From RS-232 to Object Request Brokers: Incremental Object-Oriented Networking Projects

David Janzen
Mathematical Sciences Department
Bethel College
North Newton, KS 67117
djanzen@bethelks.edu

Abstract

Selecting an appropriate set of laboratory experiences and projects for a Data Communications and Computer Networks course can be difficult due to the broad and deep nature of the topics. Emphasis may be placed on many networking aspects including design, evaluation, efficiency, security, protocols, tools, and applications. This paper presents a set of projects that attempt to integrate software engineering and systems administration topics. The projects emphasize network application programming. Particular attention will be given to a sequence of incremental projects using an object-oriented approach including the use of the Unified Modeling Language (UML) and a design pattern.

1 Context

This set of projects was assigned in the context of a one-semester undergraduate course in Spring 2000. This is the only networking course offered in the current curriculum so it is intended to be a survey of the field of networking. Lectures followed the required text [3] with supplemental material focusing on telecommunications, internet services, and object-oriented middleware based on personal expertise of the instructor and two guest lecturers. Students entering the course had completed three programming courses (CS1 in Pascal or C++, CS2 in C++, Data Structures in C++) and one computer systems course. The host institution is a small liberal arts college that participates in a cooperative of small colleges for most upper-level Computer Science courses. As a result, students came from four different campuses to a central location for course meetings one evening a week. This allowed little flexibility in creating and scheduling laboratory experiences, however it did prompt intensive use of networking applications. Course materials were distributed via the web, office hours were held in person and via email, and most projects were completed on remote computers.

2 All Projects

Project 1: Write a computer program to allow two users on a single computer to play Tic-Tac-Toe against each other.

Project 2: Extend the Tic-Tac-Toe program from Project 1 so that one of the players can be on a different computer and the two computers are connected by a serial cable.

Project 3: Write a computer program to extract frame information from packet traces. [5]

Project 4: Document a campus network using a graphical layout and a textual description of the campus network. The documents should show such information as the physical placement of routers, bridges, switches and hubs as well as identify which servers are responsible for DNS, web, email, and DHCP, what IP addresses are available, networking technologies used, types of cabling and use of firewalls.

Project 5: Write a computer program to look up hostnames and IP addresses using DNS. [6]

Project 6: Rewrite Project 2 using sockets allowing the game to be played between two players across an internet.

Project 7: Rewrite Project 6 using fork to create a game server that matches pairs of players.

Project 8: Rewrite Project 7 using an Object Request Broker. (This project was simplified as described in the Outcomes section of this paper).

Project 9: Write a simple Java client application that uses sockets to connect to an existing server written in C++ that returns the number of times it has been contacted, and separately write a Java applet of the student's choosing.

3 Incremental Object-Oriented Projects

Within the set of projects above, Projects 1, 2, 6, 7, and 8 comprise a sub-sequence that require students to create a relatively simple program in ***Project 1*** and modify it repeatedly to gain additional networking functionality. The design of the original program can be captured in the UML diagrams[1] in Figures 1 and 2. Figure 1 is a class diagram showing the primary classes Game, GameBoard and Player.

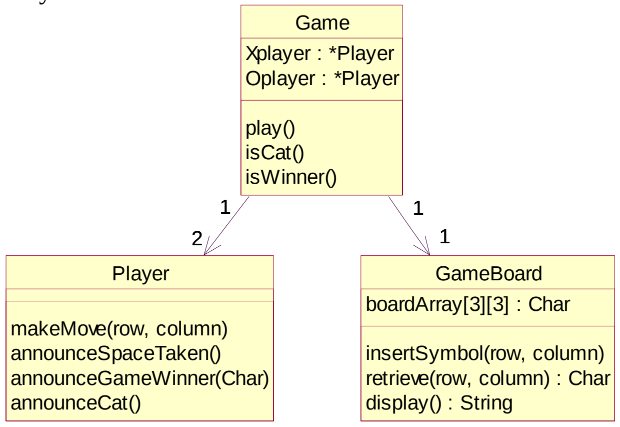

Figure 1: Class Diagram for Tic-Tac-Toe

Figure 2 is a Sequence Diagram showing the interactions between the four instances of the three classes in Figure 1.

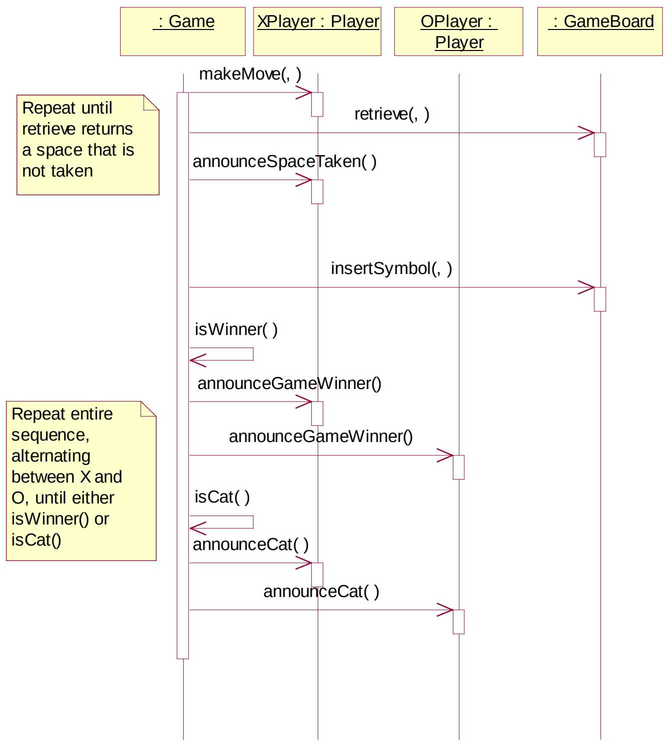

Figure 2: Sequence Diagram for TicTacToe

UML Diagrams were used to emphasize the object-oriented aspect of the projects, as well as to provide exposure to the UML. Although most students had no previous experience with UML, students had little difficulty understanding the design. The familiar and simple nature of the game of Tic-Tac-Toe and previous programming experience with classes likely contributed to students' quick understanding. Prior to presenting this object-oriented design, a structured design was developed in class by the students. A hierarchy chart and pseudocode were presented which captured this design. The structured design was then evaluated for extensibility. Additional requirements of making the players remote and creating a game server, as well as making the user interface graphical instead of character-based were considered. Frustrations with extending the structured design provided the needed motivation and appreciation for the object-oriented design.

Project 2 was introduced in the context of a lab on data communications. Students completed an exercise in which they connected two microcomputers running Microsoft Windows using serial cables and null modems, then tested and modified example client and server programs which had differing parameters on the communication ports so the data sent did not match the data received [11]. This exercise familiarized them with CreateFile, WriteFile and ReadFile plus the CommState and CommTimeout structures using the application program interface (API) defined in windows.h. Students were then introduced to the Remote Proxy pattern [9,1,4] and its application to the Tic-Tac-Toe program as in Figure 3.

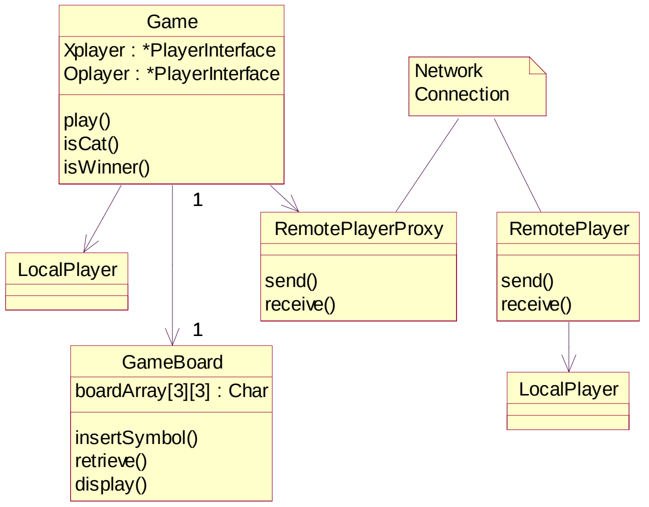

Figure 3: Adding the Proxy pattern to Tic-Tac-Toe

The proxy pattern is used to minimize code changes in the Game and Player classes. In this design, LocalPlayer is a new name for the Player class in Figure 1. The Game, GameBoard, one LocalPlayer instance, and the RemotePlayerProxy instance are all located on one computer. The RemotePlayer and another LocalPlayer instance are located on the other computer. The RemotePlayerProxy is a representative of the LocalPlayer on the remote computer. It shares the same interface as the LocalPlayer so the only change necessary in the Game class is where the PlayerInterface is instantiated. This

[1] Use of Rational Rose for UML modeling with permission of Rational Software Corporation through the Software Engineering for Educational Development program.

point is made clear in Figure 4 where the Player class has been refactored [8] so that LocalPlayer, RemotePlayer-Proxy, and RemotePlayer are all realizations of the new PlayerInterface. Notice that RemotePlayerProxy and RemotePlayer add the operations send and receive which enable the remote communication. Not shown in the UML diagram is the need for code in the constructors of RemotePlayerProxy and RemotePlayer that establishes the connection across the serial communications channel (i.e. calls CreateFile). At the instructor's suggestion, most students transmitted one of four characters to represent the four operations defined in the PlayerInterface. Based on the character, the RemotePlayer knew whether to look for additional parameters such as the character of the winner (X or O) when announcing the game winner. All communications were terminated with a special character such as '$' or '\0'. Students were given the hint to make the RemotePlayer loop until the special terminating character is received.

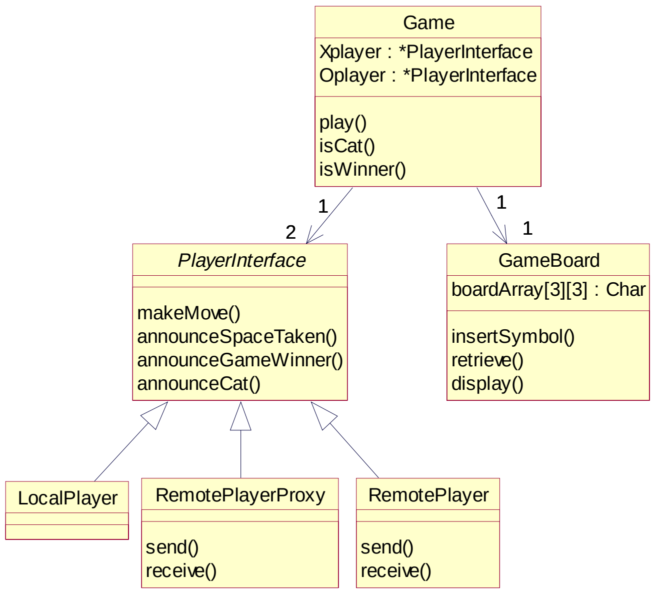

Figure 4: Create a PlayerInterface

Project 6 then had the students retain the design of Figures 3 and 4, but replace the calls to the windows.h API for serial communications with the socket API allowing the program to communicate across an internet. This project was required to run on a Linux server and used TCP rather than UDP. In ***Project 7*** students separated the Game and GameBoard instances out into a game server program that acted like a daemon listening for player clients to contact it wanting to play Tic-Tac-Toe. When two clients were received, the game server would fork a slave process to manage the game between the two players, reporting back an answer on the winner or if the game ended in a cat (tie). With this design, the clients had to know the location of the server, but neither client cared whether they were the X or O player. In other words, they were the same executable unlike in the previous two projects.

Finally in ***Project 8***, students were to rewrite Project 7 using an Object Request Broker (ORB). This approach would remove the constraint that the clients had to know the location of the server. Plus it would remove the complexities and explicit choice of using the socket API. The CORBA Naming Service [12] was employed to allow clients to look up the Game object by name rather than by IP address or DNS name. The ACE ORB (TAO) was used for this project. TAO is a free, yet mature CORBA-compliant ORB developed at the Center for Distributed Object Computing at Washington University in St. Louis, MO under the direction of Douglas Schmidt originally and now David Levine. Precompiled binaries for The ACE ORB (TAO) are available at [13].

4 Outcomes

One of the major challenges with incremental projects is that some students may fall behind on a project. If the next project builds on the previous one, then catching up is extremely difficult. In this course offering, non-incremental projects were interleaved to allow some time for students to complete a project in the incremental sequence late. Some students were still not able to complete the incremental projects even with the extra time. A working version of Tic-Tac-Toe with serial communications (Project 2) was provided when the Tic-Tac-Toe with sockets project (Project 6) was assigned. Most students chose to continue with their original source code on the subsequent project. Those students who had not completed the earlier project were able to start with a working version on the new project. This greatly improved their success in the course and their satisfaction according to the course evaluations.

Two-thirds of the students were successful in completing all of the projects. One student completed all of the projects except project 2 and 5, and the remaining two students completed none of the programming projects. External factors probably contributed to the poor performance of the last two students. Student evaluations were very positive. The two (expected) primary criticisms concerned the emphasis on programming for those planning careers in systems administration and the amount of effort and time that the projects required.

Another challenge was the inclusion of the ORB project at the end of the class. TAO was installed on a Linux server after a fairly steep learning curve and with the help of an excellent undergraduate student systems administrator. However, due to some complications with installing and using the ORB and the lack of time left at the end of the semester, Project 8 was simplified to a modification of the TimeOfDay sample [10] delivered with TAO. Students were asked to create a client-server application using an ORB where the client provides an ISBN of some publication and the server replies with additional information on the publication (author, title, number of pages). This project still seemed sufficient to provide students at least an appreciation of the significance of

Object Request Brokers and an added respect for and interest in Dr. David Levine when he presented several guest lectures near the end of the semester. The Naming Service was employed and students gained familiarity with defining data structures through the Interface Definition Language (IDL).

Students worked individually on all projects except Project 2, where they worked in pairs in a laboratory environment, and Project 4, where students from each campus worked as a team to document their local campus network. Based on experiences in other courses and the attention to eXtreme Programming [2], pairs would have been used more extensively if the constraints of multiple campuses were removed.

5 Conclusion

The set of incremental programming projects accomplished the goals of applying networking technologies such as client-server programming, serial communications, sockets, and object request brokers, while also addressing software engineering issues such as design reuse, object-oriented design, design patterns, and refactoring. Although C++ was the appropriate language for this course, the projects could have easily been completed in other languages such as Java, or a combination of languages could have been used. For instance, it may be interesting to use different languages for the two players in project 6, or to write the game server in project 7 in a language different from that used for the players. A mixture of languages might help emphasize some of the benefits of using an ORB as well.

As with any choice, there is an opportunity cost associated with this sequence of projects. For instance this class did not have time to complete projects on such topics as encryption or protocol design. The distributed nature of this course inhibited additional laboratory experiences that might have been useful as well. In particular, additional laboratories on network configuration and management [7] would have been useful. Interestingly, despite the geographic separation between most students and the instructor and the extensive use of email for office hours and project assistance, some students still drove to the instructor's campus for face-to-face assistance.

Including Dr. David Levine, with expertise in network programming and Object Request Brokers, as a guest speaker near the end of the course was a very positive motivator both for the instructor and the students. This technique of inviting guest lecturers to complement an upper-level course and tailoring projects to the speaker's expertise has been very effective. A previous experience included design reviews coupled with lectures on object-oriented design from Dr. Ralph Johnson in a software engineering course with a significant semester-long team project.

References

[1] Addison-Wesley. *NEXTSTEP General Reference: Release 3, Volumes 1 and 2*, 1994.

[2] Kent Beck. *Extreme Programming Explained.* Addison-Wesley, 2000, 100-102.

[3] Douglas E. Comer. *Computer Networks and Internets Second Edition.* Prentice Hall, 1999.

[4] James O. Coplien. *Advanced C++ Programming Styles and Idioms.* Addison-Wesley, 1992.

[5] Ralph Droms. CSCI363 - Computer Networks. Online. Internet. August 7, 2000. Available WWW: http://www.eg.bucknell.edu/~cs363/laboratories/lab03_0.html.

[6] Ralph Droms. CSCI363 - Computer Networks. Online. Internet. July 13, 2000. Available WWW: http://www.eg.bucknell.edu/~cs363/laboratories/lab04.html

[7] Bruce S. Elenbogen. Computer Network Management: Theory and Practice. *The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*. (March 1999), 119-121.

[8] Martin Fowler. *UML Distilled, Second Edition.* Addison-Wesley, (2000), 30-31.

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995, 207-217.

[10] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, 1999, 38-47.

[11] Brad Richards. Bugs as Features: Teaching Network Protocols Through Debugging. *The Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education*. (March 2000), 256-259.

[12] Doug Schmidt and Steve Vinoski. Building a Stock Quoter with TAO – A Tutorial. Online. Internet. August 7, 2000. Available WWW: http://www.cs.wustl.edu/~schmidt/ACE_wrappers/TAO/docs/tutorials/Quoter/.

[13] Online. Internet. August 7, 2000. Available WWW: http://www.theaceorb.com/TAO/Support/download.html.