Integrating Sensor-Network Research and Development into a Software Engineering Curriculum

Michael J Ocean mocean@cs.bu.edu
Assaf J. Kfoury kfoury@cs.bu.edu
Azer Bestavros best@cs.bu.edu

Computer Science Department
Boston University

September 16, 2006

Abstract

The emergence of a sensor-networked world produces a clear and urgent need for well-planned, safe and secure software engineering. It is the role of universities to prepare graduates with the knowledge and experience to enter the work-force with a clear understanding of software design and its application to the future safety of computing. The SNBENCH (Sensor Network WorkBench) project aims to provide support to the programming and deployment of Sensor Network Applications, enabling shared sensor embedded spaces to be easily tasked with various sensory applications by different users for simultaneous execution. In this report we discus our experience using the SNBENCH research project as the foundation for semester-long project in a graduate level software engineering class at Boston University (CS511).

1 Introduction and Motivation

As the commoditization of computing and sensing devices continues, we anticipate the replacement of traditional fixed and "closed" sensory networks in public or managed spaces (*e.g.*, smoke detectors, motion sensors, closed circuit video, *etc*. in malls, airports, libraries, schools, *etc*) with a flexible programmable SN infrastructure. To fully leverage such an infrastructure any novice user should be able to program the SN quickly and have their application run in parallel with other SN applications, transformming public spaces into public sensing and computing spaces. We have developed such a SN support infrastructure that we call the SNBENCH (Sensor Network WorkBench), allowing us to confront those issues surrounding these emergent SNs today.

Software and safety risks are greatly magnified in a society in which Sensor Networks (SNs) common place and nearly all computers are remotely accessible. Sensor Network applications are more distributed, concurrent, modular and interactive – in very significant degrees – than the traditional execution environment of stand-alone applications. Development environments for such applications are also undergoing great changes, frequently calling for new compatible ways to develop software to accommodate these new needs, yet typically lagging behind the enormous and relentless hardware innovations and emergent networking concepts. The development challenges inherent to distributed sensory programming place stringent demands on the software developer who must often approach novel and unique problems ill-equipped with traditional tools and approaches.

Software Engineering, in as much as it tries to be an established area of the CS curriculum with well defined topics, tends to be even more conservative. This conservatism is reflected in current SE books and in the organization of SE courses, which tend to tackle software-development problems of the traditional programming environment very well, but not so well those of the new networked and embedded environments.

At BU, we are trying to realize a SE course which can adapt to this fast-changing reality. There are limits, of course, the result of tension between the goal to adapt quickly and the need to teach well-established principles, unavoidably more suitable to the traditional programming environment than the new emerging one. Regardless, it is necessary to attempt to find such a balance, to the benefit of the students such that they may be best prepared for the future of software engineering work.

Toward this goal, this report details our experience piloting the use of our SN workBench (SNBENCH) as the foundation for a semester-long final project for a 500-level Software Engineering class here at Boston University. The SNBENCH provides a high-level programming language and complete run-time support for the deployment of programs on to a Sensor Network. There is obvious reciprocal benefit to such a collaboration; The curriculum benefits from the integration of the new paradigm of programming sensor networks, the research and development agenda of the SNBENCH benefit from both exposure and potential additions to the code base, and students benefit from exposure to an exciting domain in which their work is not a pointless intellectual exercise.

In this report we first provide an overview of the goals of the SNBENCH project and describe its architecture and relevant components. In section 3 we discus in detail our goals for the use of SNBENCH within the Software Engineering course. In section 4 we discus the projects offered to students as well as some of the project submissions. Finally we conclude in section 5 with closing thoughts and lessons learned for future efforts.

2 SNBENCH

With the increased ubiquity of sensing, computing, networking, and storage devices, a different model of Sensor Network (SN) infrastructures is emerging, whereby the owner of the SN and its users may be different entities, and the users of the SN may autonomously deploy independent applications that share the SN infrastructure. Harnessing the power of such shared SN infrastructures will hinge on our ability to (1) streamline the process whereby relatively unsophisticated "third parties" are able to rapidly develop and deploy their applications without having to understand or worry about the underlying, possibly complex "plumbing" in the SN infrastructure supporting their applications, and (2) ensure that the applications of autonomous users with potentially conflicting missions are "safe" with respect to their own operation, the operation of other SN applications and the shared SN infrastructure as a whole.

To this end we have developed SNBENCH (SN Work-Bench); SNBENCH embodies a programming environment and associated distributed run-time system that support the entire life-cycle of programming SN applications The SNBENCH is currently deployed on top of the SN infrastructure in our Graduate laboratory which we call a "Sensorium." The Sensorium is a collection of wired and wireless networked video cameras, motes, processing units and a terabyte database. The SNBENCH manages these resources and enables the execution of ad-hoc sensory programs specified over the Sensorium's monitored spaces managing compilation, resource allocation, scheduling and dispatch.

SNBENCH provides programmatic access to the Sensorium via a strongly-typed functional style programming language. A SNBENCH program is specified in the SNAFU (SN Applications as Functional Units) functional-style programming language. SNAFU applications are written for the SN as a whole, and serves as an accessible, high-level language for developers to glue together the functionalities of sensors, actuators, processing, storage, and networking units to create stateful, temporal, and persistent programs.

A SNAFU program is compiled into a Sensorium Task Execution Plan (STEP), which takes the form of a directed acyclic graph (DAG), in which the nodes describe sampling, computation, and communication operations required to execute the program. An execution plan may consist of both nodes that are either free to be placed wherever sufficient resources may be found or those nodes that require a specific physical resource. A STEP is analogous to a program that has not been linked.

The Sensorium Service Dispatcher (SSD) is a run-time agent responsible for the linking and scheduling of a STEP onto the local SN infrastructure, finding resources capable of supporting sub-graphs of the STEP graph and allocating them as appropriate. The SSD optimizes the use of resources and identifies common subexpressions across already deployed execution plans such that computation resources may be shared and/or reused. The SSD relies heavily on the Sensorium Resource Manager (SRM), a registrar of computing and sensing resources available in the system at present. The SSD decomposes a single unlinked STEP graph into several smaller linked STEP graphs and dispatches those graphs onto available Sensorium participant nodes.

Each Sensorium participant hosts a Sensorium eXecution Environment (SXE), a run-time system that realizes the abstract functionalities presented to SNAFU programmers as basic building blocks. An SXE interprets and executes partial STEP graphs that have been delegated to it by the SSD. Such a partial STEP graph may involve computing, sensing, storage, or communication with other SXEs.

The SNBENCH has been developed as a research project within Boston University to support and accelerate other research that will be developed either on top of the SNBENCH infrastructure (*i.e.*, research that leverages the SNBENCH to advance the state-of-the-art in other areas such as Computer Vision, Network Security, *etc*) or to improve future generations of the SNBENCH infrastructure (*e.g.*, resource management, allocation, negotiation, scheduling, *etc*). A more detailed overview of the SNBENCH vision may be found in [1] while implementation details may be found in [2].

The SNBENCH has already been used successfully in an extrinsic capacity in support of a Graduate course in Artificial Intelligence during the Fall 2005 semester. This report focuses on the integration of the SNBENCH into a graduate-level software engineering course during the Spring 2006 semester.

3 Software Engineering Goals

The Graduate Software Engineering course is offered by the Boston University Computer Science Department with an enrollment of roughly thirty students who are a mix of undergraduates, masters and PhD candidates. Throughout the semester students are taught software engineering principles including (but not limited to) design patterns, software life-cycle testing, type-safety, object-oriented programming principles and the use of concurrent versions system. Students are expected to complete a semester-long final project that aims to provide a realistic corporate development experience in which students work in small groups to develop modular software components that operate either within or on top of some larger work. Rather than use some contrived and static textbook project as the foundation for this work, for the Spring 2006 semester we elected to leverage the Sensor Network WorkBench (SNBENCH) research project as a platform for student projects. Sensor Networking workbench that eases the development and automates the deployment of Sensor Network applications on a shared Sensor Network infrastructure. Working within the SNBENCH students are given the opportunity to work in an active project in a new domain (SNs) with expertise and support readily available on campus. Our goal was to expose the students to engineering issues surrounding large-scale distributed sensory systems as well as the traditional topics associated with a SE curriculum. Within the domain of SNs students face concrete manifestations of what are largely theoretical concerns to most Undergraduate students (e.g., Security, Safety, Error Checking, Space and Time Complexity).

The student projects represent a significant part of their final grade for the course and easily comprise the bulk of the student's effort for the semester. Rather than a series of smaller assignments, the students are expected to work (as a group) on the same project throughout the semester. As this class integrates graduate and undergraduate students, expectations for undergraduates are adjusted appropriately, however the exposure to the graduate students helps give the undergrads a different perspective on their work. It should be stressed that students projects are non-competitive; Collaboration and discussion between groups is strongly encouraged.

To enable the SNBENCH to be used in such a capacity, the students were provided a variety of documentation including high level motivation, functional descriptions and implementation details of existing SNBENCH software components and details of the relevant interfaces/specifications of SNBENCH modules required for the student projects. To a large extent the text for this documentation already existed across multiple sources and the effort primarily consisted of pulling these resources together in a single, publicly available, well organized web page for the students [3].

In addition to the static documentation, students were given access to the lead developer on the SNBENCH project who gave a lecture over-viewing the SNBENCH vision and each of the projects in detail, met with each group individually at least once, and offered email support when meeting in person was not an option. In an attempt to ensure student efforts did not fall off track, each group gave a mid-semester project status presentation (also attended by the lead SNBENCH developer) and comments and criticisms were given at that time. In the following section we describe the individual recommended project assignments from which the students were able to choose.

4 Projects

To provide students with some initial direction, we provided five "suggested" projects that ranged in complexity, effort, integration and open-endedness. Suggested student projects were broken into two larger categories; those that enhance the SN infrastructure (those that are internal to the SNBENCH) and those that are enhanced by the infrastructure (those that rely on, run atop the SNBENCH). Given the multi-modal nature of the SNs we target, student projects were not limited to any single area of expertise. The projects topics allowed students to learn proper Software Engineering principles while implementing solutions to challenges in Computer Vision, Databases, Networking, Distributed Systems, Human Computer Interaction and other research areas.

The project suggestions and some results of these projects are listed below and are presented in order of increasing complexity, time and integration effort. We have omitted project topics that were either not selected by any group or those for which the submitted work was not selected for reuse within the SNBENCH code base.

When discusing results of student work, it must be emphasized that all groups have completed this work in very little time (approximately twelve weeks of work for a single course in addition to a standard course load and other responsibilities.) As such any omissions, flaws, or otherwise lack of completion is assumed to be due to this sever time constraint. Naturally, not all projects completed this semester were selected to be reused in the SNBENCH project. This does not speak to the quality of the work in all cases, as the selection also reflects the cost-benefit analysis with respect to the amount of time required to port the student work into the code base relative to the benefit of these modifications to the SNBENCH project overall.

4.1 STEP Programming GUI

This project recommended that the students generate a graphical user interface (GUI) with which users can compose new SNBENCH programs. The Sensorium Task Execution Plan (STEP) language is human readable and we (developers) are able to compose programs directly in STEP using a regular text editor. As STEP is a direct representation of the abstract syntax tree of a program, one can imagine a more accessible graphical programming interface in which a palette of nodes may be offered to be dropped and wired into a flow of execution that may saved directly into STEP (XML). The details of this project ask the group to build a tool that allows the user to specify the STEP program execution graphically from an extensible palette of functionalities and allow the generated program to be stored as a properly formatted STEP file (XML).

As a stand-alone component, this project was intentionally crafted to be external to the bulk of the SNBENCH code base and require no deep integration into the existing SNBENCH components. It was, however, highly recommended that students reuse the existing internal data structures to save time and avoid redundant work. The only required integration was understanding the syntax and semantics of the STEP graph. Although a group may take this work to be as simple as developing any graphical interactive development environment and in no way special to the Sensor Network domain, students were expected to consider the needs of a novice SN developer and offer an easy and intuitive interface to the capabilities of the STEP programming language that would provide graphical access to task the resources of the Sensor Network.

The work completed by [4] was the best work submitted under this topic, offering a graphical STEP programming environment that supports the loading and saving STEP files, tabbed editing and an XML based pallet of capabilities. This group has also clearly considered what user's may want from their editing tool as they offer tabbed editing, a (cleverly integrated) basic STEP validation engine and an option to post STEP programs for deployment directly from the editor. In addition, the layout of the editor they provide is straight-forward and effective and as such this work stands out as a good basis for future development of a complete STEP graphical development environment.

Although the interface holds a lot of promise, the submitted work does seem incomplete from a usability point of view, unarguably due to time constraints. This included some operational flaws discovered during our own testing and a lack of descriptive messages/prompting leading to weak in-program communication with the user.

Although this project assignment is, by design, intended to be only nominally integrated to the existing code base (it's usage is outside of the run-time scope of the SNBENCH), a surprising problem occurred when attempting to integrate this work. This work separates the graphical construction and the STEP graph validation into two distinct operations. As such, during the composition of a STEP graph, the user is allowed to construct incomplete or erroneous STEP graphs, presumably on the way to a final, complete STEP graph. Thus, many of the operations that this work allows the user to graphically perform on a STEP graph may be entirely safe/normal in this offline and transient context, however some of those intermediate STEP graphs violate data structure invariants that, in an online (*e.g.*, interpretation and evaluation) context guard against data corruption or other undesirable behavior. Since the methods of the existing STEP data structures operate under the assumption that the graph is in a structurally complete or sane state, naturally there are "assertions" in the methods of the STEP data structure to ensure this is the case. As such, running this project "off-the-shelf" with assertions enabled results in premature termination due to otherwise irrelevant assertion failures.

As such the integration work for this project almost entirely entailed changes to allow the GUI to coexist with the existing STEP data structures. This included replacing and updating some of work's interfaces to the SNBENCH objects, adding real-time checks in the GUI to prevent the construction of non-sensical STEP programs before they could be passed to the STEP data structures, and many other assorted small reliability and performance improvements.

There is little doubt that given more time this work could be polished into a superior interface. Unfortunately due to other constraints, the resources required to perfect this work are unavailable, yet benefits of this work could easily far outweigh its flaws. Thus beyond the small corrections indicated above we have included this work otherwise "as-is", repairing bugs that were discovered during integration but otherwise documenting its benign shortcomings with the intent that improvements will be added as an ongoing effort, as time permits.

4.2 Expanding SXE Capabilities

This project suggestion was by far the most open-ended and domain specific, asking students to take the perspective of "SNBENCH users" and help build support infrastructure to enable interesting, useful or otherwise desirable applications on the SN framework. A successful project in our eyes would be one in which the students enabled some useful and well motivated application to run on the SNBENCH, adding any lacking sensor functionalities (opcodes) to support those applications. The SNBENCH Sensor eXecution Environment has been designed with opcode extensibility in mind such that the code to integrate the opcode was trivial, yet they would need to understand the big picture of the SNBENCH, the provided SXE Opcode interface and the syntax and semantic of STEP such that they could stitch together the program that leveraged their new functionalities. The modularity of Opcode development also serves to aid the SNBENCHinsofar as newly developed Opcodes should easily be added to the main code base without modification.

Suggested opcodes were given in two domains; Image manipulation opcodes might include image differencing, motion detection, average image intensity, and face detection. Another suggestion included data manipulation opcodes which requires the development of some infrastructure support in addition to opcodes that interact with this new infrastructure (*e.g.*, `hash_table_init`, `hash_table_add(k,v)`, `hash_table_remove(k)`, `hash_table_check(k)`, `hash_table_get(k)`, *etc*.). Students were encouraged to implement any opcode they were inspired to provide, yet reminded to try to create useful opcodes given the domain and to discus alternate opcodes with us before proceeding.

This project's open-ended nature proved to be quite seductive as more groups selected this project than any other; Ultimately it proved difficult for the groups to simultaneously keep the larger vision of the SNBENCH in focus during their development. Most groups turned in little more than the suggested distributed storage opcodes.

Even though somewhat disorganized and informal in its presentation, report [5] by three undergraduate students was the product of a group that had seriously considered the SNBENCH's goals and domain and had strived to experience the novelties of programming within the SNBENCH head first. The group's demonstration leveraged their new contributions to demonstrate a hash server dependent motion sensor that triggers an email notification when motion is detected.

This work is a clear asset to the SNBENCH; The submission includes over 30 useful new Opcodes for use within the STEP programming language including logical, mathematic, image manipulation, e-mail and distributed storage operations. To support distributed storage, this group implemented a stand-alone centralized hash server, a client for the individual SXEs, distributed hash-table Opcodes for the SXE client and finally a web interface for users/administrators to inspect the hash-table state.

Given SNBENCHś modular Opcode architecture, after testing the new Opcodes were literally copied and pasted directly into the code-base. Integrating the hash-table opcodes was just as simple but their proper operation relies on the SXE-side hash client and hash server. Merging changes for the hash client into the SXE were straight forward due to some thoughtful design decisions and to properly fold the centralized hash server into the SNBENCH architecture it has been ported to a threaded task that is launched, controlled and monitored by the Sensorium Service Dispatcher. Closer inspection of the client and server code revealed some synchronization issues that were trivially corrected by wrapping their data storage structure with reentrant locks. We discus some general issues involving teaching synchronization details in a software engineering class in section 5.

4.3 SXE Scheduler Modification

This project suggestion recommends students make changes to the current task scheduler to improve the execution of STEP programs within the Sensor eXecution Environment (SXE). This is an inherently difficult task and considered to be the most involved project, by far. Changes to a scheduler are difficult to test and synchronization issues are easily introduced. Thus, this project was available to graduate students only on a "by permission only" basis.

Students who selected and hoped to achieve a positive result for this project needed to be familiar with scheduling and systems topics and must acquire a deep understanding of the existing SXE's scheduling and evaluation operations as well as the runtime semantic of the Sensorium Task Execution Plan (STEP).

To better understand the work involved in this task, we illustrate the SXE's basic mode of operation. Each SXE accepts a partial STEP graph which indicates the exact tasks (nodes) it is expected to execute and the dependency between the tasks (edges). As the STEP graph is directed (computations percolate upward) a STEP node higher in the graph can not be evaluated until the lower nodes have been evaluated. Thus leaves are evaluated first and so on, upward, until a result reaches the root. Some branches of the graph may not need to be evaluated at all (e.g., they are one branch on a conditional) such that checking if a node should be evaluated depends on state of both children and parents (if this nodes' children have produced "fresh" values and if the result of this node is wanted farther up the tree).

Rather than traverse the graph repeatedly, the initial SXE's scheduler is a simple round robin scheduler that checks for "enabled" nodes anywhere in the graph. Enabled nodes are moved to a queue and are separately executed by a single thread of execution . Suggested optimizations for this project include using multiple threads to execute multiple enabled nodes simultaneously (*e.g.*, nodes of independent branches) and/or adding different scheduling algorithms that ensure the nodes get a either a fair or explicitly desired share of the SXE's cycles. Both of the groups that selected this project topic achieved a fantastic result.

The scheduling work of [6] adds true hierarchical scheduling to the Sensor Execution Environment and does so with absolutely minimal changes required to the existing SXE code-base (only two methods of one class have been modified). The scheduling code is quite complete and offers the construction of a hierarchy of schedulers including Round Robin, Fixed Priority and Proportional Share scheduling. The integration to the SXE is very clean and simple yet the integration is in somewhat incomplete and inefficient due to a lack of time. Although the work supports a variety of schedulers, code augmenting the SXE to enable reading scheduler Flow-types (*i.e.*, scheduler directives) from the STEP graph was not provided, causing much of the project's benefit to not be utilized from the SXE.

The scheduling work of [7] on the other hand is considerably more ambitious with respect to integration with the existing SXE. Although the scheduling offered lacks a true hierarchical scheduler, this work offers a hybrid scheduler and the deep integration into the existing SXE implementation enables access to all their provided schedulers from within an active SXE. To support this, the group has added parsing for scheduler flow-types and modifies the scheduler to enable scheduler annotated STEP nodes to be enqueued to the correct queue. This work is exceptional in its completeness and scope (*e.g.*, excellent design, an automated regression test-suite is provided, and synchronization has been added around some existing STEP data structures.).

Unfortunately, this scheduler has an intricate enqueueing mechanism that makes its integration into the SXE more intrusive. This new approach annotates some nodes with new mechanisms for enqueueing based on partial STEP graph traversals, and flow types for unannotated nodes are rediscovered by each node re-traversing part of the graph every time this node is enabled for execution making flow-type annotations a necessity to avoid this penalty. In addition small issues surrounded this work's STEP synchronization approach, which uses a sporadic lock/release/repeat that potentially allows two simultaneous clients to manipulate data based on transient/expired data views. It is clear that these particular synchronization flaws result from an attempt to minimize changes to the admittedly complex and involved existing STEP Graph data structure. In our integration effort we have added proper synchronization that is based on our in-depth knowledge of the underpinnings of the STEP Graph data structure.

Given two excellent projects we were able to adopt the best aspects of both works into the SNBENCH code base. The scheduler of [6] has been integrated as the scheduler, adapted with [7]'s flow-type parsing and support to support programmatic access to hierarchical schedules. The new resulting SXE scheduler gives every STEP program its own proportional share of the SXE and, by default, each STEP is evaluated with its own Round Robin scheduler within its share. Scheduler flow-type specifications within a STEP graph allow the nodes within a STEP program to override their default Round Robin scheduling behavior as needed, substituting Fixed Priority, Proportional Share or a true hierarchy of schedulers with the aforementioned schedulers. When a new STEP graph or fragment is added to the SXE, flow-types are discovered for the new nodes by traversing the entire graph exactly once. From every "root" node in this SXE's STEP graph, we perform a breadth first traversal assigning new Opcodes (tasks) to their closest assigned scheduler and creating new schedulers within the hierarchy if flow-type directives are present.

5 Conclusions

The SNBENCH project has always been conceived as a foundation for accelerating various research initiatives and its use within a Software Engineering class is no exception to this mission. Integrating the SNBENCH into a SE curriculum provides the students exposure to concrete instances of new and well established design and engineering challenges in the SN domain. We conclude this report with a discussion of our achievements and areas for improvement for future semesters.

From the point of view of the SNBENCH its successful use to support this Software Engineering class is an important milestone on the path to maturity. In addition to the intrinsic stability benchmark associated with having a large experimental user base, it is clearly beneficial to have a large group of students fluent in SNBENCH programming and use. Those students are now more likely to use the SNBENCH in the future or to recommend its use to others. The project submissions described in this report have provided important and beneficial contributions to the SNBENCH framework and the effort of their authors is greatly appreciated.

From the student's perspective, this collaborative effort was well received. Students reported interest and excitement being able to work on an active project in a still evolving domain. Indeed the participation in the project became quite consuming to many of the students, some of whom petitioned for the project to be worth a greater portion of their grade and a larger part of the class over all. Those students who were informed that their work would be reused in the SNBENCH project expressed pleasure in the idea that their efforts were legitimately useful. Although not every project that was turned in was able to be absorbed into the code base, all of the students have provided useful contributions to the SNBENCH framework. Every student project has taught us valuable lessons as to how we might adjust the project assignments and student development in future semesters.

Lessons Learned and Future Adjustments

Although the collaboration was a clear success, we view this effort as ongoing and strive to improve future iterations and detail some lessons learned in this section.

Material

Adding the SNBENCH to the curriculum of the class brought many systems-centric engineering issues (*e.g.*, synchronization and resource allocation) to the forefront of the students' SE projects. Generally this material is not covered deeply in SE classes despite the clear increasing trend toward shared and distributed architectures in modern software. This trend must be reflected in the SE curriculum and as such design principles for concurrency and synchronization should become a larger part of this course.

Some students had expressed a preference for the project work as opposed to the other requirements associated with the class (*e.g.*, textbook readings, quizzes and exams). Although the textbook readings and exams can not be replaced, it may be beneficial to integrate the project material into some of the lessons more closely or to replace quizzes with small SNBENCH integrated assignments (*e.g.*, a selection of objects from the existing source may be selected to illustrate use of various design patterns).

Student Guidance

Although each group met with the lead SNBENCH developer in person once, it seems beneficial for students to discus their projects and plans more often throughout the semester. The best work done in this class tended to be that of the groups who met with the lead developer more than once during the semester. Although such meetings may be time consuming, there is little doubt that the benefit of such meetings far exceed the time investment. These meetings may also help groups stay on track and keep the big (SN) picture in mind during their development cycle.

Similarly, students present a mid-semester progress report as well as an end of semester progress report. The mid-semester report is useful to determine if groups are headed in the right direction. These presentations have a length that balances consumed lecture time and the time required to be useful. In this semester the 12 minutes per group does not shed enough light on what a group is doing and as the semester is already half over by the time of the mid-semester report, it is generally too late to recommend any drastic changes if they are needed. It may make sense to require groups to meet with faculty individually outside of class both one-quarter and half-way into the semester for a more in-depth review (in addition to the mid-semester presentation).

Student Collaboration

Despite urgings from the faculty, groups did not collaborate across groups as much as we had hoped. Although there is certainly a sense of competition that is natural when multiple groups are working on the same topic/project, there is room for groups to discus their work with one another for help. While student groups maintained public web pages that were visible by all, very few students looked at each other's web pages, and had little idea what the other groups were doing or how they were doing it. This problem is difficult to fix, but it may be the case that incentivising collaboration may solve this problem.

Time Limitations

Another difficulty is that once the semester ends students' work on projects ends as well. Given the short semester and the ambition of some groups this has the undesirable consequence of incomplete work. Although this may be corrected with better advise regarding expectation and time allotments, for particularly promising work we may wish to offer an arrangement that would allow select groups to continue, polish and integrate their work into the SNBENCH code base for credit. It might also be nice to be able to give students detailed feedback about their work after the semester has ended and the code has been thoroughly reviewed. Even if this information is too late to benefit them for the current semester, such feedback may be useful for the rest of their academic careers and beyond.

Suggested Projects

We found that while providing an assortment of projects was beneficial to the groups, a careful balance must be struck to maintain requirements of a proper academic exercise. Projects that were selected by many groups yielded more questions and answers could be shared and would in turn benefit many students. This is quite obvious as many students are disinclined to ask for help, yet will benefit from answers to questions asked. Offering too many projects can lead to a situation where there may be projects that are selected by only one group, thus increasing the overhead and decreasing the utility of support and documentation efforts. Reducing the total number of projects guarentees there will be multiple groups working on the same project and with multiple submissions on the same project, the code base increases its likelihood of a strong result being adopted into the code base, either from either a single work or a composite of submissions.

References

[1] Azer Bestavros, Adam Bradley, Assaf Kfoury, and Michael Ocean, "SNBENCH: A Development and Run-Time Platform for Rapid Deployment of Sensor Network Applications," in *Proceedings of the IEEE International Workshop on Broadband Advanced Sensor Networks (Basenets 2005)*, Boston, MA, October 2005.

[2] Michael Ocean, Azer Bestavros, and Assaf Kfoury, "snBench: Programming and Virtualization Framework for Distributed Multitasking Sensor Networks," in *Proceedings of the 2nd international conference on Virtual execution environments (VEE 2006)*, New York, NY, USA, 2006, pp. 89 – 99, ACM Press.

[3] Michael J. Ocean and Assaf Kfoury, "snBench documentation for the Spring '06 offering of cs511," `http://cs-people.bu.edu/mocean/cs511`.

[4] Ching Chang, Raymond Sweha, and Panagiotis Papapetrou, "Extending snBench to Support a Graphical Programming Interface for a Sensor Network Tasking Language (STEP)," Tech. Rep. BUCS-TR-2006-014, CS Department, Boston University, July 14 2006.

[5] Dave Cecere, Ben Freiberg, and Dustin Burke, "Extending snBench to Support Distributed Storage Across Sensors," Tech. Rep. BUCS-TR-2006-015, CS Department, Boston University, September 2006.

[6] Gabriel Parmer, Georgios Zervas, and Angshuman Bagchi, "Extending snBench to Support Hierarchical and Configurable Scheduling," Tech. Rep. BUCS-TR-2006-012, CS Department, Boston University, July 14 2006.

[7] Sowmya Manjanatha, Jorge Londono, and Zhinan Han, "An Extension to the Sensorium Execution Environment (SXE) to Provide Concurrency Support in the snBench framework," Tech. Rep. BUCS-TR-2006-013, CS Department, Boston University, July 14 2006.