Educator's Use of Concurrency When Describing Educational Simulations

Jan Erik Moström
Department of Computing Science
Umeå University, Sweden
jem@cs.umu.se

David A. Carr
Department of Computer Science
Luleå University of Technology, Sweden
david@cs.umu.se

Keywords: POP-II.A. end-users, POP-II.B. design, POP-IV.A. concurrency

Abstract

In this article, we report on an investigation into how educators describe the design of three software systems. We have been especially interested in how they describe events that are concurrent, as we believe support for concurrency will be required of an educational software authoring system. Their descriptions of concurrent processes can then be used to design the authoring system to "naturally" support concurrent programming.

Almost all participants in our investigation used concurrency as a part of their description. No one seemed to think that this was difficult. In addition, we observed that the educators frequently thought in terms of objects. These results suggest that any authoring environment targeted at teaching should include support for concurrency and objects.

Introduction

We have heard it many times: "We need computers in education." or "With computers everything becomes better." Advocates for increased computer use envision a body of interactive educational documents. This implies that authoring systems for educational software must support full programming capabilities. However, economic pressures on schools and universities make it clear; teachers will be the developers of educational applications. Furthermore, the vast majority of teachers are not software developers. One can therefore envision a large new population of end-user programmers, the teachers.

Among the applications that teachers will need to develop are complex simulations representing real-life environments such as ecosystems and transportation systems. These simulations are characterized by large numbers of independent entities acting autonomously and sometimes interacting with each other. They are, in short, highly concurrent systems. However, concurrent programming is traditionally considered a difficult subject and usually postponed until the final years of a Computer Science education. This leads us to ask, "If concurrent programming is difficult, how can it be supported for a group that may lack formal training in Computer Science?"

Clearly, one way to support concurrent programming is to make the development system's model of concurrency as close as possible to the teacher's mental model of concurrency in simulations. But, we ask, what mental models do teachers have?

In order to answer this question, we carried out an investigation of how educators thought about designing simulations where concurrent programming might be used. This investigation involved presenting eleven educators with example systems that contained varying degrees of concurrency. The educators were then asked to describe the manner in which the system would be created. We asked them to "think aloud" while creating a system solution. These sessions were then analyzed for clues as to how the educators approached concurrency.

After a short review of related work, we will present a detailed description and analysis of our investigation.

Related Work

Concurrent Programming by Novices

While no one has studied concurrent programming with educators as a target group, there have been studies with students learning to program concurrent systems. Waddington (1989) describes several studies involving concurrency and highlighting the differences between novice and expert programmers. His findings are very much in line with our experience: Novices have problems in understanding how different concurrent constructs work and do not understand their semantics. He also found that novices had trouble distinguishing relevant from irrelevant information. This leads to difficulties when debugging. Waddington also compared different inter-process communication constructs but failed to find any significant differences in understanding. However, he noted that this might be due to various effects of the experiment itself. Resnick (1990) found similar problems during his work with elementary-school children using a concurrent version of Logo. Interestingly, the children made fewer errors when modeling the real world. This suggests that concurrent programming may be easier to use in creating educational simulations.

Ben-Ari and Kolikant (Kolikant, et. al. 2000, Ben-Ari and Kolikant, 1999) have done much work studying how concurrency should be taught to different groups of students. They report experiences similar to Waddington. Their students had difficulties in both understanding the basic concepts of concurrency and understanding how the different synchronization mechanisms work. They suggest that one way of improving the understanding is to dramatize the program, having the students play the roles of different program parts. This suggests that one problem with learning concurrency is the "abstractedness" of concurrent systems. Interestingly enough, similar dramatizations are suggested to understand the design of object-oriented systems (Bellin and Simone 1997). Stein (1998) suggests that programming for Computer Science students should be taught in a new manner that emphasizes the interactive nature of programming. Her suggestions imply that the student would start writing concurrent programs in their first programming course. Weber-Wulff (2000) reports successfully using Steins approach.

We have also conducted a pilot study (Moström and Carr 1998a) of how novice programmers solved problems involving concurrent systems. In this study, a number of students were required to modify existing programs. We constructed two Pascal-like micro-languages and let the participants change the behavior of existing programs. The participants in this study were non-computer science students who had taken a basic course in Pascal, but who had no experience with concurrent programming. The results showed that using concurrency with certain problem types could make finding a solution both easier and faster.

Pane and Myers take an approach similar to the one we describe in this paper, designing languages closer to non-programmers' system descriptions (Pane and Myers 2000, Pane, et. al. 2001). They report that users expected objects to behave like independent individuals ("Objects normally moved, and remembered their state …"), and that systems were collections of individuals ("…a program is represented as a group of independent agents seated around a table.").

Educational Systems

There are currently two large projects aimed at building and sharing computer-based educational material that includes "real-life" simulations: EOE (Educational Objects Economy[1]) and ESCOT (Educational Software Components of Tomorrow[2]). A short summary of these projects follows below.

In the EOE project (Marion and Hacking 1998), teachers, developers, and industry have joined together in order to better understand how the Internet can be used to enhance education. Their goal is to build on-line communities to share ideas and software objects among the members.

[1] http://www.eoe.org
[2] http://www.escot.org

The actual objects are Java programs that illustrate some concept to be learned. The object can contain small animations of a physical event, something otherwise difficult to see, or a more complex simulation where the student is able to change different parameters to affect the outcome. In order to use an object, the user goes to an EOE library and searches the database something suitable. When an object is found, it can be included on the user's web page.

However, we believe that the use of Java creates a major problem. We have seen that teachers like to make some changes to existing material (Moström and Carr 1998b). To make changes to these objects, it is necessary to change the actual Java code. This, of course, requires the teacher to know Java, something we believe most teachers do not know.

ESCOT (Repenning, et. al. 2001) is similar to EOE in that it aims to support collaboration between educational users and sharing of software objects for educational use. But, it goes beyond the EOE model by providing a *framework* that integrates the different objects and tools into one environment. Instead of using Java, ESCOT allows the users to choose from a number of tools, for example: AgentSheets (Repenning 2000), PEN, and SimCalc. The different objects can be modified externally and even made to cooperate to form a new document. ESCOT seems promising, and it will be interesting to see what happens with this project in the future.

Studying How Educators Use Concurrency

In order to investigate educators' understanding of concurrent systems, we designed an interview study using the "thinking-aloud" method while educators created descriptions of systems that involved the use of concurrency.

Educators with varying degrees of programming experience participated in our study. They ranged from professional programmers who taught short courses to non-programming educators who regularly used computers.

To avoid biasing the participants toward (or away from) concurrent descriptions, we decided to use computer-based animations of programs that showed *how the finished program worked*. Participants were then asked to describe how the system was built. Our goal was to try to understand their descriptions and see if concurrency was used.

We did not provide a notation for the participants to use. In our earlier study of novice programmers (Moström and Carr 1998a), we noticed an influence from the notation, and we wanted to avoid biasing the solutions.

To record the thought process of the participants, we used the "thinking-aloud" method. The method requires the users to continuously and verbally describe how they reason about the problem. Our reason for choosing this method was simply that we judged that pure observation would not give us the information we were seeking and time measurements would not provide relevant results. The "thinking-aloud" method has two serious drawbacks:

- Different people have different abilities to describe how they think. Some find it very easy to do this kind of self-reflection, while others are seriously disrupted in their thought process by the requirement of describing how they think.
- As the problem becomes more difficult, our brains redirect their "computing resources" from less important tasks to the task of problem solving. This means that when the problem becomes difficult, the participant stops talking, which defeats the purpose of thinking-aloud.

In order to solve the last problem, it is common that the experimental set-up includes an experimenter whose task is to probe the participants when they fall silent. We chose this approach.

The Example Systems Used

The three example systems we used were simulated “systems”. This means that the behavior of the systems was implemented as animations. These animations were saved as a movie, and a speaker’s voice was added. The movies were then shown to the participants on a laptop.

The Information System

This was a simple information system where the users could look for contact information about the university staff. The system was built on a hierarchical model where the users were shown a new screen (or “menu”) when they clicked on an object. The system was sequential except for one simple animation.

The Foxes and Rabbits Simulation

The second system simulated the interaction between fox and rabbit populations. The representation was abstract, i.e. the creatures were represented as a large number of small colored squares. The “life” of the creatures was dependent on various individual attributes as well as on what the other creatures did.

The Road Intersection Simulation

The third and final application implemented a road intersection surveillance system. Here, it was imagined that the users should be able to monitor the traffic but not control it in any way. It was also possible to call up more detailed information about a specific car.

Procedure

Each participant took part in the study independently. The interviews occurred at different times and locations (usually at a neutral but well-known location) and followed the procedure below:

1. Each participant was given a written description of the study, which included the general goal of the study and an overview of what was going to happen.
2. The participant was given the first system description, and after the participant had read it, the experimenter asked if there were any questions.
3. When all questions had been answered, a movie containing an animation of the system was shown. Additional comments about the system were also given at this time.
4. The experimenter asked the participant to describe how the system was built. This description could be given using text, drawing, gestures, speech, or in any other way the participant preferred.
5. When the participant clearly indicated that nothing more was going to be added, the experimenter started to ask questions to make sure that the description was correctly understood.
6. When the experimenter was satisfied, the participant was asked to continue with the next system.
7. Steps 2 – 6 were repeated for systems two and three.
8. When the operation of all three systems had been described the experimenter asked if the participant wanted to change/add something to any of the descriptions.
9. In the cases where the participant did not indicate a concurrent solution, the experimenter gave a quick outline of a concurrent solution to each of the three systems and asked the participant to comment on it.
10. The participant was asked if there were any further questions, and after answering them, the experimenter thanked the participant for taking part in the study.

Step 9 warrants further explanation. During the pilot tests of the study, it was discovered that the participants *might think it was not possible to have two or more processes running at the same time or that exchanging data between the processes was not possible*. When they realized that this was indeed possible, we received comments such as:

"*Ohhh, is that possible ... I was thinking like that, but I didn't think it could be done*"

The Participants

Our goal was to enlist participants with varied backgrounds and levels of experience in programming. We divided them into three groups:

Non-programmers

These had no programming experience, whatsoever. All were using computers on a regular basis, usually daily. The group consisted of one lecturer in Media and Communication, one lecturer in Physiotherapy, one former principal of a Forestry school, and one former elementary school teacher — three women and one man.

Inexperienced programmers

These people had some basic experience with programming, typically a first course in programming. They were not programmers in their normal work, although they worked in technical areas. The group consisted of one mechanical engineer, one Computer Science teacher, who had not programmed in a long time, and one teacher in electronics and multimedia — two women and one man.

Experienced programmers

This group included people who wrote programs professionally or taught programming. Two were from the Computer Science Department at Umeå University. Another worked for a large telecommunication company, and a fourth worked for an educational software publisher — two women and two men.

All of the participants, except the educational software developer, had teaching experience, ranging from teaching at elementary school to supervising graduate students in university-level courses.

Results

As outlined above, we hoped to see a clear indication as to whether or not the participants would use concurrency in their descriptions. We also hoped that we would get some indication of the notations that participants might use when describing concurrent actions.

The participants were told that they could describe their solution in any way they wanted "from dancing to writing a program". They received pen, paper, and printouts of the pictures from the movies. However, when describing their solutions the participants all made a verbal description with gestures. This was somewhat surprising since we had expected the participants to use pen and paper in addition to a verbal description of the solution. In fact, during the pilot tests of the study, pen and paper played an important part in how the systems were described. We believe the reason for this change can be attributed to some unrecognized change in the experimenter's behavior, or possibly that the participants felt they lacked a "syntax" to describe their solution.

As could be expected, the differences among the participants were large. We could see differences in how they approached the tasks, how they explained their solutions, and also how important the description of the problem was. Most of the non-programmers seemed to have the same problem as many novices have in their first programming course; how to divide the problem into more manageable sub-problems. As one student once said, "It's like a giant stone wall and there is no way of tearing it down.” The same reaction could be seen in the answers from several of the participants during this study.

Not surprisingly, the participant’s previous experience played an important part in how they described their solutions. Because of this, the solutions were both in vague, non-technical terms and in very specific technical terms.

Descriptions of the Information System

We had not expected any descriptions that involved concurrency in this task and we were proven right. The animation could have been described concurrently, but all participants perceived it as a self-contained introductory sequence to the system. However, we found one important clue as to how an authoring system should be designed: The description of a system is heavily influenced by previous experiences. This is something that should be taken into consideration when designing an authoring system/language. It is impossible for us to say what terms/words/symbols should be used in order to make an easy-to-use system. However, one thing that caught our attention was that almost all users used “World-Wide-Web metaphors” to describe the relationship between different objects. By “World-Wide-Web metaphors” we mean expressions such as: “links to”, “jumps to”, and “another page”.

Descriptions of the Foxes and Rabbits Simulation and the Intersection Simulations

In comparing how the participants reacted to the foxes-and-rabbits and intersection simulations, it is interesting to see how different the reactions were. If we strip down the description of these problems, we can see that they are very similar: They both describe a system with several “individuals” that move around and are influenced by other “individuals”. Despite this similarity, the participants’ descriptions and reactions varied markedly. The intersection problem seemed to be much easier to understand and describe. The reasons for this could be:

- The sheer number of processes. There are many more “individual entities” in foxes and rabbits simulation than the intersection simulation. It is possible that this makes it more difficult to understand the problem and describe a solution for it.
- The descriptions of the tasks. Although both tasks were described in similar fashion (a film that showed how the system worked together with a textual description), there were differences in the description. The textual description of the foxes and rabbits simulation was much longer than the one for the intersection. There was also much less detail in the graphics for the foxes and rabbits simulation.
- The “directness” of the intersection simulation. All participants had previous experience with the situation depicted in the intersection simulation. However, few had previous experience with the problem presented in the foxes and rabbits simulation. The foxes and rabbits simulation also required a longer, more complex description.

These observations fit nicely with the first of the five design principles that Rader, et. al. (1998) describe: “Syntonicity. Programmers will be more successful if they can identify with the object they are programming”. It could be that it is easier for the participants to identify themselves with the cars and drivers than with the more anonymous rabbits and foxes. However, we can offer no explanation for the variations in descriptions encountered during this study.

Observations

The Influence of the Problem Description

As has been observed and commented many times before — see for example Norman (1990) and Nardi (Nardi and Miller 1993) — the closer the problem description is to the users’ previous experiences, the easier it is to solve the problem.

This was especially clear with the Forestry School principal who found it very difficult to understand the movie for the foxes and rabbits simulation. Yet, he was very familiar with the subject on a practical level and had long experience with similar problems. The experimenter then rephrased the problem and told the participant to envision fox and rabbit populations interacting but confined to a game board with small figures for the rabbits and foxes. After this, it took only a second or two before the participant was able to suggest a solution. A few of the other participants showed similar reactions, though less pronounced.

Thinking in Terms of Objects Seems to Be Easy

During the study, we could see that all participants were thinking in terms of objects at some time. For example, instead of describing a solution to the foxes and rabbits simulation as, "*A matrix where each cell is a record that can store the type of animal, the age of, etc.*", most used statements such as, *"...both rabbits and foxes have an age,"* and *"... The animals are placed on a matrix and moved when ...*". Many of them also described how objects communicated with each other.

Concurrency is Useful

All of the participants thought in terms of concurrency at one time or another. This seemed quite natural for them. We acknowledge that the tasks were very simple, and that there was very little communication between the different "processes". Also, the need for synchronization was limited, and only the group of experienced programmers mentioned it.

The Influence of Training in Programming

While it is clear that programming education and experience influenced the descriptions, we saw a shift toward serial solutions as training and experience increased. Experienced programmers constructed solutions that looped through objects for both the foxes and rabbits simulation and the intersection simulation. When questioned about this they cited efficiency concerns. In the case of one pilot subject who was an inexperienced programmer, the concurrent solution was rejected as not possible. This appears to be a by-product of current teaching methods that concentrate on serial solutions.

Conclusion

Our study shows that most participants found it easy to think in terms of objects and also used concurrency naturally in their description of their solutions. We believe this is a clear indication that an authoring environment for courseware should include support for both objects and concurrency. This does not mean that the smaller syntactical details of an authoring language are unimportant. On the contrary, Rader (Rader, et. al. 1998) shows how even small changes in the syntax of an authoring language can result in big differences as to the perceived difficultly in learning the language. But, before concentrating on the smaller details, we think that more basic issues such as the underlying paradigm should be addressed. As can be seen from this study, concurrency can be used, and is sometimes preferred, by non-programmers.

Furthermore, there seems to be a pattern as to how the inexperienced programmers thought about concurrency. They described each "entity" independently and expected it to behave according to that description until influenced by external events. Interaction among entities was described as occurring after certain conditions were met. This would imply that a natural representation would be to have entities (objects) and interaction rules (inter-object communication). Testing this hypothesis is a good area for future work.

Other areas for future work include more detailed investigation of available authoring systems to see how well they meet the requirements of teachers and students. We are also interested in comparing results from Computer Science education (Börstler, et. al. 2002) to see how they can be applied in designing authoring systems. Börstler's group is investigating a new approach in teaching object-oriented programming. They believe that a more all-inclusive approach to programming will improve learning as compared to concentrating on low-level syntax. The initial results are promising and could be useful when defining an authoring environment or language.

Acknowledgments

We would like to express our thanks to the eleven busy educators who volunteered to help us with this investigation. We would also like to thank Carl Rollo for proofreading drafts of this paper.

References

Bellin, D. and Simone, S. S. (1997) *The CRC Card Book*. Addison-Wesley Publishing Company.

Ben-Ari, M. and Kolikant, Y. B.-D. (1999) Thinking Parallel: The Process of Learning Concurrency. Proceedings of the *4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education*, 13 – 16. ACM Press.

Börstler, J., Johansson, T. and Nordström, N. (2002) *Teaching OO Concepts – A Case Study Using CRC-Cards and BlueJ*. Department of Computing Science, Umeå University. Submitted.

Kolikant, Y. B.-D., Ben-Ari, M. and Pollack, S. (2000) The Anthropology of Semaphores. *ITiCSE '00, Proceedings of the 5th annual SIGCSE/SIGCUE on Innovation and technology in computer science education*. ACM Press.

Marion, A. and Hacking, E.H. (1998) Educational Publishing and the World Wide Web, *Journal of Interactive Media in Education*, 98(2), http://www-jime.open.ac.uk/98/2/.

Moström, J. E. and Carr, D. A. (1998a) Programming Paradigms and Program Comprehension by Novices. *PPIG'10 — Workshop 10 of the Psychology of Programmers Interest Group*, Milton Keynes, Great Britain.

Moström, J. E. and Carr, D. A. (1998b) Teachers, teaching and computers. *STIMDI'98*, Umeå, Sweden.

Nardi, B. A. and Miller, J. R. (1993) Twinkling lights and nested loops: distributed problem solving and spreadsheet development. *In Readings in Groupware and Computer-Supported Cooperative Work: Assisting Human-Human Collaboration*, 260 – 271. Morgan Kaufmann.

Norman. D.A. (1990) *The Design of Everyday Things*. Doubleday/Currency.

Pane, J. F. and Myers, B. A. (2000) The Influence of the Psychology of Programming on a Language Design: Project Status Report. *Psychology of Programmers Interest Group 2000*.

Pane, J. F., Ratanamahatana, C. A. and Myers, B. A. (2001) Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, (54): 237–264.

Rader, C., Cherry, G., Brand, C., Repenning, A. and Lewis, C. (1998) Principles to Scaffold Mixed Textual and Iconic End-User Programming Languages. *Proceedings of the 1998 IEEE Symposium of Visual Languages*, 187–194.

Repenning, A. (2000) AgentSheets: an Interactive Simulation Environment with End-User Programmable Agents. *Interaction 2000*.

Repenning, A., Ioannidou, A., Payton, M., Ye, W., and Roschelle, J. (2001). Using Components for Rapid Distributed Software-Development. *IEEE Software*, 18(2), 38-45.

Resnick, M. (1990) MultiLogo: A Study of Children and Concurrent Programming, *Interactive Learning Environments*, 1(3).

Stein, L. A. (1998) What We Swept Under the Rug: Radically Rethinking CS1, *Computer Science Education*, 8() 118-129.

Weber-Wulff, D. (2000) Combating the Code Warrior : A Different Sort of Programming Instruction, *Proceedings of the ITiCSE 2000*, Helsinki Finland, July 11-13, 2000. 85-88.

Waddington, R. (1989) *User-Centered Design of Software Development Environments: An Example from Program Debugging*. PhD thesis, University of Nottingham.