Real-Time Simulation and Control of Large Scale Distributed Discrete Event Systems

Fernando G. Gonzalez*
Texas A&M International University, 5201 University Blvd., Laredo Texas 78041, USA

Abstract

Due to the ever-increasing computational capabilities and the increasing use of automated, distributed large-scale systems, intelligent controllers are being proposed and designed. Much of the discrete-event automated control research has been done in the context of manufacturing plants and as such this paper will present the results in the context of manufacturing systems. There is a large amount of research in the area of simulation of manufacturing systems, scheduling and some research in the control of automated manufacturing systems. Even intelligent controllers for manufacturing systems have been looked at by some. To this date there has been no realization of a general-purpose intelligent controller for manufacturing systems or any other large scale discrete-event system. Furthermore the current research efforts focusing on this problem are not considering the fact that the controller must be physically distributed among many computers in order to achieve the computer power necessary to control these large-scale complex systems. Since today’s large complex systems tend to be physically distributed over potentially thousands of miles, the control structure must be distributed as well. This results in a serious implication; the model used in the controller actually consists of a set of independent models. Very little research has focused on the design of a controller whose model is distributed. Presented in this paper is a controller that is designed to control the real-time discrete-event aspects of a large scale distributed system using manufacturing as a base problem. The paper concludes by presenting an example of a flexible manufacturing system being controlled by this software.

Keywords: Discrete-event systems, intelligent control, simulation, modeling, distributed control

1. Introduction

Today there are many large scale systems such as manufacturing plants, the Smart grid, security systems for example that are highly automated. They are generally large and their control architecture distributed among many computers possibly in different geographical areas. Even the systems contained within a single building may have a distributed control architecture. For example consider a flexible manufacturing system (FMS). These controllers run
independently on different computers. They rely on communication through a network for coordination. Many of these systems continuously run in a highly transient state. Consider such a system designed to produce many different types of parts in very small quantities filling the customer’s request as they arrive.

There are many researchers that refer to their work as the control of manufacturing system and are in fact working on a variety of areas such as scheduling, dead-lock avoidance, simulation, modelling, etc. In this paper the control of a manufacturing plant or any other type of system, is the process of coordinating the resources available along with the process of communicating with the physical machines in order to achieve the physical control of the system. Intelligent control is the process of real-time optimization running concurrently with the physical control of the system.

2. The Problem Being Addressed

The problem addressed in this paper is the real-time distributed intelligent control of large scale discrete-event systems with flexible manufacturing systems (FMS) as the target system. The challenge is to make the controller scale up to handle larger sized and more distributed systems while facilitating the modeling effort needed to provide intelligent decision making capabilities even at the most aggregate levels. In order to achieve this, the controller must make decisions in real time and be distributed. The key word here is distributed. A real time controller with decision making capabilities is relatively known and is well covered in the manufacturing literature. What is not being addressed is such a controller that is distributed among many computers. The issue is the simulation models that are used in the decision making process. If the controller is large and distributed then, given that the simulation model must include all the logic of the controller models, this leave the modeller with two situations. Either a new simulation model needs to be created that includes all the logic in all the controller models or the existing set of models used for control will be used for simulation. Note that the simulation of a system being controlled is the controller of that system running with the hardware replaced by simple software models that represent the hardware. The point is that the control and simulation models are basically the same. Unfortunately because the controller models consists of a collection of models and all the simulation software currently available require a single model, the models used for control cannot be used for simulation. The research presented in this paper is a software simulation algorithm and corresponding tool that can accommodate a collection of simulation models. The idea is to reuse the control models and avoid the extra modeling effort.

The situation is actually worse if you need to create new simulation models. Since the control architecture is hierarchical and since each controller in the hierarchy is to have decision making capabilities, the simulation modeling effort is more than just double. Presented without proof for lack of space, it can be shown that the modeling effort to create a set of simulation models, one for each controller, for a simple 2 level system with one supervisor and 5 subordinate controllers, will be almost 3 times as much than the effort for creating the control models. For a system with 7 levels in the hierarchy the simulation modeling effort is about 50 times more than the controller modeling effort. This is why the current research on intelligent controllers is not scalable. The challenge in creating a simulation algorithm that can accommodate a collection of independent models is in the coordination of these models.

In addition the controller needs to physically control the system. This is achieved by communicating with the hardware. The system being controlled must be able to communicate with the software controller. The controller must speak its language to give the hardware tasks and to receive status information. This is usually done via TCP/IP or internet communications.

3. Related Work

The following is a historical evolution of the work performed to control the discrete-event aspects of a system. Peters et al. (1996) [10], Smith et al. (1994) [11] and Smith and Peters (1998) [12] have adapted Arena, a commercial DES simulation software tool see Kelton, Sadowski and Sadowski (2001) [9] to control their experimental FMS. However, unlike the modeling approach to be discussed in this paper, their control architecture is limited to only one hierarchical level where a single supervisor, the cell controller, manages a set of subordinate processes. In their modification of Arena, they have included special events in order to facilitate message passing among the controllers. Furthermore, to model a complex FMS or other multi-level, distributed system, the set of modeling elements provided by ARENA, as well as most simulation languages, severely constrains the modeling process. This is particularly true when one attempts to assess the impact of the control architecture upon the system.
Davis (1998) [1] details the numerous restrictions that current simulation languages impose upon the modeling of hierarchically distributed systems. Mize et al. (1992) [8] further discusses the inaccuracies that ensue from using current simulation languages in order to model an FMS.

While much work has been done in the planning or scheduling of manufacturing plants, much less work has been done in the physical control of manufacturing plants. The issue of physically distributing the intelligent controller among many computers is not being addressed in the literature. There have been efforts to use the same model for both control and simulation, see Gonzalez (1996) [3] and Smith and Peters (1998) [12]. This gives the simulation high fidelity and makes the state transfer from the control model to the simulation model trivial. However when the fact that the controller needs to be distributed is considered, the other existing methodologies falls apart. This is due to the fact that the controller will use a collection of models while the simulation uses only a single model. With the current research, the models cannot be the same. Consider the additional complication resulting from the need to transfer the state from a collection of models to a single, different, model.

The latest research is to use Programmable Logic Controllers (PLC), which has been used extensively in the control of the individual machines, and apply them to the distributed control of manufacturing systems. In Korotkin et al. (2010) [14] they present a methodology to implement Petri Net logic into PLCs for the control of industrial systems. Petri Net logic is a very popular graphical method used to model discrete-event systems and is used in ARENA and most other simulation languages. Yasuda and Ge (2010) [15] present a method to decompose the Petri Net logical model into smaller pieces for implementation into PLCs with the goal of controlling a distributed manufacturing system. There is a lot of research in this area with the focus of implementing the distributed controller using a collection of PLCs. While this method may work for manufacturing systems is not scalable to larger systems such as the Smart grid because PLC’s are very simple microprocessor boards designed for the physical control of hardware. They are not designed for processing high level logic such as simulations and decision making. Furthermore even if one decides to use PLCs to implement the controller the problems addressed in our work still exist if one is to incorporate intelligence. This research is not considering the big picture but rather only working on the control of a small system albeit distributed systems. They are just concentrating on the control or optimization aspects and not considering the scalability problem.

The controller presented in this paper is based on the following concepts. It is assumed that the controller consists of a collection of controllers distributed across many computers and tied together by the network. Davis et al. (1997) [2], Gonzalez (1996) [3] and Gonzalez (2003) [4] present the architecture for the controller and the single threaded simulation of distributed systems using distributed modelling. Some of the more fundamental algorithms are described in more detail in Gonzalez and Davis (2003) [7]. These fundamental concepts are described in the following section.

4. Fundamental Underlying Concepts

4.1. The Hierarchical Architecture

The concept of the Hierarchical Architecture was originally published in Tirpak et al. (1992) [13]. The architecture introduces the simulation object as the basic element for modelling a system. Each simulation object represents a basic hierarchical element where intelligent control is to be addressed, see Figure 1a. Using this architecture gives rise to multi-resolutional control. At the top most level, the simulation object's scope of control includes the most aggregate level of decision. The planning horizon may be very long. This simulation object makes high level decisions. The controllers in the middle level have a smaller planning horizon but are still making managerial type decision. At the bottom most level, the controllers have the smallest planning horizon and are limited to the most immediate concerns of the hardware they control. The hardware is only controlled by the simulation at the bottom of the hierarchy. It’s called simulation object even though it is used for control because its algorithm is the same as the conventional discrete-event simulation with a few modifications.

4.2. The Distributed Modeling Methodology

This methodology presents a very novel approach to simulating distributed systems using a single thread. This methodology is based upon the belief that the interactions among the controllers must be considered by the
simulation model in order to accurately model a system with a distributed control architecture. The single most important characteristic of the methodology and what separates it from other object-oriented simulation approaches is its attention to modeling the flow of messages among the controllers included within the architecture. By modeling the flow of messages, the methodology allows the simulation to accommodate any number of models. Recall that the controller is distributed. That means that there are many independent controllers each running with its own model. This methodology allows us to simulate the complete system using the collection of models from the controllers. This has many advantages:

1. The simulation is more accurate since its model considers all of the communication among the distributed controllers. This was the original motivation for developing our methodology.
2. The simulation produces maximum fidelity since the same set of models is used for both simulation and control. Model verification is a major effort in modelling usually requiring more effort than to actually build the model. Furthermore there is always the risk that the model does not represent the control logic correctly. Using our method reduces the chances of discrepancies between the models used for simulation and those used for control. This simplifies the verification phase of modelling.
3. The modelling effort is shared between control and simulation. They both share the same models so only one set needs to be created. As mentioned in the problem statement, without our methodology one may think that the modeling effort will double but in reality it’s considerably worse. Since each controller has a different set of subordinate controllers in its control domain, each one will need a model that includes not only its own logic but also the logic of all of the controllers in its control domain. The same logic is being modelled over and over.
4. The control and simulation models necessarily employ the same state definition since they are in fact the same code. This simplifies the task of initializing the simulation model to the current system-state. If the simulation and control models employ a different state definition, which is the case when one employs conventional simulation approaches, then one must translate the measured system variables into values for the state variables employed within the simulation model. This is a relatively large effort. See Gonzalez and Davis (1998) [5].
5. This methodology offers multiresolutional modelling. Since the system models have a hierarchical organization, the simulation resolution can be dynamically selected by selecting the levels in the hierarchy to include models for. For example, in Figure 1a, if the top most controller wants to make a fast but rough decision it may only include the next level down. This increases the simulation speed at a cost of accuracy. If later it wants to run a more accurate simulation with a more detailed model it simply may choose to include all the models. This increases the resolution since the collection of models being used in this simulation includes much more details. The modeling elements are designed such that the removal of any model from the collection does not prevent the rest of the simulation from running. It only results in a less detailed overall model. This allows the software controller to adjust the resolution dynamically as needed.

The following is a brief description of how our simulation methodology is able to accommodate a set of models in its simulation rather a single model. The simulation of each individual subsystem is performed using the commonly used discrete-event simulation method which of course uses only a single model. An executive function manages the simulated time, event list and the list of resources. As it pulls of the next event in the chronologically ordered event list it calls its model (the single model) to execute the event. The simulated time is incremented to the time of the next event in a single discrete jump. All of these components, the event list, the list of resources, current time and the model, are encapsulated into a single object, called the simulation object. See Figure 1. In this way, each simulation object simulates a single subsystem using the standard discrete-event simulation and a single model.

In order to simulate the entire distributed system, the overall simulation model must include an instantiation of simulation object for each modelled subsystem, that is, for each model. A coordinating function for the overall simulation manages the execution of each simulation object. This function resembles an operating system in the sense that it distributes the processor usage among all of the simulation objects. This function allocates the processor to the simulation object that needs to execute the next event in time. It maintains a global event list that has a copy of each event in each of the simulation objects. It also models the network by maintaining a list of all messages sent between the simulation objects. This message queue is also used to determine which simulation object acquires the processor next.
Example 1: The Simulation of a Three Model Systems

In this example the system includes 3 models, see Figure 2a. In the simulation the coordinating function removes the next event, the one occurring in controller number 3, from the global event list. It gives the processor to simulation object number 3. Object 3 executes this event and as a result of its modelling logic it sends a message to object number 2. This message is sent to the message relay that simulates the network. Once object 3 completes its cycle it relinquishes control back to the coordinating function. The coordinating function then removes the next message from the message queue and delivers it to the proper recipient, object 2 in this case. Next object 2 receives this message and executes the appropriate event. In the execution of this event, controller number 2 schedules a new event onto its local event list and a copy is scheduled onto the global event list. At this point, the event-execution cycle of the coordinating function terminates and the function removes the next event from the global event list and begins a new event-execution cycle.

4.3. The Intelligent Controller

The intelligent controller involves the collection of simulation objects operating as controllers using actual internet communication and running in real time. The intelligence of the controller involves running simulations of the distributed system or subsystem concurrently with the controller.

Example 2: The Intelligent Control of a Three Model System

Continuing the previous example, now the 3 controllers are controlling the hardware in real time using real communications, see Figure 2b. Note controller 2 and 3 are directly controlling the hardware while controller 1 is acting as a supervisor / manager. Concurrently while running in real time, controller number 1 is making an intelligent decision and is using the simulation as described in Example 1 to produce feed forward information. It gathers the state information from itself and the other two controllers and uses it to initialize the simulation models. The decision can be related to how it manages the other two controllers. Note in the simulation there are three simulation objects with 3 models that are simply copies of the three objects and models controlling the hardware. So no new modeling effort is required and the state of the controller can be directly transferred into the simulation’s models since it’s an exact copy with no need to interpret the information. Since the controller is a software program the state is simply the contents of the variables in the objects and models. The communication agent functions like the postal system in that it routes all of the messages to their destination. It is needed for the real time collection of the state while the controllers are live. Without it the controller’s operation will need to be interrupted, see Gonzalez and Davis (1998) [5].
5. The Implementation

The merits of this work also involve how the algorithms are implemented. This section gives the reader more details of the concepts and how to implement them although still in the realm of a conceptual paper. The goal for developing this software tool is to provide a simulation approach using C++ that provides the modeling convenience of a conventional simulation language while providing the additional modeling capabilities that are needed to control a real-world distributed system. A set of C++ objects was developed in order to provide the basic necessary modeling elements that are employed in nearly all simulation and control scenarios, irrespective of which modeling methodology is being employed.

In this tool each included modeling element is represented as an object in C++. Two event lists (discussed below) are maintained where the events are stored as they wait to be executed at the proper simulated time. The simulation object,

Figure 1b, is a C++ object that contains the scheduled event list, pending event list, list of resources, the executive function, and pointers to the system model and the communication object that interacts with the hardware. The executive function manages the chronologically ordered scheduled event list and the pending event list. It manages the processing of the events as they are removed from the event lists, schedules new events that are to be placed into either the scheduled or pending event list, and manages the allocation of resources. After each event is processed, the executive function removes the next event with the smallest event time from the event list and then invokes the proper modeling object to manage the event’s execution.

In order to run, the system model must be attached to the simulation object. A communication function must also be provided to the simulation object in order to permit this object to communicate with the hardware when the model is running in a control mode. This function handles all of the communications among the distributed controllers and the various machines. The simulation object can operate in 7 modes however for simplicity the 3 most common modes are

1. Single Simulation mode: In this mode the simulation object is executing as a single simulation with no external models just as in a conventional discrete-event simulation. The system clock advances in discrete increments. Every time an event is pulled off the event list, the time is incremented to the event’s time of occurrence. No hardware is present. The executive function executes in a continuous cycle.
2. Distributed Simulation mode: In this mode the system is still only simulating with no hardware present but it is executing a collection of simulations concurrently to model a distributed system that is a system with a
collection of models. Each model is running as in Single Simulation Mode but only executing one cycle before returning control back to the coordinating function that called it.

3. Control mode: In control mode, hardware is present and the system is executing a single model but communicating with the other controllers via internet messaging. The system clock runs in real time that is, the system clock is a conventional clock, like the one on the wall, where time advances continuously. The events are pulled off the event list when their event time occurs. The pending event list is used to allow the executive function to determine when events occur.

5.1. Distributed Modeling

In order to build the complete simulation model, one simply includes all of the simulation objects into a single program and adds the coordination object. The coordination function within this object coordinates the execution of all of the simulation objects. It also has the global event list and the message relay discussed later. When the distributed model is executed upon a single computer, only one of the simulation objects can be executed at a given time because there is only one computational thread. In order to emulate all of the simulation objects operating concurrently on a single processor, the coordination function executes one simulation object for a short time and then switches to another object. The coordination function uses its global event list and its message relay to determine which simulation object to execute next and for how long to execute it. The individual simulation objects return control back to the coordination function when it is done executing all of the events that occur during that instance of time.

In simulation mode where the simulation object is running independently, the executive function in each simulation object cycles in a loop. This loop starts by checking the event list for the next event. It then executes this event. After executing the event, it checks to see whether there are any new events resulting from the execution of this event. An event may cause a second event to execute by the releasing of resources. Thus, the execution of an event may cause a chain reaction. In control mode, at the end of the current cycle, the executive function checks to see whether any message has arrived from the communication pipelines. If a message has arrived, the pending event list is then checked to find the corresponding event to be executed. In either mode, the executive function then initiates a new event-processing cycle.

In order to permit several simulation objects to operate concurrently while in the distributed simulation mode, the executive function is modified so that it cycles through the loop only once every time it is called. Furthermore, it does not check the communication pipelines at the end of the cycle. All the communication is handled by a message relay contained within the coordination object. The coordinating function within the coordinating object is responsible for calling the executive function of the model that will execute the next event. At this time, the wakened executive function cycles through the event-processing loop once and then returns program control back to the coordinating function. This method of sharing the computing processor operates on a similar principle as many multi-tasking operating systems.

Another special feature of our simulation tool is the manner by which messages enter the model. Instead of checking the communication lines every time the executive function reaches the end of an event processing cycle, the lines are not checked at all. Rather, the coordination function manages the flow of messages. Whenever a message is present, the coordination function calls the executive function of the appropriate simulation object and provides it with the message. As a by-product the system models the flow of messages among the controllers increasing the validity of the overall model. Ironically this was the original goal of this work since neglecting this can lead to inconsistencies between how the simulation assumes messages flow and how they actually flow.

5.2. The Coordination Function

The coordination function is a supervisory function that manages the execution of all of the simulation objects. The coordination object contains a global event list and a message relay. This function runs in a cycle much the same way the simulation objects do when running independently (not as part of a distributed simulation). In the typical simulation mode, the event-processing cycle starts by removing the next event from the global event list and passing it to the appropriate simulation object for execution. While the model is being executed, the message relay, which acts as the network, receives all of the messages that are generated by the model that is currently executing. These messages are stored in the rear of the message queue within the message relay, which is used to model the network. Once the simulation object finishes executing its current event-processing cycle, the coordination function
removes the first message from the front of the message queue and passes it to the recipient simulation object. The object then receives the message and executes the appropriate functions that are needed to handle the message. Additional messages may again be generated and are inserted at the rear of the message queue. Once control is returned to the coordination function, it removes the next message from the front of the message queue and recycles it though the message-processing loop. This procedure is repeated until no messages remain in the message queue. At this time the coordination executive function finishes its cycle and begins the next cycle by removing the next event in the global event list and passing it to the appropriate simulation object. See example 2 above.

The events in the global event list tell which simulation object will address the event and the time at which the event occurs. When an event is pulled off the list, the simulated time is then advanced to the next event’s event time. The event type is not recorded on the global event list, as this information is contained within the simulation object’s local event list and need not be duplicated. The coordination function then calls the appropriate object and passes it the current simulated time. The object, knowing that it is responsible for executing the next event, pulls the next event off its local event list and executes it. As stated above, it only executes its executive function’s event processing cycle up to the point where the communication lines are to be checked. At this point, program control is returned to the coordination function where the function then checks the message relay and continues its cycle.

5.3. The Overall Implementation

When controlling a system each simulation objects is executed by itself on its own computer or own thread if they are sharing a computer. They communicate using the network so there is no need for a coordinating function since the models coordinates themselves. Each simulation object is operating in control mode where the time advance is real time, the communication is via the internet, and the executive function cycles continuously. Simulation of the distributed system is required to support the decision making process. Therefore every controller that will perform decision making needs to have a simulation model that represents the system it’s controlling. This includes its own model as well as the models for the subordinate controllers that are in its control domain. How many levels down the hierarchy are included in the models depends on the desired resolution. Since each controller is in a different location in the hierarchy each controller will perform simulations using a slightly different set of models. However this does not represent any extra modeling effort since in this tool, creating the simulation model simply involves selecting models from the pool of models. The modeling effort is in fact shared between control and simulation.

To simulate the distributed system first an appropriate set of models, or actually the simulation objects that contains the modes, are collected and included into the decision making program along with the coordinating object. The objects are then run in simulation mode and the coordinating function runs the simulation as described above. The message flow that occurs between the independent controllers is in effect represented in our simulation even though no direct modeling of the network appears in any of the models.

Before the simulation begins the current state of the physical system is gathered. The coordination function asks each controller in its control domain to provide its current state. This state is simply the data in the variables and data structures of the object plus a few from the model itself. The coordination function then initializes each object in its simulation by giving it the corresponding state. Since the objects are simply copies of the objects used for control they have exactly the same variables and data structures. Therefore to initialize the models, the contents of the controlling object’s variables and data structures (the state) is copied into the simulation object’s model without any interpretation or translation. It is not necessary to know the meaning of this data, see Gonzalez and Davis (1998) [5]. This is a major advantage since without this method one will have to translate the state information from the control model to the simulation model. This involves understanding the meaning of all of the data in the variables and data structures of the controlling model. This is a very difficult task that is totally eliminated using this method.

5.4. The Decision Making Capabilities

The distributed simulations that have been presented here are a non-deterministic simulation of the complete system, that is, a single simulation run albeit of the whole distributed system. The intelligence in the decision making process involves creating a set of options then selecting the option that produces the best performance. To evaluate the different options the system must run many simulation runs for each of them so that the non-deterministic nature of the simulation explores a large portion of the random outcome space. In other words for each option many simulation runs are performed using the same initial state and statistical data is gathered and averaged.
Once all the simulations are complete the statistical data for each option is compared using a heuristic function that yields the goodness of that option. The best option is then the one that predicts the best performance. The options are generally the way the model handles something in its control domain and usually represents a value in some variable. Before evaluating an option that variable in the model is set to the value that represents that option so that the simulations predict the performance using that particular option.

The code that performs the decision making process is part of the tool and represents a standalone program rather than an object that one includes into one’s own program. The real time controller and the decision making process employing the simulations are generally executed on the same computer although this is not necessary.

6. An Illustrative Example

The following section illustrates an example of a real physical system that is controlled using a distributed model. A simulation using the same set of distributed models for the considered system is also performed. Each of the models for the included controllers was programmed using the simulation tool discussed above. In Gonzalez and Davis (1998) [6], the physical system is discussed in greater detail.

The system is a physical model of a flexible manufacturing system (FMS). The schematic diagram for the FMS emulator is shown in Fig. 3a and a photograph is depicted in Fig. 3b. This model provides a testbed for the development of the simulation and control tool that is needed to manage the system. Note that since it is assumed that the system is a real FMS, it is modeled using a collection of independent models.

![Schematic diagram of the FMS physical model](image1)

Fig. 3. (a) Schematic diagram of the FMS physical model; (b) photograph of the hardware model

The system has four Processing Centers. Each Processing Center (PC) contains one primary processing resource and a dedicated Material Handling System (MHS). Within the emulated FMS, see Fig. 3a, another process is the Fixturing Center (FC). The FC has a dedicated MHS consisting of a primary carousel capable of holding sixteen jobs and two smaller carousels for loading and unloading jobs from the Automatic Guided vehicle (AGV). The movement of these carousels is controlled by a dedicated controller. The FC has two fixtureing positions that represent the subordinate unit processes. The final subordinate process is the cell’s MHS. AGVs are employed as the primary material handlers at the cell level and are modeled with an HO-scale electric train. In this layout, there are over forty track segments that can be individually powered. Sensory switches are provided on each track segment to detect the presence of an AGV.

In constructing this control architecture, 25 independent copies of the simulation tool was employed, each with its own model of the subsystem that it is addressing and all running concurrently, see Figure 1a. The only thing that ties them together into a single-control architecture is the communication among them. The communication is performed across a local area network (LAN) connecting seven computers where the cell controller and each of the cell’s six subordinate controllers are situated on their own computer. Additional communication links are provided via RS-232 links between the FC and PC controllers and their dedicated hardware controller boards.

In this example, the developed model controlled the physical system. The controller was given a total of three jobs, each with two processing steps. Each processing step required the part to be moved to a different machine. In addition to the processing of the part, before and after each processing step the part had to be moved to the fixturing...
center for fixturing. There were four Sun Work stations and six controller boards. The software controllers were distributed among the four workstations. Each controller board controlled the hardware that was attached to it.

The same model was employed to run the simulation in order to project the future performance of the system given its current state. The state of the hardware was used to initialize the state of the simulation models. The message transcript for both the actual run and the simulation run were similar. The only difference was due to the discrepancy between the estimated and actual duration times. No changes had to be made to any of the models in order to switch from using them for control or for simulation.

7. Conclusions

This paper presented an algorithm and an associated software tool that is capable of accepting a collection of models as opposed a single model. It was shown that for a large scale distributed system with 7 levels in the control hierarchy the modeling effort that is saved when using the method presented here represent an effort of about 50 times the effort in developing the models used for control. Reusing the models used for control in the simulations is almost a necessity in creating a controller for large scale distributed systems that has intelligent decision making capabilities. Also shown are the four advantages in addition to the modeling effort savings that are a by-product of this method and include the increased fidelity of the simulation models due to them being the same code, the automatic modeling of the communications between the controllers, the dynamically adjustable simulation resolution, and the simplification of transferring the system state from the actual system to the models. And finally as a proof of concept an example of a real physical system being controlled using this software tool was presented.

References

[1] W. J. Davis, “On-Line Simulation: Need and Evolving Research Requirements,” in Simulation Handbook, J. Banks, ed., Wiley, 1998, pp. 465-516.
[2] W. J. Davis, J. Macro, and D. Setterdahl, "An Integrated Methodology for the Modeling, Scheduling and Control of Flexible Automation," 1997, Journal on Robotics and Intelligent Control.
[3] F. G. Gonzalez, 1996, “A Simulation-Based Controller Builder for Flexible Manufacturing Systems,” Proceedings of the 1996 Winter Simulation Conference, pp. 1068-1075.
[4] F. G. Gonzalez, 2003, “Intelligent Control of Distributed Large Scale Manufacturing Systems,” Proceedings of the 2003 National Science Foundation Design, Service and Manufacture and Industrial Innovation Grantees and Research Conference.
[5] F. G. Gonzalez, W. J. Davis, "Initializing On-Line Simulations From the State of a Distributed System," Proceedings of the 1998 Winter Simulation Conference.
[6] F. G. Gonzalez, W. J. Davis, "Developing a Physical Emulator for a Flexible Manufacturing System," Proceedings of the 1998 International Conference on Systems, Man and Cybernetics.
[7] F. G. Gonzalez, W. J. Davis, 2003, "A New Simulation Tool for the Modeling and Control of Distributed Systems," in SIMULATION the Journal of the Society for Computer Simulation International.
[8] J. H. Mize, H. C. Bhuskute, and M. Kamath, “Modeling of Integrated Manufacturing Systems,” IEEE Transactions, vol. 24, no. 3, 1992, pp.14-26.
[9] W. D. Kelton, R. P. Sadowski, and D. A. Sadowski, Simulation with ARENA, McGraw-Hill, 2nd ed. 2001.
[10]B. A. Peters, J. S. Smith, J. Curry and C. LaJimodiere, “Advanced Tutorial - Simulation Based Scheduling and Control,” Proceedings of the 1996 Winter Simulation Conference, Eds. J. M. Charnes, D. J. Morrice, D. T. Brunner and J. J. Swain, pp. 194-198.
[11]J. S. Smith, R. A. Wysk, D. T. Sturrock, S. E. Ramaswamy, G. D. Smith and S. B. Joshi, “Discrete Event Simulation for Shop Floor Control,” Proceedings of the 1994 Winter Simulation Conference, pp. 962-969.
[12]J. S. Smith, B. A. Peters, “Simulation as a decision-making tool for real-time control of flexible manufacturing systems,” Proceedings of the 1998 International Conference on Robotics and Automation, pp. 586-590.
[13]T. M. Tirpak, S. M. Daniel, J. D. LaLonde, and W. J. Davis, "A Fractal Architecture for Modeling and Controlling Flexible Manufacturing Systems," IEEE Transactions on Systems, Man and Cybernetics, 22(5), 1992, 564-567.
[14]S. Korotkin, G. Zaidner, B. Cohen, A. Ellenbogen, M. Arad, and Y. Cohen, “A Petri Net design Methodology for Discrete-Event Control of Industrial Automated Systems,” IEEE 26th Convention of Electrical and Electronics Engineers in Israel, 2010, pp. 431 – 435.
[15]G. Yasuda and B. Ge, “Petri Net Model Based Specification and Distributed Control of Robotic Manufacturing Systems,” Proceedings of the 2010 IEEE International Conference on the Information and Automation, pp. 699 – 705.