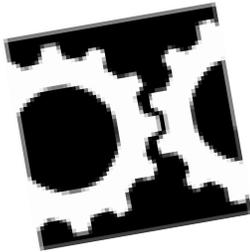


An Architecture for Real-Time Distributed Scheduling

Khosrow Hadavi, Wen-Ling Hsu, Tony Chen, and Cheoung-Nam Lee



Industrial managers, engineers, and technologists have many expectations from artificial intelligence and its application to knowledge-based systems. Although the past decade has witnessed a number of innovative applications of AI in manufacturing, the field is still in its infancy and holds even greater promise for the future.

The AAAI Press book *Artificial Intelligence Applications in Manufacturing*, (from which the following article was selected) presents a number of articles that relate to the enhancement of planning and decision making capabilities in today's automated production environments.

Scheduling problems can generally be described as allocating resources to tasks while satisfying a set of constraints (Baker 1974; Conway et al. 1967). More often than not, the constraint sets are large and diverse, the objectives conflict with each other, and the scheduling problems quickly become NP-hard. Moreover, there is the added complexity of the dynamics and the unpredictability of the environment.

Even if one could arrive at a static solution to a problem, the solution might quickly become obsolete (McKay, Safayeni, and Buzacott 1988). A typical dynamic environment is found in factory scheduling. Machine breakdown, product quality assurance, supplier delivery, operator availability, and continual arrival of new orders can all be part of the dynamics of the environment. Decision making for scheduling can also occur on many different levels: weekly to quarterly capacity planning; daily to shift-level detailed scheduling; and on-the-spot reactions to machine breakdowns, shortage of supplies, and so on. Therefore, it is desirable for a scheduling system to be able to predict, as

well as react to, the situations (Ow and Smith 1988; Smith and Ow 1985). As the dynamism of the environment increases, it becomes more difficult to distinguish between predictive and reactive behavior, so the need for a real-time system is felt (Burke and Prosser 1989).

The scheduling objectives typically include meeting committed shipping dates of the orders, reducing the lead times and work-in-process (WIP) and finished goods inventories, maximizing use of expensive equipment and the throughput of the system, and minimizing the sensitivity of the schedule to random events. These objectives sometimes conflict. For example, the urge to meet the due dates encourages the scheduler to start the jobs as early as possible. However, to minimize the WIP and finished goods inventories, the scheduler attempts to postpone starting the jobs as long as possible. Finally, to maximize the throughput of the system, the scheduler tends to ensure that the bottleneck resources are never idle, but to make the schedule flexible, the scheduler is encouraged to distribute small amounts of slack throughout the pro-

cess plan of every order (Pinedo et al. 1991). These contradictions make it even more difficult to evaluate a given schedule. Therefore, it is desirable to have a scheduling system that is capable of observing the environment from different perspectives that stem from the different objectives. If such observations can be performed on a real-time basis, the different perspectives can provide immediate feedback to the scheduler and achieve better control of the manufacturing environment. A distributed architecture will best serve the need for providing multiple perspectives on a real-time basis.

Of course, in addition to working in real time and providing multiple perspectives, a scheduling system must also include the following capabilities:

Constraint Satisfaction: Satisfy or, at least, attempt to satisfy various classes of constraints, ranging from management objectives to tool availability and vendor tardiness. This feature would indeed entail the ability to represent all the different classes of constraints.

Flexibility: Adjust or reconfigure the schedule when and if a perturbation changes the situation.

Fuzzy and Incomplete Information: Function with incomplete or uncertain information, take into account fuzzy information such as “high priority,” and find or propose a schedule in the absence of complete information. (For example, vendor delivery dates might be approximate.)

Fast Response Time: Find a new schedule as quickly as possible when the situation changes. This capability is important for supporting interactive use.

Abstraction of Information: Coordinate the knowledge extracted from expertise residing in various parts of the manufacturing environment. Appropriate information should be presented by the system as decision

Production planning and scheduling has been one of the most difficult and challenging tasks for manufacturing management. Despite the advances in scheduling theory, many actual scheduling problems are still too complex to yield analytic solutions. Much of the difficulty stems from having to deal with a large set of diverse constraints and multiple objectives that are often conflicting and ill defined. Moreover, the dynamic and stochastic nature of the environment further contributes to the complexity of the task. In recent years, the AI community has investigated factory scheduling in depth, and various paradigms have been presented. However, most of this research focuses its efforts on heuristics and constraint satisfaction; little attention has been paid to the need for a real-time, distributed scheduling system. In this article, we discuss the importance of such needs and present a recursive architecture for real-time distributed scheduling. The new architecture, composed of planning agents, has a number of advantages over the conventional hierarchical, distributed, and subsumption architectures. Finally, we present the design and implementation of a system, called REDS (requirement-driven scheduling), and discuss how this design evolved. The application of REDS to a wafer fabrication factory is also discussed.

support when and where it is needed.

User Friendliness: Provide a friendly interface to both humans and computer systems.

Good Schedules: Produce schedules that are nearly optimal according to the objectives.

In this article, we present a distributed architecture for real-time scheduling, called REDS² (requirement-driven scheduling), and describe how this architecture evolved. The distributed nature of the architecture provides the capability to attend to multiple perspectives simultaneously.

In the next section, we summarize the

relevant approaches to scheduling in manufacturing. We then describe the kinds of scheduling problem we address and the principles used in REDS². Next, we discuss the conceptual framework and architectures. Finally, we describe the implementation of REDS.

Previous Approaches to Scheduling in Manufacturing

Both operations and AI researchers have suggested numerous approaches for solving factory scheduling problems. For decades, most of the scheduling literature presented analytic approaches. These approaches, aiming at optimal solutions, proved to apply to only a small subset of scheduling problems, those with highly idealized conditions and a small number of jobs and machines (Grant 1986; Graves 1981). Typical assumptions that were made include (1) the tasks to be performed are well defined and completely known; (2) resources and facilities are entirely specified; (3) sequences for tasks are well defined; (4) each machine is continuously available; (5) there is no rework; (6) each operation can be performed by only one machine on the shop floor; (7) there is no preemption; (8) the processing times of successive operations of a particular job might not overlap; and (9) each

...It is desirable for a scheduling system to be able to predict, as well as react to the situations...

In REDS, both predictive and reactive scheduling are incorporated...

machine can handle, at most, one job at a time.

Although some of the assumptions might be reasonable in a given situation (for example, 1, 2, 3, and 7), the remaining assumptions might not. Most often, schedules generated with these idealized assumptions are inadequate in reality. In addition, problems dealt with in the analytic literature are often restricted to a small number of jobs and machines. In the real world, the numbers are significantly larger, and standard techniques such as enumeration, integer programming, or branch and bound quickly become computationally impractical. However, the theoretical aspects of this type of research can still contribute substantially to the design of a scheduling system. This contribution includes determining the computational complexity of the problems and developing efficient algorithms.

Some techniques derived from empirical research, such as the apparent tardiness cost (ATC) heuristic (Vepsalainen and Morton 1987) and the shifting bottleneck heuristic, also influenced later work by AI researchers in manufacturing scheduling (Adams, Balas, and Zawack 1988). In recent AI research, heuristics and rich domain knowledge have been used to find satisficing solutions to scheduling problems whose size and complexity are more realistic (Kanet and Adelsberger 1987; Hadavi, Shahraray, and Voigt 1990; Hsu et al. 1990; Mer et al. 1986).

Constraint satisfaction has been a dominant issue among these methods. Systems such as ISIS view scheduling from a single, order-based perspective (Fox, Allen, and Strohm, 1982; Fox 1983; Fox and Smith 1984) and use knowledge-intensive search techniques along with a rich constraint representation. OPIS addresses the weakness identified within ISIS by providing multiple perspectives with the ability to dynamically shift between perspectives. OPIS outperformed schedulers that relied on a single perspective (Ow and Smith 1988). However, by scheduling all the operations of a critical order or a critical resource, OPIS assumes that all these operations are critical. Rarely does one entire job or resource remain critical indefinitely

(Burke and Prosser 1989). Some resources only become critical over certain time intervals. Job criticality often occurs only over a subset of the operations of the orders (Pinedo et al. 1991; Burke and Prosser 1989). Furthermore, at any given moment, OPIS only considers a single perspective.

Even though the concept of a shifting bottleneck (Adams, Balas, and Zawack 1988) was discussed in Smith et al. (1986), it was never addressed in OPIS. In the Cortex project (Sadeh and Fox 1989a), the activity-based approach was proposed. This approach uses multiple perspectives and considers shifting bottlenecks by lookahead techniques. However, this approach has never been implemented in a real system.

DAS (Burke and Prosser 1989) is a planning and scheduling system formulated as a layered and distributed asynchronous system. It decomposes the scheduling problems across a hierarchy of communicating agents. DAS does not differentiate between prediction and reaction because one might consider scheduling a task that requires continuous reaction. Much of the problem-solving effort focuses on conflict resolution. Meeting due dates is the primary scheduling objective and is treated as a preference constraint. Even though various dispatching strategies are included in DAS, the scheduling process does not pay much attention to other objectives, such as machine use, inventories, and cycle times.

Finally, neither the AI community nor the operations research community has paid much attention to order release control (Glasse and Redsende 1988). In most of the literature, no distinction is made between job arrival and job release, even though it has been proven that job release control plays a significant role in producing good schedules (Glasse and Redsende 1988). With good job release control, it is possible to reduce job waiting times and WIP and finished goods inventory levels. Good job release control also helps to shorten cycle times but still allows due dates to be met (Hadavi and Shahraray 1989).

In REDS, both predictive and reactive scheduling are incorporated along with a release control strategy called FORCE (factory order release control and evaluation) (Hadavi and Shahraray 1989). REDS also provides an order perspective and a resource perspective in real time. In the next section, we present the principles used by REDS² and discuss how the architecture of REDS² has evolved from requirement-driven scheduling to real-time distributed scheduling.

The Scheduling Problem

REDS is designed to work in a variety of different environments, such as batch manufacturing and job shops. REDS has been used in factories such as a very large-scale integrated development line and a job shop mask manufacturing plant. Prototype REDS systems have also been implemented for a printed circuit board (PCB) assembly factory and a plastic molding shop. The scheduling tasks that REDS deals with include capacity planning, finite capacity scheduling, and sequencing for each machine. In addition to the multiple objectives and the diverse set of constraints mentioned earlier, REDS also needs to deal with other dynamics, such as process plan changes, rework, and dynamic test lots.

The Principles

The primary goal of REDS is to perform the scheduling task robustly but still meet the management objectives. Robustness implies that the schedules generated are feasible and that the scheduler can react to changes on the shop floor by minimally revising existing schedules. The management objectives are meeting due dates, reducing WIP and finished goods inventories, reducing product cycle times, and maximizing machine use. REDS integrates both AI and operations research techniques in its design. The major design principles used in REDS are discussed in the following paragraphs.

Abstraction of Constraints and Time: The fundamental structure used for representing constraints and schedules is a temporal tree structure, as shown in figure 1. Each node corresponds to an interval of time, and its children nodes correspond to subintervals of this interval. The intervals corresponding to the children of a node form a partition of the interval corresponding to the node. Each type of resource has its own temporal constraint tree. The schedule is also represented as a temporal schedule tree.

In a temporal constraint tree, each node i has an associated constraint pool $CP(i)$. For every descendant j of i , a mapping, a_{ji} , exists from $CP(j)$ to $CP(i)$. This mapping is referred to as an *abstraction function*. Constraint pools at different levels of the tree reflect different levels of abstraction. To plan activities farther in the future, REDS checks the constraints at higher levels of abstraction, but to make schedules for the near future, REDS goes down to the detailed constraint pools and makes sure the schedule is feasible (Hadavi, Shahraray, and Voigt 1990).

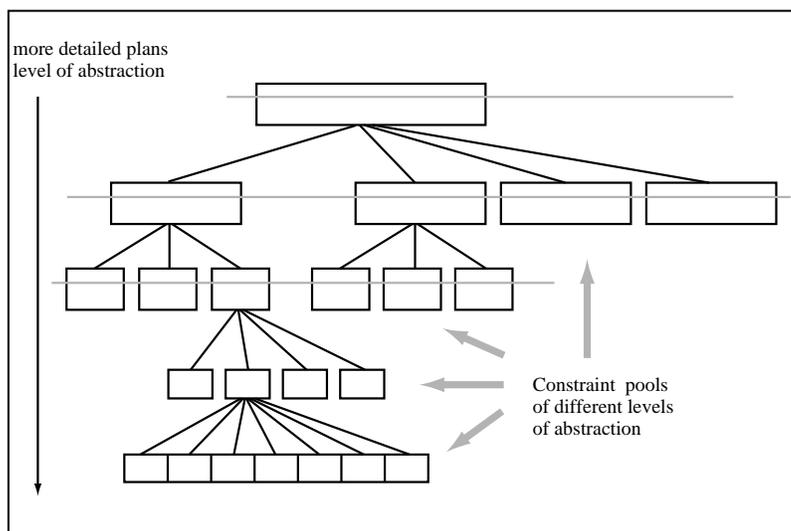


Figure 1. A Temporal Constraint Tree.

Abstraction of Orders: After an order enters the system and before the order is ready to be released, REDS performs a preprocessing step on the order. This step sets up an interval of possible starting times for the order and creates an essence function for the order. The *essence function* of an order is defined by the critical resources required to complete the order and the time intervals required for the critical resources to meet the desired due date. For example, time needed at bottleneck work centers, inventories, and special jigs and fixtures that are needed can contribute to the essence function of an order (Hadavi, Shahraray, and Voigt 1990). This essence function is an abstraction of the order.

Release Control Strategy: REDS incorporates FORCE, its release control strategy, into its scheduling process. FORCE tries not to release a job to the shop floor until it is certain that the job will be worked on. The primary purpose is to reduce the cycle times as much as possible. The concept of *continuity index*, or ci , a function of the processing time of every step allocated in the shop and the expected completion time for an order, is introduced in FORCE. Each order has a set of continuity indexes associated with it throughout its life cycle. These indexes change over time according to the dynamics of the manufacturing environment. FORCE also changes its releasing strategy accordingly, by observing the aggregated *continuity indexes profile*. In essence, this profile is an index of aggregated manufacturing parameters. We performed an extensive study to find the optimal release strategy. A

The event handler is the center of the system.

more detailed description of FORCE can be found in Hadavi and Shahraray (1989).

Constraint-Directed Reasoning with Multiple Perspectives: The temporal constraint tree of each resource provides a single resource profile, and the aggregated temporal constraint tree provides an aggregated resource profile for each type of resource. This information provides a resource perspective for the scheduling system. Using this perspective, we implemented the capacity watcher in the distributed version of REDS, namely REDS². A *capacity watcher* continuously observes the capacity of all the resources and sends warning messages when certain resource overloading is detected. The scheduler then can take either preventive or reactive measures, depending on when the overloading occurs. For example, when a machine overload is predicted, the load-leveling strategy is taken, and such an overload is avoided (Hadavi and Lehnnert 1987). By examining all the required critical resources in the essence function, an order perspective is provided to the scheduler. The scheduler then determines the earliest feasible schedule for an incoming order.

In contrast, FORCE uses continuity indexes to ensure an order is not released too early. With this order perspective, the order watcher was implemented in the distributed version of REDS. The order watcher continually observes and updates the order's status. When an event occurs, a reactive action is triggered at the most relevant level, and the effect of the event is either propagated or accommodated. Such an event can be a new order entry, a machine breakdown, inventory shortages, or a machine overload.

For example, in the event of a machine breakdown, the capacity watcher signals the event handler to pay attention to the affected orders; the order watcher updates the order's status. If the status of the order changes significantly, the effect is propagated, and some orders with lower priority might have to be delayed or suspended. Conversely, when a critical resource is underused, some new orders might be released.

Conceptual Framework of REDS

The basic conceptual framework of REDS divides the scheduling task into four subtasks, each performed by a separate module: (1) preprocessor (PPR), (2) feasibility analysis (FA), (3) detailed scheduler (DS), and (4) sequencer (SEQ). To make the concepts more concrete, we illustrate here what actually happens when a new order enters the system.

For detail on this topic, the reader is referred to Hadavi, Shahraray, and Voigt (1990). There are two modes of REDS operation: predictive and reactive. First, we explain how the four subtasks work under the predictive mode and then how the reactive mode operates.

Predictive Operations

At the time of order entry, REDS tries to fit the new order into the existing schedule by maintaining the stability of the schedule as much as possible. A schedule is more stable if it requires fewer changes to insert a new order.

The PPR module performs a backward scheduling of this order by coordinating all the sublots and the merge points. It also checks for some important constraint violations. PPR acts as an online editor to make sure that none of the requirements posed by the order is unrealistic.

After preprocessing, the FA module constructs the essence of the order. The order's essence is an abstraction of all its critical resource requirements, including bottleneck work centers, inventory items that are expensive and hard to get, and tooling. During the FA phase, if used interactively, the system maintains a dialogue with the user to find ways to meet the order requirements. This task is done by relaxing certain constraints, such as setting due dates, adding shifts, changing the quantities, and so on. If the system is not used interactively, REDS itself makes a reasonable decision based on some cost procedure that seeks the lowest cost.

The FORCE control release strategy is embedded in the FA module. FORCE is an algorithm that tries to find near-optimal release dates for the incoming orders. It attempts to maximize machine use as well as the percentage of the orders completed on time (Hadavi and Shahraray 1990).

In summary, FA can produce one of the following results: (1) No serious problems are detected. (2) Some problems are detected; relaxing some constraints will solve the problems. The required relaxation is recommended. (3) Problems detected are so serious that it would be unwise to fulfill this order.

After FA, DS works with the recommendation given by FA and produces a more detailed schedule by checking all the lower constraint levels that FA overlooked. By this point, FA should have removed all the critical problems.

This more detailed schedule is constructed by DS at lower levels of the time horizon. For example, if FA builds a schedule at the monthly level, DS will define a portion of the sched-

ule at the daily or the shift level. DS operates based on the principle of *least commitment planning*: In the immediate future (for example, the next few hours), the schedule is produced at the shift level, whereas in the farther future (for example, the next few weeks), only a weekly schedule is maintained. Hence, only a limited part of the schedule is committed.

This strategy makes it easy to change the schedule and reduces the probability of backtracking when unexpected but inevitable events occur that require revising the schedule. It also prevents too much effort from being spent on preparing detailed schedules that might soon become obsolete because of situation changes on the shop floor.

Finally, the order, blended with all the other orders in the detailed schedule, is passed to SEQ. This detailed schedule is a set of bindings of operations, resources, and a time period. Consequently, given a time period and a resource, a list of assigned operations exists. SEQ's task is to determine which of the operations listed for the current time period should be dispatched next.

SEQ contains a lot of domain-specific information regarding lot combination, setup times, choices of machines to use, and ways to avoid problems (for example, rerouting). The objectives of SEQ are to optimize the throughput, maximize machine use, and minimize flow time and WIP. A *dynamic sequencing rule* (DSR) was developed to determine an effective priority rule for sequencing. By applying DSR, sequencing decisions are made in real time to respond to constantly changing conditions within manufacturing environments. (See Shahraray [1987] for a more detailed description of DSR.) The schedule produced by SEQ contains minute-by-minute operations on the shop floor. The output of SEQ goes directly to the operators on the shop floor.

In summary, REDS forms the abstraction of orders and resources during the preprocessing phase. Meeting due dates is the major objective in this phase. FA provides release control in the hope of meeting other management objectives, such as reducing cycle times and WIP and finished goods inventories yet meeting due dates. FA also pays attention to the most critical constraints and leaves the rest of the constraints for DS to check. DS assures the validity of the schedule. SEQ receives the dispatch list in real time and makes locally optimal decisions with regard to overall management objectives.

Reactive Operations

In the reactive mode, all four modules listen

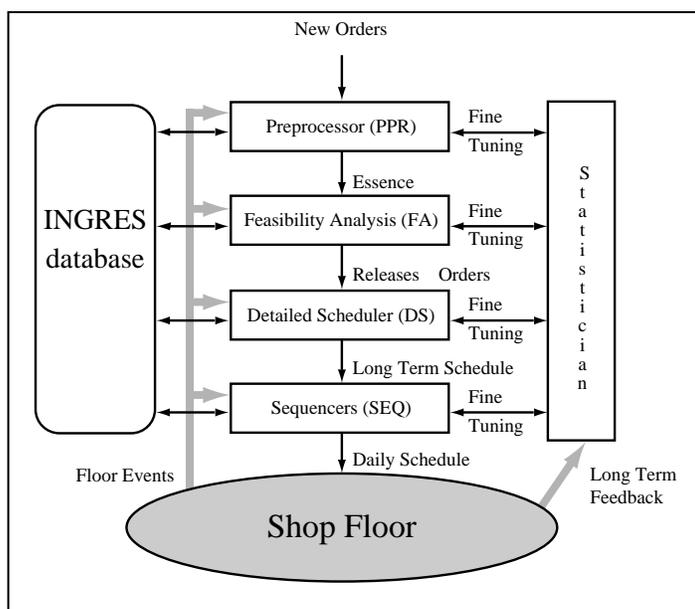


Figure 2. REDS System Architecture.

to the disturbances coming from the shop floor. Each one behaves in its own way depending on.. the complexity of the event and its horizon of impact. Their reaction times are all real time in the sense that from the user's point of view, no noticeable delay is observed. Needless to say, the actions taken by the higher-level modules respond to longer-term impacts. For example, in case of a machine breakdown, SEQ immediately tries to reroute lots and change the priority values of the affected lots. In the mean time, the planning module, DS, wakes up and starts checking on the impact of the breakdown. If necessary, the schedule is reconfigured by DS. FORCE also tries to see how much of a problem this breakdown causes as far as future orders are concerned. If necessary, it limits the entry of new incoming orders until the problem is solved.

It should be noted that all these actions are going on independently and autonomously to ensure that both short-term and long-term impacts of the problem are addressed. The average rate of events arriving from the shop floor is on the order of 50 to 100 events each minute. However, at certain periods during the day, the rate can be as low as 10 to 20 events each minute. The distribution is far from uniform and is closer to a hyperexponential distribution. These events include machine loading, unloading, breakdown, maintenance rework, scrap, and new orders.

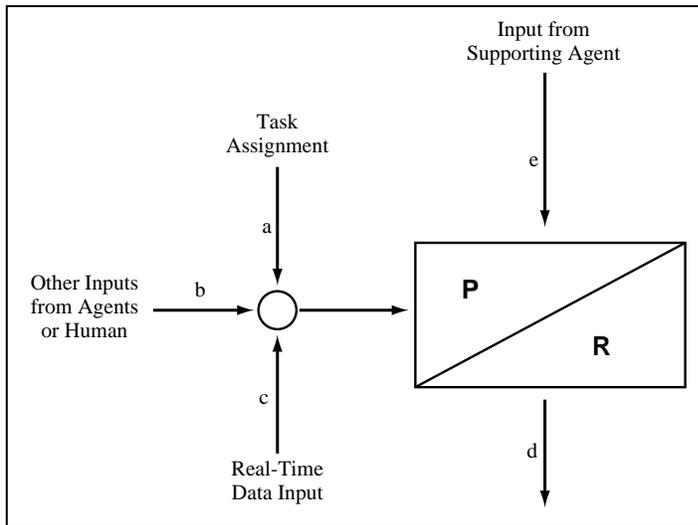


Figure 3. A Planning Agent.

The Architecture

In this section, we discuss the two versions of REDS: REDS, the first version of the system, and REDS², the current distributed system.

First Generation of REDS

Two versions of REDS have been implemented using the same conceptual framework. Architectures used in both versions can be viewed as shock absorbers in the sense that disturbances at lower levels of the temporal resource abstraction tree are dampened at low levels of the hierarchy and do not often propagate to the higher levels. The architecture of the first version is shown in figure 2.

The REDS architecture is roughly hierarchical; each module corresponds to the scheduling task described in the conceptual framework. However, it is not a strict hierarchy because information feeds back directly to different levels of the hierarchy without traversing the intermediate nodes. The lowest decision-making level reacts to events first and remedies the situation if possible. Because higher decision-making levels deal with higher levels of abstraction, the higher decision-making levels intervene when the original abstraction is disturbed beyond remedy by incoming events. The intent of the *statistician module* is to monitor, analyze, and correct long-term shop floor problems. The implementation of the first REDS ties the statistician to the sequencer only.

Problem corrections can be proposed by the statistician in two ways: (1) signaling different modules of REDS about a newly found

bottleneck work center or (2) alerting management about certain quality problems. For example, a machine that is down for more than 60 percent of the time is reported. (For a more detailed description of REDS, see Hadavi, Shahraray, and Voight [1990]).

Although this architecture was effective for its applications, multiple perspectives of the shop floor could not be maintained simultaneously. At any given moment, REDS can pay attention only to a single perspective. We felt that the dynamics of the environment could not be reflected in a timely fashion. Therefore, a real-time distributed architecture was designed for REDS² using the same conceptual framework.

REDS²: Requirement-Driven Scheduling and Real-Time Distributed Scheduling

With REDS², we introduced the concept of using a planning agent (PA) as a generic structure for designing each module. Each agent has its own clearly defined task. With a consistent structure for each module, we proposed a recursive structure for the overall design. Because it is often difficult to categorize a system into a single class of architecture, we describe REDS² as a distributed system with an implicit hierarchy.

Most systems are really hybrid, exhibiting different architectures when different aspects are examined. The subsumption architecture (Brooks 1986) is a typical example. It appears to be hierarchical when used for control, but each layer can function as an independent autonomous agent when necessary. In some situations, one layer might exercise control over the others by inhibiting communications. The major difficulty with this architecture is that each layer needs to have perfect information about the responsibility and the authority of the other layers to take control without causing chaos.

With REDS², each PA consists of a scheduling gene and a number of input and output. A scheduling gene has two components: a predictive element and a reactive element. The *predictive element* performs its routine task, and the *reactive element* reacts to irregular incoming events.

Each PA is an independent and autonomous process. Its task is assigned by agents from the module on the next-higher level. Input for each PA include signals from the shop floor, other agents, or humans. Each PA also receives input from the statistician module. Finally, the output of PA is sent to the immediate lower-level module. The output is observed by the immediate higher-level module and the statistician module without

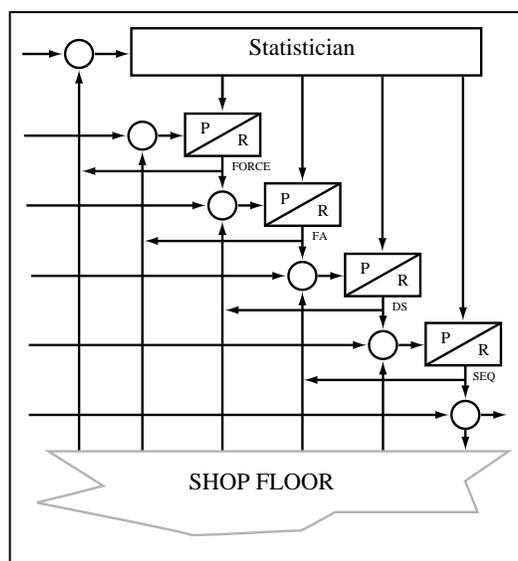


Figure 4. The Recursive Structure.

the awareness of PA. Figure 3 depicts the PA structure.

In figure 4, a recursive structure of PAs is embedded in the REDS conceptual framework. Each module (PPR, FA, DS, and SEQ) is an instance of PA, operates autonomously, and corresponds to a distinct independent software process. An implicit hierarchy exists among PAs. Although similar to the subsumption architecture in a hybrid system, our architecture works better because every module has a well-defined responsibility, reacts to incoming signals independently, and communicates with the others through a broadcast channel. Each module is unaware of the existence of the other modules. Coordination is much more efficient when no agent takes control over the others.

Based on the concepts of PAs and requirement-driven scheduling, we implemented a real-time distributed scheduling system. In the next section, we describe the scheduling system REDS² and its functions.

The System: REDS²

REDS² is implemented as an open system. It communicates with its external environment through a commercial database. It interacts with users and allows interactive scheduling. REDS² can also change its behavior at the request of a system manager during run time.

REDS² Kernel

In this subsection, we describe the REDS² system kernel and its external environmental interface.

The system kernel includes six system-level servers: three scheduling servers and three communication servers. The scheduling servers are the *order watcher*, the *capacity watcher*, and the *event handler*. The communication servers are the *data collector*, the *shop floor controller*, and the *order-entry handler*. Interfaces with the external environment include three user-level interfaces and the interface with the WIP tracker. COMETS, a commercial software product, is used as the WIP tracker. Figure 5 shows the functional diagram of REDS².

REDS² Operation. A new order arrives in the system through order entry and is sent to the order handler. The order handler performs the preprocessing tasks on this order. The output is then sent to the event handler. Some of the information (for example, process plan, routing) is also sent to the WIP tracker. The event handler takes the order with preprocessed information and stores it in a job pool. When lot move in or move out occurs, the event handler triggers the order watcher. The sequencer is a part of the event handler. It generates the dispatch list in detail for every machine.

Both the sequencer and the capacity watcher can request a job release by signaling the order watcher. The order watcher then attempts to release a job from the job pool by considering resource capacities provided by the capacity watcher. When the release conditions are satisfied, the order watcher releases a job to the sequencer; otherwise, it waits until the conditions are satisfied.

The Scheduling Servers. The order watcher and the capacity watcher perform the most important tasks for planning and scheduling. The order watcher tracks all orders and their continuity indexes. It determines when to release a job as well as which order to revoke when necessary. It consists of two modules: the planning server and the scheduling server. The planning server deals with higher levels of the temporal schedule tree, and the scheduling server deals with lower levels of the tree. Detailed constraints such as lot process route are considered by the scheduling server.

The capacity watcher updates the remaining capacity of every work center. Whenever a capacity overload or underload is detected by the capacity watcher, it sends a request for load balancing to the planning module of the order watcher. Together, the capacity watcher and the order watcher provide the profiles for resources and orders. REDS² allows

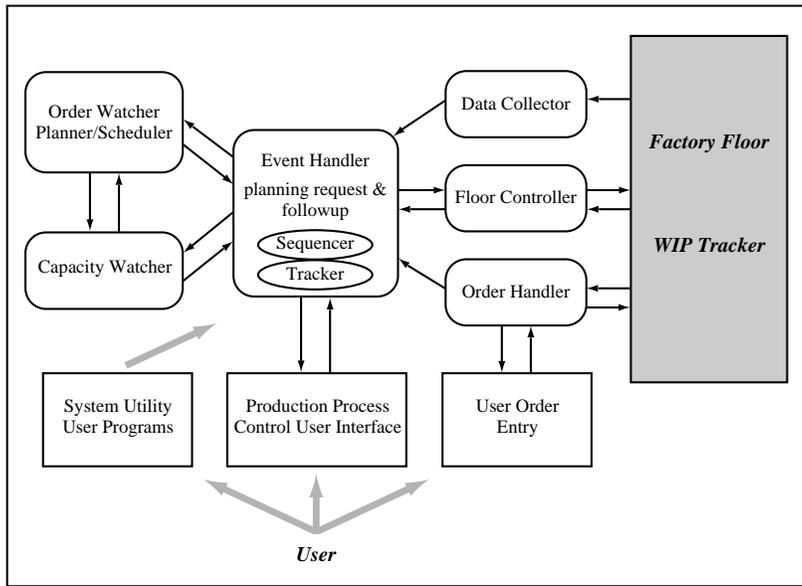


Figure 5. REDS² System Architecture.

multiple planning servers and capacity watchers to operate in parallel. This approach gives sufficiently good performance to allow scheduling to be performed in real time.

The event handler is the center of the system. It communicates with other servers and the shop floor. It handles lot tracking; sequencing; shop floor exception handling; and *housekeeping*, tracking all the shop floor information and updating the database. When an exception such as a routing change occurs, it relays the exception information to the appropriate server. The sequencer, embedded in the event handler, generates a lot dispatch list on request and sends it to the shop floor through the shop floor controller. The sequencer uses DSR in dispatching.

The event handler has an interpreter that any user can employ to describe process operation requirements. This interpreter takes a *functional script* as input and interprets it using a preloaded user library or other external programs (a *script* is a group of function calls in a Lislike language). This approach enhances system adaptability and maintainability for the following reasons: First, each function represents a knowledge chunk. This chunk can be an operational strategy, a set of heuristics, or routine data. Therefore, the factory-specific control knowledge can be captured in a script that is separate from the REDS kernel. This knowledge can be modified by changing the script while REDS² is running. Furthermore, metascheduling strategies can

be put into a script so that various scheduling strategies can be tried depending on the conditions of the shop floor.

Second, the interpreter has an if-then rule structure that is used in the following two circumstances: (1) describing the incoming floor events (one can associate the semantics of an event with a script, allowing flexible reaction to shop floor dynamics) and (2) scheduling real-time operations (this script is similar to the UNIX CRON table). When an if part is a time pattern, the associated script is executed at the time specified. REDS uses this rule to schedule housework, such as database archives and periodic lot evaluation.

The Communication Servers. The data collector, the shop floor controller, and the order-entry handler are the communication processes.

The data collector is one gateway to REDS from the external environment. The WIP tracker sends data to the data collector. It then formats the data for REDS. The shop floor controller controls the shop floor activities, such as dispatch or shop floor status query. The order-entry handler takes care of the interface between the user, the shop floor, and the REDS kernel. It confirms and preprocesses order information to be sent to the shop floor and the REDS kernel. It also guarantees data consistency between REDS and the shop floor.

User-Level Interface Programs. There are three types of user interface programs: *system utilities-user programs*, the *floor activity monitor*, and *order entry*. System utilities and user programs are used for support tasks, such as constructing calendars for scheduling. Another user program allows the user to manually schedule the orders. The floor activity monitor is used for system administration. It consists of a communication interface and a logging facility. The user can monitor the REDS² software processes and can communicate with these processes by sending messages to them. The floor activity monitor was originally designed for debugging purposes only. However, it was expanded to cover more user requests, such as changing a server's behavior. The order-entry user interface is a customized design to help the user create orders.

Coordination

Coordination is one of the most important tasks in any distributed system. In REDS², coordination is achieved by information sharing and message passing. Currently, a central database is shared by all the processes in REDS².

The push and pull behavior of REDS² is similar to that of a just-in-time system.

Push and Pull

The push and pull behavior of REDS² is similar to that of a just-in-time system. Job orders are pushed onto the shop floor whenever they're ready to be processed. REDS² pulls orders from the job pool when the shop floor can efficiently support the processing of more jobs. At the planning level, the event handler issues job release requests regularly, in a way similar to a push. When the capacity watcher detects that resources are underused, it issues a request to pull out an order to process. At the scheduling level, when an order is unable to meet its due dates, its priority rises so that the order is pushed and preempts other orders. Conversely, when an order is expected to finish earlier than scheduled, its priority drops, and it might be pulled so that critical resources can be allocated to other orders with higher priority.

Implementation

Even though REDS was initially designed for small job shops, it is currently being used for VLSI pilot lines and a mask manufacturing facility; a prototype of it was also built for PCB assembly lines. In all these applications, the major difference is the SEQ module. All the other modules are functionally the same but include some minor adjustments to satisfy local user demands.

Of all these applications, the VLSI line is probably one of the most difficult areas for producing and maintaining good schedules. In such environments, the process plans and processing times are vaguely defined; machine breakdowns are frequent; rework, lot splitting, dynamic test injection, and limited block times are common; and the products are sensitive to machines so that choices of which machines to use change dynamically. Moreover, many bottleneck machines exist in VLSI lines, the bottlenecks shift from time to time, and the processes are reentrant.

Conclusion

In this article, we described the conceptual framework for REDS and its two implementations. The major contribution of the frame-

work is that it covers all scheduling activities from planning to finite scheduling. Moreover, REDS includes a release control strategy. These planning and scheduling activities use several levels of abstraction to achieve the desired goals. In this way, avoiding resource conflicts is not the sole concern at all times. At the planning level, some minor conflicts can be allowed, but at the sequencing level, all conflicts need to be resolved. The hybrid architecture, along with the real-time system design, supports our multiple perspectives for scheduling. Both implementations have been operational in a variety of factories and have proven to be successful.

References

- Adams, J.; Balas, E.; and Zawack, D. 1988. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science* 34(3): 391–401.
- Baker, K. R. 1974. *Introduction to Sequencing and Scheduling*. New York: Wiley.
- Brooks, R. A. 1986. Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation* RA-2(1).
- Burke, P., and Prosser P. 1989. A Distributed Asynchronous System for Predictive and Reactive Scheduling, Technical Report, AISL-42, Univ. of Strathclyde.
- Conway, R. W. 1967. *Theory of Scheduling*. Reading, Mass.: Addison Wesley.
- Fox, M.S. 1983. Constraint-Directed Search: A Case Study of Job Shop Scheduling, Technical Report, CMU-RI-TR-83-22, Robotics Institute, Carnegie-Mellon Univ.
- Fox, M. S., and Smith, S. F. 1984. ISIS—A Knowledge-Based System for Factory Scheduling. *Expert Systems* 1(1).
- Fox, M. S.; Allen, B.; and Strohm, G. 1982. Job Shop Scheduling: An Investigation in Constraint-Directed Reasoning. In Proceedings of the Third National Conference on Artificial Intelligence, 155–158. Menlo Park, Calif.: American Association for Artificial Intelligence.
- Glasse, C., and Redsende, M. 1988. Closed-Loop Job Release Control for VLSI Circuit Manufacturing. *IEEE Transactions on Semiconductor Manufacturing* 1(1): 36–46.
- Graves, S. 1981. A Review of Production Scheduling. *Operations Research* 29(4): 646–675.
- Grant, T. J. 1986. Lessons for OR from AI: A Scheduling Case Study. *Journal of the Operational*

- Research Society* 37(1): 41–57.
- Hadavi, K., and Shahraray, M. 1990. In-Time Control of Factory Production via Global Feedback, Technical Report, Siemens Corporate Research, Princeton, New Jersey.
- Hadavi, K., and Lehnert, A. 1987. A Load-Leveling Algorithm for Factories, Technical Memo, Siemens Corporate Research. Princeton, New Jersey.
- Hadavi, K., and Shahraray, M. 1989. Release No Job before Its Time. In Proceedings of the Third International Conference on Expert Systems and the Leading Edge in Production and Operations Management.
- Hadavi, K.; Shahraray, M.; and Voigt, K. 1990. An Environment for Planning, Scheduling, and Control of Factories. *Journal of Manufacturing* 9(4).
- Hsu, W.; Prietula, M.; Thompson, G.; and Ow, P. 1990. A Mixed-Initiative Workbench: Integrating AI, OR, and HCI. In Proceedings of the International Society of Decision Support Systems Conference.
- Kanet, J., and Adelsberger, H. 1987. Expert Systems in Production Scheduling. *European Journal of Operations Research* 29: 51–59.
- Kusiak, A.: 1987. Designing Expert Systems for Scheduling of Automated Manufacturing. *Journal of Industrial Engineering* 42–46.
- McKay, K.; Safayeni, F.; and Buzacott, J. 1988. Job-Shop Scheduling Theory: What Is Relevant? *Interface* 18(4): 84–90.
- Mertens, P. 1986. Expert Systems in Production Management: An Assessment. *Journal of Operations Management* 6(4): 393–403.
- Ow, P., and Smith, S. 1988. Viewing Scheduling as an Opportunistic Problem-Solving Process. *Annals of Operations Research* 12:85–108.
- Pinedo, M.; Levy, D.; Hadavi, K.; Hsu, W.; and Hou, R. 1991. Dispatching Issues in Job Shop Scheduling. In Proceedings of the Joint-German Conference on New Directions for Operations Research in Manufacturing.
- Sadeh, N., and Fox, M. 1989. Focus of Attention in an Activity-Based Scheduler. In Proceedings of the NASA Conference on Space Telerobotics. Washington, D.C.: National Aeronautics and Space Administration.
- Shahraray, M. 1987. Dynamic Sequencing Rule Applied to Job Shop Scheduling. In Proceedings of the Simulation and Artificial Intelligence in Manufacturing Conference. Dearborn, Mich.: Society of Manufacturing Engineers.
- Smith, S., and Ow, P. 1985. The Use of Multiple Problem Decomposition in Time-Constrained Planning Tasks. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, 1013–1015. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.
- Smith, S.; Ow, P.; Lepape, C.; McLaren, B.; and Muscettola, N. 1986. Integrating Multiple Scheduling Perspectives to Generate Detailed Production Plans. In Proceedings of the 1986 SME Conference on AI in Manufacturing. Dearborn, Mich.: Society of Manufacturing Engineers.
- Vepsalainen, A., and Morton T. 1987. Priority Rules for Job Shops with Weighted Tardiness Costs. *Management Science* 33(8).

Khosrow Hadavi is affiliated with Intellection, Inc. in Dallas Texas.

Wen-Ling Hsu, Tony Chen, and Cheoung-Nam Lee are affiliated with Siemens Corporate Research, Princeton, New Jersey.

THE ELECTRONIC ADDRESSES FOR AAAI ARE AS FOLLOWS:

AAAI membership inquiries:	membership@aaai.org
<i>AI Magazine</i> subscriptions:	membership@aaai.org
<i>AI Magazine</i> editorial queries & letters:	ebe@sumex-aim.stanford.edu
<i>AI Magazine</i> announcements:	aimagazine@aaai.org
<i>AI Magazine</i> advertising & press releases:	aimagazine@aaai.org
AAAI national conference:	ncai@aaai.org
AAAI Press:	press@aaai.org
Workshop information:	workshops@aaai.org
Innovative Applications conference:	iaai@aaai.org
Spring Symposium series:	sss@aaai.org
Fall Symposium series:	fss@aaai.org
Other inquiries and requests:	admin@aaai.org