Scott A. Smolka

Teaching Statement

Throughout my teaching career I have taught courses at both the undergraduate and graduate levels. The undergraduate courses I have primarily offered are *Database Systems*, *Operating Systems*, and *Undergraduate Concurrency*. I am also the co-developer, along with Michael Kifer, of the *OSP Operating System Project* courseware for undergraduate Operating Systems. OSP has been used as the operating systems courseware in over 100 institutions. At the graduate level, I have mainly taught the *Graduate Database Systems* course and a course on *Computer-Aided Verification* that I co-designed with Rance Cleaveland.

In each course I teach, I strive to create an interesting and challenging *course project*, so that students can see how the concepts and theory the course teaches them can be put into practice. The project usually involves a semester-long design and implementation effort, and the students typically work in groups of two or three in order to expose them to a team-project environment. In the case of the graduate Verification course, I ask the students in the course to apply an automated verification tool to a real-life application, or to develop a new verification technique of their own. As described below, a number of these efforts have resulted in conference and journal publications.

I am also very interested in the computer science undergraduate curriculum in general. From 1994-1996, I was the Undergraduate Program Director for the Stony Brook Computer Science Department. Before that, from 1992-1995, I chaired the Department's Committee on Undergraduate Computer Science Curriculum Reform, which implemented an extensive revision of the computer science undergraduate curriculum. The new curriculum, which featured a required two-course sequence in software engineering, took effect in Spring, 1995.

In what follows, I summarize my approach to teaching the above-listed courses and, in the process, elaborate on my teaching philosophy.

1 Undergraduate-Level Courses

1.1 CSE 306: Undergraduate Operating Systems

When I teach CSE 306, I center the course around Andrew S. Tanenbaum's textbook *Modern Operating Systems, 2/E*, for the requisite concepts and theory, and the OSP courseware, for the design and implementation project.

OSP (an Operating System Project) is a flexible environment for generating implementation projects appropriate for an introductory course in operating system design. It is intended to complement the use of most standard textbooks on operating systems and contains enough projects for up to three semesters. These projects expose students to many essential features of operating systems, while at the same time isolating them from low-level machine-dependent concerns. Thus, even in one semester students can learn about page-replacement strategies in virtual memory management, cpu-scheduling strategies, disk seek time optimization, and other issues in operating system design.

OSP consists of a number of modules, each of which performs a basic operating systems service such as device scheduling, cpu scheduling, interrupt handling, file manage-

ment, memory management, process management, resource management, and interprocess communication. By selectively omitting any subset of modules, an instructor can generate a project in which the students are to implement the missing parts. This process is completely automated by the OSP Project Generator, included in the distribution. Projects can be organized in any desired order so as to progress in a manner consistent with the lecture material.

The heart of OSP is a simulator that gives the illusion of a computer system with a dynamically evolving collection of user processes to be multiprogrammed. All the other modules of OSP are built to respond appropriately to the simulator-generated events that drive the operating system. The simulator "understands" its interaction with the other modules in that it can often detect an erroneous response by a module to a simulated event. In such cases, the simulator will gracefully terminate execution of the program by delivering a meaningful error message to the user, indicating where the error might be found. This facility serves both as a debugging tool for the student and as teaching tool for the instructor, as it ensures that student programs acceptable to the simulator are virtually bug-free.

The difficulty of the job streams generated by the simulator can be dynamically adjusted by manipulating the *simulation parameters*. This yields a simple and effective way of testing the quality of student programs. There are also facilities that allow students to debug their programs by interacting with OSP during simulation.

The underlying model of OSP is not a clone of any specific operating system. Rather it is an abstraction of the features commonly found in several systems (although a bias towards UNIX can be seen, at times). With the exception of the OSP modules for interprocess communication, the OSP modules were designed to hide many of the low-level concerns one encounters in operating system design and implementation, yet still encompass the most salient aspects of their real-life counterparts in modern systems.

OSP is thoroughly documented and explained from a student perspective in the following book: *OSP: An Environment for Operating System Projects*, Michael Kifer and Scott A. Smolka, Addison-Wesley (Dec. 1990). There is also an *Instructors Manual* available from Addison Wesley as a Computer Science supplement. The Instructors manual contains detailed instructions on how to use the OSP Project Generator. It also contains a number of carefully designed sample projects an instructor could assign over the course of one or two semesters. The OSP courseware itself is available to instructors via anonymous `ftp` from Addison-Wesley.

OSP, which was developed in the late 1980s, was written in the C programming language. Due to the rising popularity of Java within the Computer Science undergraduate curriculum, and because we wanted to modernize the basic organization of OSP, Michael Kifer and I have developed and implemented a successor to OSP we call *OSP-2*. OSP-2 is an object-oriented operating system in the truest sense of the term. Being written in the object-oriented programming language Java, system resources and data structures are represented by classes, thereby providing well-defined method-call interfaces between objects à la Windows 2000. Also subclassing is used to specialize objects; for example, the I/O Request Block (IORB) is a subclass of `Event` so that threads can wait on it and be notified of its occurrence.

OSP-2 is currently in beta-test mode and Michael Kifer and I have written a 200-page textbook about the system. Each chapter of the book contains a thorough yet succinct

discussion of the requisite conceptual material for the project (subsystem) the chapter is targeting. Discussions to publish the book either in print or electronic form are underway with several Computer Science publishers.

1.2 CSE 305: Undergraduate Database Systems

I have taught CSE 305 on eight different occasions. Prior to 2005, the textbook I used for this course was *Database Management Systems* by R. Ramakrishnan and J. Gehrke. Starting in the Spring of 2005, I switched to *Database Systems: An Application Oriented Approach* by M. Kifer, A. Bernstein and P. Lewis. My motivation for switching textbooks was twofold. First, Kifer, Bernstein and Lewis are all colleagues of mine at Stony Brook, with Kifer being my collaborator on the OSP and OSP-2 courseware projects. Secondly, their book devotes an entire chapter to using SQL in applications, with extensive coverage in particular of JDBC, a call-level interface for the execution of SQL statements from a Java program.

Each time I teach CSE 305, I assign a semester-long design and implementation project for which the students are required to develop, in order, an E-R model of their database system, a relational model complete with integrity constraints expressed as functional dependencies, an implementation of the queries in SQL, and finally a web-based interface using Java, JDBC, HTML, and Javascript or JSP. Specific course projects that I have assigned include a "mini eBay" system complete with eBay-style *proxy bidding*; an online airline-reservation and car-rental system (in collaboration with Himanshu Gupta); and a university course registration system.

1.3 CSE 375: Undergraduate Concurrency

I am the Stony Brook *Course Coordinator* for CSE 375. The textbook I use to teach this course is *Concurrency: State Models & Java Programs*, Jeff Magee and Jeff Kramer, Wiley (1999). This is an excellent book for upper-division undergraduates that introduces them to the concept of *Model-Based Development* (MBD): the use of executable models to drive the design and specification phases of a software-development project. Such models can be simulated, tested and verified, allowing design flaws to be discovered much earlier in the development process than traditional methods allow. Many large companies are using MBD processes to develop their software, including those in the automotive (Ford, GM, Toyota), aerospace (Honeywell, Northrup-Grumman), and medical-device sectors (Guidant, Medtronic).

CSE 375 teaches students about MBD through the FSP modeling language, in which processes communicate via message passing. FSP is a simple yet formal modeling language, making it very amenable to formal verification techniques. The target implementation language is Java, with particular emphasis placed on Java's thread-programming constructs. I usually again offer a course project so that students can get hands-on MBD experience using FSP and Java. I also give bi-weekly homework assignments that require the students in the course to design a concurrent system using FSP and then implement the model in Java.

I would also like to design a version of CSE 375 that uses MATLAB, Simulink, and Stateflow instead of FSP, so that students can gain experience using these industry-

standard modeling notations. Course projects would focus on automated control, such as the control software for a cruise-control or flight-guidance system. Due to the technical sophistication of these modeling languages, such a course might be more appropriate as an undergraduate Honors course, or even as a graduate-level course, rather than a standard upper-division undergraduate course.

1.4 CSE 532: Graduate Database Systems

As in the case of the undergraduate database systems course, I have taught CSE 532 on eight different occasions. The textbook I have most recently used for this course is *Database Management Systems (Third Edition)* by R. Ramakrishnan and J. Gehrke. After a brief introduction to the relational model, I focus on the following advanced topics in database systems: Object-Oriented Database Systems, Knowledge Bases, Internet Databases, Data Warehousing & OLAP, Parallel DBMS, Distributed Databases, Database Design, Database Tuning, and Data Mining.

Each time I teach CSE 532, I assign a semester-long design and implementation project for which the students are required to develop an advanced object-oriented or object-relational database system using the SQL3 Object Model and DB2. As part of the project, they are asked to implement a web-based interface for their system using Java, JDBC, HTML, and JSP. Specific course projects I have assigned include a streaming Video on Demand (VoD) web site; a "mini eBay" system complete with eBay-style *proxy bidding*; an on-line job/resumè posting service à la `hotjobs.com`; and an on-line MP3 music store.

1.5 CSE 535: Computer-Aided Verification

CSE 535 is a course on the computer-aided verification of asynchronous and concurrent systems. Rance Cleaveland and I co-developed CSE 535 while he was on the Stony Brook faculty. The course has both a theory component, focusing on the semantics of process logics and process algebras and corresponding verification algorithms, and a practice component, stressing verification tools and their application to real-life systems.

For the theory component, I use a collection of slides that Rance and I authored; the following topics are covered in the slides: the semantics of temporal-logic, both linear-time (LTL) and branching-time (CTL, CTL*, modal mu-calculus); algorithms for temporal-logic model checking; process algebra, with the primary focus on Milner's CCS; structured operational semantics; bisimulation and algorithms for computing strong and weak bisimulation equivalence; and axiomatizations of bisimulation.

To complement the slides, reading assignments are given from the following books: *Logic in Computer Science: Modelling and Reasoning about Systems*, M. Huth and M. D. Ryan (Cambridge University Press); and *The Formal Semantics of Programming Languages*, G. Winskel (MIT Press).

For the practice component, the students in the course are asked to carry out a semester-long project on specifying and verifying a real-life asynchronous/concurrent system using an automated verification tool. Rance and I maintain a repository of verification tools for this purpose, including the Concurrency Workbench [1], the XMC model checker [10, 11], Murφ [2], and Spin [7]. A number of these course projects reached a level of sophistica-

tion that warranted publication in respected Computer Science conferences and journals, including the Rether real-time Ethernet protocol case study [5, 6]; the GNU i-protocol case study [4, 8, 3]; and the VISA/Mastercard Secure Electronic Transactions (SET) protocol [9].

References

[1] R. Cleaveland, J. Parrow, and B. U. Steffen. The Concurrency Workbench: A sematics-based tool for the verification of concurrent systems. *ACM TOPLAS*, 15(1), 1993.

[2] D. L. Dill. The Murφ verification system. In R. Alur and T. A. Henzinger, editors, *Computer Aided Verification (CAV '96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 390–393, New Brunswick, New Jersey, July 1996. Springer-Verlag.

[3] Y. Dong, X. Du, G. Holzmann, and S. A. Smolka. Fighting livelock in the i-Protocol: A case study in explicit-state model checking. *Software Tools for Technology Transfer*, 4(2), 2003.

[4] Y. Dong, X. Du, Y. S. Ramakrishna, C. R. Ramakrishnan, I.V. Ramakrishnan, S. A. Smolka, O. Sokolsky, E. W. Stark, and D. S. Warren. Fighting livelock in the i-Protocol: A comparative study of verification tools. In *Tools and Algorithms for the Construction and Analysis of Algorithms (TACAS '99)*, Lecture Notes in Computer Science, Amsterdam, March 1999. Springer-Verlag.

[5] X. Du, K. T. McDonnel, E. Nanos, Y. S. Ramakrishna, and S. A. Smolka. Software design, specification, and verification: Lessons learned from the Rether case study. In *Proceedings of the Sixth International Conference on Algebraic Methodology and Software Technology (AMAST'97)*, Sydney, Australia, December 1997. Springer-Verlag.

[6] X. Du, S. A. Smolka, and R. Cleaveland. Local model checking and protocol analysis. *Software Tools for Technology Transfer*, 2(3):219–241, November 1999.

[7] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[8] G. J. Holzmann. The engineering of a model checker: The Gnu i-protocol case study revisited. In D. Dams, R. Gerth, S. Leue, and M. Massink, editors, *Theoretical and Practical Aspects of SPIN Model Checking*, volume 1680 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[9] S. Lu and S. A. Smolka. Model checking the Secure Electronic Transactions (SET) protocol. In *Proc. Seventh Intnl. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99)*. ACM Press, October 1999.

[10] Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. W. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In *Proceedings of CAV '97*, LNCS Vol. 1254, 1997. Springer-Verlag.

[11] C.R. Ramakrishnan, I.V. Ramakrishnan, S.A. Smolka, et al. XMC: A logic-programming-based verification toolset. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*. Springer-Verlag, June 2000.