Work In Progress – An Integrated Programming Environment suitable for Distance Learning

Colin B. Price
Business School (Computing Division) University College Worcester,
Worcester, WR2 6AJ, UK.  c.price@worc.ac.uk

Abstract – Autonomous Robots are used as a vehicle to learn programming in C and Java, to design neural network controllers, to experiment with Finite State Machines and to explore operating system concepts. Our environment is deployed as (i) physical robots (ii) multi-robot simulation written as a Java application (iii) distributed multi-robot simulation using client-server architecture running on the Web. This paper details the design rationale, implementation, deployment and reports evaluation of the work to date. The software is freely available from the author at c.price@worc.ac.uk.

Index Terms – Computer Science Education, Java, Real-Time Operating Systems.

INTRODUCTION

Students of Computer Science and Engineering often are required to become familiar with a range of programming languages and environments to support a wide range of taught modules including operating systems, artificial intelligence, microcontrollers and various flavours of programming. Our experience is that this is often a daunting prospect and our response has been to develop an Integrated Environment for Programming where all of the module requirements can be satisfied through one simple vehicle; programming the behaviour of robots, both real and simulated.

Our environment can be used in general CS1 modules as well as the more specialist CS2 and CS3. Major parts have been in place for one year and student evaluation has been highly positive. The environment has also been successfully used with 14-16 year-olds and 16-18 year-olds.

DEVELOPMENT RATIONALE

The need to support the wide range of modules indicated quickly led to an implementation strategy. First the use of behaviour programming (such as Brooks subsumption architecture [1]) followed from the need to support multitasking for the operating systems module. A real-time operating system implemented on a simple microcontroller board would work with physical robots. Mapping these constraints onto the choice of language for the simulators suggested Java with its inbuilt multithreading. This opened up the possibility of distance and collaborative learning.

The hardware implementation chosen includes the Philips LPC932 chip on the MBC900 board supplied by Keil [3] which the students program in C. The Java simulator has been so designed that the robot C-code can be pasted directly into its editor and compiled. Students learn robotics more efficiently when a simulator is available [2], learning can also be divided between simulator and physical robot [5].

A crucial aspect of our design rationale was to provide a rich learning environment with a diversity of possible student paths and experiences. Notions of embodiment and emergence were high on our agenda [6].

Another aspect was to provide an affordable product. The entire environment is freely available, the only cost is the robot (Lego components and microcontroller) which is < $100 which compares favourably with other systems [4].

Students were involved at every stage in the development process and influenced key decisions such as not to use ‘line following’ tasks, rather to program behaviour in the sense of Braitenberg.

LOCAL IMPLEMENTATION

The simulator has been designed to support sequential programming, multitasking, subsumption architecture programming, neural net controllers and Finite State Machine (FSM) programming. All use Java syntax, though a table-driven FSM interface has also been implemented.

A built-in editor panel allows students to modify and enter behaviour code. The simulator imports javac to compile, and reloads the behaviour classes dynamically into a running application. The neighbouring ‘robot world’ panel displays robots and other objects (trees, rocks, gremlins) set up by the tutor or possibly by the student. There is extremely rapid feedback between the student’s changes to the code and the observed behaviour of the robots. A typical educational scenario will involve tutors designing a series of robot worlds and giving the students tasks to be carried out for each world. Each student then designs an appropriate behaviour for each task.

Code written in the editor panel can be cut and pasted into the physical robots IDE with almost no modification. The required calling of functions (methods) within Java was obtained by method Reflection. We have found that students choose the simulator for individual learning, for developing complex behaviour code (including debugging) and only then move on to the physical robots when a group activity often spontaneously emerges.

It is important to note that the simulator runs worlds with many robots, each running the same code consisting of several behaviours which run concurrently and are selected by robot
inputs, e.g. collision sensing selects the ‘avoid’ behaviour. Each behaviour runs as a separate Java thread and each robot runs as a thread group. This structure is maintained on the physical robot where we have developed a multitasking RTOS where behaviours are tasks and arbitration is cooperative which supports subsumption.

How do students on programming modules use the self-contained simulator? They first absorb the basic syntax of C and Java while learning about robot control and AI. We then ‘roll back’ the simulator code in a minimalist Java IDE (e.g. ‘JCreator’) to reveal the ‘OO-ness’ of the simulator and so let students analyze the simulator code and of course start to extend it and synthesize their own.

DISTRIBUTED IMPLEMENTATION

At University College Worcester we are currently introducing a VLE into our learning and teaching (‘WebCT’) and look forward to evaluating its effectiveness on our learning and teaching. Our Java simulator is a prime candidate for development into an independent ‘lightweight’ VLE. The current ‘Work In Progress’ is moving towards this end. We are using a client-server architecture with the ‘robot world’ maintained on a server machine and where individual students develop their robot behaviour code as clients. Each robot may now have unique behaviour. As the robot world runs, the system presents the entire world to each client.

In keeping with our principle of Integration, our distributed application has an identical user interface to the local implementation and so forms another part of a whole package. The main consequence of the distributed application is that one common robot world will support different individual robot behaviours. The pedagogical implications of this are beginning to be addressed – how quickly the students are willing to move away from tutor-provided behaviour code and form competitive or cooperative groups. Future extensions will almost certainly include online tutorials, and electronic forums included in the environment.

This distributed implementation is currently in development, both the pedagogy and the technology. Two alternative technologies are being evaluated, Java RMI and Java Sockets. Issues addressed are; the most efficient balance between computation and communication, the speed of response and the ease of implementation. Three architectures are being evaluated.

In the first, each client compiles the robot behaviour, and loads the class into the server. The server holds all client’s behaviours, runs them and sends updated positions to each client after each time quantum. Clearly the server is heavily loaded and communication is intensive. Yet the system is synchronous. In the second, the server distributes a copy of each client robot behaviour to all clients. Each client computes the entire robot world locally with server intervention only on addition of new clients or modification of existing behaviours. Processing is uniformly distributed, though variation in client processor speeds will prevent synchronous behaviour on all clients. The third architecture runs each client behaviour on the client machine. After each time quantum updated positions are gathered and broadcast by the server to all clients. We expect this to provide the most efficient use of processing and communication bandwidths.

EDUCATIONAL ASPECTS

The expected outcomes of this project are to see an improvement student learning of programming as reflected in their performance in CS2 programming modules. We also expect more students to choose the CS2 programming options. Our student cohorts are small, typically 20-30, too small to be subject to quantitative evaluation as in [2]. Yet feedback from UCW students (questionnaire) is very positive. Students involved in the project development are clearly enthusiastic. Our external engagements (widening participation) with 14-16 year-olds and 16-18 year-olds have elicited very positive feedback with requests for development of a tailored course structured to their needs.

The architecture is strong enough to support various learning methodologies. Competitive and cooperative teamwork is easily supported. Alternatively, the tutor may provide directed or interactive guidance to single or groups of students. The efficacy of these methodologies will be evaluated in this research.

ACKNOWLEDGMENTS

We express thanks to HITEK UK Ltd for donating Keil MBC900 (Philips LPC932) evaluation boards to implement the Robot hardware and to the Learning and Teaching Centre of University College Worcester for funding this research project.

REFERENCES


Session T1C

The Unofficial Guide to LEGO MINDSTORMS Robots O’Reilly, 1999