# 11 DESIGN PATTERNS FOR DISTRIBUTED CONTROL SYSTEM BENCHMARKING

Karthik Soundararajan and Robert W. Brennan

*Schulich School of Engineering, University of Calgary*
*2500 University Dr. N.W., Calgary, AB, Canada, T2N 1N4*
*Email: rbrennan@ucalgary.ca*

*In this paper, we describe the design and development of a simulation-agent interface for real-time distributed control system benchmarking. This work is motivated by the need to test the feasibility of extending agent-based systems to the physical device level in manufacturing and other industrial automation systems. Our work focuses on the development of hybrid physical/simulation environment that can be used to perform tests at both the physical device level, as well as the planning and scheduling level of control. As part of this work, we have extended the proxy design pattern for this application. This paper focuses on the resulting software design pattern for distributed control system benchmarking and provides examples of its use in our hybrid physical/simulation environment.*

## 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

*Please use the following format when citing this chapter:*

Soundararajan, K., Brennan, R. W., 2006, in IFIP International Federation for Information Processing, Volume 220, Information Technology for Balanced Manufacturing Systems, ed. Shen, W., (Boston: Springer), pp. 99–108.

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 investigate 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 is created using Arena® Real Time (Kelton et al., 1998) and Java socket-based communication. This paper focuses on the design and development of the Simulation-Agent interface for the hybrid distributed control system using design patterns.

We begin the paper with some background on the work on connecting simulation software with physical devices. Next, we introduce the notion of design patterns in Section 3 and follow this with details on the use of the proxy design pattern for the hybrid system in Section 4. In Section 5 we focus on the development of the Client-proxy and tests. Finally, we conclude the paper with comments on the future direction of this research.

## 2. THE HYBRID PHYSICAL/SIMULATION 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).

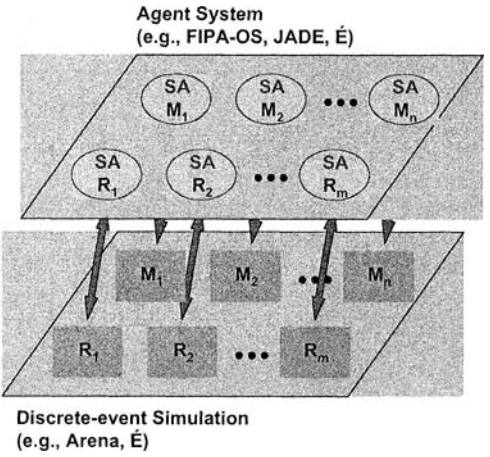


Figure 1 – Using discrete-event simulation to evaluate agent-based manufacturing systems

In this example, each entity in the discrete-event simulation model (such as a machine (e.g., $M_1$) or robot (e.g., $R_1$)) 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.

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.

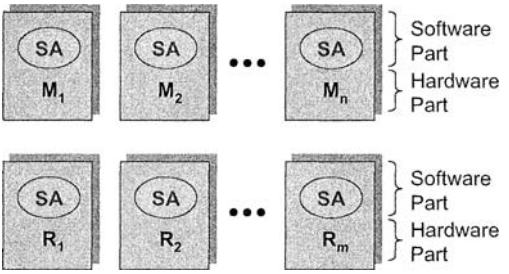


Figure 2 – Physical agents

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.

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.

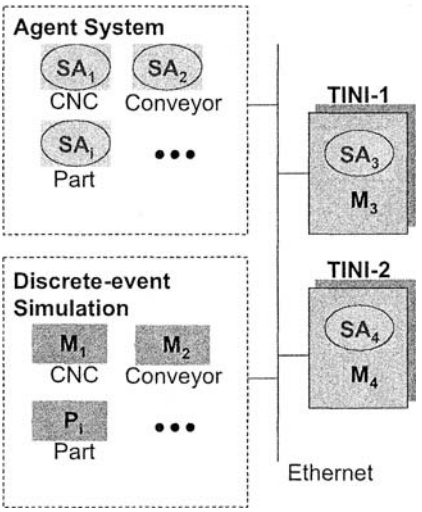


Figure 3 – A hybrid environment for experimentation

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. DESIGN PATTERNS

The hybrid prototype used for this research is comprised of Arena simulation software and a Java-based embedded controller (Soundararajan and Brennan, 2005). An important issue that was addressed during the application development was the transparency of the potential remoteness of the server (the Arena model) and the hiding and encapsulation of the means by which to contact such remote servers by the client. As a result, a design pattern that addresses this large-scale system design (the high-level architecture of the distributed system) concern was investigated.

The first step involved familiarization with the patterns literature related to distributed real-time systems. This was followed by the identification of the design

pattern that addressed the distributed system design. Applying pattern matching for this architectural design issue led to the discovery of the "Proxy" design pattern for the hybrid system.

The proxy pattern allows clients to be decoupled from their server by providing a local proxy. The proxy is a local stand-in that allows clients to access the remote server. In this way, all clients of the server request information and services from the proxy. The proxy encapsulates the information necessary to obtain data and services from the actual server, and when a client requires information from the server, the proxy marshals this request to the server (Douglass, 1999). This design pattern is one of the distribution patterns. A common situation in which the proxy pattern is applicable is where "a remote proxy provides a local representative for an object in a different address space". The pattern allows us to vary how an object is accessed and its location (Gamma et al., 1995).

This design pattern was applied discriminately for the higher level architectural design and the structural elements of the hybrid system were organized to be consistent with the pattern. The following section describes the use of the proxy design pattern in the development of the FB interface.

## 4. THE PROXY PATTERN

The hybrid system is comprised of the Arena real-time discrete event simulation model and the function block application embedded in the TINI board. This real-time embedded system is distributed between the PC where the server is located and the embedded function block application on the embedded controller that forms the client. The design problem to be addressed here is to structure the distributed client-server communication.

The simulation model is devised using Arena and the real-time features are modeled in the system using specific function blocks (FB) for routing information to the external controller. The server uses socket abstraction over TCP/IP to talk to the client. In essence the entire simulation model is the server as it knows the whole of the system: i.e. the simulation model holds the information about the job entities, process etc, as well it has a database with its event calendar to track the workings of the model and records statistics. For a system comprising of one embedded controller the structuring required on the server side is the development of the simulation model and the verification and validation of the model.

Regarding the communication aspect on the server side, for the current setup of having one real-time controller linked to the simulation, the available Arena dynamically linked library file is well suited and need not be modified. When more real-time controllers are to be interfaced then the server proxy is required to have multi-threading features to deal with the multiple threads associated with the different real-time controllers. The server side of the hybrid model conforms to the server side structural elements of the proxy design pattern as described.

Therefore, for the hybrid system, it is the client side of the setup that warrants immediate use of the proxy design pattern. In this setup it is known that the server would be located in a remote address space. However, the location of the server though now at a static IP address could be relocated. As well the clients maybe deployed in an environment much different from that of the server's, such is the case

in the hybrid model. The server is the Arena simulation model along with the server proxy and client applications are modeled as function block applications embedded in a Java-based embedded controller. The clients can be redeployed across different environments as well.

If the clients are intimately aware of the design details in such a setup then there would be problems when they are redeployed. By using the proxy design pattern we would be hiding and encapsulating the transparency of the remoteness of the server taking into account the client specific embedded controller environments. Along with this, using Java based embedded controllers is beneficial as Java based applications can be platform independent. Thus choosing Java based embedded control is beneficial when accommodating the redeployment of the clients. As long as the controller supports a java virtual machine with an Application Program Interface (API) then we could embed the developed client applications in them.

The proxy pattern adds a proxy between the Abstract Client and the Abstract Server. The pattern has two sides. In the first side, the Client-side Proxies subscribe to the Server-side Proxies, which publish the data under the command of the Concrete Servers. When the Concrete Servers call the send() operation, all the remote Client-side Proxies are notified of the new data (Douglass, 2003).

For the hybrid model a similar setup would serve well for the processing and data exchange involved. Only the deployment would be in a hybrid environment thus only part of the system is a real-time embedded controller. The data exchanged between the Arena model and the client application would be that information specific to jobs to be processed in the embedded controller, i.e., the emulator. The data exchanged is passed in the form of strings to the TINI function block application. In here the local proxy, namely the proxy function block will receive the information. This data/information would be extracted by local SIFBs (Service Interface Function Blocks), to process the information according to the nature of the job.

The hybrid system is designed and implemented conforming to the proxy pattern as illustrated in Figure 4. The pattern implementation in the hybrid system is unique since there is a simulation model and a real-time module. Therefore, the proxy pattern implementation has to be fine tuned to the specific environment. It is not possible to have the class relations in the implementation exactly as specified in the default template of the pattern but nevertheless the implementation does adhere to the proxy pattern specification (Soundararajan and Brennan, 2005).

For the hybrid setup, the pattern does a good job of isolating the subject from knowledge that the server may be remote. The client will be simplified by having the proxy, not having to deal differently with all the local FBs to subscribe data from the server. The proxy also encapsulates the knowledge of how to contact the server and what communication process it is using to talk to the server. Thus if the communication media changes or if the client environment changes the proxy can be changed to suit the needs appropriately. Also if the client environments do not support a Java virtual machine JNI's (Java Native Interfaces) can be built to serve the purpose.

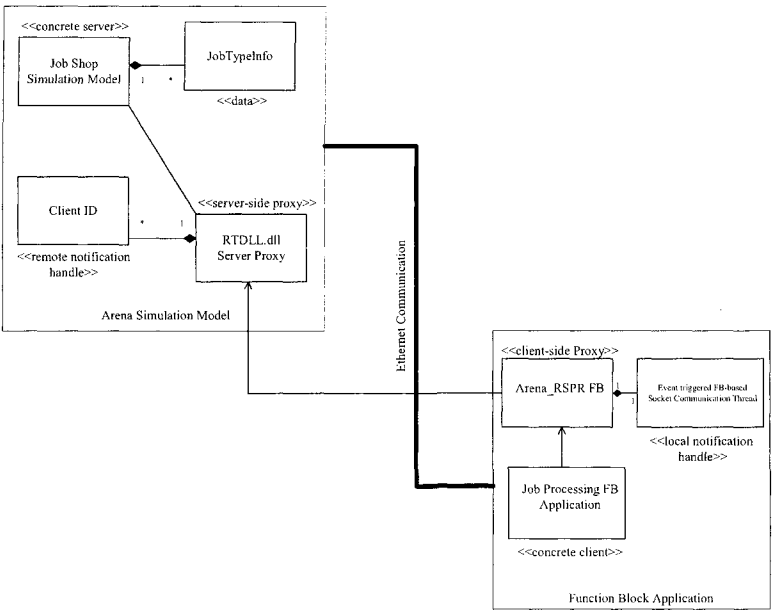


Figure 4 – Hybrid implementation of the proxy pattern

Since there is one client proxy instance than many client FB instances the traffic on the communication media is reduced. This benefit is not fully realized for the hybrid system as of now but when there would be multiple clients, this streamlining of the IPC would be very beneficial. Also, the subscription policy is event triggered due to the event triggered FB application. Although there is a continuous thread, confirm messages are used in the socket stream to request and receive messages from the server.

## 5. DEVELOPMENT OF THE CLIENT PROXY AND TESTS

The first step is to model the required SIFB, named "ARENA_RSPR", in the IEC 61499 (IEC, 2000) context by formulating the properties with appropriate event and data connections allowing for graphical representation and service representation. IEC 61499 specifies in general terms the way in which an interface to an SIFB is defined. A graphical representation of this is given in Figure 5.

The behavior of the SIFB is specified using Time-sequence diagrams. These diagrams help visualize the order in which the various messages or events occur. The Time-sequence diagram for the ARENA_RSPR depicts five transactions:

1. normal_establishment
2. unsuccessful_establishment
3. normal_data_transfer
4. server_initiated_termination

5. client_initiated_termination

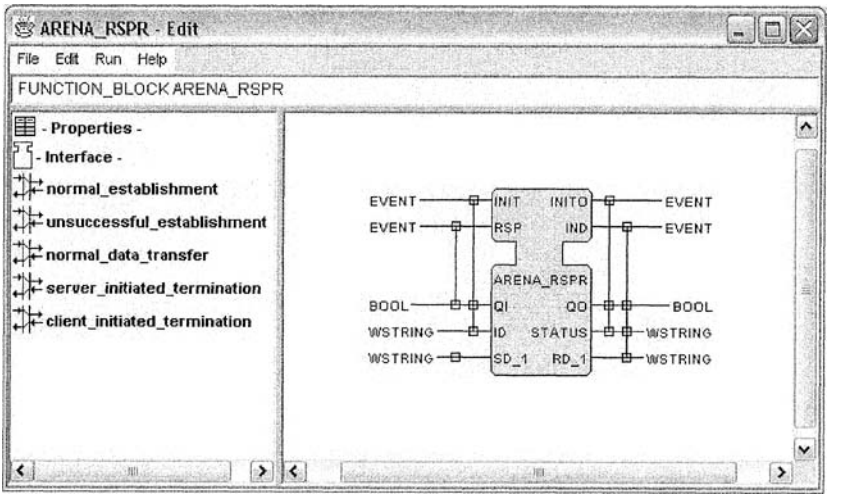


Figure 5 – Graphical representation of the client proxy

In this section, we will focus only on the "normal_data_transfer" sequence as shown in Figure 6, which specifies the data transfer between the server and the client. Arena sends the data to TINI which is acknowledged by the IND event output. When the TINI application has completed the machine execution the RSP event is triggered on the TINI and the confirmation message is received by the Arena model. The service sequence is as follows.

```
SEQUENCE normal_data_transfer
  ARENA.sendData(RD_1) -> TINI.IND+(RD_1);
  TINI.RSP+(SD_1) -> ARENA.receiveData(SD_1);
END_SEQUENCE
```

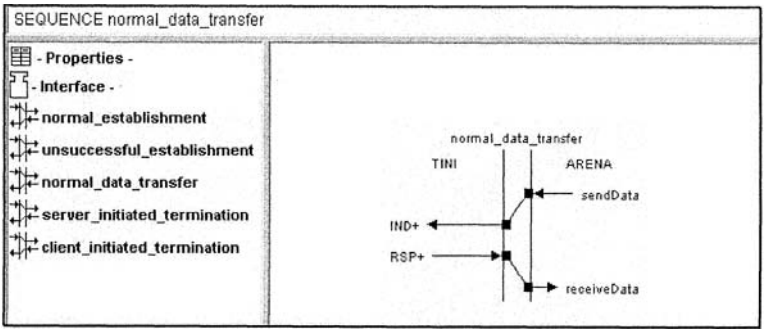


Figure 6 – The normal_data_transfer sequence

The ARENA_RSPR FB was developed as per the above specifications in Java using object-oriented principles. The process involved studying the abstract superclass for communication service interfaces in FBDK (Holobloc, 2006) and inheriting this class to develop the appropriate Arena class.

Figure 7 illustrates the program working with established client-server communication. In this case, the client proxy function block connects to the simulation model at IP Address 136.159.105.74 over port 4334.

The simulation model sends the process part instruction for job with TGID 2 to ARENA_RSPR FB and upon completion of the process instruction a confirmation message "0 2 0" is sent back to Arena. The next process instruction for job with TGID 3 is then sent to ARENA_RSPR and the confirmation message "0 3 0" is sent back to Arena model. The subsequent job to be processed, job with TGID 4 is then sent to the Arena model.

## 6. CONCLUSIONS

In this paper we have provided a summary of our work on the design and development of a simulation-agent interface for real-time distributed control system benchmarking with a specific focus on the design and development of the design pattern used for this interface.

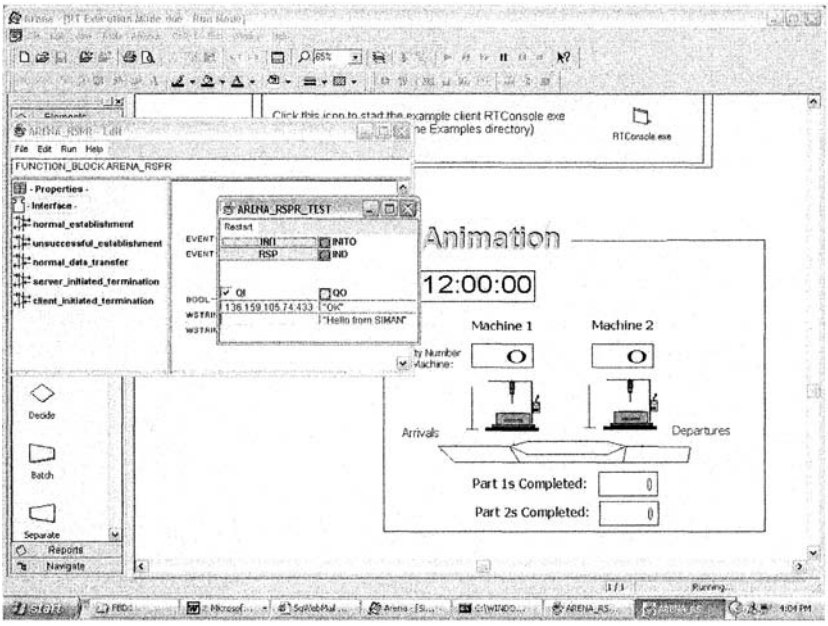


Figure 7 – Communication establishment

In this case, the proxy design pattern has provided a good framework for the large-scale design of the hybrid application. Organizing the structural elements of the hybrid system to the pattern has allowed us to model the client proxy's role and how it should allow for the client application to access the server for information.

## 7. REFERENCES

1. Brennan, R.W., and W. O, "Performance analysis of a multi-agent scheduling and control system for manufacturing", *Production Planning and Control*, 15(2), pp. 225-235, 2004.
2. Douglass, BP. Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Addison-Wesley, 2003.
3. Gamma E, Helm, R, Johnson, R, Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
4. Holobloc, Website, http://www.holobloc.com, 2006.
5. IEC TC65/WG6 Voting Draft – Publicly Available Specification – Function Blocks for Industrial Process-Measurement and Control Systems, Part I, Part II and Part III, International Electrotechnical Commission, 2000.
6. Intelligent Manufacturing Systems, Network of Excellence in Intelligent Manufacturing, http://www.ims.org/projects/outline/noe.html, 2004.
7. Kelton, W, Sadowski, R, Sadowski, D. Simulation with Arena, McGraw-Hill, New York, 1998.
8. Loomis, D. The TINI Specification and Developer's Guide, Pearson, 2001.
9. McFarlane, DC, Bussmann, S. Developments in holonic production planning and control, Production Planning and Control, 2000; 11(6): 522-536.
10. Shen, W, Norrie, DH, Barthes, J. Multi-agent Systems for Concurrent Intelligent Design and Manufacturing, Taylor & Francis, 2001.
11. Soundararajan, K., Brennan, RW. "A proxy design pattern to support real-time distributed control system benchmarking", Proceedings of the 2nd International Conference on Applications of Holonic and Multi-Agent Systems, Copenhagen, Denmark, August 22-24, 2005.