Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

EXCEPTION MANAGEMENT ON A SHOP FLOOR USING ONLINE SIMULATION

David Katz
S. Manivannan

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332, U.S.A.

ABSTRACT

A framework for a supervisory controller to manage hard and soft exceptions on a shop floor is discussed. Various tasks involved in managing exceptions, the times at which these tasks are to be performed, and their impact on the performance of a supervisory controller are studied. On-line simulation methodology is utilized to detect, classify, and handle exceptions. A synchronization procedure, an essential step to perform on-line control using a discrete-event simulation language, is developed. Its main purpose is to resolve discrepancies between the behavior of entities and resources in the simulation model and shop floor. The framework is restricted to manufacturing cells; however, it can easily be extended to manage exceptions in logistics and distribution systems. A simulation model of a flexible manufacturing cell is used to illustrate exception detection and synchronization concepts.

1 INTRODUCTION

A tremendous amount of research is being conducted in the area of supervisory control of manufacturing systems. The relationships among the essential control aspects of a manufacturing system are often captured in object-oriented frameworks. Objects representing parts, machines, and tools are utilized in a tool management system to optimize the performance of a manufacturing cell (Bu-Hulaiga and Chakravarty 1988). Also, several object-oriented control frameworks for scheduling rail guided vehicles (Rogers and Williams 1988) and for managing AGVs (Powner and Walburn 1990) are found in the literature.

Manufacturing control strategies are often represented using a hierarchical framework (Swyt 1988). Control at the machine level is performed via state-tables in real-time; whereas, the cell level control is achieved offline by search algorithms to generate a routing and scheduling combination. A hierarchical blackboard architecture for modeling and analyzing manufacturing cells is described in (Young and Rossi 1988). A control methodology for FMS encompassing the cell and work stations is presented by (Ben-Arieh *et al.* 1988; Harmonosky 1990).

Several researchers have relied upon object-oriented simulation to solve a variety of control problems in manufacturing (Bischak and Roberts 1991; Shewchuk and Chang 1991). An event-time synchronization method is designed to ensure that a simulation model linked with a production cell reflects the same behavior and complex pattern of events occurring on the shop floor (Manivannan *et al.* 1991). A timed-discrete event system is devised using a formal language to monitor the behavior of a production cell, synthesize and enforce control, and provide offline feedback (Brandin *et al.* 1992). Recently, the use of online simulation for supervisory control has been advanced. A knowledge-based online simulation architecture has been developed to handle interruptions caused by single machine breakdowns and rush orders in a flexible manufacturing cell (Manivannan *et al.* 1992).

Figure 1 shows the periodic control and exception management activities performed using offline and online algorithms on a shop floor. Offline algorithms are used to perform planning, scheduling and routing functions. These activities are managed by a human supervisor. Automatic control devices such as PLCs continuously interact with online procedures to perform both the hardware control and the schedule changes due to breakdowns, rush orders, and the major deviations in the original plans. If the pre-determined schedule is carried out as planned, then the only online controllers required are those responsible for the actual implementation of control procedures (e.g., downloading a variety of CNC programs). In such cases, offline analyses are performed only at predetermined time intervals and the resulting schedules are implemented.

However, *a priori* plans are almost never realized in practice due to unexpected events or trends that cause the behavior of the shop floor to differ from the supervisor's expectations. These events, known as *exceptions*, are difficult to manage using offline control algorithms and require online data acquisition systems, a detailed model of the shop floor operations, and fast analysis tools.

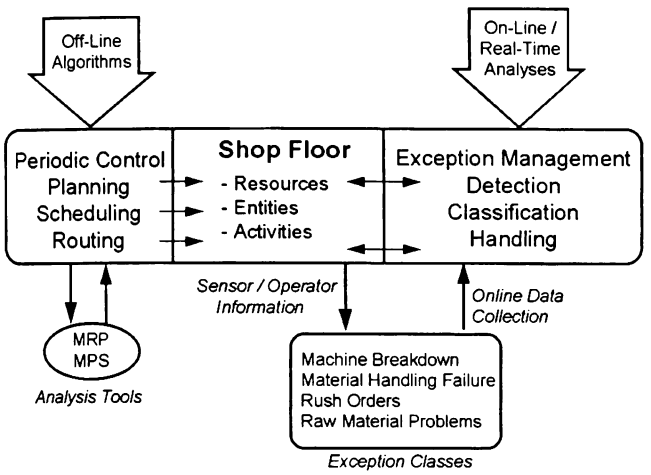


Figure 1: Supervisory Control Activities on a Shop Floor

Although many areas of offline shop floor control are well understood and thoroughly studied, an architecture to analyze the intricate patterns of online events caused by exceptions using a supervisory controller is lacking. An integrated framework is needed to solve online control problems in complex systems. In this paper, we focus on the issues of developing a supervisory control architecture for managing exceptions in a manufacturing cell via online simulation. The architecture is based on the fact that there is already an offline control mechanism in place to provide acceptable schedules and product routings.

2 EXCEPTION MANAGEMENT PROBLEM

Exception management involves solving problems in three distinct areas: detection, classification, and handling.

- *Exception detection* refers to the identification of differences between expected and actual performance of shop floor entities, resources, and activities that require a human supervisor's attention. The most recent data from the shop floor is required to ascertain its current status. In addition, knowledge of the actual and expected behavior of the shop floor and an approach to detect departures from this expectation are needed.
- *Exception classification* refers to the determination of the number and type of exception detected. For instance, an operator reports that an AGV is out of order. This exception can be classified as transporter unavailability. This exception affects the machines being serviced by that vehicle, the parts currently transported by it, other carriers, and perhaps the load/unload stations and the storage spaces on a shop floor. Exception classification requires a propagation method to identify all the entities, resources and activities that are affected by an exception.
- *Exception handling* requires the generation of a set of possible alternatives to resolve an exception, evaluation of these alternatives, selection of the best alternative, and implementation of this alternative as the control policy. The exception management system must include the algorithms necessary to accomplish these goals.

To date, several supervisory controllers incorporate simulation capabilities to provide planning, loading and scheduling decisions. However, these controllers seldom address the issues of online error detection and exception handling. Trends in simulation software including object-oriented designs, model builders and intelligent front ends have led to substantial improvements in the accuracy of models built to represent shop floor operations. Novel data collection technologies have enabled users to accurately capture the online data from production cells for control purposes. These developments allow models to mimic the shop floor more accurately; however, they still require information on various distributions to represent the stochastic events such as part arrivals, part processing, machine breakdown, material handling carrier failure, tool and fixture unavailability. Due to this reason, the behavior of entities and resources in a simulation model often does not closely resemble that of a shop floor. Indeed, this becomes a major issue during the management of exceptions; hence, a method is required to synchronize a model with the shop floor.

3 EXCEPTION MANAGEMENT VIA OLS

A supervisory control framework is designed by linking a shop floor with an online simulation (OLS) model. Various data acquisition devices and object-oriented knowledge bases are used to tie them together. The effectiveness of control depends upon the (i) accuracy of OLS models, (ii) online and offline data used to capture the deterministic, stochastic, and control events taking place on the shop floor, and (iii) methods employed in performing detection, classification, and handling.

3.1 Terms and Definitions

Prior to designing a framework for exception management we introduce the following terms and definitions:

- *Entities {E}* are raw materials, parts, or subassemblies that are transformed to become finished goods, scrap, or other byproducts.

- *Resources {R}* perform the value-adding tasks on a shop floor and include machines, transporters, and people.
- *Activities {A}* represent the interactions between entities and resources to form finished products, waste, etc.
- *Exception type* refers to a class of disruption caused by an unexpected activity that affects one or more entities and resources.
- *Control horizon* refers to the amount of time in which a control decision is made.
- *Number of exceptions* refers to the number of identifiable exceptions encountered concurrently during a control horizon. Current supervisory controllers are capable of managing single exceptions; however, they seldom handle the more common multiple exceptions that occur in large, complex manufacturing cells.
- *Current exception knowledge* is related to the learning ability of the supervisory controller. If an exception has been previously encountered, the control decision is obtained from a learning knowledge base.
- *Synchronization* is essential for managing exceptions via online simulation. It is a procedure in which the data acquired from a shop floor is used periodically to correct the variations or inaccuracies in OLS models.

Figure 2 shows the nature of shop floor exceptions and the relationships between the various terms and definitions.

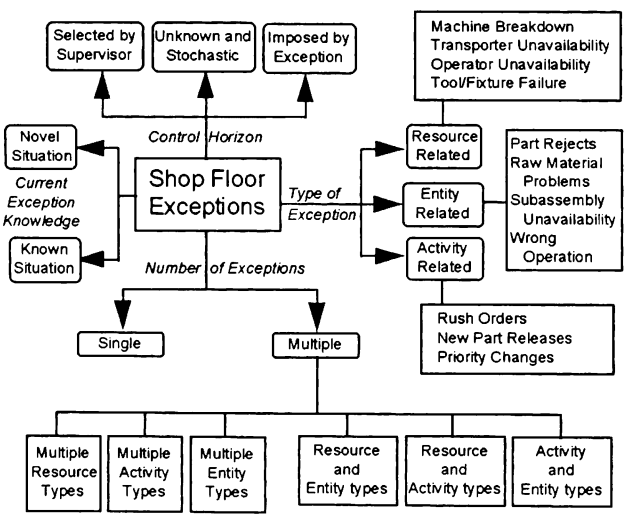


Figure 2: Exception Management Terminology and Issues

3.2 Exception Detection

Detection is the first and the most important step in managing exceptions. We classify exception detection into hard and soft. Hard exceptions are those tangible events that force the performance of the shop floor to change in some manner. Examples of hard exceptions are machine breakdowns, priority changes in processing parts, and rush orders. These events are easy to detect and usually reported by operators or automatic controllers. Soft exceptions are unexpected trends in the behavior of the entities and resources which are not so easy to detect. A soft exception occurs when the resources and entities on the shop floor behave differently than expected, without an obvious cause for the difference. For instance, if a machine does not perform as expected, then parts may begin to accumulate in the queue in front of the machine. This buildup may cause parts to be completed late, and the schedule, as planned, will not be met. In this case, a soft exception has occurred with no extraordinary evidence. For hard exceptions, the detection phase is trivial; however, the detection of soft exceptions is very difficult. A state-based approach to detect exceptions is described in this section. The terms relating to exception detection are depicted in Figure 3.

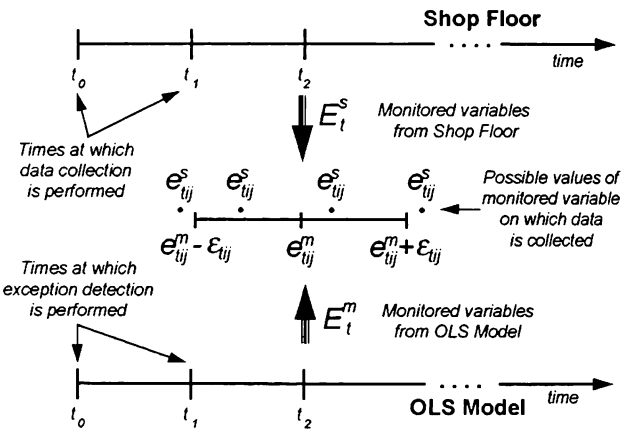


Figure 3: Exception Detection Via OLS

Let us define a matrix E_t^s representing the states of entities on a shop floor s. The components e_{tij}^s represent the value of state variable j of entity i monitored at time t. Each entity is assigned a unique number. Similarly, let R_t^s be defined with elements r_{tij}^s to represent the value of state variable j of resource i at time t. Let A_t^s be defined with components a_{tij}^s representing the value of state variable j of activity i at time t.

By keeping track of the entries in the matrices E_t^s, R_t^s, and A_t^s, the current status of the shop floor can be obtained at time t. Using an OLS model created using predefined arrival and departure patterns, process data, and schedule, we can compute the expected behavior and performance of entities, resources, and activities on the shop floor. Let us define E_t^m, R_t^m, and A_t^m as the expectations of E_t^s, R_t^s, and A_t^s respectively. The superscript m denotes the entities and resources in the OLS model developed to mimic the operations of a shop floor. Suppose ε_{tij} provide the half-

width of the interval about e^m_{tij} in which we expect e^s_{tij} to fall. Similarly, ρ_{tij} *and* α_{tij} are the half-widths of the intervals about r^m_{tij} and a^m_{tij} in which we expect r^s_{tij} and a^s_{tij} to fall respectively. The parameters ε_{tij}, ρ_{tij}, *and* α_{tij} are necessitated by the stochastic nature of the shop floor. Using these notations, we conclude that a soft exception has occurred at or before time t if and only if

$$\{e^s_{tij} \notin [e^m_{tij} \pm \varepsilon_{tij}]\} \vee \{r^s_{tij} \notin [r^m_{tij} \pm \rho_{tij}]\} \vee \{a^s_{tij} \notin [a^m_{tij} \pm \alpha_{tij}]\} \quad (1)$$

Equation (1) can be used to detect a soft exception more succinctly at time t for any (i, j). The matrix entries with superscript s are obtained from the shop floor via online data collection systems; whereas, the m values are obtained from OLS model. *Equation (1)* describes several parameters that should be selected rather cautiously for successful detection of exceptions. Their values greatly affect the performance of the detection process and can be obtained analytically, empirically, or heuristically.

The specification of E^s_t, R^s_t, and A^s_t values requires a decision by a supervisor as to which state variables of *{E}*, *{R}*, and *{A}* are monitored by the exception management system. This decision depends upon the function, cost, and availability of data collection devices on the shop floor. Further, the nature of interfaces between the operator and the exception management system play a critical role in deciding on these state variables. However, there is a tradeoff between the cost and level of detail during the selection and monitoring of state variables. In addition, higher levels of detail increase the cost of building and maintaining the OLS model.

The specification of parameters in *Equation (1)* for exception detection is a complex task. Many experiments and statistical analyses need to be performed to obtain the initial values. A learning knowledge base to systematically correct and update the parameters is required to improve the precision of the detection module. Finally, the human supervisor must be "in-the-loop", so that he or she can alter the parameters to achieve varying goals.

3.2.1 Synchronization

An event-time synchronization (ETS) to modify the events and their associated times in a simulation model was developed (Manivannan *et al.* 1991). ETS is achieved by altering the future event list of the OLS model in order to adjust for the differences between the model and the shop floor. This approach fails to consider the relationships between events and *{E}*, *{R}*, and *{A}* and does not make the necessary corrections to the OLS.

The problems are resolved using a two-step procedure. First, the current data pertaining to *{E}*, *{R}*, and *{A}* from the shop floor is transmitted to the OLS. For example, if an AGV is waiting at machine M1 on the shop floor, then the OLS model will be adjusted to ensure that an AGV is waiting at M1 at the same time. Next, the synchronization procedure utilizes an object-oriented knowledge base during the exception classification phase to determine why certain state variables disagree. For instance, let us assume that a machine takes twenty-one minutes to process a part on the shop floor; whereas, the OLS model reflects a processing time of nineteen minutes. In this case, the synchronization procedure will correct the data in the OLS model to reflect the actual performance of the shop floor. Figure 4 shows the changes in a state variable value over time before and after synchronization in OLS models.

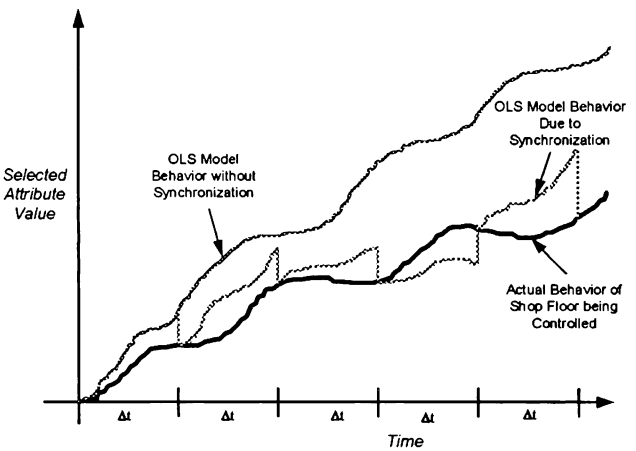


Figure 4: Changes in a State Variable Over Time in OLS Model Before and After Synchronization

3.2.2 Selection of Control Period

The exception management system is invoked by the controller whenever one of the two events occur: (i) a hard exception is reported, or (ii) a control period expires. We define the control period, denoted by Δt, as the amount of time between successive invocations of the detection procedure. The choice of Δt has a significant impact on the performance of the exception management system. If Δt is quite small then it will result in a "nervous" control system that is continuously comparing the simulation model with the shop floor. In fact, if we consider the $\lim\ \Delta t \to 0$, the exception management system tends to follow approaches currently employed in process monitoring systems. This is undesirable for a shop floor in which the status variables do not change frequently enough to warrant continuous monitoring. Further, the CPU time required by such a monitoring scheme is quite large.

Alternatively, specification of a large Δt will cause the exception management system to ignore soft exceptions for a period of time longer than desirable. This further reduces

the frequency with which the synchronization procedure is applied, resulting in divergence of the model and shop floor. Hence, when the control period expires, the shop floor will almost always be in an exception state that may or may not actually exist (the exception may be merely a manifestation of the stochastic nature of the shop floor). The choice of Δt is highly subjective and dependent upon the characteristics of the shop floor. The choice of Δt requires the human supervisor's inputs. The supervisor will specify an initial control period, and the learning knowledge base for the exception detection phase will modify the control period as necessary. The impact of Δt on exception detection is illustrated in Section 5.

3.2.3 *Selection of* ε_{tij}, ρ_{tij}, and α_{tij}

The three parameter sets ε_{tij}, ρ_{tij} and α_{tij} are responsible for controlling the sensitivity of detection procedures. These parameters allow for the implicit variation in both the shop floor and OLS model. For ease of analysis, we assume that the values of ε_{tij}, ρ_{tij}, and α_{tij} are constant for a specific i and j; i.e., we do not allow the values of these parameter to change over time.

The initial specification of ε_{tij}, ρ_{tij}, and α_{tij} requires an empirical study via simulation. Using the OLS model, first the value of Δt is determined. Estimates of ε_{tij}, ρ_{tij}, and α_{tij} are then obtained by running the OLS model using different random number streams. The matrices E_t^m, R_t^m, and A_t^m are generated from the OLS model at every Δt time units and these values correspond to the average behavior of the shop floor entities and resources.

Analogous to other exception parameters, the values of ε_{tij}, ρ_{tij}, and α_{tij} are continuously monitored and stored in the learning knowledge base. For instance, suppose the value of the parameter for the number of parts finished at operation i in OLS model is one. Then, this may lead to detecting exceptions many times without sufficient reason. In such cases, the learning knowledge base would adjust the parameter automatically to two.

3.3 Exception Classification

The exception classification procedure is activated by the controller whenever an exception is detected. It uses the online data from the shop floor and a knowledge base consisting of multiple rulebases to determine the number, type and novelty of exceptions. It then identifies whether synchronization and/or exception handling is appropriate. In addition, the exception classification procedure utilizes the object-oriented relationships among *{E}*, *{R}*, and *{A}* to describe the impact of exception (i.e., propagation effects). For instance, consider the following rule:

IF L_q^s *at M(x)* $>>$ L_q^m *at M(x) and*
arrival_rate$^s \cong$ *arrival_rate*^{m} *and*
speed (AGV)$^s \cong$ *speed (AGV)*^{m}
THEN exception is resource_degradation at M(x) and
Number of exceptions = 1

L_q^s and L_q^m denote the number of parts currently in the queue at machine *M(x)* on the shop floor and OLS model respectively. The exception classification procedure uses the relationships among machines and part types to find that *M(x)* is related to part type *P(y)*, so that the current exception affects *P(y)*. Several such rules are developed to reside in the knowledge base for classifying exceptions whenever detected.

3.4 Exception Handling

The exception handling procedures receive the data created during detection and exception phases through the controller and determine an appropriate action to resolve exceptions. A knowledge-based online simulation system (Manivannan *et al.* 1992) is implemented to handle hard exceptions due to rush orders and machine breakdowns. The architecture is being extended to handle both soft and hard exceptions. A key enhancement is a shift to a parallel processing mode so that the exception management system can continue to monitor a shop floor while selecting the best control decision via simulation. This enhancement ensures that the shop floor status does not change too much during analyses. Further, the simulation model used by the exception handling procedure greatly benefits from synchronization performed throughout the control history; the accurate OLS model allows for more precision in the evaluation of alternatives.

4 A NEW EXCEPTION MANAGEMENT SYSTEM

Figure 5 describes the operating principles of an exception management system as part of a supervisory controller. It is capable of performing exception detection *(ed_module)*, classification *(ec_module)*, and handling *(eh_module)*.

At time t, the *ed_module* determines whether the expected and actual performance of the shop floor agree. If the data captured is sufficiently similar to the data contained in the OLS model, then the *ed_module* deduces that no exception has occurred, performs synchronization, and continues to monitor the shop floor. On the other hand, if the OLS model and shop floor are sufficiently different, then it is inferred that an exception may have occurred. At this time, a message is sent to the *ec_module* describing the type and number of exceptions and conveying all the relevant data about the shop floor.

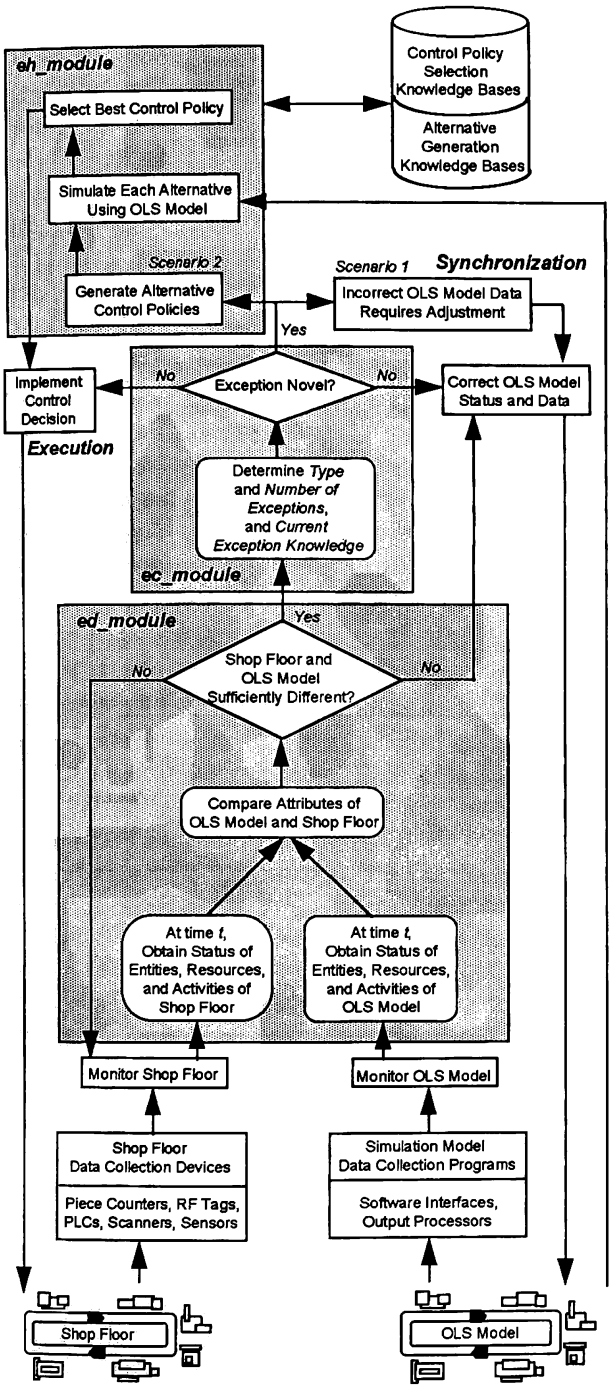


Figure 5: Exception Management Framework

Once the *ec_module* receives the message that an exception has occurred, it first determines the *type* and *number of exceptions*. It then uses the *current exception knowledge* from the learning base to check whether or not the exception is novel. If not, a control decision or OLS model update is performed immediately, and the controller resumes monitoring the shop floor. If the exception and situation are novel, one of the following scenarios exists:

scenario 1: There is an exception on the shop floor that requires revision of a control policy.

scenario 2: The data contained in a model (i.e., arrival time distribution, process time distribution) is incorrect and the model requires revision.

Other exceptions recorded during this time period also play a significant role in classification. For instance, if an AGV is already unavailable, the *ec_module* considers this fact when classifying another exception. If the *number of exceptions* during a *control horizon* is more than one, then the *ec_module* would classify this instance as a multiple exception case. The *ec_module* contains a set of rules stored in a knowledge base that determines which situation exists and what action to take next.

If *scenario 1* exists, the *ec_module* passes the type and number of exceptions and the associated data to the *eh_module*. First, the *eh_module* ascertains the *control horizon*, and initializes an internal clock to ensure timely output. Exception handling is performed using a "best control policy so far" rule to be applied when the control horizon expires. Once the control decision is reached, the *eh_module* passes the results to the shop floor and OLS model, maintaining the correspondence between the two.

If *scenario 2* occurs, then the *ec_module* invokes the synchronization procedure. Synchronization modifies the model data, model status, and future event list to reflect the actual shop floor performance and status. Once this activity is completed, the exception management system resumes monitoring the shop floor.

The exception management system also contains a learning knowledge base. This enables the supervisory controller to learn about the OLS model and shop floor during detection, classification and handling phases. This feature improves the performance and responsiveness of the exception management system.

5 APPLICATION

The implementation of the exception management system on a shop floor requires a better understanding of the different state variables and the interrelationships between them. In this section, a flexible manufacturing cell (FMS) is used to illustrate various exception detection and synchronization concepts. Instead of testing directly on a real FMS using data acquisition devices, a simulation model written in SIMAN and Cinema emulates the cell. A copy of the simulation model performs the functions of OLS. The emulated FMS uses one random number stream and the OLS model uses another. In this way, the stochastic variations found within the emulated FMS and OLS are generated.

The FMS consists of six machines and four AGVs, and produces six part types. Parts arrive according to a pre-planned loading schedule following an exponential distribution with a mean of twenty-five minutes. The parts are fixtured onto a pallet at a loading station requiring five minutes. The parts are transported by AGVs and processed by a series of machines. After all the processing steps are completed, the parts leave via an unloading station. The distance between any two adjacent machines is fifteen feet and the AGVs travel at a rate of twenty feet per minute. Table 1 provides the sequence and processing times for each of the six part types.

Table 1: Operation Sequence and Process Times

	Machine Number/Process Time for each Operation Opr(j)					
Part #	*Opr(1)*	*Opr(2)*	*Opr(3)*	*Opr(4)*	*Opr(5)*	*Opr(6)*
P1	M1/9	M2/18	M4/24	M3/8	M6/16	M5/38
P2	M2/26	M4/18	M3/34	M6/12	M5/17	M1/9
P3	M4/21	M1/16	M2/9	M6/12	M1/8	M5/4
P4	M6/17	M2/24	M3/31	M1/13	M6/14	M3/21
P5	M1/24	M2/8	M3/28	M4/31	M5/12	M6/18
P6	M2/11	M4/30	M3/23	M1/16	M4/9	M2/13

Several simulation experiments were conducted with the emulated FMS and OLS model. Two state variables related to the loading station were monitored to detect the occurrence of exceptions. These are

(i) number of parts waiting at the loading station, and
(ii) total number of parts of type six (P6) arrived at the loading station.

State variable (i) reflects the activities and behavior of resources in the FMS, and *state variable (ii)* corresponds to the part loading activity. The emulated FMS and OLS model were each warmed up for a period of 1000 minutes. At this point, synchronization was performed to ensure both the emulated FMS and OLS model would reflect the same initial behavior. From t = 1000 minutes, simulation runs were made to demonstrate the

- need for synchronization,
- impact of Δt on synchronization/OLS accuracy,
- selection of ε_{tij}, ρ_{tij}, and α_{tij}, and
- detection of exceptions.

The results generated during these simulation experiments are provided in Figure 6. This figure shows six graphs which plot the values of *state variable (i)* and *(ii)* for emulated FMS (thick solid lines), OLS model without synchronization (dashed lines), and OLS model with synchronization (thin solid lines). The results from the simulation experiments are described in the following sections.

5.1 Need for Synchronization

The impact of synchronization on the accuracy of the OLS model for supervisory control purposes was studied. By examining the thick solid and dashed lines in Figure 6, we infer that the values of *state variable (i)* or *(ii)* for the emulated FMS and OLS model exhibit greater differences over time. For instance, at t = 1050, the values of *state variable (i)* for the emulated FMS and OLS model were 10 and 9, respectively. At t = 1200, the corresponding values were 6 and 10. By viewing the thick solid and thin solid lines, we can conclude that synchronization has reduced the variability between the emulated FMS and OLS model. This ensures that the OLS model accurately depicts the dynamic state changes in the emulated FMS.

5.2 Impact of Δt on Synchronization

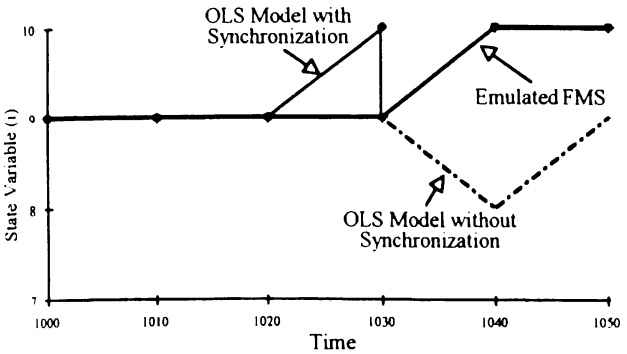
Synchronization was performed using the OLS model for various Δt values. For *state variable (i)*, three sets of simulation runs were made at control periods Δt = 10, 50, and 100 starting from t = 1000. The effect of Δt on *state variable (i)* in OLS model is shown in Figure 6 (a)-(c). A small value of Δt results in a near-perfect match between the emulated FMS and OLS model; however, this leads to frequent data collection from the emulated FMS to perform synchronization (see Figure 6 (a)). On the other hand, a large value of Δt leads to less frequent data acquisition and synchronization; however, large discrepancies between the values of *state variable (i)* in the emulated FMS and OLS model result (Figure 6(c)). At Δt = 50, a reasonable degree of accuracy is achieved with the OLS model while avoiding excessive data collection and frequent synchronization (see Figure 6(b)).

Another set of simulation experiments was conducted to examine the impact of Δt on the accuracy of the OLS model using *state variable (ii)*. The results, shown in Figure 6 (d)-(f), indicate similar behaviors for varying Δt values.

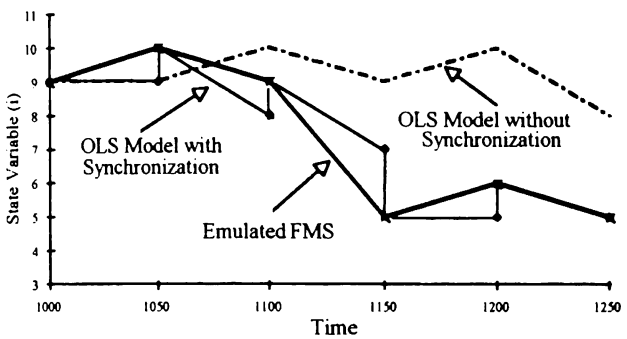
5.3 Selection of ε_{tij}, ρ_{tij}, and α_{tij}

Once an appropriate control period was chosen, the values of ε_{tij}, ρ_{tij}, and α_{tij} were obtained. Let us assume Δt = 50 minutes. Figure 6 (b) shows the changes in the values of *state variable (i)* for the OLS model after synchronization and for the emulated FMS. This variable is associated with an activity; namely, waiting at a loading station. Hence, the value of α_{tij} for this activity is required.

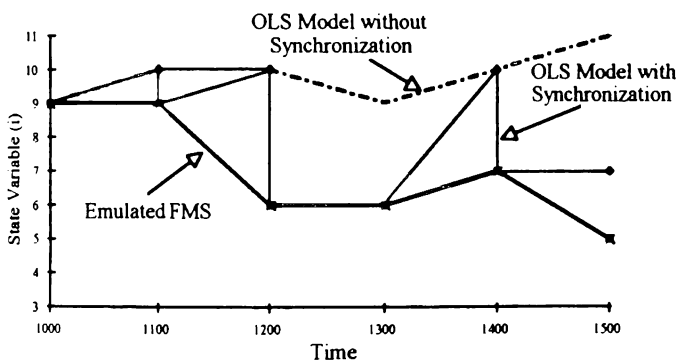
The value of α_{tij} is computed either by (a) averaging the difference between *state variable (i)* values of the emulated FMS and OLS model after synchronization over a period of time, or (b) performing an off-line statistical analysis. If method (a) is selected, then from Figure 6(b), α_{tij} is found to be one. Likewise, the values of all the other ε_{tij}, ρ_{tij}, and α_{tij} for *{E}*, *{R}*, and *{A}* are computed using the state variables monitored by the supervisory controller.



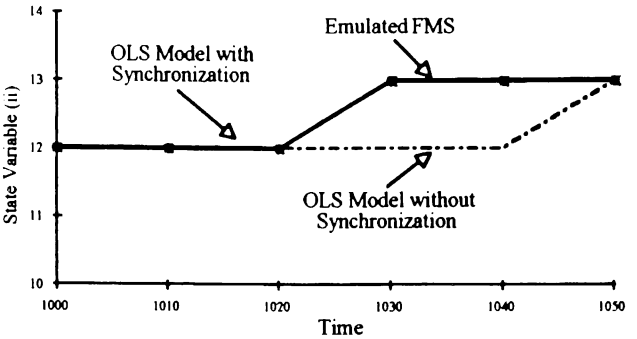
(a) Changes in *State Variable (i)* for Δt=10 in OLS Model With and Without Synchronization



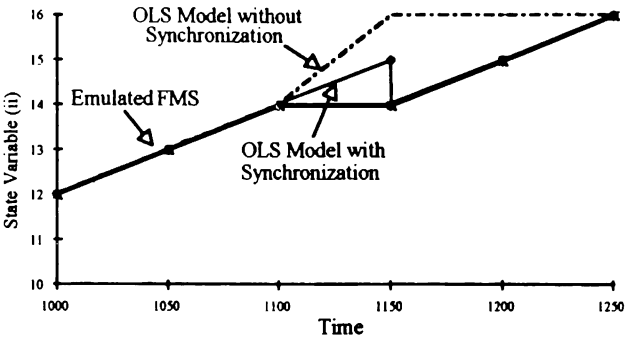
(b) Changes in *State Variable (i)* for Δt=50 in OLS Model With and Without Synchronization



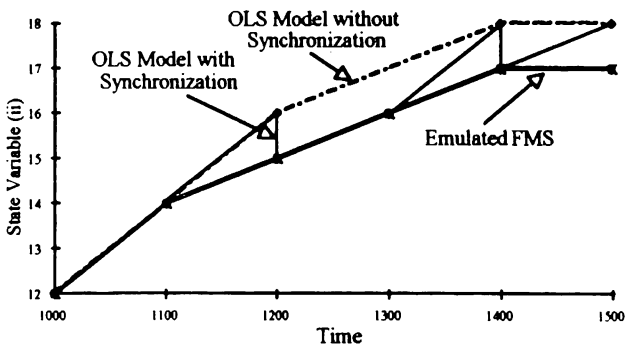
(c) Changes in *State Variable (i)* for Δt=100 in OLS Model With and Without Synchronization



(d) Changes in *State Variable (ii)* for Δt=10 in OLS Model With and Without Synchronization



(e) Changes in *State Variable (ii)* for Δt=50 in OLS Model With and Without Synchronization



(f) Changes in *State Variable (ii)* for Δt=100 in OLS Model With and Without Synchronization

Figure 6: Results of Simulation Experiments

5.4 Exception Detection

Once the OLS model and the emulated FMS were fine-tuned via synchronization to be almost identical, the OLS model was used to manage exceptions. Using the value of Δt = 50 and α_{tij} = 1, a simulation run was made from t = 1000 to detect exceptions based on the value of *state variable (i)*. For this purpose, a soft exception was injected into the emulated FMS by changing the mean of the exponential inter-arrival distribution from 25 to 23. The outputs generated by the OLS model and the emulated FMS are shown in Figure 7. At t = 1050, the difference in the values of *state variable (i)* was one. The *ed_module* found this value to be within the acceptable range and concluded that no exception had occurred during this observation period. Synchronization was performed at this point and the *ed_module* continued to monitor the emulated FMS.

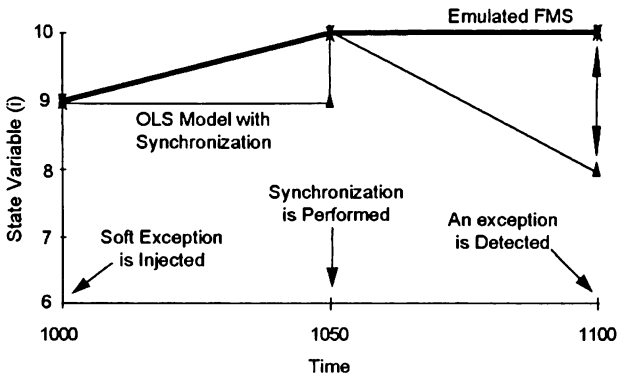


Figure 7: Detecting an Exception in the Emulated FMS

At t = 1100, the difference was two and at this point, the *ed_module* detected an exception and sent a message to the supervisory controller. Once the exception was detected, the *ec_module* classified the disruption and the *eh_module* provided an appropriate control decision.

6 SUMMARY

An integrated framework for a supervisory controller to manage exceptions on a shop floor was described. A state-based approach for detecting soft exceptions was outlined. A synchronization procedure was developed to model the shop floor more accurately. Future research will involve further enhancements to *ed_module* by interfacing it with a knowledge base and online data sources. The various modules in the exception management system will be linked via an object-oriented knowledge base, a supervisor interface, and a learning base.

REFERENCES

Ben-Arieh, D. H., C. Moodie, and C. Chu 1988. Control Methodology for FMS. *IEEE Journal of Robotics and Automation*, 4 (1).

Bischak, D., and S. Roberts 1991. Object-Oriented Simulation. In *Proceedings of the 1991 Winter Simulation Conference*, ed. B. Nelson, W.D. Kelton, and G. Clark, 194-203. Phoenix, Arizona.

Brandin, B., W. Wonham, and B. Benhabib 1992. Manufacturing Cell Supervisory Control - A Timed Discrete Event System Approach. *IEEE International Conference on Robotics and Automation*, 931-936.

Bu-Hulaiga, M., and A. Chakravarty 1988. An Object-Oriented Knowledge Representation for Hierarchical Real-time Control of Flexible Manufacturing Systems. *International Journal of Production Research* 26: 777-793.

Harmonosky, C. 1990. Implementation Issues Using Simulation for Real-Time Scheduling, Control, and Monitoring. In *Proceedings of the 1990 Winter Simulation Conference*, ed. O. Balci, R. Sadowski, and R. Nance, 595-598. New Orleans, Louisiana.

Manivannan, S., and J. Banks 1991. Real-Time Control of a Manufacuring Cell Using Knowledge-Based Simulation. In *Proceedings of the 1991 Winter Simulation Conference*, ed. B. Nelson, W.D. Kelton, and G. Clark, 251-260. Phoenix, Arizona.

Manivannan, S., and J.Banks 1992. Design of a Knowledge-Based Simulation System to Control a Manufacturing Shop Floor. *IIE Transactions*, 24 (3).

Powner, E., and D. Walburn 1990. A Knowledge Based Scheduler. In *First International Conference on Expert Planning Systems.*

Rogers, P., and D. Williams 1988. A Knowledge-based System Linking Simulation to Real-time Control for Manufacturing Cells. *Proceedings of the 1988 IEEE International Conference on Robotics and Automation* 2: 1291-1293.

Shewchuk, J., and T. Chang 1991. An Approach to Object Oriented Discrete-Event Simulation of Manufacturing Cells. In *Proceedings of the 1991 Winter Simulation Conference*, ed. B. Nelson, W.D. Kelton, and G. Clark, 302-311. Phoenix, Arizona.

Swyt, D. 1988. AI in Manufacturing: The NBS AMRF as an Intelligent Machine, Intelligent Manufacturing, The Benjamin/Cummins Publishing Company.

Tayanithi, P., S. Manivannan, and J. Banks 1992. A Knowledge-Based Simulation Architecture to Analyze Interruptions in a Flexible Manufacturing System. *Journal of Manufacturing Systems*, 11 (3).

Young, R., and M. Rossi 1988. Toward Knowledge-Based Control of a Flexible Manufacturing System. *IIE Transactions*, 20 (1).

AUTHOR BIOGRAPHIES

DAVID KATZ is a Doctoral Candidate in the School of Industrial and Systems Engineering at Georgia Institute of Technology. He is a recipient of an NSF Graduate Fellowship which supported this research. His research interests are in the area of exception management via simulation, supervisory control systems, and artificial intelligence. He is a member of IIE, SME and Alpha Pi Mu.

S. MANIVANNAN is an Assistant Professor in the School of Industrial and Systems Engineering at Georgia Institute of Technology. His current research interests lie in the area of intelligent systems engineering, with an emphasis on integrating discrete-event simulation, knowledge bases, and novel data acquisition techniques in logistics and manufacturing systems. He is a member of IEEE, SCS, SME and ASEE.