Using a Reflective Real-Time Operating System to Implement a Just-in-Time Scheduling Policy for a Flexible Manufacturing Workcell[1]

Marty Humphrey
Technical Report 95–87

UMass Computer Science Technical Report 95–87
June 16, 1995

Abstract

Research on the automated control of a flexible manufacturing workcell has generally assumed that there is a low-level computer system to control the machines, without addressing the precise functionality of this low-level system and how it might uniquely support the manufacture of products. This work addresses how Spring, a real-time operating system, can be used to implement a Just-In-Time (JIT) manufacturing policy for a flexible manufacturing workcell. The JIT policy is characterized by the right amount of manufacturing material being at the right place at the right time. We argue that two schedulers are desired to control a workcell because of two sets of scheduling requirements and responsibilities. The High-Level Scheduler (HLS) schedules machine operations need to construct products as close to their deadlines as possible. Spring schedules the fine-level computational tasks that comprise machine operations according to the Worst-Case Execution Time (WCET) of the tasks such that the individual tasks are executed before their deadline. The *reflective* properties of Spring—retaining at run-time the worst-case execution time of tasks, their resource requirements, etc.—enable Spring to provide dynamic feedback to the HLS, which the HLS uses to make scheduling decisions. Simulation studies show that the two-level scheduling system outperforms both (a) a single scheduler based on worst-case execution times, and (b) a single scheduler based on average-case execution times. Using only the worst-case execution times can lead to underutilization of resources and relatively few products manufactured, while using only the average-case execution times can lead to multiple products missing their deadlines through a ripple effect. The contribution of this work is showing how the reflective properties of the real-time operating system allow the HLS to make important decisions in a fast-changing environment.

[1]This work is funded by the National Science Foundation under grant IRI-9208920.

1 Introduction

It is increasingly important for manufacturing systems to operate at the highest level of efficiency in order to be competitive in today's global markets. Manufacturers continually evaluate new manufacturing approaches, machine purchases and machine configurations in order to gain an edge on the competition. Flexible manufacturing, in which flexibility of machine operations, process routings, and machine layout is stressed, has been touted as a breakthrough organizational and operational design. A flexible manufacturing workcell consists of many machines that collectively provide a variety of machine operations and a transport system to move material from one machine to another within the workcell. A key form of flexibility in the workcell is its ability to quickly adapt to new orders and operations. Combined with new manufacturing policies that place a higher emphasis on the customer, flexible manufacturing in principle is a attractive operating philosophy for small- to medium-sized manufacturers.

A misconception concerning flexible manufacturing systems is that it is generally assumed that the low-level computer system that will operate the complex of machines has already been built, or is at least not difficult to assemble from existing technologies. A problem with this belief is that conventional technologies will fail to be effective in situations in which the operational bounds are being tested. Flexible manufacturing is one such domain—there is a high capital investment required to adopt a flexible manufacturing approach, so management inevitably requires high utilization of machinery.

A key aspect to flexible manufacturing is the automated high-level scheduling system that they employ. Existing automated scheduling systems rarely have access to, and therefore do not consider, the second-to-second status of the machinery, buffer space, and transport system. Rather, scheduling is based on the performance of machines in some past time period and is subject to inaccuracies. Because of this, utilization will inevitably be lower than if the machinery were controlled by a system that could recognize unexpected events quickly after they occur and attempt to adjust the operations dynamically.

The goal of this research is to produce a system for a flexible manufacturing workcell in which the production of a schedule for the machines in the workcell and the execution of the schedule are tightly integrated. By integrating the scheduling of machines with the current state of the machines, operations can be scheduled on machines that are most appropriate for the current situation. Less time can be spent waiting in queues for machines. When machine operations take less than their worst-case execution times, rescheduling can opportunistically introduce new work plans into the schedule. This leads to both more orders being satisfied and a higher utilization of expensive equipment.

In this work, we show how the Spring real-time operating system [24] can be used as the basis of workcell operations. In our model of the workcell, machines of the workcell are invoked only as a direct initiative to satisfy an order that has arrived from the outside world. Each order has a deadline associated with it. In this environment, Spring is used in two ways: it is used to execute of the machines, and it is used as one-half of a two-component scheduling system for the workcell. While in principle there is no reason why Spring could not perform all of the scheduling, the approach taken is that there are two separate schedulers. One scheduler, the **"High-Level" Scheduler (HLS)** is responsible for selecting which requests for products (i.e., the orders) to attempt to fulfill and scheduling the sequence of machines to manufacture the product (i.e., the process plans) according to the desired manufacturing policy. The second scheduler, which is a modified version of the scheduling component of Spring [18], is responsible for receiving each order from the **HLS**, which has an single, overall deadline, and dynamically translating the machines in the process plan into the individual computational tasks that represent the machine sequence. Spring assigns individual deadlines on the tasks and attempts to schedule the tasks such that all resource requirements and deadlines are met. The key difference between the two schedulers is that the **HLS** maintains a high-level perspective on the manufacturing while the Spring scheduler generates a schedule in which the integrity and safety of the individual machine operations are guaranteed. For example, the **HLS** schedules activities so that the product will leave the workcell at a certain time, while the Spring scheduler determines start times for all of the computational tasks such that they will

complete before their deadline even if they take their worst-case execution times.

A key element of this research is how the *reflective* properties of the Spring kernel support multi-level scheduling [25]. Reflection is the computational process of reasoning about and acting on the system itself. The reflective properties of Spring are used by the **HLS** both to enable the **HLS** to ask Spring to execute a process plan, and to resolve the **HLS**'s uncertainties of the state of the environment that are caused by maintaining a high-level perspective. Semantic information exploited by the **HLS** can include machine status (up or down), process plan status (complete or still executing), and projected machine usage.

To evaluate our approach to workcell scheduling and operation, we simulated the machines of a workcell, and used Spring to invoke the machines. Only the manufacturing environment—the machines, arrivals, raw materials, etc.—is simulated; Spring itself is a fully-operational real-time kernel. The efficiency of the Spring kernel and the **HLS** directly impact performance, because it is not assumed that scheduling requires zero time. Rather, the deadline of an order is interpreted as the number of real seconds in which to both schedule a process plan to satisfy the order and to execute the process plan. A realistic manufacturing approach, Just-In-Time [5], was selected to show applicability to real-world manufacturing policies. The **HLS**-Spring system is compared to two single-level schedulers: one that schedules machines based on average-case duration of machine operations, and another that schedules only on worst-case execution time. In each of the three paradigms, the expected machine durations were known, but the actual machine times are not known until after the machine executed. Experiments were conducted for four environments characterized by the actual machine durations: the average duration of each machine was as expected, and there was no variance; the average duration of each machine was as expected, but there was large variance from one instance to the next; the average duration of each machine was consistently above the expected duration; and the average duration of each machine was consistently below the expected duration. The rationale for testing our system in each of these four is that each of these can appear as a phase of normal workcell operations. Therefore, the evaluation of each scheduling system is not necessarily the performance in each of the four environments, but rather how well each does on the four environments collectively. The **HLS**-Spring system is shown to easily outperform the Average-Case Scheduler, which is prone to catastrophic failure. The **HLS**-Spring moderately outperforms the Worst-Case Scheduler in three of the four cases, and readily outperforms the Worst-Case Scheduler in the fourth case. While the **HLS**-Spring system outperforms both of the single-level schedulers, preliminary analysis has shown that even better performance can be attained if the **HLS** exploited *more* of the reflective aspects of the Spring system.

The rest of this paper is organized as follows. In section 2, the basics of the flexible manufacturing environment and Just-In-Time manufacturing are presented. In section 3, other approaches to the real-time control and scheduling of manufacturing machines are discussed. These approaches include Artificial Intelligence (AI) techniques, control-theoretic approaches, and techniques drawn from Operations Research (OR). Section 4 contains the details of our scheduling approach for the operations of a flexible manufacturing workcell. Section 5 contains experimental results. Section 6 contains the conclusions, which includes work that we intend to pursue in the future.

2 Flexible Manufacturing and Just-In-Time Manufacturing

This section presents the basics of a flexible manufacturing workcell and the Just-in-Time manufacturing policy.

2.1 Flexible Manufacturing

A flexible manufacturing system is an integrated, computer-controlled complex of automated material handling devices and numerically controlled (NC) machine tools that can simultaneously process

medium-sized volumes of a variety of part types [2]. Flexible manufacturing is similar to a job shop, except that the coordination of parts, tooling, fixtures, machines and material handling equipment is more complex [12]. In theory, the real-time scheduling of a flexible manufacturing system should be based on the actual system state, such as arrival of parts, machine states, queues at machines, tool breakages, and rushed jobs [13].

The goal of flexible manufacturing is to create a manufacturing environment in which a wide variety of products can be made, and new process plans can be readily introduced, without having to dismantle and reconfigure the production line. Constructing a flexible manufacturing system requires an appreciable amount of capital expense—it is important to upper-level management that machines are highly utilized.

In many flexible manufacturing systems, machines are grouped according to functionality into *workcells*–each workcell has a group of machines (usually between 3 and 10) that collectively are capable of performing a wide variety of operations. A typical workcell is shown in Figure 1, and will be the basis of the experiments conducted for this work. There are four process plans shown in this workcell:

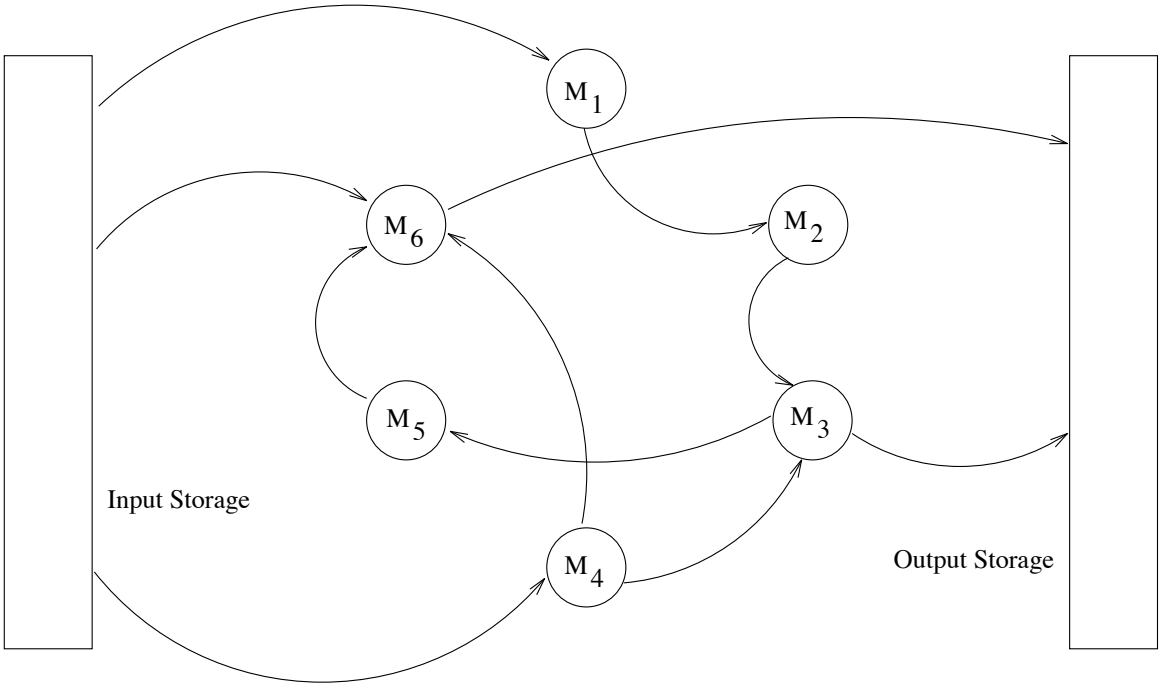

Figure 1: An Example Workcell

4–6, 1–2–3, 6, 4–3–5–6. A workcell is generally capable of more process process plans, but only four are included here both to simplify the presentation of a workcell and to simplify the experiments. In a flexible manufacturing workcell, each machine is capable of performing multiple operations (incurring a setup cost for each time the machine changes its current operation), and often two or more machines in the same workcell can perform the same operation, perhaps at different precision and different cost. Partially-completed products can be held in either the buffer space local to one of the machines, or in buffer space in a central location of the workcell. A materials transport system is used for machine-to-machine routing.

2.2 Just-In-Time Manufacturing

Just-In-Time (JIT) is a manufacturing policy originating from Japan that mandates having only the necessary items at the necessary places at the necessary times [10]. While the scope of JIT includes

shipping finished products out of the factory at precisely the time at which they are due, it also includes delivering partially-completed products from one machine in the process plan to the next machine at precisely the time at which it is needed, and having the right amount of raw material available to use as the basis of manufacture.

There are many advantages offered by effective JIT management [5]:

- JIT allows an organization to meet consumer demand regardless of the level of demand. Customers can be satisfied quickly and efficiently.
- JIT allows a reduction in raw material, work-in-progress (WIP), and finished goods inventories. Fewer goods in inventory means less time for possible damage to the product. In addition, less factory space has to be allocated for storage. This space is better allocated to operations that add value to the product (time in storage does not).
- JIT eliminates the possibility of over-producing a product (when it no longer is deemed valuable by market conditions).
- JIT allows rapid prototyping of new products and quick modification of process plans, because machinery has not been allocated to fixed production lines.

An obvious disadvantage of the JIT philosophy is the relatively small amount of inventory space for products that are projected to be required in the future, but are not actually required yet. An obstacle to implementing the JIT philosophy that is a problem in non-automated manufacturing centers is the reluctance of human workers to adopt a much more controlled work policy than traditional manufacturing. American companies using JIT techniques include Hewlett-Packard, Apple Computers, Toyota, and Harley-Davidson [21].

3 Other Methods of Scheduling and Control of Workcells

This section presents related work in the real-time scheduling and control of flexible manufacturing cells. Just-In-Time was selected as the manufacturing policy for this work because it has been shown to be an effective policy that is quite difficult to implement even in an environment that is less dynamic than the environment in which we model. This section also introduces alternatives to the two-level scheduling system adopted in this work. The shortcoming of these alternative approaches is that they are not appropriate for achieving the high level of responsiveness that the two-level scheduling system can achieve.

JIT belongs to a class of *pull* systems, which attempt to "pull" a product through its manufacturing cell as a respond to some order. The alternative to a pull system is a *push* system. In a push system, products are pushed through the system generally in an attempt to keep utilization of resources high. Finished products are stockpiled in inventory until they are required by a customer. While in inventory, finished products can become outdated, lost, or even damaged.

In industrial practice, a widely-accepted form of scheduling automation has been Material Requirements Planning (MRP) [15]. The objective of a MRP system is to determine requirements, in order to be able to generate information needed for correct inventory action (procurement and production). MRP systems meet their objective by computing new requirements for each inventory item, determine actions to meet those requirements, and time-phasing them. MRP determines capacity requirements but ignores available or installed production capacity. MRP can often prescribe machine loading in excess of 100% [19]. When the financial functions are tied into MRP, it is called Manufacturing Resource Planning, or MRP II [26]. MRP II is a system that includes manufacturing, finance, marketing, engineering, and distribution. As opposed to the JIT philosophy, MRP and MRP II are predicated on the existence of inventory, which JIT seeks to eliminate. Further differentiating our work from MRP

and MRP II systems is that the two-level scheduling system reacts much more quickly than MRP and MRP II systems, which can only generate schedules on a daily or weekly basis.

The alternative approach to MRP that is being more accepted is Finite Capacity Scheduling (FCS), in which the currently-available production capacity is considered (and generally attempted to be optimized). One area of FCS that is receiving considerable attention is Optimized Production Timetables (OPT), in which identified resource bottlenecks are the priority for scheduling–by keeping the bottlenecks occupied, production flow is maximized [11]. OPT may require significant time to generate schedules, because it attempts to optimize bottleneck resources. Thus, it may be inappropriate for highly dynamic environments, such as the flexible manufacturing workcell of this research.

Research into the scheduling of automated scheduling systems include work from the areas of Operations Research (OR) and Control Theory. Buzacott and Yao review the application of queueing network models to flexible manufacturing systems [4]. The OR approach to scheduling is covered by [8]. Heuristic dispatch rules are often applied at the machine level; an extensive review is offered in [17]. OR results are generally limited to simplified single-machine environments and are not directly applicable to the flexible manufacturing workcell. Recent results in the control-theoretic approach to the production planning, scheduling, and control are covered in [7]. Control theory is a viable approach to the control of a flexible manufacturing, but, because it is based on *rates* of operation and product flow, it offers limited applicability to the goal of this work, which is to manufacture products solely as a result of an explicit order.

The hierarchical framework for controlling automated manufacturing has been adopted in numerous work [1, 16, 3]. In these approaches, analysis occurs at more than one time (e.g., before the job is released to the workcell, at the time the job is released to the workcell to determine priority, and at the time the machine selects the next job). Our approach attempts to more closely unify these three levels in order to make better decisions before the job is released to the workcell.

Numerous approaches to control of automated manufacturing arise from the world of Artificial Intelligence. Job-shop schedulers include ISIS [6], which took solely an order-centered perspective, and its successor OPIS [22], which attempted to balance the order-centered approach with a resource-centered approach. Other knowledge-based schedulers include ReDS [9] and Micro-Boss [20]. AI schedulers are limited in real-time situations because of their inherent lack of predictability concerning the amount of time required to generate a schedule.

A distinguishing feature of work presented in this paper is, once selected for manufacturing, a process plan is *guaranteed* to be executed through the scheduling of worst-case execution times. None of the alternatives discussed here guarantee that a product will be produced by its deadline (if the product is not already in inventory). At this level, all schedules are "best-effort". Another difference between our work and the work discussed here is that our system both builds *and* executes a schedule. Real-time guarantees in combination with another agent that is not constrained to operate according to worst-case execution times is also adopted in the architecture of CIRCA [14], but has yet to be applied and evaluated in a manufacturing domain.

4 HLS-Spring Scheduling Methodology

If a flexible manufacturing workcell is going to be truly automated, it must include both a facility for scheduling and a facility for executing the schedule. In our approach, the process that is responsible for scheduling is directly tied to the process that is issuing instructions to the machines of the workcell. For simplicity, we refer to process by which steps in the process plans of products being made are assigned to machines for discrete periods of time as *scheduling*, and we refer to the execution of the schedule as the *control* of the machines. This section discusses our unified approach to scheduling and control, and emphasizes that responsive scheduling is achieved through this unified approach.

4.1 Control of Machines in the Workcell

The control of the machines in the workcell requires the ability to issue precise instructions to each machine, and the ability to later check some status information on the machine to determine if the operation was completed successfully. We assume that each machine is capable of receiving instructions through a computer network and has the necessary functionality to perform self-regulatory checks. For example, if the machine is already set up to perform a particular action, then an instruction code can be issued to the machine to direct the machine to load the part from its local buffering space, perform the desired operation, and terminate at any point that it encounters a problem. Our control mechanism is responsible for issuing the command to the machine, and then checking its status after a period of time equal to the worst-case execution time (WCET) of the action. It is assumed that the machine indicates when it is done by writing some status register at the point at which the operation terminates successfully.

4.2 Scheduling of Machines in the Workcell

The scheduling and execution of the tasks that collectively execute machines in the workcell is performed by the Spring kernel [24]. The Spring multi-processor architecture consists of three application processors (AP_1, AP_2, and AP_3) and a single system processor (*SP*). The SP is devoted to scheduling (running the Spring scheduler) and interacting with outside entities (such as the **HLS**), while the APs execute the schedule created by the Spring scheduler. To execute a machine, the Spring scheduler schedules four *tasks* (T_1–T_4) on the same AP for every machine invocation. T_1 performs some generic setup code. T_2 acquires the machine resource and issues the instruction to the machine. T_3 checks the status of the machine, performs error-correction code if necessary, and releases the machine resource. T_4 performs some additional bookkeeping. T_2 can be scheduled immediately after T_1, and T_4 can be scheduled immediately after T_3, but there must be a delay between the end of T_2 and the start of T_3 equal to the WCET of the machine.

Process plans are scheduled by allocating time for the individual machines that comprise the process plans. Because Spring schedules (and executes) the machines, it can be argued that Spring itself is capable of scheduling all the operations for the workcell. However, we do not take this approach for the following primary reason. While we believe that real-time is *not* synonymous with fast [23], we believe the Spring scheduler must be fast. Scheduling orders from the outside world inevitably consumes a large amount of time; requiring the Spring scheduler to schedule orders as well as low-level tasks might compromise the safety and integrity of the machine operations. *The Spring kernel is responsible for the control of the machines, to ensure that products are made correctly and safely.*

Instead of creating a monolithic scheduler, we decided to split scheduling into two systems according to functionality. The Spring system controls the low-level operations of machines, while the **HLS** manages uncertainty in the environment and selects between competing sets of alternatives. The split in functionality is very rough and is an open research issue: which level–the "real-time system" or the "real-time AI system"–should be responsible for performing each action? We experimented with numerous versions of the **HLS**, but settled on a scheduler that schedules based on the worst-case time requirements of machine operations, but is expected to react when operations take less than their worst-case. The Spring system continues to schedule based on worst-case execution times of computational tasks.

The way an order is manufactured by the workcell is as follows. First the **HLS** attempts to schedule the process plan in its schedule. Its schedule maps operations in the process plan to machines at specific times, based on worst-case performance. If the **HLS** cannot fit a process plan into its schedule, such that the order completes before its deadline, but not so early as to violate the nature of the Just-In-Time philosophy, the order is retained until a minimum amount of time needed to be scheduled at a remote, lightly-loaded workcell. The rationale for this approach is that there are multiple workcells that can be used to satisfy an order, but that orders should be scheduled locally in order to alleviate the issues of

distributed scheduling. Once the process plan is scheduled, the **HLS** waits until a release time that it has determined, based on the time at which the first machine operation in the process plan is scheduled, and its understanding of the current load on Spring, and asks the Spring scheduler to schedule the process plan. The Spring scheduler receives the request, dynamically constructs the task graph that, when executed, will produce the product intended by the process plan, and attempts to schedule the task graph with its current schedule. Because Spring is a guarantee-based operating system, if the Spring scheduler cannot find a schedule in which all of the tasks of its previous schedule, as well as all of the tasks in the task graph that corresponds to the process plan, are allocated the resources they need for the time they need, the task graph introduced will be rejected. If the task graph is rejected, Spring continues executing the schedule it had before the request by the **HLS**. If a schedule can be found, it accepts the new task graph, and informs the **HLS**. In a manufacturing environment, the advantage of the guarantee is that it provides a mechanism for ensuring that *all* machine operations of a process plan will complete before the due date of the product. Without this guarantee approach, it would be possible for an order to be unpredictably preempted by other orders, causing the overall performance of the workcell to degrade as products are generated late.

There is a range of information that Spring can provide to the **HLS**, because Spring retains a significant amount of information at run-time. When the **HLS** requests Spring to schedule a process plan, Spring can return simply *yes* or *no*, indicating whether or not the process plan could be scheduled by the Spring scheduler. However, it is also capable or providing more information, such as the projected start and finish times of the order if it is schedulable, and the order(s) preventing its schedulability if it is not. Spring it also capable of replying to queries concerning the current status of an order. Again, a range of information is capable of being replied—Spring can reply with the up-to-date estimate of start times and finish times, or it can reply with the *machines* that have yet to execute. All of this information is useful to the **HLS**, because the **HLS** has a view of the environment that is uncertain. The main source of uncertainty in the environments tested in this paper are the actual machine durations. In this work, we configured Spring to reschedule as tasks execute and finish earlier than their worst-case execution times. By rescheduling, Spring can reclaim the time that had been previously allocated to the task. By operating in this manner, Spring guarantees that tasks will execute correctly even if they require their worst-case execution times, but resource utilization will not degrade if they take much less than their worst-case execution times. While this rescheduling results in higher machine utilization, it can also exacerbate the uncertainty captured in the schedule of the **HLS**. For example, assume the **HLS** schedules an order to execute on machine M_1 from time $T = 10$ to $T = 20$, and then on machine M_2 from $T = 21$ to $T = 28$. Further, assume that when the **HLS** submits the order to Spring, the Spring scheduler schedules tasks such that machine times initially match the expectations of the **HLS**. If M_1 takes less than 10 seconds, the Spring scheduler will reschedule, causing the operation on M_2 to execute earlier. At this point, the view of the **HLS** is no longer accurate. If the inaccuracies persist in the schedule of the **HLS**, the **HLS** will schedule orders for particular times that are already being used by the Spring scheduler. In this case, when the orders are submitted by the **HLS**, they will be rejected, and utilization will decrease.

In general, the **HLS** can use the semantic information capable of being provided by Spring to resolve its inaccurate view of the state of the machinery. However, in this paper, only a limited amount of information is actually used. When the **HLS** submits an order, Spring returns the scheduled start time and scheduled finish time if it is schedulable, and *no* if it is not. Spring informs the **HLS** when a process plan has completed. The **HLS** uses this to update its understanding of the current schedule, by removing the executed order from its current schedule. The **HLS** reclaims this time, and attempts to fill the time with other orders. Note that the **HLS** does *not* move already-scheduled process plans earlier, because of the Just-In-Time nature of the manufacturing. The ability to return the status of an order is not exploited. The intent of this paper is to illustrate the usefulness of this level of feedback, without attempting to optimize the performance of the workcell. Where appropriate, however, an attempt is

made to specify that certain information generated by Spring would aid performance. More complex feedback is the subject of further research.

5 Experimental Results

To show the utility of our scheduling design, we simulated the operations of a single simplified flexible manufacturing workcell (Figure 1). The workcell has six machines; each machine is capable of performing only one operation. In order to test a highly-dynamic environment, the worst-case machine times were chosen to be on the order of seconds: the WCET for machine M_i is $(4 \times i)$ seconds. For example, the worst-case execution time of machine 6, M_6, is 24 seconds. We deferred the issues of buffer space and scheduling the transport system for future experiments.

To adequately illustrate the performance of the **HLS**-Spring system, we compared its results with two other schedulers, the "Worst-Case Scheduler" and the "Average-Case Scheduler". The Worst-Case Scheduler is discussed in Section 5.1, and the Average-Case Scheduler is discussed in Section 5.2.

To simulate the arrivals of orders to the workcell, we used an environment file containing various information. While the WCET for each machine was fixed for these experiments, the distributions for the actual durations were specified in this file. The amount of time to simulate was 10 minutes, which was sufficient to establish steady-state operation of the workcell given the loads used in these experiments. The arrival rate of each order also was specified in this file. To generate a moderately-heavy load, the arrival rate of each order was Poisson with an average interrarrival time equal to the sum of the worst-case durations of the machines in the process plan. Deadlines of each arrival were set fairly tight–deadlines were the arrival time plus two times the sum of the WCETs of the machines in the process plan. The environment file, along with parameters that were held constant for these experiments, together were used to generate a *scenario*, which is a list of order arrivals for a 10-minute period. Multiple scenarios were used to statistically evaluate the performance of a scheduler in an environment.

In all three scheduling policies, machines were scheduled to perform an operation of a process plan only if a contiguous block of time could be found. This was done to adhere to the principles of Just-In-Time philosophy, in which machine operations are very carefully planned. In order to reduce work-in-progress, which is very important in Just-In-Time, it is important to attempt to maintain an accurate view of where each process plan is currently executing. The used of non-contiguous blocks compromises the ability to regulate what orders are in queues, and what orders are currently executing. Also, it must be emphasized that we believe that, while "the right amount at the right place at the right time" is strived for, a little too early is manageable, but a little too late is unworkable. This is because machines are assumed to have a little buffer space in which to store materials. If a material transport system has been scheduled to move the material to the next machine at a very precise time, the material can wait in local storage until the material transport system arrives. However, if the order has not completed processing, the material transport system cannot wait, because it has key to the overall performance of the workcell. A second way in which we attempted to implement the Just-In-Time policy is to not schedule locally a process plan that would finish too soon before its deadline. It is assumed that there are multiple workcells, and an order that can not be scheduled to finish within a threshold of its due date is shipped to another cell. This action is taken without addressing whether or not the other cell has time to schedule this process plan or has time in its schedule for this process plan. Experiments were conducted to determine what effect the value of this threshold had on the results; there appeared to be little effect, so a value was chosen that resulted in most orders being scheduled locally for the Worst-Case Scheduler, and was fixed across all three schedulers.

There are numerous metrics used for determining performance: the number of orders successfully executed by the workcell before the deadline, the time before the deadline these are executed, the number of orders executed but completed after their deadlines, the average lateness of these orders, and the expected and actual utilization of each machine. The overall model of the manufacturing site

is not necessarily that the workcell produces finished products; rather it manufactures according to the principles of Just-In-Time. The workcell is heavily penalized if it manufactures products after the deadline of the order. Because there is limited space in which to store orders after they are completed, the amount of time that an order completes early is not as significant.

5.1 Worst-Case Scheduler

In this scheduling and control policy, activities are scheduled for machines for the entire worst-case execution time of the machine operation. If the machine operation takes shorter than the worst-case duration, the machine sets idle, and the part being processed resides in local buffer space until the end of the worst-case duration. As opposed to the **HLS**-Spring system, parts are not transported early because there's no reason to—the Worst-Case Scheduler directs the machines without feedback, so it's not going to schedule any operation to use that time. Intuitively, this scenario produces operations that begin at precisely the time at which they are scheduled (which is the prime objective of the Just-In-Time methodology). The negative aspects of this scheduling policy are that, in environments in which the worst-case machine duration is much longer than the expected duration, machine utilization can be low, and a relatively low number of process plans can be accepted for production.

Because parts are not transported early when they complete earlier than the WCET of the machine, the amount of time that the product leaves the workcell earlier than expected is equal to the amount of time that the product finishes early on the last machine in its process plan.

5.2 Average-Case Scheduler

An alternative to scheduling by only the worst-case duration of machines is to use the average-case durations instead. However, this can only be realistically done if the low-level dispatching mechanism is different than in the environment of the Worst-Case Scheduler. The rationale for using the average case is that for every machine operation that takes more than the average-case execution time, there is a machine operation that takes less than the average-case execution time. Therefore, products should be completed on time if scheduled by average-case. In the case of a single-machine process plan, the machine is not scheduled to start its machine operation at a time equal to the deadline minus the worst-case machine time, but rather at a time equal to the deadline minus the expected machine duration. Intuitively, the positive aspects of this policy are that it should increase machine utilization and allow more orders to be accepted for production as compared to the scheduler based only on the worst-case execution time. The negative aspects are that there might be multiple machines in a process plan that requirelonger than their average-case durations. This can cause the process plan to be completed after its deadline. The problem with scheduling based on the average case is that one order taking longer than average, and thus completing after its deadline, will likely cause other orders to miss their deadlines as well if the schedule is tightly packed. This can be seen by a simplified example of two orders. If O_2 is scheduled to begin executing immediately after O_1 and O_2 requires many of the machines that were used by O_1, then, if O_1 takes more than its average time, O_2 will be late even if it takes its average time.

5.3 Performance Results

In order to compare the effectiveness of the **HLS**-Spring system with the other two schedulers, we evaluated each in four different environments. Each environment is characterized by the amount of time each machine took with respect its expected average-case duration, which was always defined as one-half of its WCET. In the first environment, machines executed for exactly their average-case duration with no variance. In the second environment, actual machine times were drawn from normal distributions with mean equal to the expected mean of the machine, with variance equal to the expected mean as well. For example, each time M_6 was executed, the actual duration was drawn from a normal

distribution with $\mu = 12$ seconds and $\sigma^2 = 12$ seconds. In the third environment, machines in reality consistently required much more time than expected—actual durations were drawn from distributions of mean equal to $3/4$ WCET and variance equal to 1 second. For M_6, this meant drawing from normal distribution with $\mu = 18$ seconds and $\sigma^2 = 1$ second. In the last environment, machines consistently took much less time than expected–actual durations were drawn from distributions of mean equal to $1/4$ WCET and variance equal to 1 second. For M_6, this meant drawing from normal distribution with $\mu = 6$ seconds and $\sigma^2 = 1$ second.

Each of these environments is presented and discussed separately. For each of the four environments, 10 scenarios were generated. The average number of arrivals for each scenario is 73.9, with a standard deviation of 1.04. Each scheduler was executed on each of the 10 scenarios. The tables contain values for 10 scenarios of 10 minutes. Each entry in the table is the average over the 10 scenarios, with the standard deviation in parentheses.

	Worst-Case Scheduler	Average-Case Scheduler	HLS-Spring
On-Time Executions	38.00 (1.84)	55.40 (1.80)	42.30 (2.49)
Late Executions	0.00 (0.00)	(0.10) (0.30)	0.00 (0.00)
Average Earliness	9.21 sec. (0.16)	0.47 sec. (0.17)	19.46 sec. (0.60)
Average Lateness	0.00 sec. (0.00)	1.10 sec. (3.30)	0.00 sec. (0.00)
M_1 util (planned)	11.65% (0.98)	7.27% (0.38)	12.33% (1.08)
M_2 util (planned)	23.31% (1.96)	14.55% (0.79)	24.66% (2.12)
M_3 util (planned)	48.97% (2.93)	28.96% (1.26)	51.86% (3.11)
M_4 util (planned)	35.78% (1.86)	24.22% (1.32)	37.86% (2.67)
M_5 util (planned)	23.39% (0.96)	11.93% (0.48)	24.79% (1.01)
M_6 util (planned)	67.04% (3.75)	58.14% (2.68)	87.31% (6.01)
M_1 util (actual)	5.83% (0.51)	7.27% (0.38)	6.15% (0.52)
M_2 util (actual)	11.65% (0.98)	14.55% (0.79)	12.33% (1.08)
M_3 util (actual)	24.48% (1.46)	28.96% (1.26)	25.94% (1.57)
M_4 util (actual)	17.89% (0.96)	24.22% (1.32)	18.94% (1.33)
M_5 util (actual)	11.71% (0.46)	11.93% (0.48)	12.37% (0.51)
M_6 util (actual)	32.50% (1.87)	58.14% (2.68)	43.67% (3.00)

Table 1: Performance When Machines Take Exactly Their Expected Time

Machines that take exactly their expected time. When machines perform exactly as expected, with no variance, the Average-Case Scheduler performs the best–as shown in Table 1, it successfully completes the most orders on time (55.40), and completes on the average within a second of their deadlines (0.47 sec.). Machine utilization is half the expected value in the case of the Worst-Case Scheduler, because machines always take exactly half of the time allocated, and the Worst-Case Scheduler does not reschedule. The machines were slightly more utilized in the **HLS**-Spring system as compared to the Worst-Case Scheduler, because process plans finish in half of the time expected, and the **HLS** is able to reclaim time in the schedule that it had previously allocated to an order that completes early. This is reflected in the planned utilization of M_6: 87.31%. In the case of the **HLS**-Spring, the expected utilization reflects that a time unit on a machine can be scheduled for two different operations and thus is counted twice. In this case, it is theoretically possible for a machine to be planned for more than 100% utilization. The reason why machines are *not* more utilized in the **HLS**-Spring system is because, as machines take less than their worst-case time, the **HLS**' view of the world is increasingly less accurate—machine operations are not occurring at the same time as it expects. Therefore, when it places

an order in its schedule, and eventually submits the order to Spring, Spring often cannot guarantee the time, because another process plan is currently executing. Therefore the order is rejected. The reason why this does not cause the **HLS**-Spring to be unable to submit *any* orders is because as orders *complete*, (a) the **HLS** view of the world becomes less uncertain, and (b) the **HLS** is able to reuse the space previously allocated to the just-completed order. A reason why machines in general are not utilized more by any of the three schedulers is because of the mix of orders: given these particular arrival rates for the four process plans, M_6 is the bottleneck resource, and indirectly controls the utilization of the other five machines. Because the Spring scheduler reschedules as machines take half of their worst-case durations, process plans finish much earlier than anticipated (19.46 seconds on the average). In future work, we intend on exploiting more of the information retained by Spring to attempt to reduce this time, in an effort to reduce the amount of buffer space required for products leaving the workcell. It should be noted that is is unreasonable to expect machines to always execute for their expected durations. As such, this environment is meant solely as a baseline.

	Worst-Case Scheduler	Average-Case Scheduler	HLS-Spring
On-Time Executions	38.30 (1.18)	25.10 (4.04)	42.20 (2.52)
Late Executions	0.00 (0.00)	30.40 (4.15)	0.00 (0.00)
Average Earliness	9.79 sec.(0.54)	4.88 sec. (1.04)	17.79 sec. (1.08)
Average Lateness	0.00 sec. (0.00)	6.84 sec. (1.23)	0.00 sec. (0.00)
M_1 util (planned)	12.05% (0.78)	7.29% (0.52)	12.81% (0.91)
M_2 util (planned)	24.12% (1.54)	14.53% (1.08)	25.60% (1.82)
M_3 util (planned)	50.77% (2.64)	29.25% (1.83)	53.86% (3.27)
M_4 util (planned)	35.72% (2.02)	23.48% (1.38)	36.36% (3.11)
M_5 util (planned)	24.30% (0.87)	12.37% (0.52)	25.78% (1.11)
M_6 util (planned)	65.54% (3.18)	58.08% (3.32)	84.44% (7.09)
M_1 util (actual)	6.20% (0.65)	7.24% (0.77)	6.57% (0.66)
M_2 util (actual)	11.60% (1.86)	13.82% (2.14)	12.32% (0.66)
M_3 util (actual)	23.55% (2.71)	27.26% (3.12)	24.95% (2.75)
M_4 util (actual)	17.97% (2.07)	23.46% (3.08)	18.20% (2.59)
M_5 util (actual)	12.33% (2.51)	12.60% (2.76)	13.12% (2.87)
M_6 util (actual)	32.37% (1.29)	55.86% (4.77)	42.72% (5.07)

Table 2: Performance When Machines Take Expected Time with Large Variance

Machines that take expected time with large variance. When machines require in reality on the average a time equal to its expected average but with large variance (Table 2), the cascading effect of multiple orders missing their deadlines becomes evident in the case of the Average-Case Scheduler, which on the average executes 30.40 process plans that complete after the deadline of the order has passed. Even though there is a high utilization of machines (85.66% on M_6), this is unacceptable for the Just-In-Time approach. Because so many process plans complete after the deadline of the order, the Average-Case Scheduler is inappropriate for the operation of the workcell. In comparing the **HLS**-Spring to the Worst-Case Scheduler, the relatively large uncertainty in the view of the **HLS** is apparent. That is, the **HLS** did not execute significantly more process plans than the Worst-Case Scheduler, for the reasons discussed concerning the previous environment. In fact, both the Worst-Case Scheduler and the **HLS**-Spring system performed approximately the same as the last environment, implying that both schedulers are unaffected by variance in the execution time of machines, as long as the times are on the average as expected.

	Worst-Case Scheduler	Average-Case Scheduler	HLS-Spring
On-Time Executions	37.60 (1.27)	0.20 (0.40)	38.90 (1.14)
Late Executions	0.00 (0.00)	55.30 (2.41)	0.00 (0.00)
Average Earliness	5.25 sec. (0.17)	0.00 sec. (0.00)	9.99 sec. (1.08)
Average Lateness	0.00 sec. (0.00)	17.06 sec. (3.49)	0.00 sec. (0.00)
M_1 util (planned)	11.71% (0.87)	7.04% (0.55)	12.02% (0.86)
M_2 util (planned)	23.41% (1.75)	14.09% (1.11)	24.01% (1.76)
M_3 util (planned)	49.31% (2.71)	28.11% (1.83)	50.61% (2.73)
M_4 util (planned)	36.19% (2.40)	23.40% (1.22)	35.17% (2.03)
M_5 util (planned)	23.73% (1.04)	11.63% (0.60)	24.34% (1.12)
M_6 util (planned)	67.23% (3.91)	57.11% (3.57)	73.88% (5.89)
M_1 util (actual)	8.79% (0.83)	10.65% (0.81)	9.04% (0.82)
M_2 util (actual)	17.56% (1.36)	21.01% (1.88)	18.02% (1.36)
M_3 util (actual)	36.90% (2.19)	42.04% (3.09)	37.87% (2.22)
M_4 util (actual)	27.25% (1.84)	35.22% (1.86)	26.48% (1.53)
M_5 util (actual)	17.69% (0.83)	17.35% (0.93)	18.15% (0.87)
M_6 util (actual)	50.41% (2.87)	85.66% (5.43)	55.51% (4.32)

Table 3: Performance When Machines Take Consistently More Time Than Expected

Machines that take consistently more time than expected. The inadequacy of the Average-Case Scheduler is markedly apparent when machines take much longer than expected (Table 3): on the average, it executed less than one (0.20) process that finished before its deadline. Again, because the **HLS** does not correctly view the schedule of the low-level machinery, it is unable to significantly capitalize when process plans complete early. However, performance is still comparable to the Worst-Case Scheduler: 38.90 process plans are completed on time by the **HLS**-Spring system, while the Worst-Case Scheduler completed 37.69 process plans on time.

Machines that take consistently less time than expected. When machines take much less than expected time, the Average-Case Scheduler performs admirably, without executing any orders past their deadlines. However, the **HLS**-Spring system performs almost as well as the Average-Case Scheduler, and significantly outperforms the Worst-Case Scheduler. The only drawback to the **HLS**-Spring system is that on the average each completed order must wait a significant amount of time in the buffer before leaving the workcell (27.83 seconds vs. 14.10 seconds for the Worst-Case Scheduler). The reason why the **HLS**-Spring system significantly outperforms the Worst-Case Scheduler in this environment, and not in the other three environments, is because the inaccurate view of the actual status of an executing process plan as maintained by the **HLS** exists for much less time than in the other three environments. That is, the actual amount of time that an executing-order requires is much less. Given this, even if the **HLS**'s view is inaccurate, it is inaccurate for much less time, and thus has little ramifications on other process plans. Thus, fewer order submitted by the **HLS** are rejected by Spring.

5.4 Discussion

The importance of these four environments is that they capture phases of the normal life of a workcell. It is expected that during the normal operation of the workcell, activities will take much longer than expected, much shorter than expected, and as expected (with or without a large variance). Therefore, the expected performance of implementing either the Average-Case Scheduler, the Worst-Case Scheduler, or

	Worst-Case Scheduler	Average-Case Scheduler	HLS-Spring
On-Time Executions	38.10 (1.64)	54.90 (2.39)	52.90 (1.58)
Late Executions	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
Average Earliness	14.10 sec. (0.25)	7.84 sec. (0.36)	27.83 sec. (0.77)
Average Lateness	0.00 sec. (0.00)	0.00 sec. (0.00)	0.00 sec. (0.00)
M_1 util (planned)	11.77% (0.92)	7.50% (0.42)	13.82% (1.00)
M_2 util (planned)	23.56% (1.79)	14.97% (0.84)	26.62% (1.97)
M_3 util (planned)	49.76% (2.55)	30.02% (1.25)	60.14% (2.76)
M_4 util (planned)	36.10% (3.49)	24.47% (1.65)	54.26% (2.82)
M_5 util (planned)	24.10% (0.78)	12.57% (0.39)	36.21% (1.87)
M_6 util (planned)	68.78% (2.98)	60.25% (2.95)	114.01% (13.07)
M_1 util (actual)	2.79% (0.46)	3.67% (0.36)	3.23% (0.49)
M_2 util (actual)	5.77% (0.46)	7.30% (0.48)	6.48% (0.49)
M_3 util (actual)	12.30% (0.99)	14.92% (0.76)	14.48% (2.08)
M_4 util (actual)	8.97% (1.00)	12.27% (1.07)	13.32% (2.04)
M_5 util (actual)	6.02% (0.26)	6.27% (0.26)	9.53% (1.29)
M_6 util (actual)	17.28% (0.86)	30.33% (1.75)	32.4% (3.91)

Table 4: Performance When Machines Take Consistently Less Time Than Expected

the **HLS**-Spring scheduling system should be evaluated across all four environments. The Average-Case Scheduler has been shown to perform particularly poorly when machines durations are much longer than the expected time (only 0.20 process plans were completed on-time in the third environment). Hence, the Average-Case Scheduler should not be used. The Worst-Case Scheduler performs reasonably well in all environments but results in low resource utilization, particularly when the average duration of a machine execution is much less than expected (in the last environment, the actual machine utilization of M_6 was only 17.28%). The **HLS**-Spring is appropriate for controlling a flexible manufacturing workcell under these conditions, because it is effective in all four environments. In terms of on-time executions, it performed as well as the Worst-Case Scheduler in three environments, and much better in the fourth (52.90 vs. 38.10), and does not fail as the Average-Case Scheduler did in two of the environments. During the normal life of a workcell, in which the workcell can operate in each of the four environments, the **HLS**-Spring effectively manages the machine operations.

In addition, we foresee ways in which the performance of the **HLS**-Spring system can improve performance. As mentioned, the machine utilization can be improved if the **HLS** uses more of the reflective properties of Spring to improve its current state of the machinery. As discussed in Section 4.2, Spring has the capability of providing more information as feedback. As the usage of this feedback is developed in the **HLS**, the view of the current state of the world by the **HLS** will more reflect the true state of the manufacturing environment. This will complicate the scheduling process of the **HLS**, but should result in fewer process plans being rejected by Spring. With higher resource utilization and more process plans executed, the advantage of the **HLS**-Spring system becomes more apparent.

6 Conclusions and Future Work

We have shown that the Spring real-time operating system can be used successfully in a dynamic flexible manufacturing setting. The reflective properties of Spring enable it to either respond and adapt quickly, or provide timely information to an outside agent such as the High Level Scheduler, which can then guide the response of the system.

For each of four instances of a representative manufacturing environment, the **HLS**-Spring system performs as well as or better than the single scheduler that bases its decisions on the worst-case execution time of machines. In the fourth case, the **HLS**-Spring system significantly outperformed the Worst-Case Scheduler. It has been shown that scheduling machine operations in this environment based on average-case performance can in general lead to poor performance.

There are three main directions in which the research will continue. The first direction is to make a more robust model of the flexible manufacturing environment. This includes removing some of the simplifications imposed on the workcell itself, as well as investigating distributed scheduling across multiple workcells. Some features we anticipate adding to the model of the workcell include limited buffer space, a non-trivial transport system, and machines that can perform more than one operation (multiple machines can perform the same operation, as well.) Both the High Level Scheduler and the Spring kernel are capable of performing distributed scheduling, so it will be interesting to investigate which is more appropriate given the semantics of the domain. The second direction we will pursue is to more closely study the two-level scheduling system in hopes of determining how the system performs as a function of the amount and type of information passed between the two levels. Thirdly, the postulated improved performance when the **HLS** uses Spring's ability to reply with more detailed status information regarding executing process plans will be investigated. This should lead to improved performance in each of the first three environments, in which the **HLS** Spring system performed only marginally better than the Worst-Case Scheduler.

Acknowledgment

The author wishes to thank Professor Jack Stankovic for his important and constructive comments on earlier drafts of this paper.

References

[1] R. Akella, Y. Choong, and S.B. Gershwin. Performance of hierarchical production scheduling policy. *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, CHMT-7(3):225–240, September 1984.

[2] J. Brown, D. Dubois, K. Rathmill, S. Sethi, and K.E. Stecke. Classifications of flexible manufacturing systems. *FMS Magazine*, pages 114–117, April 1984.

[3] J. A. Buzacott and J. G. Shanthikumar. Models for understanding flexible manufacturing systems. *AIIE Transactions*, 12(4):339–349, December 1980.

[4] J. A. Buzacott and D. D. Yao. On queueing network models of flexible manufacturing systems. *Queueing Systems*, 1:5–27, 1986.

[5] T.C.E. Cheng and S. Podolsky. *Just-in-Time Manufacturing: An Introduction*. Chapman & Hall, London, 1993.

[6] M.S. Fox. ISIS: A retrospective. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, pages 3–29. Morgan Kaufmann, San Francisco, 1994.

[7] S.B. Gershwin, R.R. Hildebrant, R. Suri, and S.K. Mitter. A control perspective on recent trends in manufacturing systems. *IEEE Control Systems Magazine*, MCS-6(2):3–15, April 1986.

[8] S.C. Graves. A review of production scheduling. *Operations Research*, 29(4):646–675, July–August 1981.

[9] K. Hadavi, W.-L. Hsu, T. Chen, and C.-N. Lee. An architecture for real-time distributed scheduling. *AI Magazine*, 7(4):45–61, Fall 1992.

[10] R. Hall. *Zero Inventories*. Dow Jones-Irwin, Homewood, Illinois, 1983.

[11] F. R. Jacobs. OPT uncovered: many production planning and scheduling concepts can be applied with or without the software. *Industrial Engineering*, pages 32–41, October 1984.

[12] M.V. Kalkunte, S.C. Sarin, and W.E. Wilhelm. Flexible manufacturing systems: A review of modeling appraoches for design, justification and operation. In Andrew Kusiak, editor, *Flexible Manufacturing Systems: Methods and Studies*. North-Holland, New York, 1986.

[13] M.H. Kim and Y.-D. Kim. Simulation-based real-time scheduling in a flexible manufacturing system. *Journal of Manufacturing Systems*, 13(2):85–93, 1994.

[14] D.J. Musliner, E.H. Durfee, and K.G. Shin. CIRCA: A cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6), 1993.

[15] J. Orlicky. *Material requirements planning*. McGraw-Hill, New York, 1994.

[16] I.M. Ovacik and R. Uzsoy. Exploiting shop floor status information to schedule complex job shops. *Journal of Manufacturing Systems*, 13(2):73–84, 1994.

[17] S.S. Panwalkar and Wafik Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, January–February 1977.

[18] K. Ramamritham, J.A. Stankovic, and P.-F. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):184–195, April 1990.

[19] F.A. Rodammer and K. P. White, Jr. A recent survey of production scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(6):841–851, November 1988.

[20] N. Sadeh, S. Otsuka, and R. Schnelbach. Predictive and reactive scheduling with the Micro-Boss production scheduling and control system. In *Proceedings, IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling and Control*, Chambery France, August 1993. International Joint Conferences on Artificial Intelligence (IJCAI).

[21] M. Sepehri. *Just-in-Time, not Just in Japan*. American Production and Inventory Control Society (APICS), Falls Church, VA, 1986.

[22] S.F. Smith, P.S. Ow, N. Muscettola, J.-Y. Potvin, and D.C. Matthys. OPIS: An integrated framework for generating and revising factory schedules. *Journal of the Operational Research Society*, 41(6):539–552, 1990.

[23] J.A. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 21(10):93–108, 1988.

[24] J.A. Stankovic and K. Ramamritham. The Spring kernel: A new paradigm for real-time systems. *IEEE Software*, 8(3):62–72, May 1991.

[25] J.A. Stankovic and K. Ramamritham. A reflective architecture for real-time operating systems. In Sang Son, editor, *Principles of Real-Time Systems*. Prentice-Hall, 1994.

[26] O.W. Wight. *Manufacturing resource planning: MRP II: unlocking America's productivity potential*. Van Nostrand Reinhold, New York, 1984.