Computer Science Education
1998, Vol. 8, No. 1, pp. 64–81

0899-3408/98/0801–0064$12.00

A Multi-Purpose Simulation Project for Engaging Students and Teaching Object Concepts*

Hossein Saiedian
Department of Computer Science, University of Nebraska at Omaha

ABSTRACT

This article proposes that laboratory projects which require group activities can be beneficial to undergraduate students in computer science. Team projects expose students to activities that otherwise might only be encountered in the workplace after graduation. A group project can provide better preparation and motivation than laboratory assignments which isolate students. A desirable project is one that allows the concepts of a course to be integrated into it as they are introduced in the lectures. The description of a simple networking simulation is given which exemplifies a project that is easily managed by students. This project possesses the ability to be expanded in a variety of ways so that it provides a suitable starting point for a wide spectrum of computer science courses such as computer networks, programming languages, object-oriented programming, and object-oriented software engineering.

INTRODUCTION

University teaching is often assessed in comparison with the way in which industry works outside the university. As Denning (1992, p. 84) has noted, "Employers and business executives complain that graduates lack practical competence. Graduates, they say, cannot build useful systems, formulate or defend a proposal, write memos, draft a simple project budget, prepare an agenda for meeting, work on teams, or bounce back from adversity." While a university career is largely judged in terms of individual achievement and assessment, a computing career is largely judged by group cooperative activity. Since work in business organizations is a team-oriented activity, university students should be exposed to such activity in order to meet the needs and demands of industry. This can sometimes be achieved through internship, but internship opportunities

* This work was partially supported by a UCR grant, University Committee on Research, University of Nebraska at Omaha. Rick McBride's suggestions have improved the content of the article.

Correspondence: Hossein Saiedian, Department of Computer Science, University of Nebraska at Omaha, Omaha, Nebraska 68182-0500, U.S.A. E-mail: hossein@unomaha.edu.

are not available to all. Experience with a realistic group project in an academic environment can be a valuable substitute for an internship (Bruegge, 1992; Etlinger, 1990; Moore & Potts, 1994; Saiedian, 1992).

An early participation and exposure to team projects helps a student to understand that written and oral communication, interpersonal skills, and the ability to statistically analyze results are vital for being professionally competent in the real world. Introduction of a team project early in the curriculum can have the ancillary effect of sparking a student's interest in writing, speech or mathematics courses. A project-oriented course also highlights some of the practical issues that face software development teams in industry. A student on a team gains an overall knowledge about complex software systems (and thus practical knowledge of many of a course's principles) that would be difficult to obtain without the support of a group. Team projects are becoming an essential part of courses like software engineering, database management systems, computer networks, and systems analysis and design courses.

Classroom projects have a high positive impact on students and are becoming the preferred method of teaching the computer science courses mentioned above. In fact, the objective is not to merely prepare students for working in a business environment, but also to enhance teaching in the educational institutions (Tucker, 1991). Learning by doing is essential for a software engineering course (Moore & Potts, 1994), and a project-intensive approach gives an opportunity to students to apply the concepts they have been exposed to in the classroom and readings. The students gain experience from actually applying methods and techniques for analysis, design, implementation, and testing presented in the classroom and gain some practical experience in project management. Practical exposure to a subject gives better understanding of the concepts as learning comes through experience. For example, Saiedian (1992) illustrates that theoretical concepts are not sufficient in teaching a database course and that actual database development is required to encounter the idiosyncrasies of developing a database system.

Ideally, several of the principles in a course should be captured in a project. This follows the approach taken by the ACM/IEEE *Computing Curricula* report (1991) which often suggests laboratory assignments that can be used in conjunction with a *knowledge unit* (i.e., a collection of related subject matter such as the scheduling and dispatch strategies used in operating systems). The ACM/IEEE report does not specifically designate any of these suggested laboratory assignments as a group project. A team project can incorporate many more principles than a laboratory assignment designed for one person. Even

though a student implements just a portion of a group project, there is practical exposure to other students' modules which embody different course concepts.

Other desirable properties that the project should possess include the following:

- An ability to be partitioned into well-defined modules
- The capacity to add new functionality or higher level layers of software that rely on the original project's software
- The ability to transform the original modules of the project so that a variation on the project can be implemented

PROJECT DESCRIPTION

In this section, we describe a project that is given to the students in an upper-level computer network class. However, as shown in the section called "Multi-Purpose Applications", this project can be used for a variety of purposes. Although a minimal description is given here, instructors can easily expand the project description throughout the semester in order to include additional features in the simulation or make it more challenging for the students.

The project description asks the students to use a high-level language such as Ada, C, C++, Java, or Pascal to simulate a local area network using a non-persistent CSMA (Carrier-Sense Multiple Access) local area network protocol. A programming language that supports concurrency is most effective for this project. Examples of such a language include Ada and concurrent versions of C and C++. A non-concurrent language such as C or Pascal can still be used to simulate a non-concurrent version of the project.

In a CSMA environment, several user stations are connected to a communication channel (known as the carrier), as shown in Figure 1. When a station wishes to transmit some data *packet* to another station, it first listens to the carrier to determine if another transmission is taking place. If the carrier is busy (i.e., some transmission is in progress), the station backs off and tries again later using one of the many algorithms available. If the carrier is available (i.e., no transmission is in progress), the station may transmit its data packet. Collision may still occur if two or more stations are simultaneously listening to the carrier and simultaneously sense an idle carrier. There are a number of algorithms for handling collisions. The simplest one is to allow transmissions to take place. Because transmitted data will be corrupted, the receiving station(s) will not acknowledge the complete arrival of a packet and therefore the

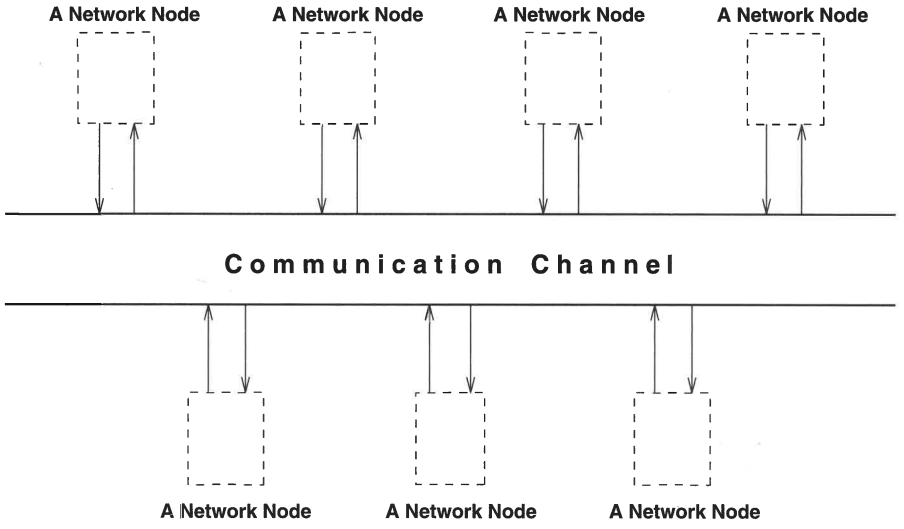

Fig. 1. A CSMA environment.

sender will have to transmit the data packets all over again. For a complete description of the CSMA protocols and algorithms, an interested reader may refer to Tanenbaum (1996, chapter 3).

Students are asked to simulate and model a CSMA network over which data packets with the following fields are transmitted:

`Dest Addr` 2 Bytes	`Src Addr` 2 Bytes	`Data Field` 1–15 Bytes	`CheckSum` 1 Byte	`Postamble` 1 Byte

where

- `Dest Addr` is the packet's destination address,
- `Src Addr` is the packet's source address,
- `Data Field` represents the packet's data,
- `CheckSum` is the packet's checksum, and
- `Postamble` represents the last character in a packet that is not used in the `CheckSum` calculation (this combination of bits is not permitted within the packet either)

There are several categories of processes which should be modeled in the simulation. These processes include:

- The `Carrier` Process. This process is responsible for carrying the data packets to all stations.
- `Transmitter` Processes. Each such process is responsible for obtaining data from a user application, assembling a packet, and transmitting it over the network.
- `Receiver` Processes. These are responsible for obtaining each packet from the carrier, determining if it belongs to its user application, and, if so, disassembling it and sending its data part to the application process.
- User `Application` Processes. These are the application processes run by the user.

A minimal network topology that illustrates the relationship among these processes is shown in Figure 2. A more detailed description (but still simplified for project purposes) of each of these processes is given below.

The `Carrier` Process

The `Carrier` process models the communication media. A `Transmitter` process should be able to *sense* the `Carrier`'s status (idle or busy). The status of the `Carrier` is changed from idle to busy when the first character in a

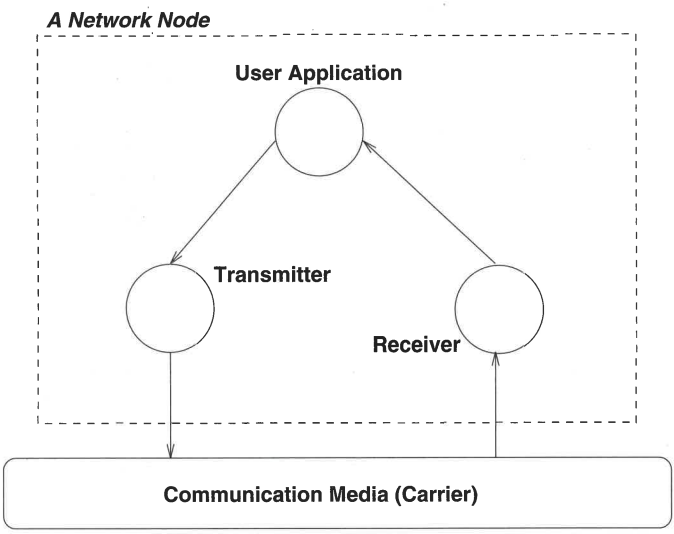

Fig. 2. A minimal simulation topology.

packet is sent over it. Whenever a character is given to the `Carrier` process, it will deliver it to all of the `Receiver` processes that are connected to it. Whenever a `Postamble` character has been delivered to the `Receiver` processes, the `Carrier` process resets its state to idle. Note that a packet represents a data structure (or package) used by the `Transmitter` and `Receiver` processes; the `Carrier` will be delivering individual characters.

The `Transmitter` Process

The `Transmitter` process has data delivered to it by an application process via a transaction in the form:

```
SendPacket(SrcAddr, DestAddr, data)
```

The `Transmitter` process proceeds to assemble a data packet which contains the above items as well as the `CheckSum` and `Postamble` and then uses a non-persistent CSMA algorithm to access the `Carrier`. Each byte of the data packet is delivered to the `Carrier` process via a separate transaction call.

For illustrative purposes, a `Transmitter` process should print out all of the information which is supplied to it whenever `SendPacket` is invoked; it should also print out each complete data packet which it transmits.

The `Receiver` Process

A `Receiver` process has individual characters delivered to it by the `Carrier` process. It is responsible for recognizing whether or not the packet is free of errors. If the destination address, in a correct packet, indicates the local application process then the packet will be locally buffered and later sent to the local application process when it is requested. A correct packet can be overwritten by another correct packet which is received for a local destination if the local application process is slow in requesting a received message from its `Receiver` process.

A `Receiver` process should print out each message that is destined for the local application process or which is found to be in error. A packet is in error if it is indicated to be so through the `CheckSum`, or if it is less than the minimum size for a valid packet, or greater than the maximum size of a valid packet.

The `Application` Processes

The `Application` processes are responsible for sending and receiving packets over the `Carrier` through the `Transmitter` and `Receiver`

processes. Each `Application` process corresponds to a user node and is therefore associated with its own `Transmitter` and `Receiver` processes. The dashed box in Figure 2 therefore represents a user node. The transaction call that is made to the local `Transmitter` has the form:

```
SendPacket(SrcAddr, DestAddr, data)
```

and the call to the local `Receiver` has the form:

```
RecvPacket(SrcAddr, DestAddr, data)
```

If a packet is to be received, then an `Application` process should block itself until one can be delivered to it by the local `Receiver` process. Each packet which is correctly received by an `Application` process should cause an acknowledgement to be sent in return.

Whenever a packet is sent by an `Application` process, that process should next execute a call to the local `Receiver` in order to await an acknowledgement of the packet which was sent. While awaiting an acknowledgement packet, an `Application` process should be able to receive and acknowledge data packets destined for it. If an acknowledgement is not received within a preset time limit, the `Application` process uses a binary exponential backoff procedure to calculate a random time to wait before retransmitting the packet. The transmission of a packet is abandoned after four unsuccessful attempts and, in this case, the `Application` process should print out the information regarding its failure and then proceed with its next operation. For simplicity, it is assumed that any time which elapses while waiting for an acknowledgement is canceled by a data packet which the `Application` process receives. After a data packet has been received (when an acknowledgement was expected), the wait for an acknowledgement starts anew.

Each `Application` process should output trace information which identifies:

- the identity of the `Application` process
- the type of operation being requested (`SendPacket`, `RecvPacket`, `timeout`)
- the status of the operation (starting or completed)

Recalcitrant and Error Checking Processes

Students should generate at least four `Application` processes which exercise `SendPacket` and `ReceivePacket`. (We require at least four user

nodes. A more challenging project would make the number of nodes variable—or an input parameter to the project.) One of these processes should be *recalcitrant*. A recalcitrant process:

- ignores (i.e., receives, but does not acknowledge) *all* attempts by one of the `Application` processes to send it packets
- ignores every first attempt made by the other `Application` process to send it a new packet

In addition to the four classes of processes which are needed to support communication, students should have one instance of a process which will allow error testing. This process combines the functions of an `Application` with those of a `Transmitter` and it should send four erroneous packets:

1. A packet for a non-existent destination address
2. A packet for a destination that exists (this packet to have a bad checksum)
3. A packet for a destination that exists (this packet to be smaller than the smallest legitimate packet)
4. A packet for a destination that exists (this packet to be larger than the largest legitimate packet size)

Again, appropriate information should be printed out whenever any of these packets is sent to the `Carrier`.

These process classes permit an arbitrary network topology to be configured and tested. The `Application` process only simulates the functions of the Medium Access Control Layer (IEEE, 1985). An additional protocol layer which performs higher level services (e.g., data flow control) can be added as another process that relies on the `Application` process. This expansion of the simulation requires two service access points to be added to the `Application` process which permit it to be called and to return information to the process above it. Additional protocol layers can be added in an incremental manner to the simulation as they are needed to illustrate the concepts of a course.

MULTI-PURPOSE APPLICATIONS

While the main use of the project described here is for a computer communication network course, it can be successfully used in a number of other courses, such as those listed below.

Object-Oriented Programming

As shown by Saiedian and Wileman (1993), each component of the network shown in Figure 2 can be modeled as an object. In fact, a very effective and interesting application would be to model each process as an object. An even more interesting approach would be to have only one base *class* representing a process with a number of *virtual methods*. The base class would declare and define features shared by all of the processes. Individual *subclass* objects can be derived from the base class that define their own implementation of the virtual methods to model the behavior of objects such as the `Transmitter`, `Receiver`, `Carrier`, etc. A similar approach is taken by Saiedian and Wileman (1993), where each process is treated as an object. Essential code (in Ada) is also provided. Class relationships can be modeled by a notation such as the Unified Modeling Language (UML).

Ada Multitasking

The simulation project can best be carried out in a language that supports objects and concurrency (since real network components must be able to run independently and concurrently). Because one of the most powerful features of Ada includes support for concurrency and objects, this project can serve as an effective exercise for the students of an (advanced) Ada course in which concurrency and objects are emphasized.

Programming Languages

The project can be successively implemented in different high-level languages. This demonstrates the different features of languages in a context that is familiar to the students; namely, the simulation project. For example, students can be asked to do the project in:

- C and then C++ to realize the abstraction concepts available in C++
- C and Concurrent C to evaluate the differences in the program's behavior in sequential versus concurrent runs
- Ada, Java, and C++ to note and compare the readability, flexibility, and the significance of standard libraries in these three competing languages

Simulation

This project essentially consists of simulating the behavior of a non-persistent CSMA and thus it can be used for a computer simulation course or an advanced programming course dealing with simulation and optimization.

More interesting features and simulation factors can be added. For example, we do not currently ask students to model timing factors; however, timing factors are crucial in a more detailed simulation of a real network. Likewise, physical distance, propagation delays, etc., can be added to make the simulation project more challenging. The author has successfully used this project in a computer network course where students were required to develop concurrent simulations (in Ada and Concurrent C).

Software Engineering

Needless to say, this project, with some additional features (e.g., modeling packet collisions, transmission of jam signals, variable number of `Application` processes, more complicated packet structure, etc.), can serve as a one-semester software engineering project. The author has also successfully used this project in an object-oriented software engineering course where students were asked to use use-cases, UML, and C++ (or Java) to develop analysis, design, and implementation models. This approach is briefly evaluated below.

AN EVALUATION FRAMEWORK

The above project has been used by the author for two purposes: (1) as a course project for a communication network course where the primary objective was simulation and an experiment in protocol modeling, behavior and performance analysis, and (2) as a course project in an object-oriented software engineering course. The second use, which involved developing use-cases, CRC cards, class diagrams, interaction diagrams, state diagrams, and an object-oriented implementation, is briefly described below. Rational Rose's UML CASE tool (version 4.0) was used for modeling purposes and both C++ and Java were used for implementation. The objective of this section of the article is to provide an account of what a reasonably successful project should offer and to introduce a basis (or a set of "metrics") for evaluating students' projects. For each component of the project, an example is selected and shown. Grading teams and individual participants is discussed at the end.

Use-Cases

Ivar Jacobson's (1992) use-case approach was used for modeling and documenting the requirements. This approach allows capturing and recording the

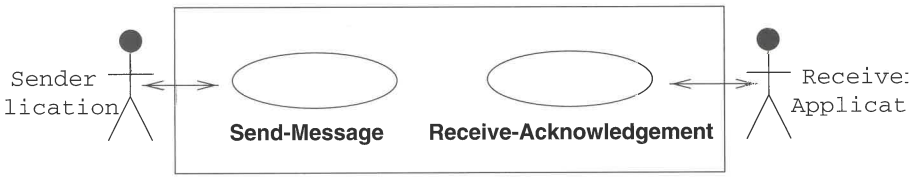

Fig. 3. Project use-cases.

requirements from the systems users' (or *actors'*) point of view. The use-case approach for this project included at least two primary actors: namely, `sender-application` and `receiver-application`. (Actually, `applications` represented the primary actors in the simulation, but they played two roles, the `sender` role and the `receiver` role.) Two primary use-cases were identified; namely, `send-message` (used by the `sender-application`) and `receive-acknowledgement` (used by the `receiver-application`). This is shown in Figure 3 using UML's use-case diagram. Each of the use-cases were further analyzed and "extended" to model optional, alternative, abnormal, and unexpected uses. On average, for each use-case, 4–5 alternatives (or "extensions") were identified. Thus, a reasonably successful team project should include around 10 use-cases. The "basic" course of action for the `send-message` use-case is shown in Figure 4.

CRC and Class Diagrams

On average, ten different classes were identified by the students. These included classes such as `Packets`, `Acknowledgement` (a sub-class of `Packet`), `Carrier`, `Receiver`, `Transmitter`, `Application`, `Recalcitrant` (a sub-class of `Application`), `Log`, `TopologyManager`, and so forth. Each of these classes were documented via CRC cards as well as UML's class diagrams. An example of a CRC card documentation (for the `Carrier` class) is shown in Figure 5 and a UML class diagram (for the `Transmitter` class) is shown in Figure 6.

Class Relationships

Using UML, a variety of relationships and associations between classes (such as "inheritance," "composition," "aggregation") were captured and documented. A sample UML diagram showing certain relationships is shown in Figure 7. The diagram was produced by Rational Rose UML CASE tool 4.0. The

Use-Case: `send-message`

1. The `Sender-Application` invokes `SendPacket` with parameters `SourceAddr`, `DestAddr`, `DataBlock`.
2. The `Sender-Application` invokes `ReceiveAcknowledgement`.
3. The `Transmitter` receives the packet.
4. The `DataBlock` is written to the log.
5. The `DataBlock` is parsed into one or more data block fragments that are less than or equal to the largest block size being handled by the network.
6. The message block is formatted into a packet with `DestAddr`, `SourceAddr`, `DataField`, `CheckSum`, and `Postamble`.
7. The `Carrier` status is checked to determine if it is busy or idle.
8. If the `Carrier` is idle, the packet is sent to it one character at a time until all characters are sent.
9. The packet is logged for trace reporting.
10. The `Carrier` sends the packet to all available nodes one character at a time.
11. The subsequent data block segments are sent until all data block fragments are sent.
12. The `Receiver` (within the sender's node) starts a counter to test for time-out threshold.
13. Once the acknowledgement has been received, the `Receiver` will pass it to the `Sender-Application`.

Fig. 4. Basic course of action in `send-message` use-case.

UML CASE tool was used by all teams to create many different kinds of diagrams for their projects, including use-case and class diagrams.

Interaction Diagrams

On average, eight UML sequence diagrams were employed to graphically depict interactions among the objects. Each of these diagrams corresponded to (or implemented) one of the use-cases. The UML's sequence diagram which shows a time-ordering of communication among objects participating in `receive-message` use-case (an extension of `send-message`) is shown in Figure 8. The diagram was produced via Rational Rose's UML CASE tool.

Class: Carrier SubClass: SuperClass:	
Responsibilities: GetCarrierSense SetCarrierState GetDataFromTransmitter SendDataToReceiver	Collaborators: Transmitter Receiver, TopologyManager

Fig. 5. A CRC representation of the Carrier class.

Transmitter
− packets []: Packet − appl: Application
+ receiveFromApplication () + createPacket () + senseCarrierStatus () + transmitChar () + Logtransaction ()

Fig. 6. A UML class representation of the Transmitter class.

State Diagrams

State diagrams are used to characterize the dynamic behavior of an individual object. The UML's state diagram for the Receiver object is shown in Figure 9. It was produced via Rational Rose's UML CASE tool. State diagrams were developed for each class of objects in the project.

Component Diagrams

Finally, the component diagrams were used by the students to show a configuration (or topology) of the network nodes. Each network node consists of an application unit which depends on a receiver and a transmitter unit. The executable application depends on the nodes that constitute the network. The component diagram (created via the Rational Rose CASE tool) is shown in Figure 10.

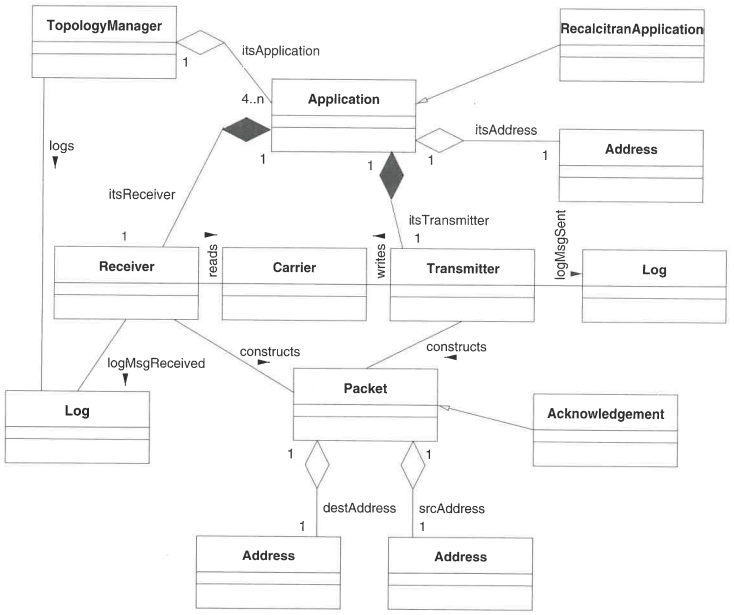

Fig. 7. An example of UML class relationships.

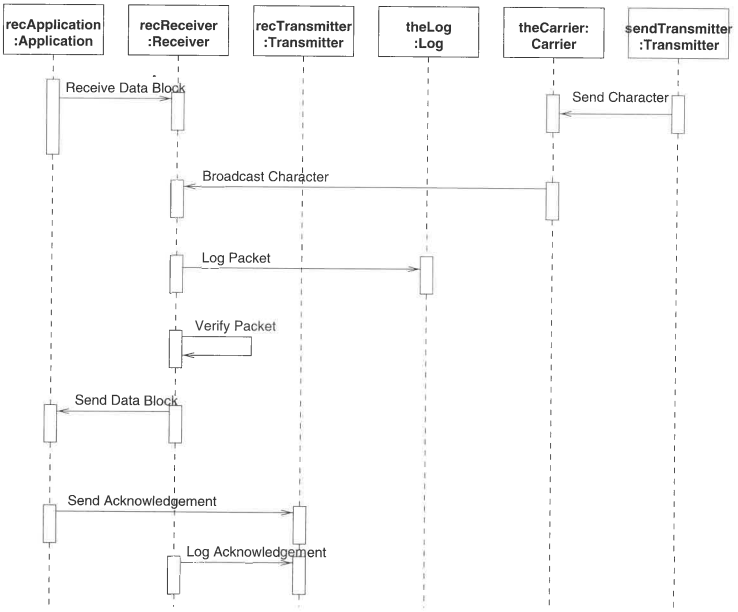

Fig. 8. UML sequence diagram for `receive-message` use-case.

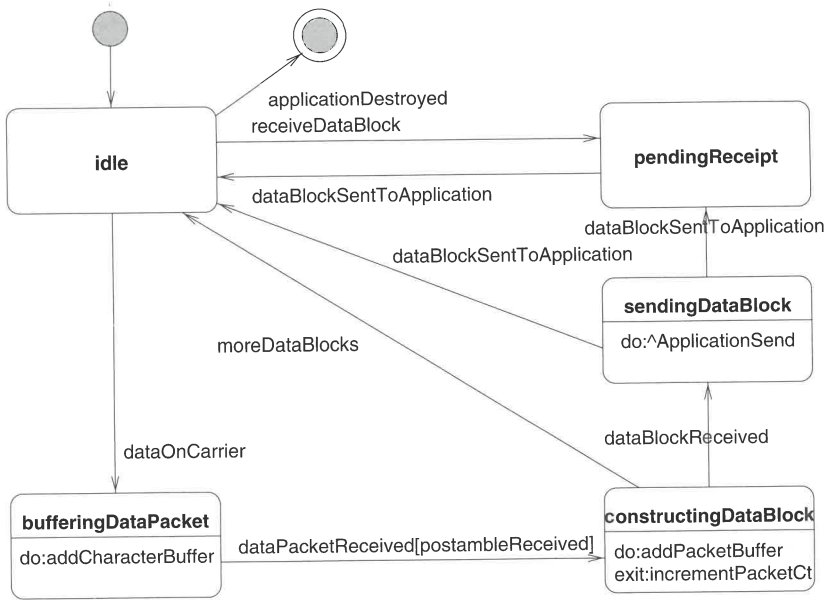

Fig. 9. An example of UML state diagram.

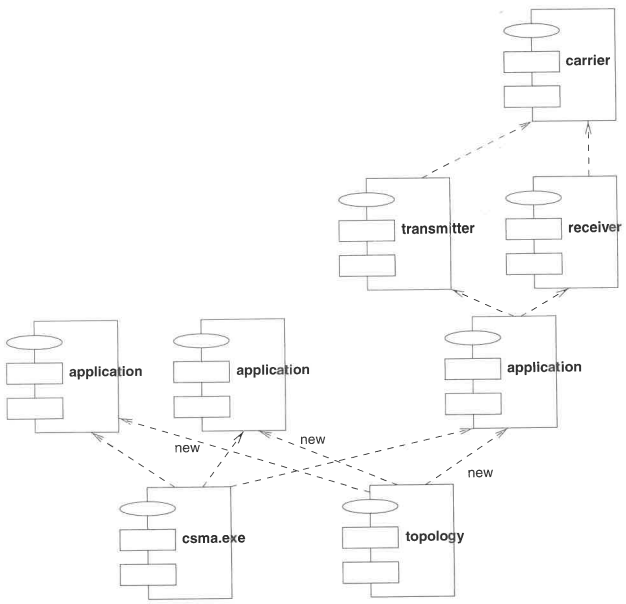

Fig. 10. An example of UML component diagram.

Grading

Students were graded on the quality of the work they produced, not on how many hours a week they spent on it. However, they were requested to create professional-looking documents, not only for "clients," but also for communication among themselves. Portfolios, labeled theme binders, etc., were recommended. Each part of the project was graded based on the accuracy, consistency, and completeness of its content as well as its organization (e.g., appropriate title, section and paragraph names) and appearance (e.g., consistent page numbers).

A similar approach, described by Saiedian (1996), was used for grading each team project and individual team members. Each part of the project was graded based on functionality, accuracy and completeness of content, as well as organization (e.g., appropriate title, section and paragraph names) and appearance (e.g., consistent page numbers). Each graded part was given back to the students, who then made the necessary corrections and modifications. At the end of the semester, all project parts were assembled and resubmitted as the final version of the project. The final project was once again graded for completeness and consistency.

A major issue in offering team-oriented project courses is establishing a fair system for grading the final product of a successful (or failed) project and determining the grades for individual members of a team. The grading process requires a lot of thought and care, particularly when judging whether a team member has done his or her share of the work. The instructor must determine whether the students be evaluated individually or as a team. For example, if the final project earns n points, should each team member receive n points? Should members who contributed less earn less points? What would be the criteria for determining who contributed less? Should contributions be viewed in terms of technical, time, or leadership contributions? The instructor has to find ways to evaluate the performance and contribution of each individual student. A useful method that we have been utilizing (especially when there are conflicts) is to have each student provide a "performance review." Each performance review includes the following:

- A written description of what he or she has done for the project
- An evaluation of the project by assessing its strengths and weaknesses
- A brief explanation of how differently the student would have approached the project if he or she had to do it again (Students must indicate what they thought was most successful about the project and what they thought was most unsuccessful.)

Students are required to prepare the performance review form, and their responses to the items in the performance review usually serve as a good indication of how actively they have participated in the development of the project. In addition to the above, students are asked at the beginning of the semester to keep a log. Students are told that it is a good idea to maintain a log of the time they spend on the project and a description of what they did during that time. The logs help students see how they have spent their time and helps them make better predictions about the time needed for the different phases of the design and development; they also help students in justifying and "proving" what they claim they have done for the project should they have to "prove" their participation. Etlinger (1990) and Scott and Cross (1995) suggest peer grading. The rationale for this is that if the students realize that they will be evaluated and graded by their fellow team members, they will be more sincere in doing their part of the project.

CONCLUSIONS

The CSMA simulation project is believed to be representative of projects that are manageable by groups of undergraduate students. As was illustrated, the functions required for the simulation can be decomposed into several simple modules, each of which can be assigned to a team member. The initial project can be used as the basis for designing and implementing new, higher level functions. This incremental approach for constructing a team project can be used to introduce new course concepts in a natural and practical way. Also, when alternatives to an existing concept are introduced (e.g., 1–persistent CSMA), existing modules in the project can be altered to reflect the variation.

A suitable team project not only enhances learning the principles of a course, but also provides students with an environment similar to that found in industry. Thus, courses utilizing team projects can better serve the needs of both graduates and their employers. What this project possesses is the ability to be expanded in a variety of ways to provide a suitable starting-point for the wide spectrum of courses described in the last section.

Our approach in using the project for an object-oriented software engineering course and an evaluation framework were introduced at the end of the article.

REFERENCES

ACM/IEEE-CS Joint Curriculum Task Force (1991). *Computing Curricula 1991.* New York: ACM Press.

Bruegge, B. (1992). Teaching an industry-oriented software engineering course. In C. Sledge (Ed.), *Software Engineering Education*, LNCS 640, proceedings of the 6th SEI Conference on Software Engineering Education (pp. 65–87). New York: Springer-Verlag.

Denning, P. (1992). Educating a new engineer. *Communications of the ACM, 35,* (12): 82–97.

Etlinger, H. (1990). A retrospective on an early software projects course (proceedings of the 21st ACM SIGCSE Symposium on Computer Science Education). *ACM SIGCSE Bulletin, 22,* (1): 72–77.

IEEE (1985). *Carrier sense multiple access with collision detection*, technical report, 802.3. IEEE.

Jacobson, I. (1992). *Object-oriented software engineering: A use-case driven approach.* Addison-Wesley.

Moore, M., & Potts, C. (1994). Learning by doing: Goals and Experiences of Two Software Engineering Project Courses. In J. Diaz-Herrera (Ed.). *Software Engineering Education*, LNCS 750, proceedings of the 7th SEI Conference on Software Engineering Education (pp. 151–164). New York: Springer-Verlag.

Saiedian, H. (1992). Guidelines for a Practical Approach to the Database Management System Course. *Journal of Information Systems Education, 4,* 23–29.

Saiedian, H. (1996). Organizing and Managing Software Engineering Team Projects. *Computer Science Education, 7,* 111–134.

Saiedian, H., & Wileman, S. (1993). A concurrent object-oriented framework for simulation of network protocols. *Journal of Systems and Software, 23,* 139–150.

Scott, T., & Cross, J. (1995). Team selection methods for student programming projects. In R. Ibrahim (Ed.). *Software Engineering Education*, LNCS 895, proceedings of the 8th SEI Conference on Software Engineering Education (pp. 295–303). New York: Springer-Verlag.

Tanenbaum, A. (1996). *Computer Networks* (3rd ed.). Prentice-Hall.

Tucker, A. (Ed.) (1991). Computing curricula 1991: A summary of the ACM/IEEE-CS joint curriculum task report. *Communications of the ACM, 34,* (6): 68–84.