Working with the CONSUL Middleware to Create an Instrumented Construction Site Application

Kelly Chang
Department of Computer Science
University of California at Santa Cruz
kpchang@ucsc.edu

1. INTRODUCTION

Research regarding mobile ad-hoc networking has expanded over the past few years. Ad-hoc networks are similar to normal wireless networks in several ways, including the technology used (e.g. the 802.11 protocol). The critical differentiating factor of an ad-hoc network from a normal wireless network is the lack of a fixed network infrastructure; in an ad-hoc network, communication links are formed between devices within wireless communication range. When the devices are mobile, this lack of infrastructure makes it very difficult for an application programmer to create an application for an ad-hoc network; the network topology changes rapidly in response to the mobility of nodes, making it impractical to know in advance the location and nature of resources in the network. Accessing information across a changing network is one of the key pieces in an application for mobile ad-hoc networks. An application running on a single node in the network isn't as powerful as an application that spans the entire network; therefore it is important that programmers are able to create applications that have that capability.

To aid in the development of applications for ad-hoc networks, a middleware is often used. A middleware is a library that supports a collection of low-level functions. These functions can be used by the programmer through the simplified interface provided by the middleware. CONtext Sensing User Library, more often referred to as CONSUL, is one such middleware. [1] CONSUL covers all of the details of acquiring information from the network, thereby allowing the application programmer to focus on the application. Our goal is to demonstrate that the use of CONSUL in application development involving ad-hoc networks is easier than developing an application without it. To show this, the project will use CONSUL in the development of an ad-hoc network application designed to support management of an instrumented construction site.

2. BACKGROUND

Ad-hoc sensor networks may support a number of useful applications, but it can be difficult to develop an application that uses them. Sensor networks are dynamic and constantly evolving. Nodes often enter and leave the network, making it difficult to know if a certain node exists within the network at any given time. This makes issues such as mobility, uncertainty, node discovery, and routing very important when it comes to sensor networks. [2] There are also the issues of power and resource constraints. The sensors used in the networks do not plug into an electrical outlet and instead run off of batteries, which greatly limits the options available to the programmer. In addition to their power supply being limited, the sensors also have a restricted amount of processing power and memory. [3]

Several middleware solutions have been designed with these problems in mind and can be used to aid the programmer with application development. Some of these solutions include the context toolkit, CONSUL, EgoSpaces, and Agilla. For brevity, EgoSpaces [4] and Agilla [5] will not be discussed.

2.1. THE CONTEXT TOOLKIT

The context toolkit [6] uses context widgets to provide the application with information from the networked sensors. These context widgets are similar to normal GUI widgets. The context widgets hide the complexity of the actual sensors used and abstract the information to suit the expected needs of applications. They provide reusable and customizable building blocks that manage sensing of a particular sensor or group of sensors. The developer includes these context widgets into the application, allowing the application to access the information on the sensors easily.

Despite all of its pros, the context toolkit has two very large cons: it has a large and complex overhead, and it has limited aggregation. As stated earlier, sensor networks are often power and resource constrained. The larger and more complex of a middleware means that the actual application running on the sensors will have to be that much smaller. Aggregation is also a key issue because it is how data is collected from the network and obtained by the application. The context toolkit needlessly limits the types of aggregations it is able to perform by moving the context aggregation functionality away from the client and into the middleware. [1]

2.2. CONSUL

CONSUL uses a simplified interface to allow application developers access to context information. It is designed to give novice programmers the ability to create context-aware applications. Unlike the context toolkit, CONSUL has less overhead and more options for data aggregation.

CONSUL handles sensing by allowing software to interface with sensors connected to a host. CONSUL provides a unified interface to sensors through the use of monitors. Any kind of sensor can be interacted with simply by using the "getValue" or "setValue" methods of a CONSUL monitor. Since there are lots of physical sensors out there, each with a different way of accessing information, the use of CONSUL monitors can make programming easier since developers do not need to know the specific implementation details of each physical device in order to access its data. CONSUL also simplifies the process of discovering the sensors in the network. CONSUL provides a MonitorRegistry that automatically discovers and manages a collection of CONSUL monitors that are available in the network; applications simply ask the MonitorRegistry for a monitor by a descriptive name.

Instead of including an entire widget, like the programmer would have to do with the context toolkit, the only thing needed to incorporate CONSUL into the application are a few lines of code that are used by the application to get the monitor value. CONSUL also provides a library for the types of monitor values that are typically provided by physical sensors. Application developers are able to add new monitor values to the library as needed, thereby allowing for additional types of data and sensors to be considered and for new types of aggregation of data values to be incorporated. [1]

3. DEVELOPING THE CONSTRUCTION SITE APPLICATION

In a real life construction site, both the workers and supervisors need to be advised on the status of the materials and equipment for any safety hazards, malfunctions and shortages. Our application would interface with devices that would keep these people aware of any possible problems or delays. Motes could be deployed across the construction site, providing both the workers and the supervisors with information on materials and equipment. For example, if there were a possible hazardous chemical leak, chemical sensors would be able to detect the leak and inform the workers and the supervisors of it. They would then be able guide anyone in the surrounding area to evacuate the site safely.

Our original plan for the construction site application was to create a simulation using iRobot Create robots and the motes. The iRobot Create would serve as a worker or piece of equipment, and the motes would function as sensors attached to equipment and materials. Our goal for this project is to develop CONSUL monitors for the sensors to support the construction site application.

Extensive research regarding ad-hoc networks [2, 3], middlewares [1, 4, 5, 6] as well as TinyOS [12] and NesC [13, 14] was done prior to any actual development.

3.1. WORKING WITH TINYOS AND NESC

Working with the motes proved to be a difficult task. The motes run on an operating system called TinyOS, which is programmed using a language called NesC. The reason behind using a specially designed operating system and language on the motes is due to their power and resource constraints. TinyOS is aptly named as it is specifically designed to run on "networked sensors with minimal hardware requirements". [7]

The unfortunate consequence of TinyOS and NesC being specifically designed for the motes is that the applications for the motes must also be programmed using NesC. NesC is similar to C, but it uses a different compiler, making it a language unique to TinyOS. [7] The TinyOS tutorials [8] provided some guidance for programming in NesC, but adjusting to the language can be a time consuming process.

3.2. CREATING A CONSUL MONITOR

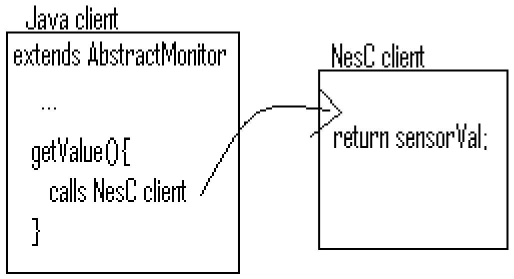
To create a CONSUL monitor for the motes we need to extend the AbstractMonitor class and implement the getvalue and setValue methods. Specifically, the getValue method would have to be implemented such that it uses the physical sensing devices native method for getting values from its sensors. This would mean getting values for the temperature, humidity and light sensors as they are the most common sensors on the motes we had.

We began by determining how to best access the sensors on them. The TinyOS developers provided sample applications with all of the operating system's distributions. Two of these applications sampled one sensor on the mote, a generic DemoSensor, and showed the values either through the mote's LEDs or a GUI application, also included in the TinyOS distribution. Through these applications and the accompanying tutorials [8] we learned how to access the sensors.

Our next step was to direct the output to the terminal for debugging and for application display purposes. The first approach taken for this was through the printf command. However, unlike languages such as C and Java, implementation of printf in NesC is more than simply calling printf. The printf client must be wired to the application, and the buffer needs to be flushed in order to actually view anything. A few fruitless attempts were made at incorporating the printf command into the sensing application, including one which required very little wiring. [11]

After our unsuccessful tries with the printf command, we switched to using the serial port. This turned out to be a much less complicated approach than the printf command. The application still needed to be wired to the serial port client; however, documentation for this process was much more plentiful and straightforward than the printf command. [8]

From here we were able to view the various values on the motes as output to the terminal. Work began on trying to adapt this NesC application into a NesC client that could be called from the Java client to return the value on the sensors. The Java client would act as a monitor and have the getValue and setValue methods within it. The getValue method would call the NesC client, which would return the value from the specified sensor on the motes in the network.



4. FUTURE WORK

Ultimately we would like to be able to compare the development process between using CONSUL and using QueryME, a middleware created by Dr. Payton. [9] In order for this to be possible, however, we must first create a monitor that interfaces with the motes by extending the Abstract Monitor. We hope to accomplish this by completing our NesC client application which will return the values from the motes' sensors to an outside monitor programmed in Java. This way we will be able to avoid porting the entire middleware to NesC.

Once this part is completed we will be able to test the monitors using the application created by Robert Goodrich during the REU at UNCC. [10] Providing that the application and motes work together successfully, we will then be able to rewrite the application using QueryME. QueryME is a more sophisticated middleware. It uses CONSUL to provide a unified interface to sensors while simultaneously providing additional functions and features for the application programmer. [9]

We are also looking to deploy the application and a similar sensing network in a real construction environment. This system would, theoretically, provide the workers and supervisors with information on the condition of the material, possible equipment failures and safety hazards.

5. CONCLUSION

A significant amount of time was spent researching background information on mobile ad-hoc networks, sensor networks, middlewares, TinyOS, and NesC. Despite this, there were still a few issues with adjusting to programming in NesC and debugging applications. Given more time a testable application could have been constructed, and we already have a general outline of the steps needed to complete this project. Much was learned about these topics as well as the research experience itself, and this alone was well worth the time and effort.

6. REFERNCES

[1] G. Hackmann, C. Julien, J. Payton, and G.-C. Roman. “Supporting Generalized Context Interactions,” *Proceedings of the 4th International Workshop on Software Engineering and Middleware, co-located with ASE'04, Linz, Austria, T. Gschwind and c. Mascolo (editors), Lecture Notes in Computer Science 3437*, pp. 91-106, March 2005.

[2] C.-Y. Chong and S. P. Kumar. “Sensor Networks: Evolution, Opportunities, and Challenges,” *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247-1256, Aug. 2003.

[3] J. Blumenthal, M. Handy, F. Golatowski, M. Haase, and D. Timmermann. “Wireless Sensor Networks – New Challenges in Software Engineering,” *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA ’03. IEEE Conference,* vol. 1, pp. 551-556, Sept. 2003.

[4] C. Julien and G.-C. Roman. “Egocentric Context-Aware Programming in Ad Hoc Mobile Environments,” *In Proc. of 10th Int'l Symposium on the Foundations of Software Engineering*, pp. 21-30, Nov. 2002

[5] C.-L. Fok, G.-C. Roman, and C. Lu. "Mobile Agent Middleware for Sensor Networks: An Application Case Study," *Fourth International Symposium on Information Processing in Sensor Networks, 2005. IPSN 2005*, pp. 382-387, April 2005.

[6] D. Salber, A. Dey, and G. Abowd. “The Context Toolkit: Aiding the Development of Context-Aware Applications,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: the CHI Is the Limit (CHI '99)*, pp. 434-441, May 1999.

[7] “TinyOS FAQ,” [Online]. Available: http://www.tinyos.net/faq.html. [Accessed: Aug. 28, 2008].

[8] “TinyOS Tutorials,” May 16, 2008. [Online] Available: http://docs.tinyos.net/index.php/TinyOS_Tutorials. [Accessed: Aug. 28, 2008].

[9] J. Payton. “A Query-Centered Perspective on Supporting Context Awareness in Mobile Ad Hoc Networks,” pp. 1-20, March 2006.

[10] R. Goodrich. “Developing Instrumented Construction Site Applications using the CONSUL Middleware,” *2008 Research Experience for Undergraduates. REU 2008*, pp. 1-2, Aug 2008.

[11] “Modified printf tar file/thread,” [Online] Available: http://www.mail-archive.com/tinyos-help@millenium.berkeley.edu/msg15182.html. [Accessed Aug. 15, 2008].

[12] P. Levis. “TinyOS Programming,” June 28, 2006. [Online] Available: http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf. [Accessed July 14, 2008].

[13] D. Gay, P. Levis, R.v. Behren, M. Welsh, E. Brewer, and D. Culler. “The *nesC* Language: A Holistic Approach to Network Embedded Systems,” [Online] Available: http://www.cs.berkeley.edu/~pal/pubs/nesc.pdf. [Accessed July 14, 2008].

[14] D. Gay, P. Levis, D. Culler, and E. Brewer. “nesC 1.2 Language Reference Manual,” Aug 2005.