors are quite popular in research and industry communities due to their simplicity, ease of use and application readiness. The motes run TinyOS [17]. TinyOS is a lightweight event-based operating system that implements the networking stack, handles communication with the sensors and provides a programming environment for this platform.

As we started experimenting with wireless sensor networks, the need for a GUI for demonstration and debugging purposes quickly became apparent. The first version of the GUI was built to address our internal needs. On the basis of this version, we designed a GUI for the "Pursuer-Evader" project conducted by a research team at Ohio State University (OSU). We describe this GUI in Section 2. We then refocused the GUI for the "Line in the Sand" project described in Section 3. We are currently finishing our design of the GUI for "Extreme Scaling" project. This effort is described in Section 4. All these sections are structured similarly. We start by describing the project itself and the requirements it placed on the GUI. We then describe what GUI features were implemented for this project. We proceed to describe the internal architecture of the GUI. We conclude by discussing the implementation process, the usage history and the feedback that we received from the users.

Related Literature. There is a number of generic visualization tools for distributed systems. Carr et al designed ConcurrentMentor [10] to externally monitor the execution of a distributed system. Tropol et al [21] built a visualization tool for logging the activities in a distributed system. Neither of these tools are de-

signed specifically for sensor networks. ATEMU [20] is a interesting simulator for an individual sensor network node. ATEMU emulates the architecture of a Berkeley prototype networked sensor [6]. ATEMU allows to debug and visualize the processes of a single sensor node. However, it is not designed to visualize a sensor network as a whole. Basic visualization primitives for sensor networks are available in TinyViz [19] and EmView [15].

2 Pursuer-Evader Tracking

Problem requirements. We implemented the GUI to support the OSU implementation of a pursuer-evader tracking application [12]. The application was to be demonstrated at DARPA Network Embedded Systems Technology program retreat in Bar Harbor, Maine in June 2002. This project used Mica release of the motes [11]. Each mote has an embedded 4MHz microprocessor, 4KB of RAM, 4MB of FLASH memory, Monolithics asynchronous short-range radio transmitter, and an array of sensors.

The experiment consisted of two Lego Mindstorm robots serving as an evader and purser on top of a foam panel where a 4-by-4 grid of motes were embedded. The evader was human-controlled. The pursuer was autonomous. It gathered the information from the embedded motes and decided on the direction of pursuit. Both the pursuer and the evader were only allowed to move along the grid lines of the panel. The pursuer moved faster than the the evader. The objective of the pursuer was to catch the evader. Using its light sensor, an embedded mote detected the evader passing over it. When the evader was sighted, the mote notified the neighbors over the radio. The motes indicated the direction of pursuit by blinking an infrared LED. As the pursuer passed over a mote it read the direction to go. In essence, the motes maintained the pursuit direction tree with the evader at the sink. The application was designed to withstand a mote failure and tree corruption.

An extra *base-station* mote was attached to a PC. This mote eavesdropped on the messages exchanged by the embedded motes and relayed this information to the GUI application running on the PC.

GUI descriptions and features. For the purpose of the demonstration, the operator needed to see pursuit direction tree matching the ongoing experiment. The programmer also needed visualization of the tree as the amount of information generated by the motes was overwhelming — each mote generated four messages per second.

The Pursuer-Evader GUI contained a debugging panel that showed the messages received by the base-station and a network topology panel where the pursuit direction tree was displayed. A screenshot of Pursuer-Evader GUI is shown in Figure 1. For debugging purposes, the pursuer-evader application code was sometimes run under a PC-based simulator. In this case, the simulator relayed the messages over a TCP/IP socket. The GUI was designed to accomodate both a physical experiment and a simulation.

Architecture. The architecture diagram for the Pursuer-Evader GUI is shown in Figure 2. The GUI consists of four components. To ensure adequate responsiveness of the GUI, each component was implemented as a separate thread. The *Message-Processing Component* receives the raw message from either the COM-port of the socket and converts it to byte format. The component also checks the integrity of the message by computing its CRC and discards corrupted messages. The *Message Display Panel* receives byte-format messages from the Message Processing Component and displays them as text. Similarly, the *Network Topology Panel* receives messages, interprets them and updates the pursuit direction tree. The *Control Panel* starts and stops the application, configures the message input channel and other parameters of the experiment.

The GUI was written in Java. Swing class API was used to organize various panel displays as well as the pursuit direction tree display. Mote message processing was done with the help of Java communication API.

Experience. The GUI was quite helpful during the development of the pursuer-evader application. The GUI was successfully used during the intended demonstration and further experiments.

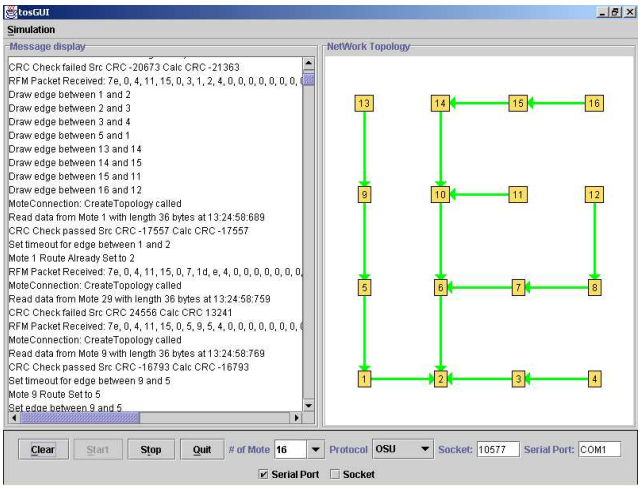

Figure 1: A screenshot of the Pursuer-Evader GUI

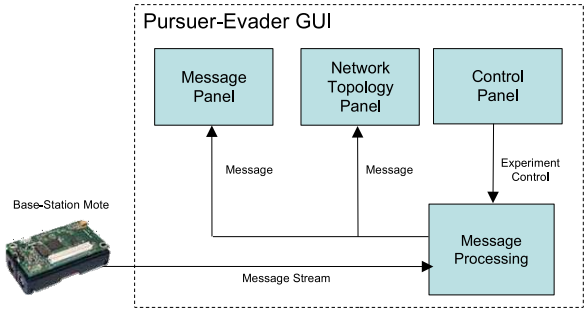

Figure 2: The architecture of the Pursuer-Evader GUI

3 Line In The Sand

Problem requirements. "Line is the Sand" (LITeS) [9] was a DARPA-sponsored experiment led by the Ohio State University. The objective of LITeS was to demonstrate the capabilities of sensor networks in target detection and tracking. The demonstration was to take place at McDill Air Force Base, Florida in August, 2003. The experiment required the protection of 60 by 25 ft area by 78 sensor nodes arranged in a grid formation. The sensor nodes were to be able to self-organize into a network, track and classify intruders as well as withstand failures. The entire project was to take about 8 months.

The sensor nodes were Mica2 motes [6] with a 4KB static RAM and 128KB of programmable flash memory. The motes had a Chipcon 916.5 MHz radio transceiver. The motes were also equipped with magnetometers and Micro Impulse Radars (MIR). The motes were placed in pre-determined locations on the field. They then configured themselves into a connected network. Messages from the motes were relayed through the network to the exfiltration point (base-station). Various classes of intruders passed through this sensor grid. The network detected, classified and tracked them. The motes used their magnetometers to sense the metal content of the intruder and classified it as a human, a soldier, or a car based on increasing metal content. The MIRs were used to judge the speed and trajectory of the intruder. The sensor network was supposed to differentiate multiple intruders simultaneously passing through different portions of the protected area. The network was also to detect and track a succession of intruders passing through the same area.

GUI description and features. We provided the GUI with features to suit both groups of users: operators and programmers. For operators of the system, the GUI was required to represent accurately the tracks of one or multiple intruders in a way that related to the actual topography of the monitored area. In particular, the operators were interested in the entry and exit points of each intruder. The operator also needed to be able to record and playback a certain sequence of events as necessary. During the course of the experiment, the operator needed to have the ability to zoom in on a particular target location, or zoom out to gain a wider perspective.

In order to fulfill these requirements, we decided to depict the motes over a bitmapped picture of the protection area. Operators could choose to display evenly spaced grid lines on the GUI to get a sense of the distance between the motes and the relative

locations of the detected intruders. We provided on-demand zooming capability which was controlled by the mouse and allowed arbitrary magnification factors. The operator was given the ability to freeze the GUI and stop it from displaying real-time events, as well as to playback up to five minutes of past events. The playback did not interfere with the recording of real-time events which became available to the operator once he exited the playback mode.

To the programmers, debugging and monitoring were important. In particular, they wanted to be able to view each raw message received from the base-station, and constantly monitor the health of every mote in the network. Detailed information about every mote and intruder needed to be available. In order to monitor the state of the network as a whole, we needed to represent the network topology graphically.

We provided a Summary Display panel on the GUI, which reported average network statistics such as average mote power levels, the number of intruders detected and the number of functioning motes. A separate Debug Panel displayed raw messages received from the base-station. The on-screen representation of the motes depicted the network topology using arrows. Whenever an intruder was detected, the GUI clearly indicated which of the motes actually participated in the detection of the target. At a large zoom level, every mote and intruder displayed a digital readout of its key properties. In order to make these programmer requirements unobtrusive for the operator and to reduce on-screen clutter, most of these features could be turned on or off on demand. A screenshot of the LITeS GUI is shown in Figure 3.

Architecture. The architecture diagram for the LITeS GUI is shown in Figure 4. The initial processing of the messages received by the base-station mote is handled by the component developed by the OSU team. The message is then transferred to the LITeS GUI. The incoming messages pertain to two main objects to be displayed by the GUI: motes and targets. A *Mote message* describes the state of a particular mote and includes its ID, parent ID, coordinates, power level and readings of its on-board sensors. A *Target message* describes a detected intruder and includes the ID of the target, its classification (civilian, solider, car or unknown), coordinates, and a list of IDs of all the motes that participated in target detection. To facilitate the message transfer we used the Java Management Extensions (JMX) [4] toolkit developed by Sun Microsystems. The incoming messages are stored in internal object repository of the GUI.

The operation of the GUI components is event-driven. The events are generated by either the user or the incoming messages. The events are handled by separate threads. As soon as the new object information arrives, the event listener of the *Display Panel* component forces it to repaint the screen based on the newly available information. The Display Panel also notifies the *Debug Panel* and the *Summary Panel* components about newly arrived information. The Debug Panel presents information message-by-message, while the Summary Panel shows the aggregate network statistics. Due to the amount of processing and screen display activities, the Debug Panel has a separate thread that runs asynchronously with the Display Panel thread.

The Display Panel receives image magnification requests from the user. These requests are handled with the help of a Zooming Toolkit. The *Zooming Toolkit* is based on Jazz [8] — an application framework for building a zoomable user interface developed by the University of Maryland.

On the basis of user input, the *Control Panel* determines the visibility of on-demand features of the Display Panel such as the coordinates grid and the network topology. The Control Panel also drives the Playback Module. The *Playback Module* freezes the real-time presentation in the Display Panel and replays the past events. The user inputs the number of seconds of past events that should be played back. The Playback Module contains a separate thread that controls playback. This enables the Display Panel to continue to accumulate real-time information during playback.

Experience. The GUI was successfully used for the demonstration of the experiment. It was developed, tested and debugged on schedule.

The GUI design process was iterative. We created

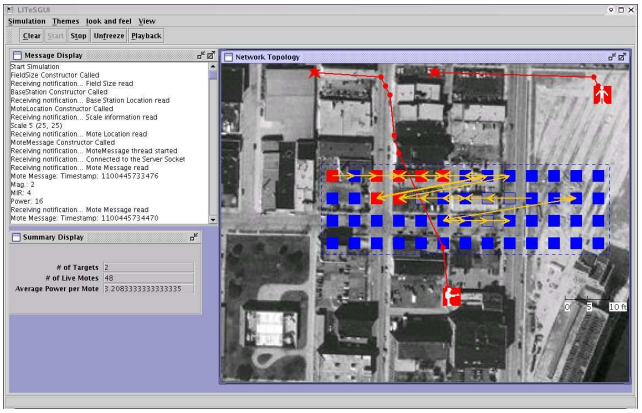

Figure 3: A screenshot of Line in the Sand GUI

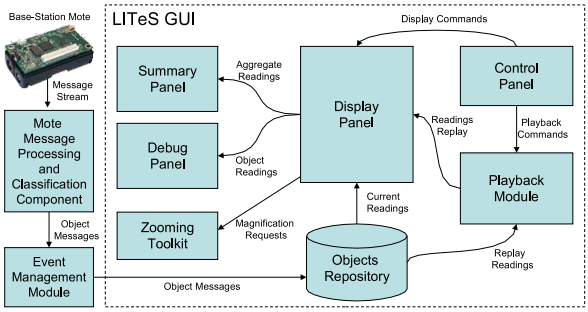

Figure 4: The architecture of Line in the Sand GUI

a formal specification based on the verbal GUI requirements communicated to us by the OSU team. Feedback and extra feature requests were then incorporated into the design. After the design was relatively stable, we negotiated the external interfaces. These interfaces were made easily extensible as we envisioned further changes as the application evolved.

The GUI was initially debugged and tested by our team using simulated trace data and once it was stable, it was sent to OSU for field testing. After reports of excessive memory usage, we employed JProbe [1] to detect and eliminate memory leaks in the GUI.

As we observed the usage of our GUI, we noted that some features were heavily used. The ability to playback events was quite popular. The OSU programmers liked the debugging capabilities into the GUI. They found it quite useful to be able to visualize the current state of the application and determine exactly which motes participated in the detection of a particular target. The bitmapped background representing the topology was not found to be as useful, mainly because of the lack of resolution of available aerial views.

Some features required further refinement. The users found it more convenient to select an appropriate level of magnification at the beginning of the experiment. The mouse-based magnification control proved to be less useful. Hence, we implemented a menu-based selection of discrete magnification levels.

Our LITeS GUI proved rather popular. It was used multiple times by OSU in demonstrations after the LITeS project was completed. In particular the OSU team used the GUI in the initial stages of the extreme scaling project to be described in the next section. Furthermore, a team from Michigan State University used LITeS GUI to debug and demonstrate a wireless sensor network reprogramming application [22]. In this experiment, the team successfully adapted this GUI to display reprogramming of 40 motes. The motes were preprogrammed with the standard TinyOS BLINK application. After successful reprogramming with LITeS code, the motes started sending messages to the base-station and the GUI displayed the tracking data sent by the motes, thereby demonstrating that the reprogramming was successful.

4 Extreme Scaling

Problem requirements. After the success of LITeS [9], Extreme Scaling (ExScal) [2] was the next project for which we have to develop a GUI. The objective of ExScal was to use a wireless sensor network to protect a linear asset, such as a pipeline, against sabotage. The final demonstration was to take place at Avon Park Air Force Base, Florida in December 2004. In ExScal, 10,000 sensor nodes were required to protect an area of 10km by 500m. The sensor

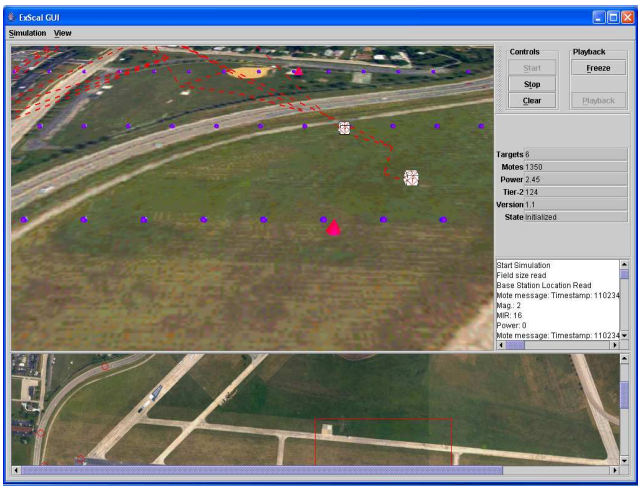

Figure 5: A screenshot of the ExScal GUI

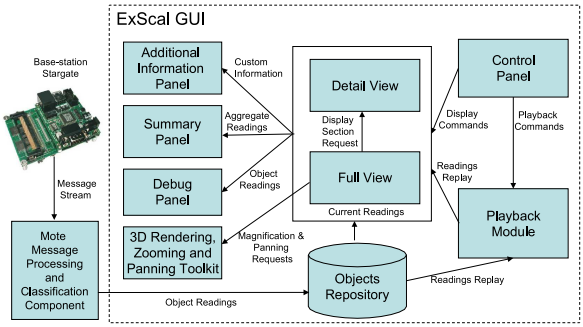

Figure 6: The architecture of the ExScal GUI

network was expected to detect, classify and track the intruders as they approach the protected asset. To satisfy the problem requirements, the following architectural decisions were made. The sensor network was to be multi-tiered. The first tier was composed of the XSM [13, 14] motes. XSM is a modification of Mica2, custom built for ExScal. The XSMs have low power sensing, improved radio performance, passive infrared sensor (PIR), acoustic sensor and a grenade timer. By analyzing the readings of these sensors, targets are classified. The second tier was composed of Stargates [7]. A Stargate is a PDA class device capable of communicating through PCMCIA, COM-port, Ethernet, USB. A Stargate runs Linux. In ExScal, Stargates were configured to communicate via IEEE 802.11. The XSMs and Stargates were grouped into sections. Each Section has 198 XSMs and 5 Stargates. The XSMs and Stargates were to be placed in pre-determined locations in the field. They were to self-configure themselves into sections and form a connected network. The Stargates first initialized themselves and reported their location to the base-station. The messages from the XSMs in a section were relayed to the Stargates which sent them over the tier-two network to the base-station. Due to the limited memory resources of the motes, they had to be reprogrammed after they were deployed.

All in all, 9184 XSMs and 274 Stargates were to be used in the experiment [18]. The final experiment was to be monitored from a 300ft-high observation tower.

GUI description and features. The GUI design was driven by the scale of the experiment, both in the number of computing devices and the geometry of the protected area. This required enhanced navigational capabilities. The user needed to have a global picture of the experiment as well as a detailed view. The user also needed to access the experiment from various vantage points, such as the observation tower or the ground level.

To accommodate the requirements, we split the application window into two parts: a Detail View and a Full View. The *Detail View* presents a specific section of interest in three dimensions. The user can zoom and pan the area as well as view the area at different angles and magnifications. The *Full View* displays a two dimensional map of the entire protected area with the two main objects: Stargates and target tracks. This gives the user a complete view of the area at a glance. The user can quickly obtain a detailed view of a particular section of interest by clicking on it.

As the application increased in complexity, the

amount of information to be graphically displayed also increased. An *Additional Information Panel* was added to accomodate extra data to be provided to the user. It displayed the current program version running on the network, number of active targets, number of XSMs alive in the currently viewed section. This panel was made configurable so that the developer could quickly add other parameters to display. The *Display Panel* was designated to display generic network statistics. Additional color coding was added to distinguish various program states of stargates. Due to their large number, the motes were to be displayed only when they participated in target detection. The target icons were brought in compliance with military map conventions. The other features of the previous version of the GUI were updated to accomodate the current changes. The screenshot of the ExScal GUI is shown in Figure 5. In this picture, the small spheres designate the motes, the dashed line is the track of the target, and the cube is the actual detected target classified as a soldier.

Architecture updates. The architecture diagram for the ExScal GUI is shown in Figure 6. This version of GUI underwent a number of modifications. We replaced Jazz used in LITeS with Java3D [5]. Java3D provides a set of object-oriented APIs that can be used to build, render and control the behavior of 3D objects and visual environments. Java3D also provides a set of zooming and panning functions that are more powerful than those of Jazz. To provide a useful reference to the outside world, we had to use a high-resolution image of the protected area. Quick redrawing and magnification of this image required advanced image manipulation functions. We used Java Advanced Imaging (JAI) [3] APIs by SUN Microsystems for image manipulation. We streamlined the message transfer by replacing JMX [4] with our own routines for event notification.

The messages processed by the GUI reflected the multi-tiered architecture of the application. There are three types of messages. A *Mote Message* describes the state of a particular mote and includes its ID, parent ID, co-ordinates, power level and readings of its on-board sensors. A *Stargate Message* describes the state of a particular stargate and includes its ID, parent ID, real space co-ordinates, program version it is running. A *Target Message* a detected intruder and includes the ID of the target, its classification, coordinates, and a list of IDs of all motes participating in its detection.

Experience. Due to the complexity of ExScal, the application development process was more structured. Several times we participated in milestone demonstrations. There we had to show the evolving GUI prototype, discuss interfaces and receive feedback from the other teams. Currently, the GUI development is on track for the GUI to be used in the final demonstration.

5 Conclusion

The need for a graphical user interface that visualizes the operation of the application is inherent in the field of wireless sensor networks. However, due to the wide range of potential applications for sensor networks, a tool that accommodates the requirements of them all is hardly a possibility. A more viable option is to develop a programmer's toolkit implementing the features most frequently used to visualize sensor network applications. This toolkit needs to be easily extensible and customizable to the needs of a particular application.

We propose to develop such a toolkit on the basis of the GUI we described in this paper. We plan to capitalize on the experience we gained developing the GUI for the various programming projects to design the toolkit so that it will suit many a sensor application.

References

[1] Comprehensive java performance tuning. http://www.quest.com/jprobe/.

[2] Exscal's home page. http://cast.cse.ohio-state.edu/exscal/.

[3] Java advanced imaging (JAI). http://java.sun.com/products/java-media/jai/.

[4] Java management extensions (JMX). http://java.sun.com/products/JavaManagement/.

[5] Java 3D. http://java.sun.com/products/java-media/3D/.

[6] Mica 2: Wireless measurement system. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/6020-0042-06_A_MICA2.pdf.

[7] Stargate: A platform X project. http://platformx.sourceforge.net/.

[8] A structured graphics 2D framework. http://www.cs.umd.edu/hcil/jazz/.

[9] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y-R. Choi, T. Herman, S. S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: A wireless sensor network for target detection,classification, and tracking. *Computer Networks*, 46(5):605–634, December 2004.

[10] S. Carr, C. Fang, T. Jozwowski, J. Mayo, and C.-K. Shene. Concurrentmentor: A visualization system for distributed programming education. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 4:1676–1682, 2003.

[11] D. Culler. Networks of tiny devices embedded in the physical world. *Workshop III: Massively Distributed Self-Organizing Networks*, 2003.

[12] M. Demirbas, A. Arora, and M. Gouda. A pursuer-evader game for sensor networks. *Sensor Network Operations, IEEE Press*, 2004.

[13] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. *Submitted to International Workshop on Information Processing in Sensor Networks (IPSN'05) Special track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS)*, 2005.

[14] P.K. Dutta. On random event detection in wireless sensor networks. Master's thesis, The Ohio State University, 2004.

[15] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin. EmStar: An Environment for Developing Wireless Embedded Systems Software. Technical Report CENS Technical Report 0009, Center for Embedded Networked Sensing, University of California, Los Angeles, March 2003.

[16] J. Hill, R. Szewczyk, A. Woo, D. Culler, S. Hollar, and K. Pister. System architecture directions for networked sensors. *ACM SIGPLAN Notices*, 35(11):93–104, November 2000.

[17] J.L. Hill and D.E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November/December 2002.

[18] S. Kumar and A. Arora. Exscal topology for node deployment. *ExScal Note Series: ExScal-OSU-EN00-2004-01-30*, 2004.

[19] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[20] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J.S. Baras. Atemu: A fine-grained sensor network simulator. *IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[21] B. Tropol, J.T. Stasko, and V. Sunderam. Integrating visualization support into distributed computing. *Proceedings of the 15th International Conference on Distributed Computing Systems*, 1995.

[22] L. Wang. MNP: multihop network reprogramming service for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 285–286. ACM Press, 2004.