SPREAD: A Distributed Simulation TOOLKIT

P.Chernett V. Callaghan, M. Colley, J.Standeven
email vic@essex.ac.uk

Abstract - **this article describes "SPREAD", a simulation tool kit and its use in building "Virtual Robots", a simulation of multiple mobile robot vehicles used in the teaching of Computer Science at university level. A novel aspect of the simulator is the use of PVM [1] to achieve high performance at low cost by using spare CPU cycles on large numbers of networked workstations.**

1. INTRODUCTION

The computer science department at the University of Essex (in common with other universities contributing to this issue) has for some time used simple robot vehicles, which we characterize as Intelligent Autonomous Vehicles (IAVs), in the teaching of undergraduate computer scientists. We believe that having to test their science in the physical world provides an antidote to the increasingly theoretical direction that computer science has taken in recent years. Further, in building, from the ground up, an even modestly autonomous robot, it quickly becomes evident to students that they must draw upon and integrate many areas of the computer science curriculum from register-level interfacing, through adaptive control to full-blown AI issues such as planning, machine vision and world modeling..

Fig. 1-1 "Trillian", one of the Essex robots.

One way to provide wider access to limited and expensive physical resources such as robots is by simulation. By allowing students to prepare their work before their allocated time-slot on the real robots, that time can be more efficiently and productively spent. In addition, once programs have been verified in the "real" world, simulation can be used to run experiments that go beyond the physical resources of the laboratory. A prime example of this is in the area of multi-agent AI where simulation can allow experimentation with large numbers of virtual robots where only a few may be available in reality.

There are, of course, several commercially available packages for supporting this type of simulation. Unfortunately the cost of these, and the specialist workstations needed to run them put them beyond the means of departments that only require the facility as a relatively small part of their overall activity. There are several low-cost or shareware simulation packages available but many (for example Simderella [2]) assume static robots with manipulator arms rather than IAVs. Other popular robot vehicle simulators, such as Xmouse [3], are usually built on very simple models that have little relation to any real robots and can often only deal with single robots. Those that do simulate multiple robots such as Mission Lab [4] are often at a very high level and don't allow detailed simulations of individual robots. Yet others such as Khepera [5] are tied to particular vehicles.

Our approach has thus been to develop the SPREAD simulation engine to supports the simulation of complex worlds, inhabited by multiple autonomous vehicles, each of which may be modeled as many parallel embedded processes. Overall performance is maintained by distributing execution across a network of computers. An important design aim was machine independence so that the system could be used in a wide range of institutions while still taking advantage of whatever hardware they happened to have available. This portability extends to the worlds to be simulated as well as to the simulator itself.

In this paper SPREAD is discussed in the context of the "Virtual Robots" simulation tool used in the Brooker Laboratory for Intelligent Embedded Systems at Essex. Virtual Robots was built using the SPREAD toolkit and has been in use in the department for several years.

We will first describe the architecture of SPREAD and its use in Virtual Robots. This will be followed some performance figures and a brief description of our future plans.

2. THE SPREAD ARCHITECTURE

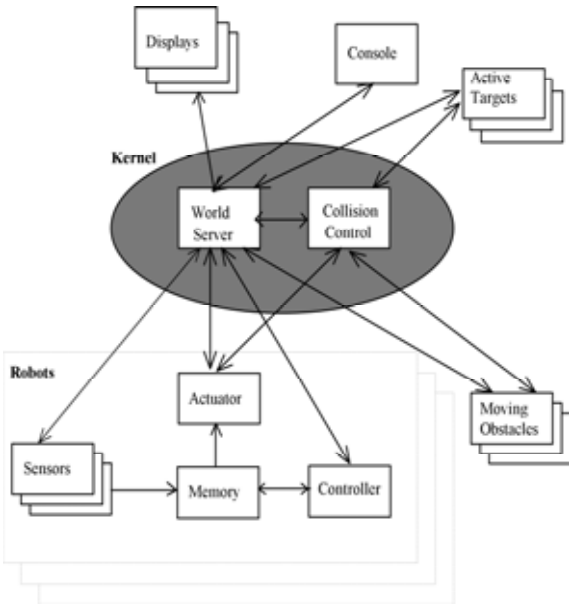

Fig. 2-1 Overall Architecture of the Simulator

The simulator structure (see Fig. 2-1) consists of the following types of module: console, display, world, collision control, actuator, sensor, memory, controller, moving obstacle and active target.

A single world server and collision control server form the kernel of the simulator. The world server maintains a database of the position of all objects in the simulated world. The collision control

module negotiates with objects that have collided and reports back the result to the world server.

Each robot consists of a controller, an associated set of sensors and, if the I/O to be modeled is memory mapped, a memory module.

The console, of which there must be exactly one, provides control of the simulation itself and is responsible for starting and terminating the simulation as well as making run-time adjustments such as the level of detail to be simulated. Display modules, of which there may be several, present the current state of the simulation to the users.

Along with the modules of the simulation itself the complete Virtual Robots package includes graphical editors for designing environments and setting up initial conditions for simulator runs, such as the position and orientation of the IAVs.

3. THE USER INTERFACE

1. There are four classes of SPREAD user:
2. those that simply use it via a GUI like Virtual Robots, together with an accumulated set of robot and robot parts,
3. those that want to write and test robot controllers, normally in parallel with testing the same controllers on real robots,
4. those that want to add new simulations of robot parts, and finally
5. those that want to build new GUIs on top of SPREAD

The way each of these groups interacts with SPREAD is discussed below:

3.1 Using an existing GUI

The only reasonably complete GUI to be written so far is the "Virtual Robots" simulator that we use at Essex although a start has been made on an X version. Virtual Robots is described in more detail below.

There are three tasks that the GUI has to perform:

First the initial conditions for a simulator run have to be set up. This includes placing all the static objects into the environment, including boundary walls, obstacles, detectable tracks on the floor and so on. Active targets such as beacons need to be placed and have their characteristics defined such as aperture angles, ranges and identification numbers. Robots need to be "built" by placing sensors and actuators and deciding their characteristics. Finally the initial position and orientation of the robots need to be decided.

The second task of the GUI is to start up the display processes. There can be as many of these as desired and they can be placed on any of the participating workstations. Virtual Robots has only one sort of display that is described below.

Finally provision must be made for controlling the simulation itself. There can only be one of these and in Virtual Robots it comprises a simple panel with "START", "Stop", "Reset" and "Exit" buttons. In addition it has a slider that sets the amount of real time that each "tick" of simulated time represents.

3.2 Writing new robot controllers

Assuming that a simulated robot has been set up that matches the real robot under test the SPREAD API imposes very few changes on the C or C++ source code in which controllers are normally written. Programmers must provide a parameterless function that reads from memory mapped sensor locations and writes to similarly mapped actuator "registers". Code must be "bracketed" by initialization calls that subscribe the controller program to the simulation and cleanup code that is called on simulation exit.. Assignments to mapped memory have to replaced by the "peek" and "poke" functions of the API.

A final optional job is to add a simple polygon definition to a textual configuration file that enables the simulation GUI to display the new robot in a distinctive way.

3.3 Writing simulations of new hardware

The main task here is to write a program that has three main functions:

The first registers the program with the simulation, retrieves parameters from a text based configuration file including the location of its memory-mapped registers, and retrieves a copy of a list of all the static objects in the world and the current length of the time quantum.

The second is called when the simulator terminates and allows the program to perform any cleaning up such as removing temporary files.

The third implements the model itself and is called at the start of each time quantum. At each call the program must retrieve the new locations of all mobile objects and use these together with the list of static objects to calculate its output values. For sensors this is straightforward; it just has to "poke" them into the relevant locations and send an "end quantum" notification back to the world server.

For actuators, the output value represents the new position that the controlled object would occupy in the absence of any mobile obstacles. This, may lead to anomalies if some other actuator process has placed another object in the same position. In the real world the objects would have collided and rebounded in some way. At present the provisional positions are sent to a collision server that resolves the anomaly and returns the resultant position back to the actuator process for use in the next time cycle.

3.4 Writing new GUIs

SPREAD makes this easy by providing many functions to register with and leave the simulation, to register as a member and to leave "process groups" such as the display group, to send and wait for architecture independent messages, to start, pause, and reset a simulation run, to terminate properly the engine itself and all the processes on participating machines, to read and write configuration files and so on.

4. THE VIRTUAL ROBOTS INTERFACE

SPREAD itself only provides a machine independent computational engine. To be useful as a teaching tool an interactive graphical user interface must be provided.

4.1 The configuration file editor

The purpose of this is to set up the initial conditions for a simulation run. It uses the "drag and drop" paradigm to place and parameterise robots and active targets. When complete it will also allow the placing of all object types such as obstacles and followable tracks which at present have to be entered into the configuration files by hand. We also intend to take a similar approach to the

"building" of virtual robots so that a kit of robots parts can be assembled interactively.

Fig. 4-1 shows the editor in action. The smaller polygon represents an obstacle and the larger one represents the a track that can be detected and followed. The icon at bottom left has been used to add several robots by dragging and dropping in the desired position. When this is done the user is asked to supply the path to that robot's configuration file. Once placed the arrow representing the robot's initial heading can be dragged to point in the desired direction.

Active targets can be placed similarly by using the "lighthouse" icon. Double clicking on the placed target brings up the "Goal Parameters" panel shown in the illustration. This allows the identification code, aperture angle and range to be selected. These are also shown in the view window although this is not apparent in the illustration. Clicking on any object also causes its current settings to be shown at the bottom of the window. Also not shown is a menu to load and save configuration files.

Robots and targets can be removed simply by dragging them out of the editor window. Two methods of tracing the robots' routes are given: a trail of dots or, as in Fig 4-2 a trace of the robot's bounding polygon. The particular robot shown was equipped with eight ultrasonic range finders. These only give the distance to the closest sensed object within the sensing angle. This is shown by drawing both the circle segment representing the area that can be sensed by each sensor and the arc representing the possible positions of a sensed object. This can most clearly be seen as the robot in the picture senses the obstacle on its left hand side. An alternative method of showing the possibilities of accumulating the range data is provided where a trace is left of the arcs where an object may possibly be. This has proved invaluable in illustrating methods of building maps from such cumulative and uncertain data such as the popular "occupancy grid" methodology pioneered by Elfes [6] Other sensors' activity can be viewed by similar methods.

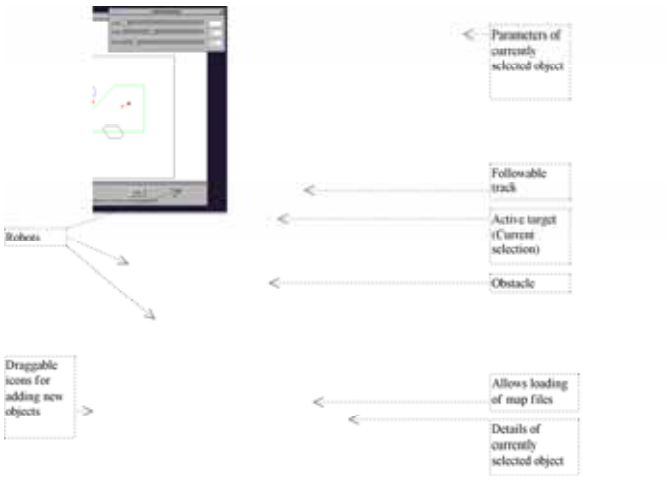

Fig. 4-1 A screen dump showing the interactive configuration file editor

4.2 The Virtual Robots Simulation Viewer

This is the most developed part of Virtual Robots and has many facilities to help users to see the world as the robots see it by making the sensors' fields of view and their outputs visible. Some of these facilities are shown in Fig 4-2 below. The depicted robot has started at the top right of the picture moved in a straight line until detecting and following the track. Near the bottom left it has detected the obstacle and switched from track following behavior to an edge following behavior. The test run subsequently shows an error in the control algorithm where rather than reverting to track following as intended when the track is found again it continues to follow the edges of the obstacle.

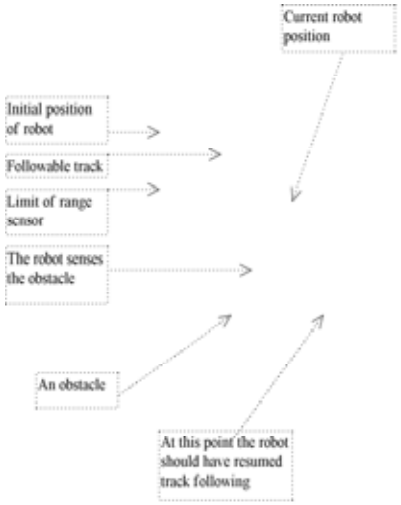

Fig 4-2 A screen dump of a simulator run.

Options are also provide for zooming into particular areas for more detailed observation and for scrolling around the simulation area.

5. PERFORMANCE ANALYSIS

Throughout the development of the SPREAD project we have striven for simplicity, portability, transparency with respect to programming, and ease of use. In particular this has led us to make the simplest of mapping of "objects" to processes. Ideally, to preserve transparency this should be without regard to the placing of processes on machines and the computational and communication demands of each process. Finding practical ways to overcome the performance inefficiencies that arise from this naïve approach is to play an important part in future development of the project. As a first step we have made some tests to establish a benchmark against which to measure future improvements.

The testing that is described below was performed on a typical cluster of 12 '486 DX2 66MHz PC's running NeXTSTEP. The network was a standard Ethernet isolated from the rest of the campus network by a managed bridge. The experiments were run at night when none of the machines were being used. The tests were only of the SPREAD engine and none of the graphical front end was involved.

All tests were run at least 10 times and the figures shown below represent a straightforward mean of all runs. The method was to run a typical simulation with on a varying number of machines and simulating a varying number of robots. All robots were configured identically and were running the same control program. In each case the simulation was run for a fixed number of time quanta

representing the same amount of simulated time. Performance was measured by the mean (real) time taken per time quantum.

The general trend of the curves met our expectations; times obtained for more machines are generally shorter than for fewer machines. However there are many cases were adding machines actually make the performance worse. It can be seen from

Fig. 5-1 that the best performance is for six machines whereas the worst is for seven. The traffic between controller and memory is the key to this anomaly. In the current implementation no attempt is made to place processes in the best places so it is quite possible that processes that communicate a great deal, might be placed on different machines. In the 6 host case controllers and their associated memory processes happened to be placed on the same physical nodes and the network traffic did not play a prominent role. With one more machine (7 hosts) memories and controllers tended to be placed on differing machines needed the network to communicate, creating an I/O bounded system. The best effective speedup we obtained was for 35 robot simulations which ran about 5 times faster on 12 machines than on one.

We conclude from this that we need to:

- place processes more intelligently - maybe simply placing all processes for a robot together would be sufficient;
- reduce network traffic - use of true multicast will be a great improvement as much information such as updates to the world database could be read simultaneously by many processes;
- reduce network collisions. Switching hubs are now inexpensive. Even at a nominal 10Mb/s with one machine on each switched outlet an order of magnitude increase in total bandwidth. This would also enable gains to be made by delegating collision processing to the colliding actuator processes. At present the collision server is a bottleneck.
- use faster network technology. We will soon be installing fast (100Mb/s) Ethernet in the laboratory with uplinks to FDDI that will eventually link to other laboratories

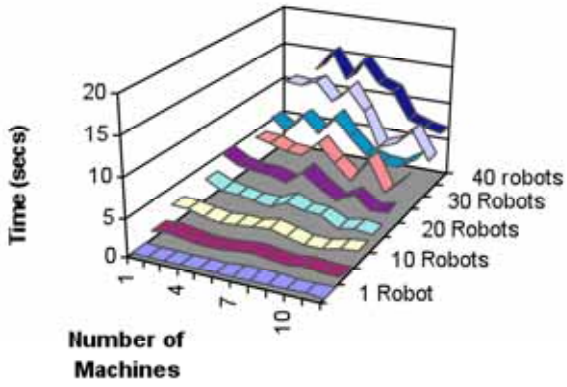

Fig. 5-1 Average time per quantum using different numbers of computers and simulated robots

6. FUTURE WORK

Future versions will allow the definition of processes to be kept together where possible and eventually we hope to allow process to migrate during a simulation run.

Although PVM gives high portability and a simple programming model it has many inefficiencies in its implementation and search for alternatives are a high priority.

Work is underway to make the underlying world model fully 3-dimensional. Virtual Reality type viewers are being constructed to take advantage of this.

We are installing a fiber network that should make the network bandwidth an order of magnitude more favorable.

7. SUMMARY

The high quality of work completed by many students over the years that the simulator has been in use have convinced us that there is much to be gained by making this type of simulator more widely available.

The advantages to be gained by using cluster computing methodologies to improve performance remain tantalising but not yet proven. We certainly feel encouraged by our experience so far.

We are pleased to have received funding for two years of development of the system for commercial use in the offshore industry.

8. BIBLIOGRAPHY

[1] Parallel Virtual Machine (PVM), Oak Ridge National Laboratory, Oak Ridge, Tennessee.

[2] P. van der Smagt (1994) `` Simderella: a robot simulator for neuro-controller design,'' *Neurocomputing* 6 (2), Elsevier Science Publishers, pp. 281-285

[3] Webber A.D. Xmouse. Details at ftp://ftp.essex.ac.uk/pub/robots/Simulators/Xmouse

[4] Douglas C. MacKenzie, Jonathan M. Cameron, and Ronald C. Arkin, "Specification and Execution of Multiagent Missions," July, 1995. Available at http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab

[5] O. Michel, The Khepera Simulator, University of Nice, France. Details can be found at URL http://wwwi3s.unice.fr/~om/khep-sim.html

[6] Elfes, Alberto, "Occupancy grids : a probabilistic framework for robot perception and navigation", 1989. Thesis (Ph.D.), Carnegie Mellon University. Photocopy. Ann Arbor, Mich. : UMI Dissertation Information Service, 1994.