A HYBRID SIMULATION/PHYSICAL ENVIRONMENT FOR BENCHMARKING REAL-TIME DISTRIBUTED CONTROL SYSTEMS

Robert W. Brennan and Karthik Soundararajan

Department of Mechanical and Manufacturing Engineering, University of Calgary, 2500 University Dr. N.W., Calgary, Alberta, CANADA, T2N 1N4

Abstract: In this paper we propose a hybrid physical/simulation environment for benchmarking real-time distributed control systems. This environment uses Arena® Real Time for simulation and the Java-based TINI platform for real-time control. Experimental results show that the characteristics of the physical device can be accounted for through the use of this hybrid environment.

Keywords: simulation, real-time control, intelligent control.

1. INTRODUCTION

Given the difficulty of practical manufacturing scheduling and control problems, recent research has moved away from traditional, analytical approaches that have been the domain of operations research for many years and towards new approaches that rely on artificial intelligence, holonic and multi-agent systems (Shen et al., 2001; McFarlane and Bussman, 2000). In order to make this research relevant however, it is important that realistic and industrially relevant test cases are available to address specifically the evaluation and stress of the performance of scheduling and control systems based on these new technological paradigms. As well, it is important that these test cases span the realm of research in this area from planning and scheduling systems to real-time control.

In order to address this need, a special interest group on benchmarks of multi-agent systems was established under the umbrella of the Network of Excellence on Intelligent Manufacturing Systems (IMS-NoE, 2004). This paper focuses on one aspect of the work being performed at the University of Calgary in this area.

In this paper, we describe a hybrid physical/simulated environment that is currently being developed for manufacturing systems control experimentation that incorporates both simulated and physical manufacturing devices. The objective of this work is to extend benchmarks of multi-agent systems to the full manufacturing hierarchy: i.e., from device control to planning and scheduling. In order to accomplish this, an important aspect of the project involves developing an interface between the simulation software and the physical devices. In our work, we are currently investigation the use a Tiny Internet Interface (TINI) board (Loomis, 2001) that runs Java programs (allowing us to develop local software), and has access various I/O (e.g., discrete/analog I/O, Ethernet). The link to the discrete-event simulation model will be created using Arena® Real Time (Kelton et al., 1998) and Java socket-based communication.

We begin the paper with an overview of this hybrid simulation/physical environment in Section 2 and provide details of its implementation in Arena® and Java in Section 3. In Section 4 we describe our preliminary experiments with the hybrid environment and a benchmark test case. Finally, we conclude the paper with comments on the future direction of this research.

2. A HYBRID SIMULATION/PHYSICAL ENVIRONMENT

In the manufacturing domain, discrete-event simulation is a very powerful tool that can be used to evaluate alternative control policies. For example, discrete-event simulation has been used to evaluate agent-based scheduling approaches by interfacing agent-based or object-oriented software with a discrete-event simulation model of a plant to be controlled as is illustrated in Figure 1 (Brennan and O, 2004).

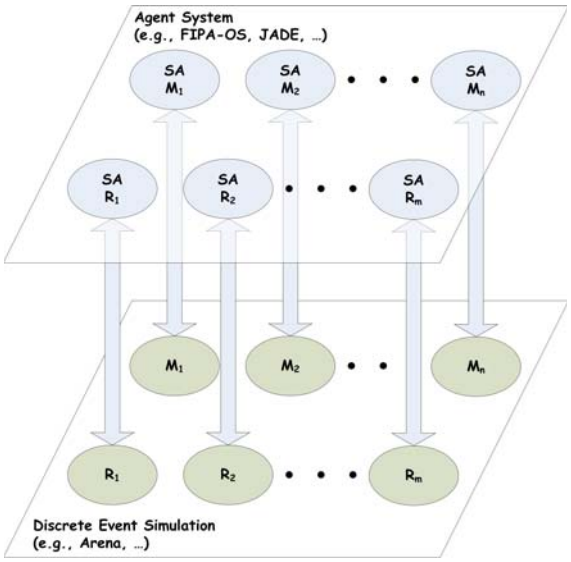

Fig. 1. Using discrete-event simulation to evaluate alternative agent-based control policies.

In this example, each entity in the discrete-event simulation model (such as a machine (e.g., M_1) or robot (e.g., M_1)) is represented by a corresponding software agent (SA) in the control module. For example, the software agents can be thought of as the "reasoning" part of the entity that is responsible for scheduling etc.

The reason for this direct correspondence between SA and entity is that (given recent advances in hardware and software) is it possible to have intelligent agent software running directly on a machine (e.g., computer numerical control (CNC), robot, etc.). This software can be thought of as the "brains" of the machine that can potentially allow it to act autonomously and/or cooperatively.

Much of the work that has been done on benchmarking multi-agent systems has followed this basic model (Terzi et al., 2004). For example, the Remote Factory project (Cavalieri et al., 2000; Cavalieri et al., 2002) uses an online emulation of the manufacturing plant as part of its Test Bench Assistant.

The use of simulation to emulate the plant behaviour is the most common approach for planning and scheduling problems. However, if one is interested in the real-time behaviour of the manufacturing system (i.e., at the device level), the next step is to have these SA's run directly on the machines. This will allow us to test the real-time capabilities of the system (i.e., its ability to meet deadlines), and also allow us to incorporate extra functionality concerned with "execution". For example, SA's (running directly on a machine) may be used to perform fault diagnosis and preliminary recovery services. SA's may also be capable of reconfiguration (e.g., allowing new hardware to be added/removed/modified dynamically). This arrangement is shown in Figure 2.

From an experimental and research point of view, there are some problems with this second approach even though it represents the ultimate goal (i.e., industrial implementation) of the agent-based system. The main problem is that, given financial and space resources, most researchers using this approach are limited to experimenting with relatively small systems. As well, even if a large system is possible, it is debatable whether it would be a good time and cost investment. For example, we may only need a relatively small number of physical devices to test and validate the execution capabilities of our agent system. However, we would like to have a reasonable number of emulated machines to test the scheduling capabilities.

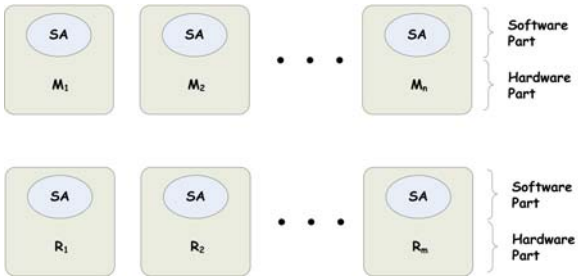

Fig. 2. Physical agents.

The former requirement (execution) is typically hard real-time (i.e., deadlines must always be met very quickly), while the latter requirement (planning, scheduling, and dispatching) is typically soft real-time (i.e., deadlines must be met on average and much more slowly). As a result, we need physical hardware to test the execution environment and could use a simulated environment to test the "higher reasoning" part of the system. Of course, physical hardware could also be used to test this latter part of the system.

A second problem (from a research perspective) with a pure physical system is that we lose many of the experimental benefits of discrete-event simulation software (e.g., statistical analysis, graphics, the ability to easily change the system configuration).

As a result, we suggest that a hybrid physical/simulated environment is developed for manufacturing systems control experimentation. In

order to accomplish this, one aspect of the project involves developing an interface between the simulation software and the physical devices. One approach is to use a Tiny Internet Interface (TINI) board. This board runs Java programs (allowing us to develop local SA's), has access to discrete/analogue I/O, and has Ethernet capabilities. The general idea is illustrated in Figure 3.

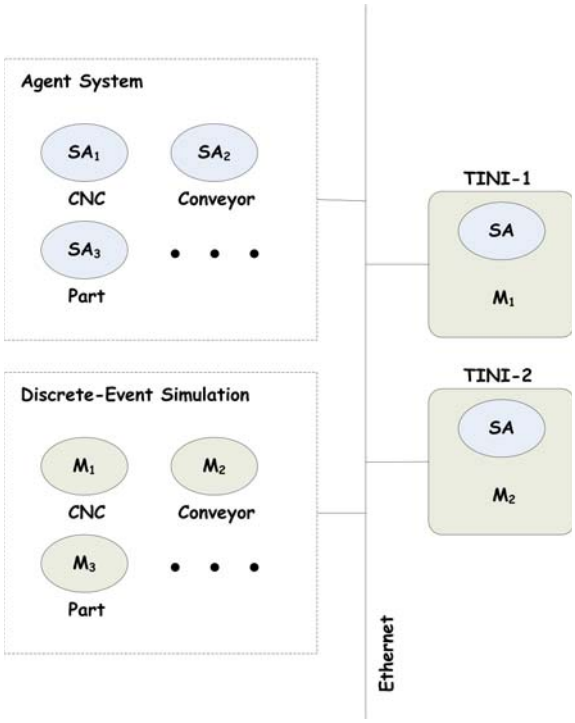

Fig. 3. A hybrid simulation/physical environment for experimentation.

In this example, M_1, M_2, and P_i have software agents associated with them (SA_1, SA_2, and SA_i respectively). M_3 and M_4 are also part of the system, but they are real physical devices. For example, the TINI boards could act as controllers for robots or conveyors. Their agents (SA_3 and SA_4 respectively) communicate with the other parts of the system via a UNIX socket.

3. IMPLEMENTING THE HYBRID ENVIRONMENT WITH ARENA

In this section we begin with a brief overview of the simulation software used for the hybrid environment. This software is well suited to the needs of this research because of its ability to operate in a real-time mode, allowing the simulation to synchronise with the physical parts of the system. Next, we describe the mode of communication used between this simulation software and our physical devices.

3.1 Arena Real-time.

Arena Real Time (RT) is the real-time version of the Arena® suite of discrete-event simulation software environments (Kelton et al., 1998). Although the Arena RT version of the software is required for full real-time support, all of the real-time features are provided in the Standard Edition of Arena®. The only limitation is that Arena RT only allows simulation models to be run for a limited time (10 minutes).

Arena RT runs the simulation model in execution mode. When in this mode, Arena can coordinate simulation logic with an external process of a real system. The external processes and Arena communicate via a messaging system, whereby entities in the Arena model send messages to the external applications to indicate simulated tasks, and the external applications send "message responses" back to Arena to indicate the tasks have been completed. Unsolicited messages can also be sent to Arena to indicate special events (e.g., the arrival of raw material or customer orders). The Arena simulation clock speed is set to the real-time clock speed of the computers operating system.

The Arena RT extension is used in this research to coordinate a benchmark simulation model with the process of the TINI board described previously. This setup allows us to assess the hybrid model and the control software running on the TINI Board.

Two alternative programming approaches for implementing inter-process communication can be used with Arena RT: Visual Basic for Applications (VBA) events, or routines provided in SIMAN's C++ user-code library. In each case, the following basic functionality can be accessed from within the simulation model:

- Initialization – This is called at the beginning of the first simulation replication. This code initializes the inter-process communications (e.g., opening a communication socket).
- Message Transmission – This is called when a simulation entity tries to send a message to the external process. Code that sends the message to the inter-process communication (IPC) queue is placed here.
- Message Receipt - Code that retrieves messages from the IPC queue and passes them to Arena is placed here.
- Termination – This is called at the end of the last simulation replication. Code that terminates the inter-process communications is placed here (e.g., closing a communication socket).

In order to implement inter-process communication in Arena RT, the appropriate VBA code or C++ code (in the form of a DLL) must be linked to the Arena model. Typically, the Arena model acts as a server and the external processes act as clients.

3.2 Client-server communication.

In this section we provides an overview of the client-side communication program that was developed in

Java to allow communication between the Arena simulation model and the physical devices illustrated in Figure 3.

The TCP/IP protocol suite is used in this case to implement the basic network communication. As well, as noted previously, the mode of communication between the distributed applications used is the client/server mode. In this mode of communication, a client program initiates communication while the server program waits passively for and then responds to clients that contact it. As a result, the client program needs to know the server's address and port initially, but not visa versa. For this application, Arena uses a C++ program to establish Arena as a TCP/IP server. The server's job is to set up a communication endpoint and passively wait for connections from clients. In this case, the TCP server first constructs a socket instance at a specified port (this socket listens for incoming connections to the specified port). Next, the server repeatedly calls "accept" methods to get the next incoming client connection. When a client connection is made, an instance of the socket for the new connection is created and returned by an "accept" method. The server can then communicate with the client using the returned socket's input stream and output stream.

The client-side program can be implemented in any programming language that allows TCP/IP sockets to be created (Java is used here to allow the TINI boards to be implemented). The TCP client first constructs a socket instance to establish a TCP connection to a specified remote host and port. It then communicates using the socket's I/O streams as in the case of the server.

In order to test the hybrid environment proposed previously, an initial test was performed using a benchmark simulation model (described in the next section) and a single TINI board as shown in Figure 4.

In this case, the Java client program is running on the TINI board (m197.enme.ucalgary.ca). Once the TINI client connects, the Arena server responds with a "hello" message and part arrival information. For example, the Arena server informs the TINI board that a Type 3 part has arrived at time = 0 (TNOW = 0) with a part ID of 4.

The TINI board then "processes" the part. For our initial experiments, the TINI board generates a random delay time (using a triangular distribution in this example). In future versions, the TINI board's I/O will be used to allow real physical control.

As can be seen in Figure 4, Part #4 is delayed by 4.73 seconds. The TINI board then responds with a "part processed" message. In this case, the message "0 4 0" is sent to the Arena server indicating that a "response to task" (i.e., the first "0") message is being sent concerning Part #4 (i.e., the "4") and the return code is "0" (i.e., the second "0"). The Arena server responds by indicating that this part is being sent to the next machine.

4. EXPERIMENTS WITH A MANUFACTURING BENCHMARK

In this section, we report on our initial experiments with this approach implemented with a benchmark simulation model. We begin with a brief description of the benchmark test case then report on our preliminary results with the hybrid environment.

4.1 The manufacturing benchmark.

The experiments reported in this section are based upon the benchmark system proposed by Cavalieri et al. (2000). This test environment contains four types of resources, each having non-overlapping capabilities, with the third resource type as the bottleneck. The benchmark demands two of each type of resource, or eight in total. Transportation between resources is accomplished though autonomous ground vehicles (AGV's), which for simplicity, are assumed to be always available and require zero transportation time. It should be noted that this transportation assumption, though included in the benchmark, is not strictly necessary. Any transportation activities can themselves be considered resources, and included accordingly in the approach presented in this paper. As noted previously, one of the resources is represented by a physical device (i.e., a TINI board) as is illustrated in Figure 4.

4.2 Preliminary experiments.

For the experiments reported in this paper, the Scenario 1 (deterministic processing times) and Scenario 2 (stochastic processing times) experiments suggested by Cavalieri et al. (2000) were run. However, our main interest for these initial experiments is to evaluate the latencies associated with a physical device rather than the performance of the manufacturing system itself. As a result, only the deterministic processing time results (Scenario 1) are reported here. Further details on our implementation of the agent-based scheduling system for this test case can be found in (Brennan and O, 2004).

In order to evaluate the latencies associated with the physical device, the client application was first run on the local PC with the Arena server and then on a remote device (i.e., a TINI board). The results for a type 1 part (processing time = 8 seconds) are shown in Figures 5 and 6.

Each figure shows the results of 10 simulation runs. The error bars provide an indication of the latencies associated with processing the part outside of the simulation environment.

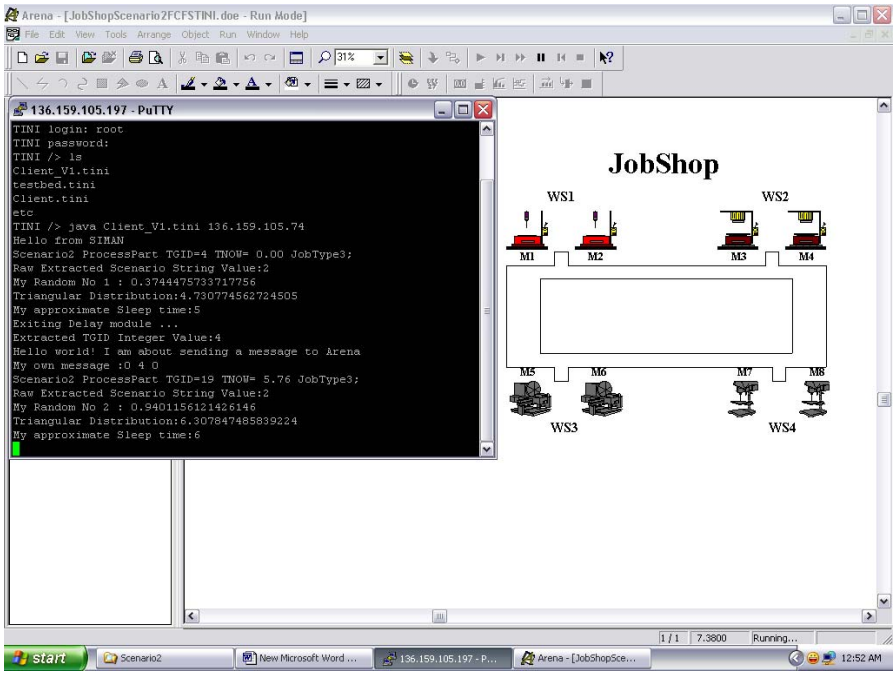

Fig. 4. Java client connection from the TINI board.

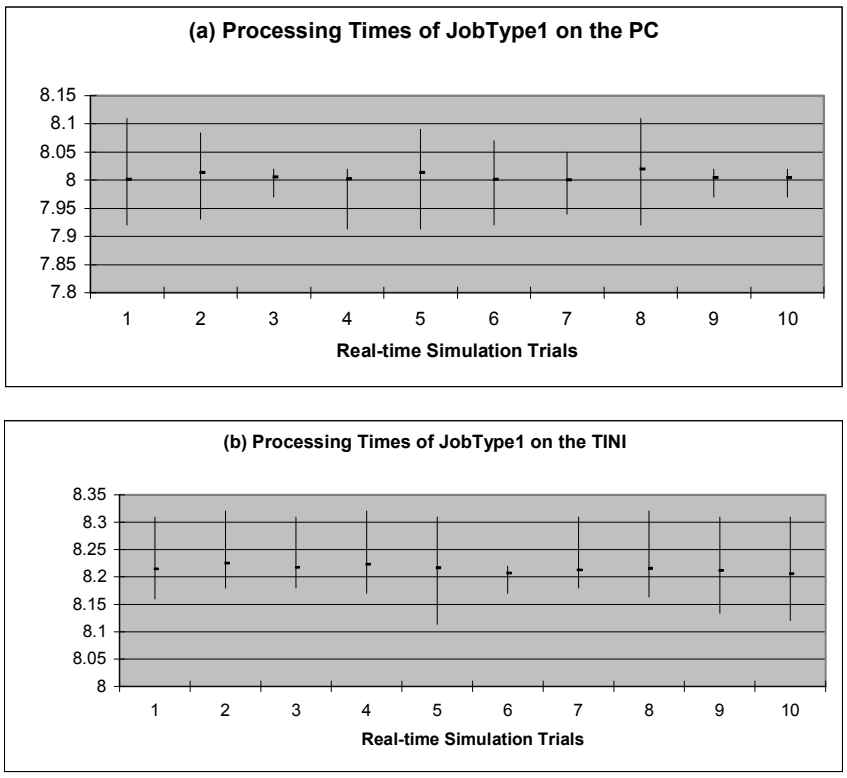

Fig. 5. Processing time results for a type 1 part on the local PC.

For example, the maximum latency on the PC is 0.11 seconds (trials 1 and 8), while it is 0.32 seconds on the TINI (trials 2, 4 and 8).

As well, the maximum latency for the PC falls below 0.10 seconds for 8 out of 10 simulation trials with the PC, while it is above 0.30 seconds for 9 out of 10 trials with the TINI board.

It is interesting to note that processing time falls below the deterministic value of 8 seconds in all cases on the PC (i.e., the minimum processing time is 7.91 seconds for trials 4 and 5). This result provides an indication of the resolution that is possible using Arena's statistics collection facilities for this type of timing test. Despite this small source of error, the results in Figure 6 clearly show that the latencies associated with the TINI platform are higher than those associated with a local client application (e.g., the TINI board's minimum processing times never fall below 8.10 seconds).

5. CONCLUSIONS

The latency results described in the previous section are very promising in the sense that they clearly show that characteristics of the physical device can be accounted for in the hybrid simulation. In other words, by incorporating actual physical devices in the simulation environment, effects that are often difficult to quantify such as communication latency and processor latency can be observed.

This is particularly useful given that real-time embedded platforms such as the TINI board are typically resource constrained. For example, the main drawbacks of the TINI board implementation are its limited memory and its execution speed. Java class files are stored in non-volatile SRAM, which supports a maximum of 1 MB on current versions of the TINI platform. Both tasks and threads are scheduled in a round-robin fashion on by the TINI operating system: the task scheduler is launched every 8 milliseconds and the thread scheduler is launched every 2 milliseconds (Loomis, 2001).

This relatively slow execution speed is the most likely cause of the latencies observed in the Figure 5 TINI board results. Some of the delay may be a result of communication network latencies, however given that the PC results show virtually no delay, the effect is very negligible.

However, given that many the new technological paradigms for manufacturing scheduling and control rely heavily on distributed artificial intelligence it follows that communication latencies have the potential to become a very important consideration in many cases. As more physical devices are connected and the network becomes more heavily loaded, this will be an issue. As a result, one of the main motivations for this hybrid approach is to be able to observe these physical characteristics of the system without resorting to a full-blown physical testbed.

As noted previously, a second objective of this research is to be able to extend the work on benchmarks to the lower, real-time control level. Our current work on the hybrid environment is concerned with this aspect of the research. In particular, we are currently implementing physical agents or "holons" (McFarlane and Bussman, 2000) on the TINI platform that will interact with the hybrid environment in order to provide intelligence at the physical device level. These physical agents will be implemented using IEC 61499 function blocks (IEC, 2000) as described in (Olsen et al., 2004). This will allow our work on manufacturing scheduling and control (Brennan and O, 2004) to be extended to include real-time aspects.

REFERENCES

Brennan, R.W., and W. O (2004) Performance analysis of a multi-agent scheduling and control system for manufacturing, *Production Planning and Control*, 15(2), pp. 225-235.

Cavalieri, S., M. Macchi, S. Terzi (2002) Benchmarking manufacturing control systems: development issues for the performance measurement system, *Proceedings of the IFIP Performance Measurement Workshop,* Hannover, Germany.

Cavalieri, S., M. Garetti, M. Macchi, and M. Taisch (2000) An experimental benchmarking of two multi-agent architectures for production scheduling and control, *Computers in Industry*, 43(2), pp. 139-152.

Dilts, D.M., N.P. Boyd and H.H. Whorms (1991) The evolution of control architectures for automated manufacturing systems, *Journal of Manufacturing Systems*, 10(1), pp. 79-93.

IEC TC65/WG6 (2000) Voting Draft – Publicly Available Specification - Function Blocks for Industrial Process-measurement and Control Systems, Part 1-Architecture, International Electrotechnical Commission.

Intelligent Manufacturing Systems (2004) *Network of Excellence in Intelligent Manufacturing*, http://www.ims.org/projects/outline/noe.html.

Kelton, W., R. Sadowski, and D. Sadowski (1998) *Simulation with Arena*, McGraw-Hill.

Loomis, D. (2001) *The TINI Specification and Developer's Guide*, Pearson.

McFarlane, D.C., and S. Bussmann, (2000) Developments in holonic production planning and control, Production Planning and Control, 11(6), pp. 522-536.

Olsen, S., J. Scarlett, R.W. Brennan, and D.H. Norrie (2004) Contingencies-based reconfiguration of holonic control devices, *6th BASYS*, Vienna, Austria.

Terzi, S., S. Cavalieri, R. Bandinelli, and M. Tucci (2004) Experiences in distributed simulation for bechmarking production scheduling systems, *Proceedings of the IMS International Forum*, Cernobbio, Italy, pp. 1017-1026.

Shen, W., D.H. Norrie, and J-P. Barthes (2001). *Multi-agent Systems for Concurrent Intelligent Design and Manufacturing*, Taylor & Francis.