Convit, a Tool for Learning Concurrent Programming

Hannu-Matti Järvinen, Mikko Tiusanen, and Antti T. Virtanen
Software Systems Institute, Department of Information Science
Tampere University of Technology
Finland
{hmj,mikko,virtanea}@cs.tut.fi

Abstract: Concurrent programming is an important and difficult topic for most computer science students. This paper presents a visualization tool called Convit to help students master and understand concurrent programming problems and their solutions. The tool, a Java applet, is basically a debugger or simulator, which runs concurrent programs written in a simple pseudocode. The code is translated into Java run in the applet. The students are supposed to work on their own with the tool exploring the examples and problems created by teachers. The tool also allows modification of the synchronization statements, to a limited degree, so that the student may try out various choices.

Introduction

Concurrent programming is difficult to learn and master. It involves making multiple programs co-operate and synchronize to achieve a mutual goal or, at least, not to interfere with each other. Since these problems appear in computer networks, distributed systems, operating systems, and embedded systems, it is nevertheless an important area of programming: there is a constant pressure to find better methods for teaching it.

Convit (CONcurrent VIsualization Tool) is a simple Java-based tool developed for the operating systems course at the Tampere University of Technology [1]. The focus of the tool is on the actual implementation of algorithms and solutions used in concurrent programming. In this paper we will briefly discuss some aspects of the problem of teaching concurrent programming, our approach embodied in Convit, and, finally, some of its qualities and future use.

For further information about the project and it's status see convit homepage at http://www.cs.tut.fi/~convit/

Concurrent programming

Many students find concurrent programs hard to understand. Such programs consist of multiple, mostly independent but interacting parts in execution simultaneously. In a sense, this is similar to juggling balls or drawing with colors instead of a pencil: the interaction of programs adds a dimension to the thought process needed to write the program. Especially, the proper use of *mutual exclusion* preventing simultaneous or interleaved access to data or peripherals, is difficult to master. This is not surprising since concurrent programming requires a way of thinking different from that taught by most initial programming courses. Moreover, it is much more difficult to detect programming errors in concurrent programs than in ordinary sequential ones, not an easy problem even there.

Shene [2] suggests that the best place to introduce multithreaded programming to students is the operating systems course. This seems to be common practise in nearly all universities around the world. Indeed, this is also the latest possible point in the curriculum, since teaching operating systems without concurrent programming is unthinkable, hopefully.

How to teach this particular hard topic is the usual predicament of the teacher of the operating systems course. We are, therefore, aiming to write a small system, Convit, that could be used by the students to visualize concurrent execution and interaction of small pseudocode concurrent programs.

From the teacher's point of view an ideal tool would be easy to use, portable, extensible, easily maintainable and require very little installation and setup, if any. Naturally, it should help the students to get a grip on concurrent programming, at least on mutual exclusion and synchronization. Obviously, the students with a kinesthetic inclination would most benefit from such a tool, since they could learn hands-on about how such

programs behave.

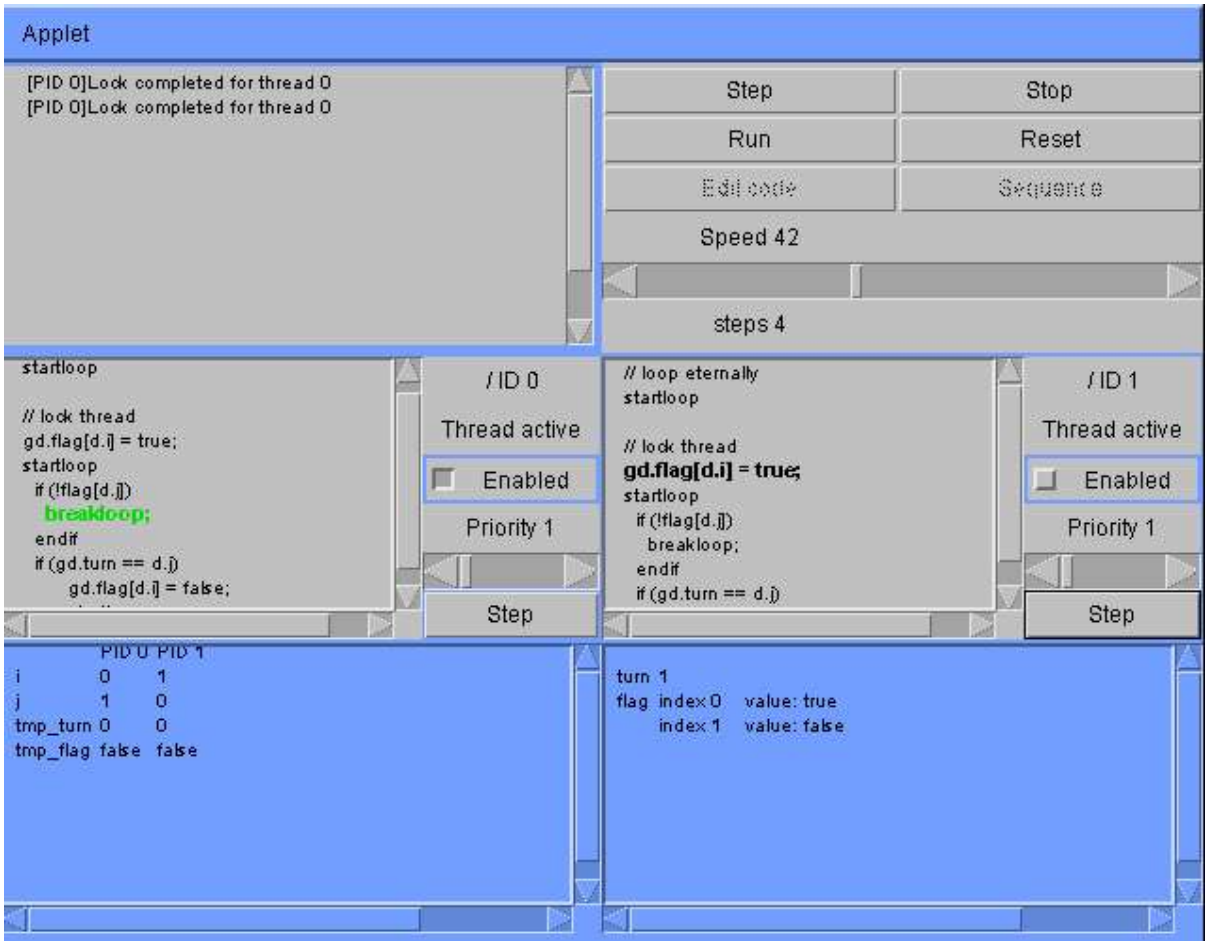

Figure 1: A screenshot of the tool

The approach

Convit lets the students test their ideas and experiment with various examples and problems, with instant visual feedback from the tool (Fig.1). Convit operates on source code and variables, shared or not. In addition to simple debugging and simulation, Convit provides means to modify the source code and detects global deadlocks as they occur. It also implements a simple scripting language to specify predefined action sequences for demonstration purposes. Currently all examples and interactive problems must be created by the teacher beforehand, since this involves a compilation into Java of the code of the concurrent program parts; this restriction is to be relaxed later.

There are other tools employing more abstract concepts [3,4], while Convit aims to demonstrate how these concepts show up in the source code and how they are actually implemented. Basically a multithreaded debugger, the programs Convit runs have no outside effect, however. The programming language employed is a simple pseudocode specifically designed for the tool. Systems similar in concept to Convit are Ben-Ari's BACI (Ben-Ari Concurrency Interface) [5] and ConcurrentMentor by Carr et al [6].

The Convit programming language is very simple, but powerful enough to cover all elementary concurrent programming problems and algorithms. The syntax looks like a mixture of Pascal and C, and should be easy to understand for anyone with a little knowledge about computer programming.

```
startloop        // begin program
  P(g->mutex);
  g->read_count := g->read_count+1;
  // do something
  println("working on ...");
  V(g->mutex);
endloop
```

The scripting language is even simpler and has only four basic commands: advance thread X / delay / wait for user to press a key / display message. These are meant to be used mainly by the teacher to set up demonstrations.

Convit is currently implemented as a Java applet. This was a natural choice for a number of reasons:

- The system can be made portable and can be used in practically any computer.
- Java is supposed to be more maintainable than alternatives, should the hardware and operating systems change.
- Java has facilities, like *reflection* (below), which are convenient for this sort of project.

The system is divided into three main parts: the parser of the language, the engine, and the graphical user interface (GUI). The engine and the GUI are not dependent on the actual syntax of the language, nor on the parser otherwise.

The engine handles mutexes, semaphores, thread priorities, and the global deadlock detection. It is the dispatcher of the system, but needs not restrict what the threads, the concurrent programs, actually do.

The GUI initializes the threads and the engine, but the details of the thread initialization is separate from the GUI. Each algorithm or example run by Convit should provide its own version of the thread initialization code. As pointed out by [3], selecting the right method for visualization is crucial. This will need some work later: the current implementation is sufficient as a concept study and as a basis for further development in this respect.

The reflection API and classloader system of Java [7] provide a clean way to handle the separation of actual data and threads (running the source code) from the rest of the system. With it, the system basically only needs the name of the class responsible for the thread initialization.

Discussion

Currently, the students are provided with a limited possibility to modify the example source code, specifically, to add and remove synchronization statements. In the future this limitation is ought to be removed by using a different approach, allowing more radical changes.

Adding a new problem or example to the system and testing it takes approximately one hour for a knowledgeable user. This should be acceptable.

As Convit was programmed as a Java applet, it can be used virtually from any computer with a web browser. JDK 1.1 was used to ensure this: many browsers only support this and not the later versions, though these would have provided some useful libraries.

Since Convit has not currently been used on a course we cannot demonstrate how useful it is. We plan, however, to use it during the operating systems course of our institute [1] during the fall term of 2003 and try to measure its usefulness. At least we will survey the students' opinion of its usefulness. Also, it may be reasonable to divide the students into two groups: students in the other group using the tool, and others not. By comparing the results of these groups it should be possible to estimate the value of the tool.

References

[1] URL http://www.cs.tut.ti/kurssit/8104000/ (TUT Operating systems course, referenced Aug 28, 2003; mostly in Finnish).

[2] Shene C-K: "Multithreaded Programming Can Strengthen an Operating Systems Course". Department of Computer Science, Michigan Technological University, Houghton, MI 49931-1295, USA.

[3] Gruia-Catalin R, Cox KC, Wilcox CD, Plun JY (1991). Pavane: A System for Declarative Visualization of Concurrent Computations. Washington University in St Louis, School of Engineering & Applied Science, WUCS-91-26.

[4] Hartley SJ (1994). "Animating Operating Systems Algorithms with XTANGO". Math and Computer Science Department, Drexel University, Philadelphia.

[5] URL http://www.mines.edu/fs_home/tcamp/baci/ (referenced Aug 27, 2003).

[6] Carr S, Fang C, Jozwowski TR, Mayo J, Shene C-K (2003). "ConcurrentMentor: A Visualization System for Distributed Programming Education". The 2003 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, June 23-26, 2003, pp.1676-1682.

[7] Gosling J, Joy W, Steele G, Bracha G (2003). Java Language Specification, 2nd edition. Addison-Wesley Publishing Company, 2000.