An Undergraduate Distributed Computing Course

Dr. Daniel C. Hyde
Department of Computer Science
Bucknell University
Lewisburg, PA 17837 USA
hyde@bucknell.edu

Abstract - *This paper proposes an undergraduate distributed computing course that focuses on the fundamental principles common to multimedia, client-server, parallel, web and collaborative computing. This computer science course should actively engage the students in exploring the concepts of distributed computing. Several extended projects using the language Java are described.*

Keywords: distributed computing, computer science undergraduate education, Java, Constructivism, projects

1 Introduction

Like many universities and colleges, the Computer Science Department at Bucknell University teaches an undergraduate parallel computing course. However, parallel computing is currently out of vogue. There are several reasons for this. First, the promises of parallel computing have not been fulfilled. Second, many major parallel computing vendors have gone bankrupt or merged with other companies. Third, the computing industry and researchers have awakened to the fact that less than 1% of the market is the traditional high performance scientific computing market – the main target for parallel computing courses. The action is in the commercial and multimedia markets. We have learned that the market place rules!

Undergraduate parallel computing courses are outdated and need to be re-evaluated. These courses no longer serve the needs of our students. Times have changed; the students really need a distributed computing course rather than a parallel computing course. Our computer science students need to understand new concepts and principles in order to design and program multimedia, client–server, web-based, and collaborative systems as well as parallel systems. A major activity our graduates do today is develop middleware, e. g., using distributed objects-based software, such as CORBA, to interface databases, centralized services and legacy software systems. Therefore, the author proposes to replace the undergraduate parallel computing course with a new "Distributed Computing" course that retains many of the fundamental principles covered in the parallel computing course.

2 What is Distributed Computing?

"Distributed computing" is an ill-defined term. In the past, the term could mean (1) distributed operating systems [3]; (2) distributed data processing as used in many data processing departments to replace their old mainframes with client-server systems; or even, (3) parallel computing on distributed memory parallel machines. We propose a much broader definition of the phrase. *Distributed computing* means designing and implementing programs that run on two or more interconnected computer systems.

For this level of course, we are assuming students will design and implement distributive programs for a fully functioning internet with its associated networking hardware and protocol software. The intent is to use a network; not study or design networks. The network community would call these "network applications."[2]

In this new proposed undergraduate course, distributed computing includes multimedia systems; client-server systems; parallel computing; web programming; and collaborative systems, i. e., mobile agents. What are the key characteristics of each of these distributed computing systems? What common fundamental principles do they possess that will serve our students' needs for the next five or ten years?

2.1 Multimedia Systems

Many of our computer science students will be involved in the design and construction of multimedia systems as stand-alone systems or on the Web. To be successful, they need to understand such topics as creation of graphics and animations; generation of sound; concurrency; event handling; real time demands of sound and video; importance of standards; abstraction of data representations as in MPEG4; data compression; networking; performance measurements; and software engineering techniques and tools.

2.2 Client-Server Systems

The main characteristics of client-server systems that our students need to understand are networking; designing communication protocols; concurrency; graphical user interfaces (GUIs); fault tolerance; exception handling; abstractions, such as distributed objects used in middleware, e. g., CORBA; security; interacting with databases, centralized services (e. g., name server) and legacy software systems; performance measurements; dealing with low bandwidth situations; and software engineering techniques and tools.

2.3 Parallel Computing

Parallelism is subtly different from concurrency. Parallelism is the performing of actions at the same time, e. g., a program running on multiple CPUs. For some researchers, concurrency is the *illusion* of a program running in parallel. For example, several Java threads may appear to run at the same time, but in reality the threads are interleaved in time on one CPU. The Java Virtual Machine (JVM) performs instructions for one thread, then switches to perform instructions for another thread. I call this *virtual parallelism*. I reserve the word *concurrency* to mean capable of being performed in parallel in the abstract. An underlying implementation for this abstraction may be either *physical parallelism*, e. g., running on multiple CPUs, or *virtual parallelism*, e. g., time slicing on one CPU.

The field of parallel computation's primary focus is in the use of parallelism to achieve higher performance, i. e., a program that runs faster or a program that handles a larger problem in the same amount of time. For parallel programming, students need to understand message passing; bandwidth; network latency; load balancing; task scheduling; concurrency; problem partitioning – both domain and functional [8]; communication structures; algorithmic synchronization due to data dependencies; performance measurements; and parallel algorithms.

2.4 Web Programming

Web programming is the design and construction of a program, e. g., an applet, to perform a task on a web page. For this activity, students need to understand GUIs; concurrency; event handling; graphics; network communication; and software engineering techniques and tools.

2.5 Collaborative Systems

Collaborative systems or mobile agents are autonomous programs that may move around a network as well as collaborate with other agents to perform a task. To design mobile agents, students need to understand concurrency; network communication; security; importance of standards; the power of abstraction, e. g., mobile agents; fault tolerance/reliability; serialization of objects which allows a system to send executable code, program state and other data to another system which can reconstruct the agent; and persistent objects.

3 Common Fundamental Principles

Reading the above descriptions of these five types of distributed computing systems, one immediately recognizes commonalities. Objects and their associated concepts such as distributed objects, serialization, and persistence, are fundamental principles. Concurrency and control of threads and processes are fundamental. "Concurrency control is at the heart of system design."[1] Network communication and designing communication protocols are fundamental. GUIs and concern for user interactions are fundamental. Software techniques and tools are common. But most important is the ability for the student to create new abstractions as well as understand established ones. These common fundamental areas form the basis for a new course in distributive computing.

Clearly, not all the material in the five types of distributed computing deemed "important" by faculty members will fit into one course. Also, most universities, especially the smaller ones, cannot afford to teach an undergraduate course for each one. Therefore, the real trick is how to package the material into one course which not only covers a significant number of the fundamental principles, but also can be fun and educational for the students.

4 The Distributed Computing Course

Several years ago such a course would not have been possible. However, today the objected–oriented programming language Java makes the course possible. Java has many features suitable for teaching distributed computing, including built-in threads and its extensive Application Programming Interface (API) with support for GUIs; networking (sockets and streams); remote method invocation (Java RMI); security model; serialization; and interfaces to CORBA and databases using SQL. Java is a golden opportunity to improve the undergraduate curriculum in the area of distributed computing.

We propose that the course be taught as a new elective course targeted primarily to junior and senior computer science majors. We assume the typical student has completed the CS1 and CS2 sequence using Java or C++, covering topics in object-oriented programming, data structures, and

[1] Jean Bacon, "Defining our scope: the Computer Society's 'systems' magazine," *IEEE Concurrency*, Vol. 7, No. 1, Jan-Mar 1999, page 2.

software engineering. Knowledge of operating systems would be useful but is not required. Variations of the distributed computing course could be taught with or without the prerequisite of an operating system course, depending on the local circumstances.

If the students do not know Java but have extensive C++ experience, I suggest the Java textbook by Deitel and Deitel [4]. This text covers the basics of Java and contains good introductory chapters with working Java programs on GUIs, exception handling, multithreading, multimedia, and networking (sockets and streams). For more experienced students with introductory knowledge of Java, I recommend the pair of books by David Flanagan [6, 7].[2] His second book, *Java Examples in a Nutshell*, contains many valuable examples on GUIs, networking, multithreading, Remote Method Invocation (RMI), database access, and security.

The second text will depend on the emphasis of the course. For an emphasis on client-server systems and distributed objects, I suggest the book *Java Distributed Computing* by Jim Farley [5], which provides an excellent coverage on distributed objects (Java RMI and CORBA), threads and security. It also contains a small chapter on collaborative systems. To provide an in-depth coverage on concurrency, I suggest the text *Concurrent Programming: The Java Programming Language* by Stephen Hartley [10]. Hartley's text has a chapter on parallel computing. If the emphasis is on distributed systems, I highly recommend [3], which covers networking and traditional topics in distributed operating systems, such as remote procedure calls and logical clocks. The instructor will want to supplement the texts with articles. For a good article on Java-based collaborative computing see [16].

For a 13-week semester, I suggest the following break down of topics:

(1) one week on introduction to distributed computing

(2) two weeks on GUIs, event handling, exceptions, manipulating images, and animations

(3) two weeks on client-server systems, including networking with sockets and streams

(4) three weeks on concurrency, including multithreading

(5) three weeks on parallel computing, including domain and functional partitioning, message passing and performance measurements

(6) two weeks on collaborative systems, i. e., mobile agents, including security and reliability models.

The author is a student of the constructivist theory of learning [9, 1]. From the constructivist point of view, knowledge is constructed, not transmitted. Learning is an active process in which learners construct new concepts based on their prior knowledge. Therefore, I suggest the course be driven by a series of extended projects which actively engage the students in exploring the concepts. With this approach the lectures and reading assignments provide the background and the conceptual frameworks for the projects.

[2] His two books together cost less than many Java texts.

The weekly laboratory exercises[3] provide practice on individual concepts and introduce needed skills. While the laboratory exercises may support the projects, the projects are envisioned to be performed by the students outside of class.

4.1 Racetrack Project

The first project is proposed for the first six weeks of the course and consists of multiple assignments to create an interactive racetrack program.

Phase 1: Using a tool such as `xpaint`, the student creates a GIF image, 50 by 50 pixels, of a racecar viewed from the top. The student then creates 16 versions of the image rotated around its center. The student writes a Java applet to create a simple animation, say to spin the racecar. [Concepts – beginning Java, generating images, simple animation]

Phase 2: In the second part, the student creates a Java application for a racetrack in which he or she can change the speed and direction of the racecar by pressing the arrow keys. The project switches to Java applications at this point since many web browsers do not support the Java 1.1 event model of listeners. [Concepts – Java application, event handling, model-view-controller (MVC) design pattern [15]]

Phase 3: The third part is to allow two racecars to run on the track using two sets of four keys to control the speed and direction of the cars. Students add sounds. [Concepts – limit detection for collisions and driving off track, generation of sounds]

Phase 4: The fourth part is to create a server and two clients which will allow the two cars to be controlled and viewed on two workstations. [Concepts – client-server, sockets and streams, design of message protocol, threads and concurrency, protecting shared variables]

Phase 5: The fifth part is to improve part four by packing (marshalling arguments) and unpacking messages to speed up the performance and to use double buffering to reduce flicker in the screens. [Concepts – double buffering, painting model in Java, marshalling arguments]

Phase 6: The sixth part is to use threads to allow the server to handle many racetracks with pairs of racecars. [Concepts – threads and server robustness]

The project has been carefully selected to be fun, leave lots of room for creativity and not be too gender specific -- no shoot 'em up and blast them away interactions -- as well as to cover a wide variety of concepts in GUIs, multimedia, animations, client-servers and threads.

4.2 Stock Market Game

We suggest the second project span three weeks. The student creates a fast-paced on-line interactive game of stock market, which has many of the characteristics of electronic commerce. The student designs a server (the stock exchange) to update the prices of stocks *once a minute* depending on interactions with a collection of clients (both human and computer) on different workstations who are buying and selling from the stock exchange. The prices of the stocks change dynamically based on the law of supply and demand. The server design must have several threads to handle the

[3] At Bucknell University, we organize our programming-oriented courses with three lectures and a two-hour structured laboratory a week.

clients and the small database. Random good and bad events (e. g., good quarter sales at a company boost the price of its stock) are introduced to make the game more interesting.

Phase 1: In the first part, the students submit a design of their game. This includes a design of the human client's screen including the human interactions. An important aspect of the game is effective presentation of the stocks and effective interaction mechanisms to allow the player to make quick and meaningful decisions. [Concepts – GUI design, client-server design, human interface design]

Phase 2: The second part of the project is to implement a simple server (Java application) and one human client (Java applet suitable for web browser). [Concepts – client-server, communication message design, multithreads, simple database, concurrent reads and writes, persistent objects]

Phase 3: In the third part, the students implement the full server which allows many clients to register and unregister for service during the course of the game. The server sends the stock updates to all registered clients every minute. The students design and implement a computer client which analyzes the current stocks and buys and sells to make a gain. A parameter is passed to each computer client on start up to characterize its behavior in the range from very bearish to very bullish. The computer clients provide variety and fluctuations in the market and, therefore, more interest to the game. [Concepts – threads, server design and server robustness]

The stock market game provides opportunities to introduce topics in class which are not specified in the project. For example, the class could discuss security, reliability and scalability issues.

4.3 Matrix Multiply

The third project spans three weeks and explores parallel computing.

Phase 1: The student uses the Message Passing Interface (MPI) [14] standard and Java[4] to write parallel computing programs to perform a matrix multiply of matrices using several different domain partitions. [Concepts – parallelism, message passing, processes, communication structures, domain partitioning, load balancing]

Phase 2: In the second part, the student measures timings of the parallel programs to construct a simple performance model of the time to communicate between two workstations (network latency) and the computational time to compute an inner product. [Concepts – performance measurements, network latency, speedup, Amdahl's law]

5 Conclusions

The author has described an undergraduate computer science course in distributed computing that covers the fundamental principles common to multimedia, client-server, web, parallel and collaborative computing. We suggest three extended projects written in Java that actively engage the student in exploring the concepts of distributed computing.

[4] The instructor may wish to use C or C++ with MPI to explore parallel computing.

6 References

[1] Ben-Ari, Mordechai. “Constructivism in Computer Science Education,” *SIGCSE Bulletin*, Vol. 30, No. 1, Mar 1998, pages 257-261.

[2] Comer, Douglas E. *Computer Networks and Internets*, Prentice Hall, second edition, 1999.

[3] Coulouris, George, Jean Dollimore and Tim Kindberg. *Distributed Systems: Concepts and Design*, Addison-Wesley, second edition, 1994.

[4] Deitel, H. D. and P. J. Deitel. *Java: How to Program*, Prentice Hall, second edition, 1998.

[5] Farley, Jim. *Java: Distributed Computing*, O'Reilly and Associates, 1998.

[6] Flanagan, David. *Java in a Nutshell*, O'Reilly and Associates, second edition, 1997.

[7] Flanagan, David. *Java Examples in a Nutshell*, O'Reilly and Associates, 1997.

[8] Foster, Ian. *Designing and Building Parallel Programs*, Addison-Wesley, 1995.

[9] Hadjerrouit, Said. “A Constructivist Framework for Integrating the Java Paradigm into the Undergraduate Curriculum,” *SIGSCE Bulletin*, Vol. 30, No. 3, Sept 1998, pages 105-107.

[10] Hartley, Stephen J. *Concurrent Programming: The Java Programming Language*, Oxford University Press, 1998.

[11] Holub, Allen. “Programming Threads”, series of articles in on-line magazine *JavaWorld*, http://www.javaworld.com

[12] Lea, Doug. *Concurrent Programming in Java: Design Principles and Patterns*, Addison-Wesley, 1997. New edition due Dec. 1998.

[13] Oaks, Scott and Henry Wong. *Java Threads*, O'Reilly and Associates, 1997.

[14] Pacheco, Peter S. *Parallel Programming with MPI*, Morgan Kaufmann, 1997.

[15] Rumbaugh, James. "Modeling Models and Viewing Views - A Look at the Model-View-Controller Framework", *Journal of Object-Oriented Programming*, Vol. 7, No. 2, May 1994, pages 14.

[16] Wong, David, Noemi Paciorek and Dana Moore. “Java-based Mobile Agents,” *CACM*, Vol. 42, No. 3, March 1999, pages 92-102.