Reactive Scheduling Systems

STEPHEN F. SMITH

Center for Integrated Manufacturing Decision Systems, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, sfs@cs.cmu.edu

Abstract

In most practical environments, scheduling is an ongoing reactive process where evolving and changing circumstances continually force reconsideration and revision of pre-established plans. Scheduling research has traditionally ignored this "process view" of the problem, focusing instead on optimization of performance under idealized assumptions of environmental stability and solution executability. In this paper, we present work aimed at the development of reactive scheduling systems, which approach scheduling as a problem of maintaining a prescriptive solution over time, and emphasize objectives (e.g., solution continuity, system responsiveness) which relate directly to effective development and use of schedules in dynamic environments. We describe OPIS, a scheduling system designed to incrementally revise schedules in response to changes to solution constraints. OPIS implements a constraint-directed approach to reactive scheduling. Constraint analysis is used to prioritize outstanding problems in the current schedule, identify important modification goals, and estimate the possibilities for efficient and non-disruptive schedule modification. This information, in turn, provides a basis for selecting among a set of alternative modification actions, which differ in conflict resolution and schedule improvement capabilities, computational requirements and expected disruptive effects.

key words: reactive decision-making, constraint-based problem solving, blackboard systems and production scheduling.

1. Introduction

The broad goal of manufacturing production management, like other resource-constrained, multi-agent planning problems, is to produce a coordinated behavior where demands are serviced in a timely and cost-effective manner. In most manufacturing environments, the construction of advance schedules is recognized as central to achievement of this goal; it enables anticipation of potential performance obstacles (e.g., resource contention) and provides opportunities to minimize their harmful effects on overall manufacturing system behavior. In practice, however, two factors confound the use of schedules as operational guidance. The first is the inability to extract useful guidance from generated schedules in any effective, systematic manner; most existing production planning and scheduling systems operate with models that ignore important operating constraints and conditions, and the correspondence of generated schedules to factory floor operations is missing. The second confounding factor is the dynamically changing nature of the production environment. Machines break down, materials fail to arrive on time, changing market conditions present unexpected production demands, etc., all of which work against attempts to follow prescriptive plans. In fact, the performance of the manufacturing organization ultimately hinges on an ability to rapidly adapt schedules to fit changing circumstances over time. In this respect, production planning/scheduling is not a static optimization problem, but an ongoing reactive process. Optimization objectives (assuring a good global manufacturing behavior) must continually be balanced with concerns of continuity in operations (because execution of a schedule sets a large number of interdependent processes in motion) and responsiveness (to keep the manufacturing system moving).

In this paper, we present an approach to incremental reactive management of schedules based on a view of scheduling as an iterative, constraint-directed process. Under this view, schedule revision/adaptation is driven by detection and analysis of conflicts and opportunities that are introduced by changes to current solution constraints. Constraint analysis is used to prioritize outstanding problems in the current schedule, identify important modification goals, and estimate the possibilities for efficient and non-disruptive schedule modification. This information, in turn, provides a basis for selecting among a set of alternative modification actions, which differ in conflict resolution and schedule improvement capabilities, computational requirements and expected disruptive effects. This approach to reactive scheduling is implemented in OPIS (OPportunistic Intelligent Scheduler), a knowledge-based system developed originally for manufacturing production scheduling. OPIS combines a modeling framework suitable for capturing essential operational constraints and objectives with a blackboard-based control architecture to provide a general infrastructure for configuring constraint-based schedule revision methods and strategies. The current OPIS manufacturing scheduler instantiates this infrastructure

with a specific set of methods and a constraint-based model for directing their application in responding to reactive scheduling problems. Experimental studies have demonstrated the viability and effectiveness of the scheduler in both generative and reactive scheduling contexts in a variety of complex manufacturing scheduling domains.

To provide a context for discussion, we first examine the characteristics and complexities of the reactive scheduling problem in practical domains. We then sketch the origins and evolution of the current OPIS scheduler, highlighting the basic concepts that underlie the approach and summarizing the results obtained. Next, we describe the principal components of the current OPIS scheduling architecture. This provides a basis for summarizing the heuristic methodology defined in the current OPIS manufacturing scheduler and illustrating its operation in responding to reactive scheduling problems. The approach is then summarized and contrasted with other recent work in reactive scheduling, and we conclude with a brief discussion of various ways in which the OPIS approach has been and is currently being extended and generalized for use in other application domains.

2. The Reactive Scheduling Problem

The general problem of interest in this paper is that of managing prescriptive solutions to scheduling problems. In brief, scheduling problems involve allocation of resources to the activities of multiple independent processes over time to achieve a targeted global behavior. Coordination of production in a factory, management of space missions, and transportation scheduling to support crisis management are representative examples. To be viable as operational guidance, a solution - or schedule - must first be *feasible*; i.e., it must satisfy the physical constraints in the domain relating to usage of resources and execution of processes. In practical domains, these constraints are often wide ranging and complex in nature. In manufacturing production environments, for example, resource allocation decisions must be consistent with capacity limitations, machine setup requirements, batching constraints on parallel use, work shift times, etc. Similarly, production activities have associated duration and precedence constraints, and may require the availability of multiple resources (e.g., machines, operators, tooling, raw materials).

However, feasibility alone is rarely the goal of scheduling; typically the task is one of optimizing (to the extent possible) a set of objectives and preferences. For example, processes to be coordinated typically have requested start and due dates, and one scheduling objective is to attend to these constraints. Because it may not be possible to satisfy all of these constraints (nor is it generally computationally feasible to determine precisely whether or not this is the case), the common operational objective is minimizing tardiness. Other global objectives (often conflicting) relate to the efficiency with which the processes are executed and resources are utilized:

minimizing wait time between constituent process activities, maximizing resource utilization, etc. In some cases, performance objectives can be approximated by tactical operating biases (or preferences). In a manufacturing context, for example, there might be a preference for a new machine over an older machine with overlapping capabilities because of reliability considerations. The *quality* of a schedule is a function of the extent to which it achieves (or effectively balances) stated objectives and preferences.

Scheduling research has traditionally focused on generating optimal solutions to classes of problems that make specific assumptions about the nature of domain constraints and objectives (e.g., [11]). Unfortunately, practical scheduling domains rarely meet these assumptions. But more generally, scheduling can rarely be treated as a static optimization problem. Aside from unpredictability in the execution environment, schedule generation in practice also tends to be a dynamic reactive process (particularly when multiple decision-makers are involved). An initial schedule is built, problematic or unsatisfactory aspects of the result are identified, requirements are relaxed or strengthened, schedule modifications are made and so on. Here, the current schedule provides the context for identifying and negotiating constraint changes (with the user or other scheduling agents). Although the focus is on improving the acceptability/quality of the solution, there is considerable pragmatic value placed on maintaining continuity in the schedules produced across iterations. Likewise, once execution begins, it is important to preserve continuity in domain activity while those changes are made that are necessary to ensure continued feasibility and attendance to overall performance objectives.

These pragmatic requirements argue strongly for approaches to scheduling based on incremental revision/adaptation of an existing schedule. It is this net change perspective that leads to what we refer to as the reactive scheduling problem. We assume that a schedule consists of a set of constraints that delineate (1) start time, end time and resource assignments for each activity of each process, and (2) available resource capacity over the scheduling horizon. The need for schedule revision arises in response to the introduction of new constraints or the removal of existing constraints, which might reflect the receipt of status/requirements updates from the environment or might be due to the results of prior modification actions. Schedule modification can be initiated for two purposes: (1) to restore the feasibility of a schedule now known to be infeasible because of the introduction of new conflicting constraints or (2) to attempt to produce a higher quality solution if one or more constraints respected by the current schedule have been relaxed. In the first case, modification is necessary to insure continued executability. In the second case, the potential gain in schedule quality may be weighed against its likelihood.

There are several characteristics of the schedule revision problem that influence the approach taken in OPIS toward its solution:

- It is generally not possible to bound the scope of change required to

the current schedule in advance. Given the tightly coupled nature of scheduling decisions, changes to one portion of the schedule often have ripple effects. Heuristic guidance can minimize this phenomenon, but problem combinatorics prevent its elimination. Schedule modification must necessarily proceed *opportunistically* (i.e., with the understanding that revision actions may have unforeseen interactions with other portions of the schedule that must subsequently be resolved).

- Striking an appropriate balance between attending to various scheduling objectives, minimizing disruption, and being computationally efficient when reasoning about possible modifications is a difficult task. There are often simple, fairly non-disruptive changes that can restore schedule feasibility. For example, if resource capacity is suddenly lost, affected activities could simply be delayed until a time in the schedule when required resource capacity is available (disregarding implications with respect to scheduling objectives such as minimizing tardiness). At the same time, however, it is not possible to enforce rigid bounds on schedule quality (e.g., degree of tardiness allowed) with assurance that a solution actually exists.

- What about exploiting the expertise of human schedulers? Although we do not discount the experience accumulated by veteran human schedulers, our experience in many manufacturing environments is that schedulers often fall prey to the complexity of interacting constraints and decisions, and tend to adopt myopic "fire-fighting" tactics (where extinguishing one fire ignites the next). Such tactics keep execution moving, but global system behavior deteriorates rapidly. In OPIS, use of knowledge about current problem structure (i.e., properties of current solution constraints) is advocated as an alternative basis for directing the schedule revision process.

3. Origins and Evolution of Approach

The approach to the reactive scheduling problem taken in the OPIS scheduler is rooted in earlier work with the ISIS job shop scheduling system[8, 9]. The ISIS scheduler was the first attempt to formulate and operationalize the view of scheduling as a heuristic, constraint-directed activity. ISIS emphasized complete representation of all constraints that impact operational decision-making, and a representational framework that recognized the conflicting and negotiable nature of many of these constraints. A representation of preference (i.e., relaxable) constraints was defined to encode knowledge relating to various factory objectives and operating preferences, including their relative importance, possible relaxations of preferred choices, the utility of each alternative, and the types of decisions that the constraint impacts. This knowledge about preferences was embedded in a larger relational framework for modeling the entities and physical con-

straints of the production environment. The OPIS modeling framework was built directly on these representational concepts.

ISIS also introduced a heuristic search framework for using constraints to guide the scheduling process. At the core of the ISIS approach was a beam search strategy for adding a new job (order) into the developing shop schedule (or alternatively revising the schedule of a job previously added to the schedule). Within this search, physical constraints (i.e., operation precedence constraints, resource requirements and availability) were used as a basis for generating alternative sets of decisions. Relevant preference constraints provided the basis for evaluation and pruning of alternatives at each step of the search, thus implementing a generative approach to constraint relaxation. This core search process was augmented in two ways to define the overall scheduling procedure. First, it was bracketed by rule-based analysis steps. Domain specific rules were applied both prior to any invocation of the search to fix specific relaxable constraints (e.g., specify "backward" scheduling to ensure satisfaction of due date) and after the search to assess results and propose constraint relaxations if appropriate (e.g., relax the due date and attempt "forward" scheduling if a feasible schedule cannot be found during backward scheduling). Second, this extended beam search procedure was embedded in an iterative, hierarchical control regime where additional types of constraints were considered at each successive level. After selecting the next job to schedule (or reschedule) on each iteration, a high level analysis of remaining resource availability was used to emphasize specific allocation intervals (through the introduction of additional preferences). After detailed forward or backward beam search scheduling, which incorporated these additional preferences, final local optimization to minimize WIP time was performed.

One difficulty encountered with this search architecture was its inflexibility with respect to strategic problem decomposition. Although the kernel heuristic beam search procedure provided a flexible basis for local search in the presence of diverse constraints and objectives, its placement within an overall decomposition framework based rigidly on relative job priority and stepwise construction of job schedules was found to impose significant limits on the system's ability to achieve a good compromise with respect to conflicting global objectives.[29]. The problem can be simply seen by considering two common factory scheduling objectives: minimizing work-in-process (WIP) time and maximizing resource utilization. The job-oriented problem decomposition framework of ISIS provides an opportunity to minimize WIP time, because subproblems are solved that contain all constraints involved in optimizing this objective. At the same time, a job-oriented decomposition works against the objective of optimizing resource usage because the constraints relevant to this tradeoff are spread across subproblems. A resource-oriented decomposition strategy, alternatively, provides the complementary opportunity to optimize resource usage (at the expense of leverage in minimizing WIP). The subproblems solved

in this case contain the competing requests of multiple jobs for a given resource, enabling resource setups to be minimized. Similar arguments against a fixed decomposition strategy can be made in reactive scheduling contexts. Localized revision of individual job schedules (the reactive strategy in ISIS) can provide a direct basis for resolving operation precedence violations (e.g., resulting from quality control failures and the subsequent introduction of "part repair" operations) with continued attendance to WIP minimization, but provides only indirect leverage, at best, in rearranging resource assignments to maximize throughput in response to an unexpected loss in resource capacity.

To broaden the range of constraints and objectives that could effectively be dealt with, the OPIS scheduler adopts a more flexible approach to problem structuring, referred to as *multi-perspective scheduling*. The concept of multi-perspective scheduling, as originally conceived, advocated the selective use of a complementary set of local scheduling methods, each conducting its search under different problem decomposition assumptions. An initial implementation (OPIS 0) focused on integrating use of the job-oriented beam search procedure of ISIS with a resource-oriented search procedure built around the "idle time" priority rule [19] and designed to maximize usage of substitutable resource groups. Schedules were generated according to a predefined strategy, first constructing a schedule for a pre-designated bottleneck resource group, and then completing the shop schedule on a job-by-job basis. Comparative experimental analysis carried out in the context of a specific Westinghouse job shop environment convincingly demonstrated the power of this configuration over both ISIS and a well-regarded resource-centered dispatch scheduling method in balancing weighted tardiness and WIP minimization goals over a broad range of factory conditions.[22].

Although OPIS 0 experiments confirmed the utility of incremental schedule construction from both resource-centered and job-centered decomposition perspectives, it did so under rigid and simplifying control assumptions. By presuming a static problem structure comprised of a single pre-identified bottleneck resource group, it was possible to perform the necessary computation to ensure feasibility of the partial schedule each step of the way (and avoid the complications of backtracking). However, dependence on any static assumptions about problem structure is ultimately confining (this was the original criticism of the ISIS search architecture). Resource bottlenecks are typically not stationary but float over time according to production characteristics (e.g., job mix, shop load); attendance to primary resource bottlenecks can lead to the emergence of secondary bottlenecks; and in some cases there is no dominant locale of resource contention in the overall manufacturing system. Each of these circumstances suggests different problem decomposition/structuring decisions, and indicate the need for dynamically determined schedule building strategies.

To dynamically control the use of local search methods oriented

around job and resource loci respectively, the OPIS 1 scheduler [27] introduced the concept of *constraint-based control*. The general idea here is that monitoring and analysis of characteristics of the evolving structure of solution constraints (in particular, flexibilities and inflexibilities in the solution space) can provide useful problem structuring knowledge. As foreshadowed in the design of the initial hard-wired, multi-perspective strategy of OPIS 0, resource contention was incorporated in OPIS 1 as the essential aspect of current solution structure upon which to focus the schedule building process. A resource capacity analysis procedure was defined and used in conjunction with the heuristic that decisions at "bottleneck" resources are the most critical to the overall quality of the solution and should be considered first[19, 15]. In moving to this *opportunistic* scheduling strategy, the OPIS 0 assumption of a guaranteed feasible partial solution at each intermediate solution state was also abandoned, giving rise to the possibility of inconsistent decisions. It was felt that repeated generation of "throw away" schedule extensions to ensure feasibility (as was done in OPIS 0) would not only become computationally over-bearing but would also add undesirable bias to the opportunistic scheduling strategy (because of the influence of the heuristics used to generate extensions). In OPIS, inconsistent intermediate solution states are seen as equivalent to inconsistencies arising from unexpected external events. They are reactive scheduling problems to be solved. In the OPIS 1 scheduler, a simple "schedule shifting" method was added to reconcile all detected conflicts and allow the schedule building process to proceed.

This opportunistic approach to schedule building was supported by an underlying system architecture based on standard principles of blackboard systems[7]. The OPIS 1 architecture reflected the central role of constraint management in the scheduling process and the constraint-based approach to coordinating local search. The design anticipated a wider array of solution constraint metrics upon which to base strategic control decisions and a larger repertoire of potential scheduling actions. Constraint analysis routines and scheduling methods were encapsulated as knowledge sources which, when triggered, operated on a globally accessible representation of the current solution. Strategic decision-making was localized within a single controller, that maintained and executed an explicit plan (i.e., an agenda of analysis and scheduling tasks) for solving the problem. The OPIS 1 architecture provided an initial infrastructure for investigating constraint-based, multi-perspective approaches to reactive schedule revision and adaptation.

Focus on the reactive scheduling problem necessitated a shift in perspective with respect to strategic control. Whereas problem solving in generative contexts can be driven top-down from global analysis of problem requirements and identification of critical decisions, the situation is different in reactive contexts. Here the starting point is a set of identified problems in the current solution (e.g., constraint conflicts), and focusing heuristics must be based on localized solution analysis. Moreover, control

decisions must balance pressing (re)optimization needs with potential opportunities to make revisions with limited non-local impact. Investigation of reactive scheduling strategies also sharpened understanding with respect to scheduling architecture requirements and highlighted problems in some of the assumptions made in the OPIS 1 scheduler. Most important was recognition of the inappropriateness of attempting to maintain a strategic control plan for solving a reactive scheduling problem. Because it is generally not possible to predict the non-local effects of a given conflict resolving action (e.g., the number and types of conflicts, if any, that will be introduced as a result of applying the action, the potential serendipitous effects of this action on other currently pending conflicts), it was rarely the case that an initially formulated control plan could be carried out to completion. Reassessment and revision of the agenda of pending tasks was typically required at each strategic step.

These considerations and experiences contributed to development of the OPIS 2 scheduler [25]. Work in reactive scheduling led to expansion of the set of scheduling methods, extending the search-based methods utilized in OPIS 1 to provide schedule revision capabilities and incorporating additional, more-specialized repair actions. The underlying scheduling architecture was revised to provide a more reactive, blackboard-based control framework. Within the OPIS 2 architecture, schedule generation and revision is uniformly cast as an iterative process of subproblem formulation and subproblem solution, opportunistically directed by analysis of current problem structure. A specific heuristic model, which maps the optimization needs and opportunities implied by specific reactive scheduling states to the differential capabilities of the expanded set of revision methods, was developed and implemented as a means of validating the approach [20]. A series of experiments carried out in the context of an IBM computer board assembly and test line demonstrated the comparative advantage of this reactive model over several less flexible revision strategies as well as a random selection model biased by the choice percentages observed using the model across all experiments (to verify that the model did, in fact, encode useful knowledge) [25]. Performance was evaluated in this study with respect to weighted criteria reflecting optimization, stability, and efficiency objectives across a diverse range of reactive problems involving unexpected resource loss and job delays due to quality control failures. More recent experimental analysis has evaluated this reactive model in support of generative decision-making[1]; reruns of the original multi-perspective job shop experiments with the OPIS 2 scheduler, using dynamic, bottleneck-based problem decomposition and the reactive model to resolve conflicts introduced along the way, yielded significant further improvements in schedule quality over the original hard-wired multi-perspective strategy.[1]

In the next few sections, we summarize the organization and opera-

[1] The reader is referred to [22, 20, 25, 1] for details of the experimental results obtained with variants of the OPIS scheduler.

tion of the OPIS 2 scheduling system (referred to hereafter simply as OPIS). We first describe the infrastructure for constraint-based, reactive scheduling provided by the OPIS scheduling architecture . We then turn attention to the methods and heuristics implemented within this architecture in the current OPIS reactive scheduler.

4. The OPIS Scheduling Architecture

Figure 1. schematically depicts the blackboard-based control architecture defined in OPIS. The architecture presumes a collection of base scheduling methods (or knowledge sources) that carry out designated *scheduling tasks* and make changes to a commonly accessible representation of the current solution. An additional "model update" knowledge source is invoked upon receipt of external notification of constraint changes (e.g., new requirements, execution status updates) to reflect their consequences on the current solution. The introduction of changes to the current schedule (whether they are externally imposed or due to execution of a scheduling task) results in the posting of *control events* in a global description of the system's current control state. The control state at any point characterizes the set of outstanding problems that remain (i.e., current conflicts in the schedule, set of commitments that remain to be made, unexplored opportunities for improving the solution). A separate set of analysis knowledge sources extends this control description to provide the information necessary to support the formulation of subsequent scheduling tasks.

Two additional system components provide the distinguishing characteristics of the OPIS architecture and the infrastructure for opportunistic, multi-perspective scheduling: a *schedule maintenance* subsystem, which incrementally maintains a representation of current solution constraints, and *a top level manager (TLM)*, which holds responsibility for coordinating the use of scheduling, analysis and model update methods, and implements an event-driven control cycle. The former provides both a basis for analyzing aspects of the current scheduling state and a means for communicating scheduling constraints among different formulated subproblems. The latter defines a structure for specifying and a mechanism for applying the control knowledge necessary to implement constraint-based scheduling strategies.

Both the representation of the schedule maintained within OPIS and the methods incorporated to analyze and modify this schedule are defined relative to an underlying domain model, which contains a specification of the constraints and objectives on process execution and resource allocation that must be accounted for in the target environment. Before considering the schedule management subsystem and the TLM in more detail, we first summarize the structure of domain models constructed within the OPIS modeling framework.

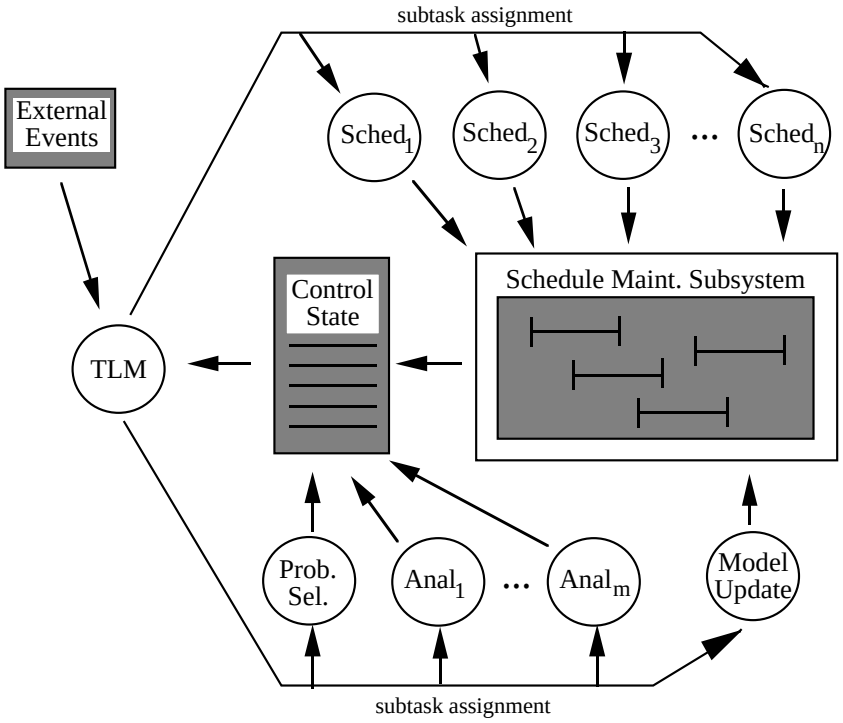

Figure 1. The OPIS Scheduling Architecture

4.1. Modeling Framework

Within a OPIS domain model, prototype process descriptions (e.g., manufacturing production plans for various part types) are represented as hierarchies of operations, with aggregate operations designating either more detailed sub-processes (i.e., sequences of operations) or sets of exclusive alternatives. Operation descriptions at any level specify precedence relations (predecessor and successor operations), duration constraints, and resource capacity and setup requirements. Resources are correspondingly represented hierarchically, with atomic resources grouped into increasingly larger resource pools, to provide a description of resource allocation constraints (e.g., available capacity, hours of operation) at each level of abstraction in the prototype plans. Thus, for example, if a disjunctive leaf operation specifies usage of a particular manufacturing machine as a resource requirement, the aggregate operation to which it is related at the next level specifies usage of capacity in an encompassing machine group as a resource requirement. This hierarchical domain model provides a structure

for representing and maintaining solution constraints at different levels of precision; in reactive contexts this hierarchical model provides one basis for reasoning about the scope of potential schedule modifications.

The OPIS modeling framework provides an extensible set of primitives for specifying a wide range of constraints on resource allocation. Resource representations enable specification of unit capacity resources, which must be allocated exclusively to a given process (e.g., a machine), batch capacity resources, which can simultaneously be allocated to multiple processes over the same interval (e.g., an oven), and a variety of disjunctive and conjunctive aggregate capacity resources, where capacity can simultaneously be allocated to multiple processes without temporal synchronization (e.g., machine, operator groups). Resource setup constraints can be modeled as temporal delays, expressed as arbitrary functions of the operations that consecutively utilize a given resource. Operation durations, similarly, can be expressed as functions of work content (e.g., the number of parts in the batch being processed). Work shift constraints can additionally be imposed on resource availability, with allowance for preemption of process execution across non-working hours. Priorities and priority classes can be associated with process requests (e.g., orders for manufactured parts) in addition to requested release and due dates. The utility-based representation of preferences originally developed in ISIS provides a basis for specifying idiosyncratic biases with respect to choice sets defined in the domain model (e.g., preferences over substitutable resource groups). Details of the OPIS representational framework can be found in [26].

4.2. Schedule Maintenance

To provide a representation of the current schedule, an appropriate prototype plan in the domain model is instantiated for each job or process to be accomplished in the current scheduling horizon. These instantiated plans designate the set of process operations that must be scheduled to obtain a complete solution (actually a superset, because some instantiated operations designate alternatives that will become undefined as choices are made). Given these instantiated plans and the resource hierarchy defined in the domain model, the schedule maintenance subsystem incrementally maintains the following solution constraints:

- the current time bounds (an earliest start time, latest end time pair) on the execution of each instantiated operation that has been or might be scheduled, and
- a specification of the current available capacity of resources at each level of the hierarchy over time at each level in the hierarchy (represented as an ordered sequence of intervals of the form *(st, et, capacity available)* that covers the scheduling horizon).

As additional scheduling decisions are made or the constraints implied by external status updates are introduced, the schedule maintenance system

combines these new constraints with the constraints on process execution and resource utilization defined in the underlying domain model and specified problem constraints (e.g., job release and due dates). This results in an updating of the time bounds and available capacity representations, respectively, of related operations and resources at all defined levels of abstraction. Thus, an unscheduled operation's time bounds at any point reflect the set of allocation decisions compatible with domain and problem constraints, and any scheduling decisions that have been made.

Constraint propagation in response to schedule changes can lead to the detection of two types of conflicts:

- time conflicts - situations where either the time bounds or scheduled execution times of two operations belonging to the same process instantiation violate a defined temporal precedence constraint, or the time bounds of a single operation violate a rigid absolute bound on process execution.[2]

- capacity conflicts - situations where the resource requirements of a set of currently scheduled operations exceed the available capacity of a specific resource over some interval of time.

The recognition of conflicts signals the need for schedule revision. Detected conflicts are posted in the current control state as *elementary conflict events* which require subsequent scheduling attention.

Constraint propagation can also lead to detection of rescheduling *opportunities*, situations where time and capacity constraints are loosened by introduced schedule changes. In the current implementation, such situations are treated in a somewhat specialized manner; *opportunity events* are posted only in response to changes originating from external events that imply additional resource capacity (e.g., cancellation of a process request) to ensure that a rescheduling process is triggered. The final type of control event that can be posted by the schedule management subsystem is an *incomplete hypothesis event*, signifying that some set of scheduling decisions still remains to be made. Details of this approach to schedule maintenance can be found in [14].

4.3. Strategic Control

Events posted by the schedule management subsystem are responded to within a control cycle defined by the TLM. The TLM control cycle identifies four stages of control decision-making that must occur in specifying the next scheduling action to perform and correspondingly four types of knowledge sources necessary to support the process:

[2]Some types of imposed time constraints, in particular due date constraints, are specified as relaxable instead of rigid. Violations of such relaxable constraints are not seen as conflicts, but are instead interpreted as relaxation decisions made by the originator of the change. In such cases, time bounds of other temporally related operations belonging to the same process are updated to reflect this newly determined end time bound.

- *event aggregation* - The first decision-making step contributes to selection of the particular control event (or events) of those currently posted to serve as the focal point of reaction on the current cycle. It is often the case that individual events are related in some manner and would be better addressed simultaneously. During event aggregation, knowledge of such relationships is applied to the set of posted events. In cases where specific relationships are detected, *aggregate* events are created and added to the list of posted events in the current control state.

- *event prioritization* - After this preprocessing of posted control events, prioritization heuristics are applied to select the specific event to be responded to in the current cycle. All events other than the highest priority event are left pending until the next cycle.

- *event analysis* - Having identified a focal point for problem solving, the next step in the control cycle is problem analysis. The goal of this step is to summarize essential aspects of the current solution state (e.g., the relative looseness or tightness of current time and capacity constraints), providing a basis for determining how to best respond to the event. Analysis results are appended to the event description in the current control state.

- *subproblem formulation* - During the last step, subproblem formulation knowledge is applied to the results of problem analysis, resulting in generation of a particular *scheduling task* to execute. A scheduling task designates (1) a particular component of the overall schedule to extend or revise, (2) a particular scheduling method to apply, and (3) depending on the KS selected, appropriate parameterization of the solution procedure.[3]

Once an appropriate scheduling task has been determined, it is carried out, the results are introduced into the current solution, and the TLM control cycle repeats. When, on any given cycle, the set of posted events becomes empty, a complete and consistent solution has been obtained and the process terminates.

5. Constraint-Based Schedule Repair in OPIS

The OPIS scheduling architecture allows specification of a range of methods and heuristics for addressing the reactive scheduling problem. The current OPIS manufacturing scheduler implements one such configuration of methods and heuristics, emphasizing principles of constraint-based focus

[3]It is possible in some circumstances for the subproblem formulation step to produce a sequence of scheduling tasks. In this case, all specified tasks will be executed before further consideration of the current control state.

of attention and multi-perspective scheduling discussed earlier in the paper. In this section, we examine the various heuristic components of this constraint-based repair methodology. We start with an overview of the base methods defined as strategic schedule revision alternatives.

5.1. Strategic Alternatives

The set of possible modification actions available in the current OPIS scheduler range from general heuristic search procedures oriented toward generating and revising sets decisions associated with a specific process or resource to more specialized revision procedures for sliding schedule components forward or backward in time and performing pairwise resource assignment exchanges. In describing these methods below, we restrict attention to describing revision capabilities, and characterizing their differential behavioral characteristics.

The **Order Scheduler (OSC)** provides a method for revising the schedule of some contiguous sequence of operations in the plan of a given process (e.g., the plan associated with a given manufacturing order). Revision of a designated process (or subprocess) schedule is accomplished by first retracting the current time and resource assignments of each constituent operation (i.e., releasing previously allocated intervals of resource capacity), and then applying the augmented beam search strategy of ISIS (see Section 3) to determine new resource assignments and execution intervals for the operation sequence. In addition to identifying the specific subprocess to be revised, an OSC scheduling task also designates a level of "visibility" with respect to resource availability. The search can be constrained to consider only execution intervals for which resource capacity currently exists, which we designate as the complete visibility (CV) OSC, or can be allowed to consider capacity allocated to lower priority jobs as available, which we designate as the prioritized visibility (PV) OSC. These two modes of operation are illustrated in Figure 2. Because prioritized visibility search admits the possibility of introducing additional capacity conflicts into the schedule (leading to "bumping" of lower priority processes), a decision to invoke the PV-OSC trades off potential additional disruption for some ability to perform resource-based optimization.

The **Resource Scheduler (RSC)**, provides a second general revision method, in this case for resequencing operations on a designated resource (or substitutable resource group) from a specified point in time onward so as to consistently accommodate a designated set of conflicting operations. Schedule revision is accomplished by "assuming" that the schedule must be completely regenerated from the revision start time onward and applying an iterative forward-dispatch search procedure to accomplish this goal. However, the decisions in the current schedule are not actually retracted prior to invoking this procedure; rather operations in the current schedule are only retracted when they are chosen to be "dispatched on a resource" on a given dispatch cycle. After all designated conflict op-

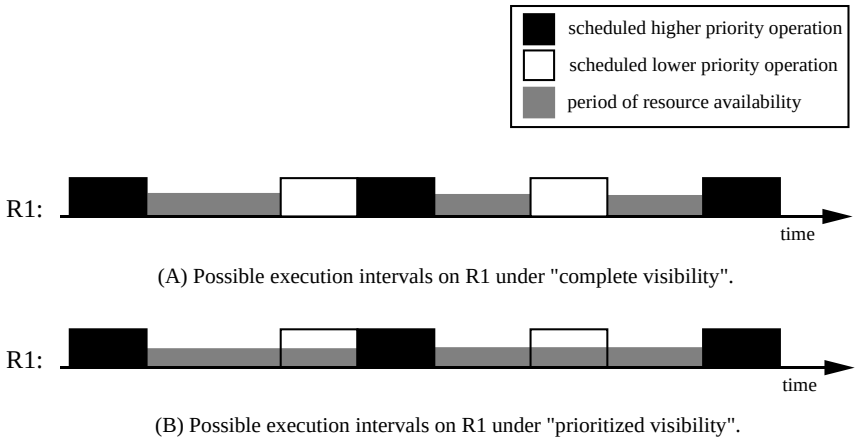

Figure 2. OSC visibility of allocation decisions

erations have been rescheduled, the procedure terminates on the first cycle where retraction and re-insertion of the chosen operations leave the overall resource schedule consistent. Thus, RSC attempts to preserve as much of the original resource schedule as possible while it resolves the problem at hand. The dispatch cycle itself makes selective use of a collection of priority rules to balance weighted tardiness and setup minimization concerns when selecting among alternative resource assignment and operation dispatch decisions. A detailed description of these heuristics can be found in [22]. The RSC method is designed from the assumption that contention for the resource(s) to be rescheduled is high; use of a dispatch-based search framework presumes that it is not necessary to consider insertion of resource idle time, and places emphasis instead on efficient resource utilization. Since revision of the schedule of a resource (or resource group) by RSC typically results in some amount of resequencing of scheduled operations (forcing some to be scheduled later than before) it is possible that its application will introduce new time conflicts with downstream process operations into the overall schedule.

The **Right Shifter (RSH)** implements a considerably less sophisticated reactive method that resolves conflicts by simply "pushing" the scheduled execution times of designated operations forward in time ("jumping" over any scheduled operation on the same resource whose end time falls before the end of the shift). Execution of these designated shifts can introduce both time conflicts (with downstream operations belonging to the same process) and capacity conflicts (with operations scheduled downstream on the same resource). However, these conflicts are internally resolved by recur-

sively propagating the shifts through resource and process schedules to the extent necessary. Thus, the RSH will not introduce any new conflicts into the overall schedule. Figure 3 shows the result of an RSH action to resolve a time conflict involving $op-a1$ by shifting its scheduled start time on $R1$. In this example, each process i follows the operation sequence $op-i1 \rightarrow op-i2$ on resources $R1$ and $R2$, respectively.

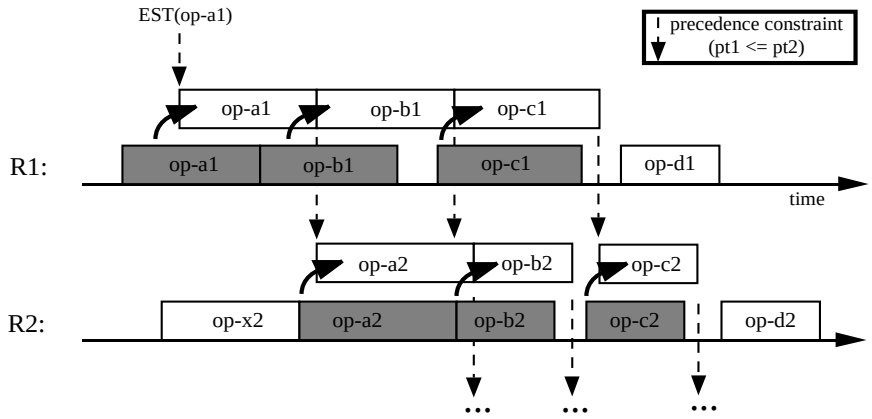

Figure 3. Application of RSH to $op-a1$ on $R1$

The **Left Shifter (LSH)** provides a similar but totally non-disruptive reactive method that "pulls" operations backwards in time (i.e., closer to execution) to the extent that current resource availability and temporal process constraints will permit. The method proceeds by sliding operations on a designated resource R to exploit an identified interval of available resource capacity (and any capacity intervals created by this sliding), and then recursively applying the procedure to the resources associated with the successor operations of processes who have had their scheduled execution interval on R changed. The recursion terminates whenever a downstream resource schedule is encountered that does not provide opportunities for left shifting or when process schedules have been completely traversed. Within the current implementation LSH is used exclusively for responding to opportunity events. Figure 4 shows the result of a LSH action invoked on resource $R1$ upon indication that $R1$ has become available earlier than expected. In this case, $op-a1$ is first rescheduled to start as soon as $R1$ is available, $op-c1$ is then shifted into the time interval vacated by $op-a1$ (because $op-b1's$ earliest start time constraint does not allow it to be moved), and finally $op-d1$ is shifted left as far as possible.

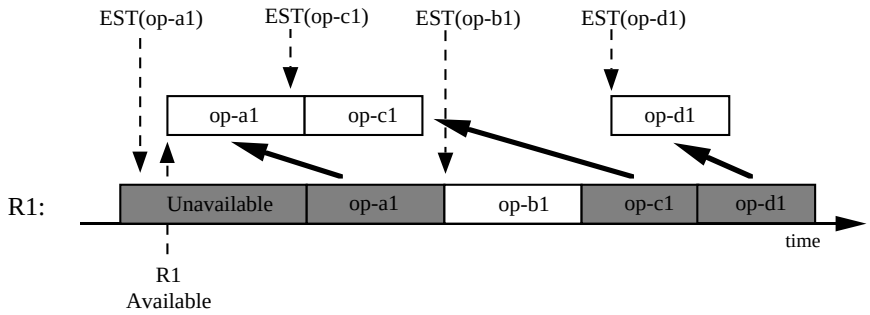

Figure 4. LSH revision in response to unexpected resource availability

The **Demand Swapper (DSW)** implements a final revision method based on pairwise exchange of the respective resource assignments and execution intervals of two similar processes. It is applicable in situations where a given process has unexpectedly been delayed and is now expected to be tardy. When invoked, DSW first identifies a set of suitable candidate processes for exchange (i.e., similar processes that are currently scheduled to complete ahead of their due dates). If at least one candidate is found, then an exchange of the remaining unexecuted portion of the problematic process's schedule with the corresponding portion of the schedule of another is made to minimize their combined tardiness. In essence, this action has the effect of redirecting the pair of processes to fulfill each other's respective demands. Note that the DSW is not necessarily a conflict resolution strategy. It is more appropriately viewed as a scheduling action that improves the character of the conflict. Figure 5 illustrates the result of applying a DSW action, in this case in response to the insertion of an extra "rework" operation into $Process1's$ operation sequence and the subsequent detection of a precedence violation between operations $rework-1$ and $op-1-2$. The depicted exchange of downstream execution intervals leaves the schedule of $Process1$ conflict free and ending with a small amount of tardiness. The precedence violation has moved to $Process2's$ schedule, but given that $Process2$ was originally scheduled to finish well ahead of its due date, there is now additional slack available for resolution of the conflict.

5.2. Responding to Conflicts

The above revision methods provide a range of capabilities for responding to reactive scheduling problems, and the strategic control problem faced by the scheduler is that of intelligently mapping relevant capabilities to detected problems. As indicated in Section 4, three general sources of knowledge are required by the TLM to solve this control problem: the first supporting selection of the particular conflict or set of conflicts to focus on

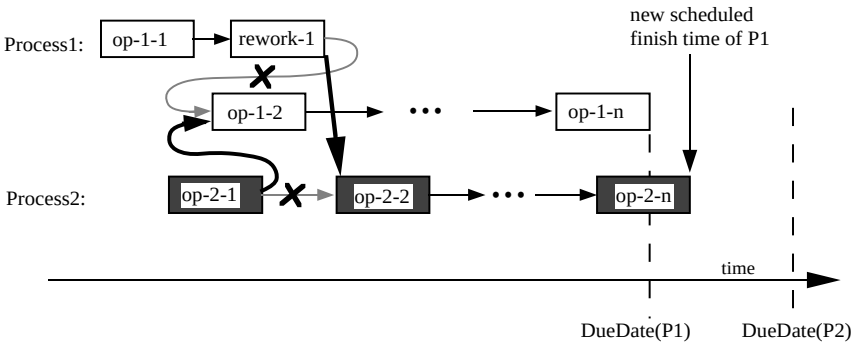

Figure 5. DSW revision in response to an unexpected process delay

next, the second providing characteristics of essential aspects of the current solution state, and the third concerning formulation of the most appropriate revision action to take to resolve this conflict (or set of conflicts).

5.2.1. Conflict Selection

Posted conflict descriptions provide basic characterizations of problems indicating the type of conflict (time or capacity), the operation(s) whose current commitments are in conflict, the resource (or resources) involved, the processes (or jobs) involved, the start time of the conflict and its temporal magnitude. Given this information, a variety of criteria can be defined for aggregating and prioritizing the current set of posted conflicts on any given revision cycle. The heuristics incorporated in the current scheduler reflect our experiences to date.

With respect to aggregation of posted elementary conflicts for simultaneous consideration, the heuristics currently utilized emphasize recognition of two types of relationships in posted elementary capacity conflicts that have been observed to occur with some regularity in specific reactive contexts.[4] The first relationship that triggers aggregation is based on *commonality in the resources involved* in elementary capacity conflicts. It is often the case that two or more capacity conflicts involving the same resource have intersecting sets of conflicting operations. For example, in the simple case of a unit capacity resource, if a given operation is scheduled over an interval that spans the scheduled execution times of several other operations previously allocated to that resource, individual capacity conflicts will be detected for each pair of operations. This situation is illustrated in Figure 6, which depicts (in Gantt chart form) portions of the schedules of

[4] We have experimented with various relationships for aggregating time conflicts but have as yet found none that add substantial value to rescheduling performance.

two unit capacity resources $R1$ and $R2$, and indicates two capacity conflicts CC_1 and CC_2 in $R1's$ schedule that involve $op - x1$. Because problems of over-allocation often require resource resequencing, it makes little sense to consider these conflicts individually.

The second situation under which elementary capacity conflicts are aggregated is based on *commonality in the processes involved* in the conflicts. This aggregation is motivated by a type of situation that can result from execution of a PV-OSC revision task (the only modification action that can introduce additional capacity conflicts). If PV-OSC determines new scheduling decisions for a given process that bump a lower priority process with a similarly structured plan (as is common in manufacturing environments with flow shop characteristics), then it is quite possible for capacity conflicts to be introduced on several resources which involve operations belonging to these same two processes. This situation is also depicted in Figure 6, where it is assumed that jobs are processed first on $R1$, then on $R2$. Conflicts CC_1 and CC_3 involve the same two jobs at two consecutive process steps. Because multiple components of the same process schedule now require revision, it is natural to consider the aggregate problem.

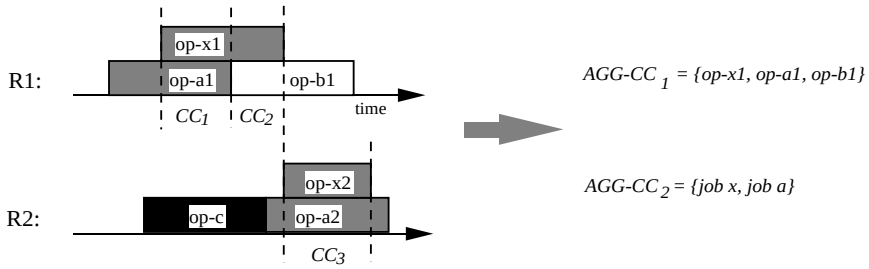

Figure 6. Aggregation of conflicts

Recognition of either relationship leads to the generation of aggregate events (as depicted in Figure 6) and their introduction into the list of currently posted events. Note that distinct aggregate events are not necessarily mutually exclusive groupings of elementary events. Any created aggregate event that is not selected as the current focal point for reaction is discarded.

Within the current scheduler, events are selected (prioritized) first as a function of event type. Aggregate conflicts are higher priority than elementary conflicts, and process-based aggregations (by their linkage to an ongoing reactive strategy) are higher priority than resource-based aggregations. Elementary time and capacity conflicts are of equal priority.

Opportunity events are lowest priority and are only selected in conflict free situations (because conflict resolution can actually exploit any posted opportunities). The second selection criteria applied (in situations where event type does not identify a unique event) is the urgency of the event (i.e., temporal proximity to execution).

5.2.2. Conflict Analysis

Determination of the most appropriate revision action to undertake to resolve a given conflict requires knowledge about the continued validity of various scheduling decisions, pressing (re)optimization needs, and current sources of rescheduling flexibility. The base assumption underlying the OPIS scheduler is that analysis of current problem structure can provide this knowledge. In this section, we describe the set of metrics that are currently computed for this purpose. These metrics summarize various characteristics of current solution constraints, and computations are confined to a local region of the solution containing the focal point conflict. A *conflict horizon*, defined as an interval that temporally spans the conflict by a pre-specified margin, restricts attention to a subset of operations in addition to those directly involved the conflict.[5]

Two metrics are defined to estimate the severity of conflict itself, and five others are defined to characterize the tightness or looseness of current time and capacity constraints in the schedule. Several are defined with respect to a particular resource involved in the conflict, designated R_C. For all types of capacity conflicts except process-based aggregate conflicts, R_C is the single resource that is over-allocated (which might well be an aggregate capacity resource representing a pool of more atomic resources). Unless otherwise stated, we will assume that relevant metrics are computed relative to each constituent $< conflict, R_C >$ pair in the case of a process-based aggregate conflict. In the case of a time conflict, which designates a precedence violation between two consecutive process operations, we assume that R_C is the resource assigned to the downstream operation. We also designate $Confl_C$ as the set of operations whose commitments are in conflict (the union over all constituent conflicts in the case of all aggregate conflicts). Figure 7 graphically depicts a portion of a schedule for aggregate resource R_C that contains a capacity conflict, identifying both the conflict horizon and $Confl_C$.

Conflict Duration is defined as the temporal magnitude of the inconsistency (e.g., $t2-t1$ in Figure 7) normalized with respect to the average duration of all operations scheduled on R_C within the conflict horizon (e.g., the average duration of operations $op-b$ through $op-h$ in Figure 7). This metric provides one indicator of the continuing validity of R_C's schedule. Appealing to sensitivity analysis[2, 17], if the duration is low, then it is reasonable to assume that the sequencing decisions previously made at R_C

[5] This use of metrics is quite similar to the concept of "textures" described in [10], although the focus there is in generating schedules.

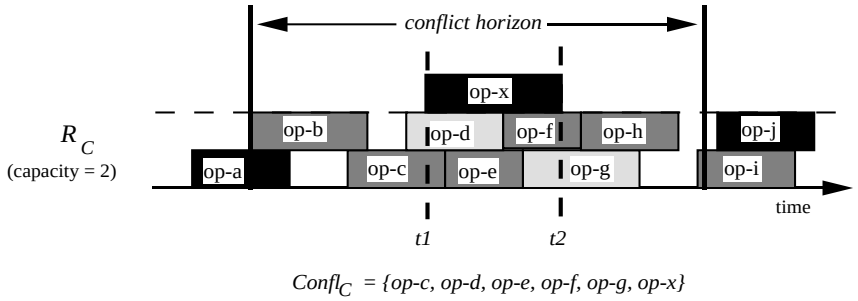

Figure 7. A capacity conflict

remain valid. The problem lies only in the timing details. If the duration is high, however, this assumption is not valid.

Conflict Size is defined as $\mid Retract_C \mid$, where $Retract_C$ is an identified (minimal) set of operations in $Confl_C$ that must be retracted (and hence rescheduled) to restore consistency (in the case of process-based aggregate conflicts, $Retract_C$ is computed for each constituent capacity conflict and conflict size is the average size of these sets). $Retract_C$ is constructed with the assumption that the operation whose commitment created the conflict will not be moved, and this set is retained for later use in formulating revision tasks. In Figure 7, for example, $Retract_C = \{op-d, op-f\}$ and $\mid Retract_C \mid = 2$ if we assume all operations are of equal priority. If the conflict size is low, then it is reasonable to assume that most sequencing decisions relative to R_C are valid. For example, if conflict size is 1, then the operation in $Retract_C$ may be the only one out of sequence. If the number is high, then sequence optimization at R_C is an important concern.

Resource Idle Time is defined as the average amount of capacity available at R_C over the conflict horizon (recall we are typically speaking of an aggregate resource). Figure 8 indicates periods of available capacity (or resource idle time) in the case of the conflict contained in Figure 7. If average idle time is low, indicating a bottleneck situation, then optimization of R_C's schedule to achieve maximum throughput is a primary concern. If average idle time is high, then resource-based optimization is unimportant. Resource Idle time is also computed for any disjunctive aggregate capacity resource that contains R_C in the underlying hierarchical model. An increase in resource idle time moving upward in the resource hierarchy indicates flexibility to assign alternative resources to operations in $Confl_C$.

Local Upstream Slack measures the flexibility in the scheduled start times of the operations scheduled on R_C. The upstream slack of a given job is defined simply as the difference between the scheduled start

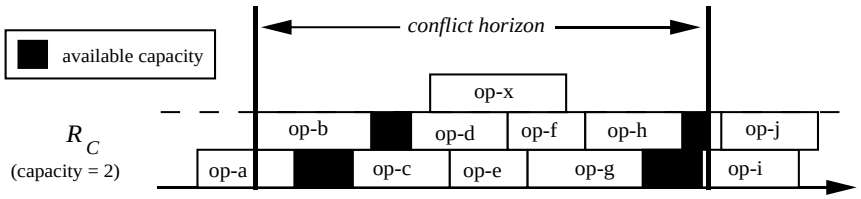

Figure 8. Periods of idle time within the conflict horizon

time of its operation on R_C and the scheduled (or actual) end time of its immediately preceding operation, and local upstream slack is the average slack over all jobs scheduled on R_C within the conflict horizon. Figure 9 expands the schedule given in Figure 7 to include portions of the upstream and downstream schedule (on resources R_i and R_j respectively), and indicates the upstream slack associated with jobs c, e, g, and h (in this figure the operation sequence $op-i' \rightarrow op-i \rightarrow op-i''$ is assumed for each job i). If the local upstream slack of operations requiring R_C is high, then there are opportunities for resequencing on R_C. It might be possible to place operations into the "holes" of available capacity that will be vacated by operations in $Confl_C$.

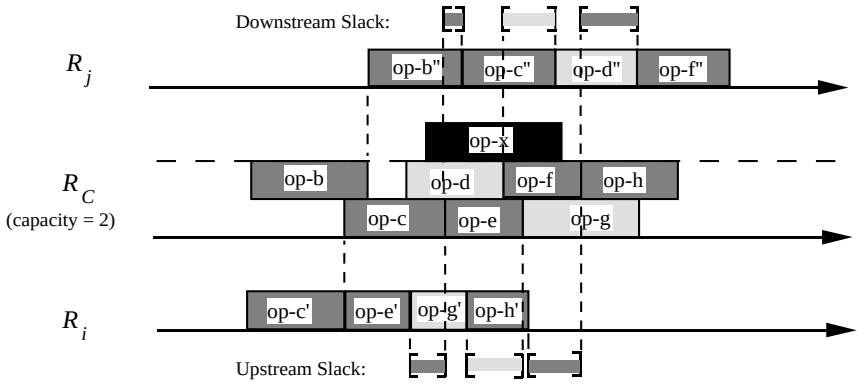

Figure 9. Upstream and downstream slack

Local Downstream Slack, alternatively, captures the local flexibility in the scheduled end times of operations currently scheduled on R_C.

In defining this measure, we appeal to an assumption concerning characteristics of a good schedule, namely that good schedules will exhibit queue times only before bottleneck resources. Given this assumption, we define local downstream slack to be the average of the durations of the first scheduled delay in the downstream schedules of each process/job scheduled on R_C within the conflict horizon. If there are no scheduled delays in the schedule of a given process, then there is no local slack. Figure 9 (top) indicates the downstream slack for jobs b, c, d, and f scheduled on R_C in the simple case where there is only one successor operation. If downstream slack is low, then downstream resource contention is not likely to be severe (i.e., there are no apparent downstream bottlenecks). If downstream slack is high, there is evidence of at least one downstream bottleneck and optimization of resource schedules further downstream can be important.

Projected Lateness defines another average measure of temporal flexibility, this time relative to the operation(s) in $Confl_C$. The projected lateness of an operation is defined similarly to local downstream slack, except now we are interested in the difference between the earliest time that the process/job can arrive at the downstream bottleneck and its scheduled start time on that resource. If there is no downstream bottleneck, then we are interested in the difference between the earliest the process can finish and its due date. If the projected lateness of an operation in $Confl_C$ is negative (i.e., the operation is still "early" relative to its current deadline), then resource-based optimization is unimportant. If the lateness is positive, then resource-based optimization is important.

Variance in Projected Lateness is defined with respect to all operations scheduled on R_C within the conflict horizon. This provides an indication of the opportunities for pair-wise optimization of process/job schedules. If the variance is high, then it might be possible to trade off positive and negative projected latenesses of specific processes by swapping demands.

5.2.3. Subtask Formulation

Determination of how to best resolve a given conflict is based on a qualitative model that combines the above stated implications of computed analysis metrics with knowledge of the differential optimization and conflict resolution capabilities of alternative revision actions. Given the results of conflict analysis, the model yields the revision method whose behavioral characteristics best match recognized revision needs and opportunities. The reasoning process underlying construction of the model is illustrated in Figure 10. In this case, the presence of a "large" conflict duration indicates the need for some amount of resequencing at R_C. Because resource idle time at R_C is "low", efficient utilization of R_C to maximize throughput is needed. RSC is the strongest candidate from the standpoint of these needs, with the potential disadvantage of introducing time conflicts with downstream process operations. However, because upstream slack is "high", there are

opportunities for non-disruptive sequence changes at R_C. Hence, RSC is selected as the method to apply.[6]

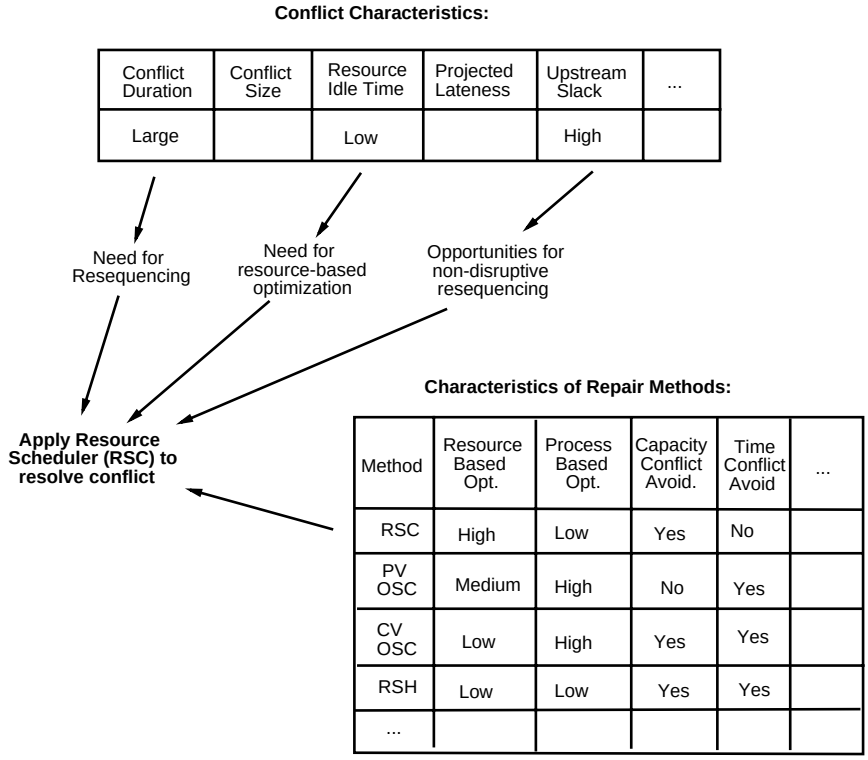

Figure 10. Mapping Analysis Results to Repair Actions

The set of rules that make up the constructed model is given in Table 1. Only one rule does not specify a unique choice. In this case, DSW is always selected first (because it typically acts to improve the character of the conflict), and then removed from consideration on the next cycle. As indicated previously, handling of opportunity events is restricted to those reflective of unexpected gains in available resource capacity. Such events are responded to only after a feasible schedule has been obtained, and always by applying LSH.

[6]Currently, simple thresholds on conflict analysis metric values are used to produce qualitative interpretations.

Confl. Dur.	*Confl. Size*	*Idle Time*	*Upstr. Slack*	*Proj. Late.*	σ^2 *(P.L.)*	*Action*
small						RSH
large		high		−		CV-OSC
large		high		+	high	DSW, PV-OSC
large		high		+	low	PV-OSC
large	large	low				RSC
large	small	low	high			RSC
large	small	low	low			PV-OSC

Table 1
Action Selection Model

Formulation of a scheduling task involving the selected action requires determination of scope (i.e., what particular set of scheduling decisions is to be revised). This decision is also based on information provided by conflict analysis, as well as knowledge relating to prior modification actions. Because revision methods differ in the types of schedule components they manipulate, decisions about scope depend on the specific method selected. If either OSC and RSH actions are to be applied to resolve a capacity conflict, then the process (or processes) to be rescheduled must be selected. Here the operation whose commitments originally introduced the conflict are left intact; scheduling tasks are created for the processes associated with each operation contained in the set $Retract_C$ (computed during conflict analysis).[7] For OSC actions, it is also necessary to delineate which portion of its downstream schedule to revise. If downstream slack is high (indicating the presence of downstream bottleneck resources), then scope is limited to the portion of the process's plan that precedes the downstream bottleneck operation.

In the case of RSC actions, decisions about scope concern the level of aggregation of the focal point resource (or, equivalently, how large a set of substitutable resource schedules to revise). This decision is a function of the increased rescheduling flexibility that can be gained by broadening scope. If a disjunctive aggregate resource is found moving upward the hierarchical model with significantly higher estimated resource idle time than that of R_C, then that resource is selected as the focal point of the RSC action; otherwise scope is restricted to R_C. DSW tasks, finally, require designation of a set of processes to consider, which is taken to be the set of operations scheduled on R_C over the conflict horizon.

5.3. An Example

To illustrate the behavior of the OPIS reactive scheduler, we consider a sample reactive scenario involving response to an unexpected loss in

[7] Given the current action selection model, $| Retract_C |$ is rarely > 1 when OSC or RSH is selected.

resource capacity. Figure 11 graphically depicts (again in Gantt chart form) the current schedule for a portion of a hypothetical factory. The diagram indicates machine and time assignments of operations for nine jobs (or orders) covering three stages of the manufacturing process. The operations associated with each particular job are related by temporal precedence constraints which are designated by directed arcs in the diagram. The pattern used to designate the operations of a job reflects the job's "part type", which has implications with respect to machine setup requirements at different stages of the manufacturing process (there are four types of parts being manufactured in this example). We assume in this example that all assigned machines are unit-capacity resources (i.e., capable of servicing only one job at a time); thus, the time line associated with a given machine in the diagram indicates either a scheduled operation, a scheduled setup operation (represented as a black rectangle), or a period of availability (represented as blank space).

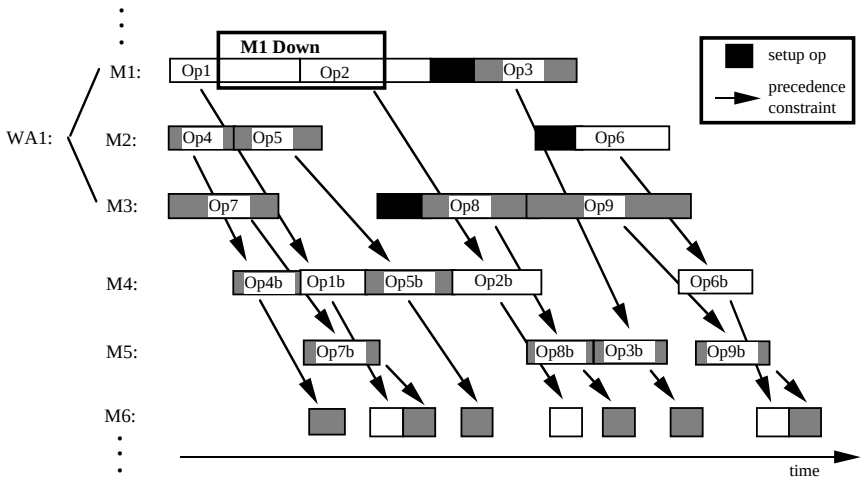

Figure 11. Unexpected Machine Breakdown

The three stages of the manufacturing process covered in the diagram impose the following allocation constraints. Machines *M1, M2,* and *M3* are organized into work area *WA1* and are utilized in the first stage of the process. Each of these machines is capable of manufacturing any of the four part types; however, a machine setup must be incurred in switching from one part type to another. Machines *M4* and *M5* are used in the second stage of the process. Each of these machines is specialized to the

manufacture of two unique part types and there is no setup required for part changeovers. Machine *M6*, finally, is used in the third stage for all part types (again with no setup requirements).

As indicated in Figure 11, machine *M1* is expected to be down for an interval that overlaps the ongoing execution of *Op1* and the scheduled execution of *Op2*. In this case, the already completed portion of *Op1* is salvageable. Introduction of these newly known constraints into the current solution (via execution of a model update task) results in (1) a split of *Op1* into a two operation sequence *(Op1 Op1*)* reflecting completed and uncompleted portions of the original *Op1*, (2) allocation of a "machine repair" operation for the designated interval, and (3) subsequent detection and posting of two capacity conflicts. Because both conflicts involve the same resource, they are aggregated for simultaneous consideration. Analysis of this aggregate event indicates high resource contention over the conflict horizon, positive projected lateness in the jobs involved in the conflict, and a large conflict size, leading to formulation of an RSC revision task. Given that resource contention is lower at the aggregate *WA1* level in the hierarchical model (suggesting resequencing flexibility if alternative machine assignments are considered), *WA1* is chosen as the focal point of the RSC task. The revisions made by the RSC in this step are indicated in Figure 12. The resulting solution state is given in Figure 13.

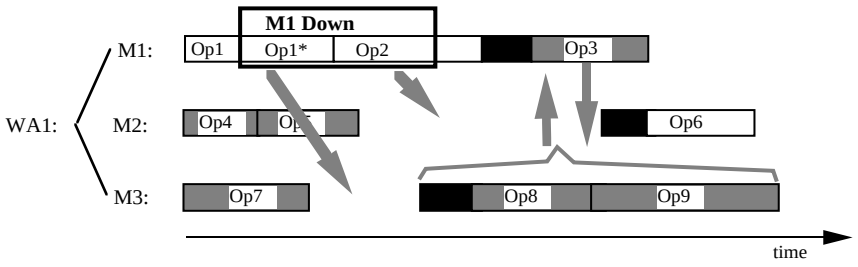

Figure 12. Action of RSC in revising *WA1* schedule

Execution of the RSC scheduling task introduces two new time conflicts, reflecting precedence violations between operation pairs *(Op1* Op1b)* and *(Op2 Op2b)* respectively. On the basis of conflict urgency, problem selection determines an initial focus on the conflict involving Job1. Analysis of this conflict indicates relatively low resource contention on $M4$ over the conflict horizon, but positive projected lateness in the job involved in the conflict. Accordingly, a PV-OSC scheduling task is formulated to provide an aggressive approach to revising the downstream portion of Job1's sched-

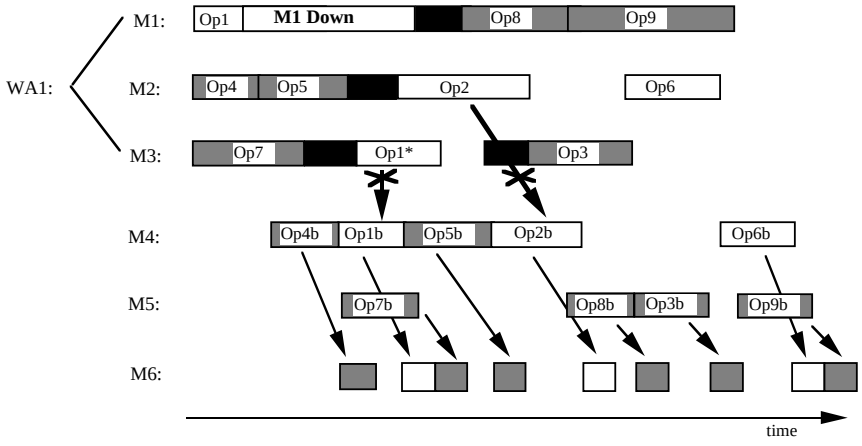

Figure 13. Solution State after revision of *WA1* schedule

ule. Figure 14 indicates the result of this action. Because weighted tardiness prioritization favors Job1 over both Job5 and Job2, decisions are made to utilize machine capacity currently allocated to both of these jobs. Upon commitment to these decisions, the depicted capacity conflicts are detected and posted. The previously detected precedence violation between the operation pair ($Op2Op2b$) persists also (the earliest feasible start time of $Op2b$ is indicated by the "[" in Figure 14).

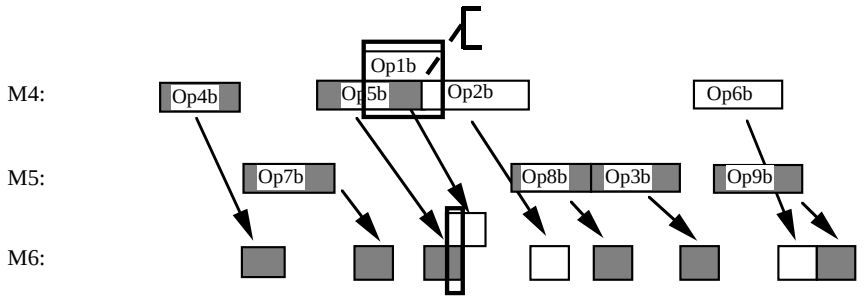

Figure 14. Scheduling state after PV-OSC (init-op = Op1b)

In this case, commonality in the jobs involved in each capacity con-

flict results in creation of an aggregate conflict event. Subsequent conflict analysis again indicates relatively low contention at resource focal points as well as the presence of rescheduling flexibility in the form of upstream slack. A CV-OSC scheduling task is formulated to reschedule the downstream operations of Job5 from *Op5b* forward using existing intervals of available capacity. The results of this action are indicated in Figure 15.

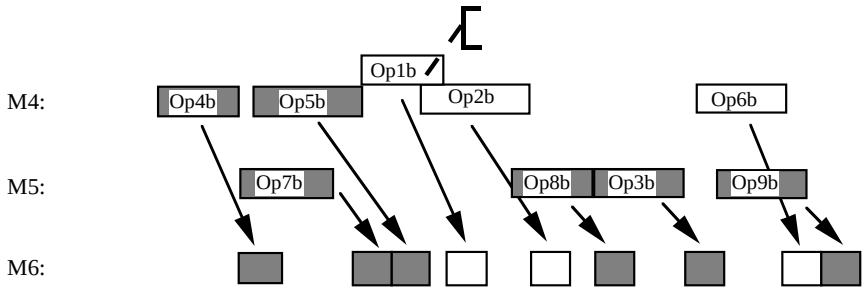

Figure 15. Solution state after CV-OSC (init-op = Op5b)

The final remaining conflict is also resolved using CV-OSC, because of the absence of significant downstream resource contention and negative projected job lateness. The final revised schedule is shown in Figure 16.

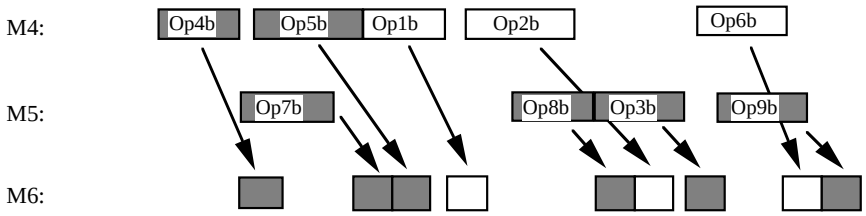

Figure 16. Final solution

5.4. Characteristics of the Approach

The OPIS scheduling methodology makes several pragmatic assumptions to deal with the special characteristics and complexities of the schedule modification task pointed out in Section 2. Each of these assumptions

involves compromise in some respect, but it is also these assumptions that make the approach viable in practical scheduling domains. In this section, we examine these assumptions in relation to the issues previously raised.

Bounding the scope of a given reaction. On each cycle of the repair process, OPIS makes heuristic decisions as to the scope of the next schedule modification action. These decisions are based on implications of the structure of current solution constraints as encoded in its subproblem formulation model. However, the conflict analysis metrics on which these implications are founded are, by computational necessity, aggregate and interpreted qualitatively. There is no guarantee that solution of the delineated subproblem (which limits visibility to a subset of the current solution constraints) will not introduce additional constraint conflicts into the schedule, and there are accordingly no a priori bounds that can be placed on the extent to which change will ripple through the solution. The expectation is that, on average, the heuristic model will limit the amount of revision performed to what is required (given expected performance biases - see below). Experimentation has borne out this expectation. Although missed opportunities for more localized change have been observed in specific reactive circumstances (e.g., when the current state falls at the boundaries of the model), overall results indicate that, on average, this model significantly outperforms other, more rigid and less informed schedule revision strategies [25]. Moreover, because the model is based on generic properties of the underlying solution constraint graph, it is applicable across a range of scheduling domains (and certainly generalizable to others).

Trading off attending to scheduling objectives for being non-disruptive. The degree of emphasis given toward either of these concerns is also a function of the heuristic subproblem formulation model employed. Abstract characterizations of the current solution state have implications with respect to both optimization needs and opportunities for non-disruptive modification, and these implications are not always synergistic. In such circumstances, the current OPIS subproblem formulation model is biased toward modification actions that preserve schedule quality. Other models are of course possible. For example, within the NEGOPRO software project scheduler [23], which adopts a similar constraint-directed, multi-perspective approach, a subproblem formulation model biased in the opposite direction is employed (in this domain, continuity of resource assignments is preferred even at the expense of acquiring more resources when milestones slip).

"Time to execution" bias. Although the above tradeoff has implications with respect to computational efficiency, the iterative nature of the modification process allows the need for responsiveness in reactive situations to be treated separately. On any given cycle of the iterative process, constraint propagation produces descriptions of the currently pending set of constraint conflicts and improvement opportunities. If prioritization of this set is based first on the temporal proximity of these problems to the current time then it can be ensured that conflicts blocking execution are

attended to in bounded time. Moreover, because new external updates are imported on each cycle, focus can shift to more pressing problems as they arise.

Guarantee of a feasible solution. The existence of a feasible solution is guaranteed in any modification context by assuming that constraints are "infinitely relaxable" along at least one dimension (e.g., activities can always be delayed, additional resource capacity can always be acquired, processes can always be dropped from the schedule). This flexibility is counterbalanced by the inclusion of corresponding scheduling objectives to minimize relaxation to the extent possible (e.g., minimize tardiness), and the subproblem formulation model's overall bias toward maintaining solution quality.

6. Relationship to Other Work

Work in reactive scheduling and schedule revision techniques has gained considerable prominence within the field of knowledge-based scheduling in recent years. Relative to manufacturing scheduling, several alternative constraint-based approaches have appeared. In [5], a blackboard-based system for generating and revising factory schedules based fairly directly on the concepts of OPIS (although employing different revision operators) is described. One interesting aspect of this work, from an architectural perspective, was its use of a decomposable framework for temporal constraint propagation that provided a basis for explicitly controlling the amount of constraint propagation performed in different problem solving circumstances, and thus an ability to trade off the time spent revising the schedule against the quality of the reaction.

In [4], a distributed constraint satisfaction problem solving (CSP) approach to reactive scheduling based on a hierarchically organized set of scheduling agents is defined. Each agent is given responsibility for making and retracting specific commitments (time assignments on a particular machine, machine assignments in a given work area, due dates of current jobs). Schedule revision proceeds as a collective activity based on dependency-directed backtracking search, with the single strategic agent bearing responsibility trading off global objectives when it has been established that current constraints cannot be met. From a behavioral standpoint, the interaction protocols employed give rise to a reactive strategy where scope is systematically enlarged from individual machine schedules, to groups of substitutable machines, and finally to consideration of process due date relaxation. The real issue here is scalability, given the reliance on systematic backtrack search.

Recent work in iterative repair scheduling approaches [3, 13, 16, 33] shares the same incomplete search assumption as OPIS in approaching conflict resolution. However, this work has principally focused on a different class of scheduling problems (e.g., space mission scheduling) where absolute

temporal constraints are not relaxable and there is generally less temporal structure to processes. The optimization problem here can be seen as minimizing the number of capacity conflicts (because any conflicts that cannot be resolved will require processes to be dropped from the schedule), and thus the tradeoffs emphasized in the OPIS reactive model concerning tardiness and WIP minimization are not relevant. One exception is the space shuttle processing domain of [33], which presents a complex "project scheduling" problem that is much closer in character to manufacturing scheduling problems. However, the approach taken here treats due dates as non-relaxable and attempts to minimize the number of capacity conflicts in the final schedule.

Work in iterative repair scheduling also differs in the approach taken to control of the revision process. Whereas OPIS relies on knowledge about problem structure to direct application of sophisticated local search methods, most of this work has instead emphasized more extensive global search with simpler (typically smaller granularity) repair heuristics. In [13, 16], a "min-conflict" heuristic, which revises a single conflicting commitment with bias toward minimizing the number of new conflicts introduced, is repeatedly applied to attempt to find a conflict free schedule from an initial randomly generated set of commitments (using periodic restarts as a hedge against local minima traps). In [33], a more complex heuristic that ensures continued consistency of relative temporal process constraints while it moves a conflicting operation is repeatedly applied within a simulated annealing search framework. In this case, provisions are provided for introduction of additional repair heuristics to deal with constraint violations other than capacity conflicts, and utility-based penalty functions associated with constraints and objectives are used to evaluate alternative solutions. The approach taken in [3] is perhaps the closest in concept to the OPIS methodology. Here extensive initial global search, which randomly shuffles activities to other regions on the time line to relieve capacity conflicts, is used as a basis for discovering inherent bottleneck regions in the space and this information is used to direct the application of more informed schedule revision operators.

7. Extensions and Current Focus

Subsequent research in manufacturing scheduling domains has explored frameworks for distributing and refining variants of the scheduling methodology employed by OPIS. In [28, 12], a structural decomposition of the factory, employing hierarchical descriptions of time and resource capacity constraints, was used as a basis for extending the OPIS framework to manage schedules at different levels over different time horizons. The CSS scheduler [21], alternatively explored a decomposition of scheduling responsibility among a set of "resource broker agents" (each concerned with efficient utilization of a specific set of resources) and a single "work order

manager" agent (concerned with process oriented constraints and objectives). In [24], a framework for partitioning responsibility between a global scheduler and a set of local "dispatching" agents was defined, emphasizing exploitation of existing temporal flexibility in the global schedule as a means for localized, execution-time schedule repair as circumstances warrant. This framework is currently being further developed for operational use in real-time control of an INTEL wafer fabrication facility.

One area of current research is investigating the broader applicability of these results and the basic OPIS schedule repair methodology to the problem of distributed management of large-scale transportation schedules. Work here has led to development of the DITOPS (DIstributed Transportation Scheduling in OPIS) system [32], which is currently being applied to problems in military crisis-action deployment logistics. Generalizations and extensions have been made to the original OPIS representational framework, the repertoire of revision methods available, and the heuristic subproblem formulation model to better reflect the character of the important constraints in this domain and the dimensions along which constraint relaxation and schedule modification should proceed. We have demonstrated capabilities to intelligently respond to a range of reactive problems (e.g., port closings, unexpected unavailability of transport vehicles, changes to material movement requirements), and are currently engaged in integrating DITOPS with other, complementary transportation planning technologies to support a large-scale demonstration in an operational scenario.

A second thrust of current research focuses on integration of incremental, reactive scheduling techniques with user decision-making processes, and the development of flexible interactive scheduling tools. As we argued at the outset of this paper, scheduling in most practical environments is very much an iterative process of "getting the constraints right". Incremental schedule revision techniques such as those developed within the OPIS scheduler are particularly well suited to the form of decision support that this process implies, particularly in substantial scheduling environments (manufacturing-related or otherwise) where it is unreasonable to expect users to productively interact with a scheduling system at the level of individual decisions in the system's solution model. In these domains, there are simply too many decisions to consider and effectively manipulate on an individual basis (e.g., via manual Gantt chart editing with constraint checking), and the details of most decisions are generally unimportant from the standpoint of user tasks and goals. The reactive scheduling methodology of OPIS, alternatively, provides the foundation for an interactive framework that preserves the user's ability to exercise explicit control over the solution change process (essential for exploration, understanding and calibration of possible solution improvements and adjustments) while promoting user interaction in higher-level and more comprehensible task-oriented terms (e.g., reschedule job x to complete by Friday, add sufficient overtime to complete next week's orders on time). Building on the notions of hierarchical model-

ing and opportunistic subproblem formulation defined in the OPIS control architecture, we are developing an interactive scheduling framework that operationalizes this view. Our vision is a graphical, spreadsheet-like model of user/system interaction, where schedules are visualized, analyzed and manipulated from aggregate decision-making perspectives, and the system incrementally "manages the details" of solution change in a manner consistent with user expectations. [31].

A third complementary component of our current research aims at simplification of the application building process. Though we believe that reactive scheduling methodology developed in OPIS is relevant to a broad range of applications, different scheduling environments invariably present different challenges. There is diversity along several dimensions: in the structure of different domains (e.g., type of manufacturing system and discipline), in the types of constraints that dominate, in the performance objectives and preferences that must be attended to, and in the types of uncertainties must be accommodated; and problem characteristics along each of these dimensions will dictate the modeling assumptions, scheduling heuristics, and solution procedures most appropriate for successful application. The OPIS scheduling architecture provides a modeling and scheduling system infra-structure for encoding and integrating application-specific solution components. However, at the software level, its reliance on frame-based implementation techniques provide only limited support for encapsulation of component functionality and exploitation of a layered system semantics. We have recently completed an initial redesign and reimplementation of the basic OPIS framework, based on more modern object-based analysis and programming techniques. Our goal here is a application building "toolbox", which provides an a compositional library of system building blocks (e.g., constraint representation and propagation techniques, (re)scheduling methods and heuristics, subproblem formulation policies) and an explicit hierarchy of protocols for configuring and customizing schedulers to meet the requirements of specific applications.[30]

A final area of current research is exploring the development and use of constraint-based schedule revision strategies with more flexible "temporal data base" representations of current solution constraints (e.g., [6]). Within such representational frameworks, start and end times of scheduled activities are generally not fixed to specific points in time, but instead are constrained to intervals that satisfy posted temporal constraints. This allows, for example, the possibility of developing schedules that anticipate executional uncertainty and defer commitment on timing details whenever possible and appropriate. This work is being carried out with the HSTS system[18], an integrated planning and scheduling architecture that provides such a representational framework.

8. Acknowledgments

The OPIS scheduler is the result of the contributions of many individuals spanning a period of many years. Peng Si Ow provided inspiration for many of the ideas underlying the approach and was a research partner throughout the original system development. Discussions with Mark Fox have also had a major influence in shaping the OPIS approach. Other individuals who have made valuable contributions include Gilad Amiri, Dina Berkowitz, Casper Cheng, Geir Hasle, Juha Hynynen, Ora Lassila, Claude LePape, Jyi-Shane Liu, Dirk Matthys, Bruce McLaren, Pedro Muro, Nicola Muscettola, Kevin Neel, Kikuo Ogaki, Jean-Yves Potvin, Ali Safavi, Toshihiro Satomi, Katia Sycara, and Christopher Young.

This research has been sponsored in part by the Air Force Office of Scientific Research under contract F49620-82-K-0017, International Business Machines Inc. under contract number 71223046, the Advanced Research Projects Agency under contract F30602-90-C-0119 and the CMU Robotics Institute.

An earlier version of this paper appeared as "OPIS: A Methodology and Architecture for Reactive Scheduling" in *Intelligent Scheduling*, (eds. M. Zweben and M.S. Fox), Morgan Kaufmann Publishers, San Mateo, CA, 1994.

REFERENCES

1. Amiri, G. and S.F. Smith, "A Comparative Analysis of Constraint-Based Scheduling Strategies", The Robotics Institute, Carnegie Mellon University, Tech. Report, September, 1992.
2. Bean, J.C., and J.R. Birge, "Match-Up Real-Time Scheduling", Technical Report No. 85-22, University of Michigan, Department of Industrial and Operations Engineering, June, 1985.
3. Biefeld, E. and L. Cooper, "Operations Mission Planner: Final Report", Jet Propulsion Laboratory, Tech. Report 90-16, March, 1990.
4. Burke, P., and P. Prosser, "A Distributed Asynchronous System for Predictive and Reactive Scheduling", Tech. Report AISL42, Dept. of Computer Science, University of Strathclyde, 1989.
5. Collinot, A., C. LePape, and G. Pinoteau, "SONIA: A Knowledge-Based Scheduling System", *Artificial Intelligence in Engineering*, 2 (2), 1988, pp. 86-94.
6. Dean,T.L. and D.V. McDermott, "Temporal Data Base Management", *Artificial Intelligence*, 32 (1), April, 1987, pp. 1-56.
7. Erman, L.D., F. Hayes-Roth, V.R. Lesser and D.R. Reddy, "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty", *Computing Surveys*, Vol. 12, No. 2, 1980.
8. Fox, M.S., "Constraint-Directed Search: A Case Study in Job Shop Scheduling", PhD. Thesis, Dept. of Computer Science, Carnegie Mellon University, 1983.

9. Fox, M.S., and S.F. Smith, "ISIS: A Knowledge-Based System for Factory Scheduling", *Expert Systems*, 1 (1), July, 1984, pp. 25-49.
10. Fox, M.S., N. Sadeh, and C. Baycan, "Constrained Heuristic Search", *Proceedings IJCAI-89*, Detroit, MI, August, 1989.
11. Graves, S.C., "A Review of Production Scheduling", *Operations Research*, 29 (4), 1981, pp. 646-675.
12. Hynynen, J., "A Framework for Coordination in Distributed Production Management", Ph.D. Thesis, Helsinki University of Technology, Laboratory of Information Processing Science, October, 1988.
13. Johnston, M., "SPIKE: AI Scheduling for NASA's Hubble Space Telescope", *Proceedings 6th IEEE Conference on Artificial Intelligence Applications*, Santa Barbara, CA, March, 1990, pp. 184-190.
14. LePape, C. and S.F. Smith, "Management of Temporal Constraints for Factory Scheduling", Technical Report TR-CMU-RI-87-13, The Robotics Institute, Carnegie Mellon University, June, 1987.
15. Meleton, M.P., "OPT: Fantasy or Breakthrough", *Production and Inventory Management*, Second Quarter, 1986, pp. 13-21.
16. Minton, S., M.D. Johnston, A.B. Phillips and P. Laird, "Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method", *Proceedings AAAI-90*, Boston, July, 1990.
17. Johnson, L.A. and D.C. Montgomery, *Operations Research in Production Planning, Scheduling and Inventory Control*, John Wiley, New York, 1974.
18. Muscettola, N., S.F. Smith, A. Cesta, and D. D'Aloisi, "Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Framework", *IEEE Control Systems*, Vol. 12, No. 1 (February, 1992).
19. Ow, P.S.,"Focused Scheduling in Proportionate Flowshops", *Management Science*, 31 (7), July, 1985, pp. 852-869.
20. Ow, P.S., S.F. Smith and A. Thiriez, "Reactive Plan Revision", *Proceedings AAAI-88*, St. Paul, MN, August, 1988.
21. Ow, P.S., S.F. Smith and R.E. Howie, "CSS: A Cooperative Scheduling System", in *Expert Systems and Intelligent Manufacturing*, (ed.) M.D. Oliff, Elsevier Science Publishing Co., 1988.
22. Ow, P.S. and S.F. Smith, "Viewing Scheduling as an Opportunistic Problem Solving Process", in *Annals of Operation Research*, Vol. 12, (ed.) R. Jareslow, Baltzer Scientific Publishing Co., 1988.
23. Safavi, A. and S.F. Smith, "A Heuristic Search Approach to Planning and Scheduling of Software Projects", in *Advances in Artificial Intelligence: Natural Language, and Knowledge-Based Systems*, (ed.) M. Golumbic, Springer-Verlag Publishers, 1990.
24. Smith, S.F., N. Keng, and K. Kempf, "Exploiting Local Flexibility During Execution of Pre-Computed Schedules", in *Applications of AI in Manufacturing* (eds. D. Nau and C. Tong), MIT Press, July, 1992.
25. Smith, S.F., P.S. Ow, N. Muscettola, J.Y. Potvin, and D. Matthys, "An Integrated Framework for Generating and Revising Factory Schedules",

Journal of the Operational Research Society, 41(6), June, 1990.

26. Smith, S.F., "The OPIS Framework for Modeling Manufacturing Systems", Technical Report TR-CMU-RI-89-30, The Robotics Institute, Carnegie Mellon University, December, 1989.
27. Smith, S.F., "A Constraint-Based Framework for Reactive Management of Factory Schedules", in *Intelligent Manufacturing*, (ed.) M.D. Oliff, Benjamin Cummings Publishers, 1987.
28. Smith, S.F. and J.E. Hynynen, "Integrated Decentralization of Production Management: An Approach for Factory Scheduling", in *Intelligent and Integrated Manufacturing Analysis and Synthesis*, (eds.) C.R. Liu, A. Requicha, and S. Chandrasekar, ASME PED-Vol. 25, New York, 1987.
29. Smith, S.F. and P.S. Ow, "The Use of Multiple Problem Decompositions in Time-Constrained Planning Tasks", *Proceedings IJCAI-85*, Los Angeles, CA, August, 1985.
30. Smith, S.F. and O. Lassila, "Configurable Systems for Reactive Production Management", in *Knowledge-Based Reactive Scheduling*, (eds.) R. Kerr and K. Szelke, IFIP Transactions 15-B, North Holland Publishers, Amsterdam, 1994.
31. Smith, S.F. and O. Lassila, "Toward the Development of Flexible Tools for Mixed-Initiative Transportation Scheduling", *Proceedings 1994 ARPA Planning Workshop*, Tuscon, AR, February, 1994.
32. Smith, S.F. and K.P. Sycara, "Distributed, Multi-Level Management of Transportation Schedules", Technical Report CMU-RI-TR-93-17, The Robotics Institute, Carnegie Mellon University, December, 1993.
33. Zweben, M., M. Deale, and R. Gargan, "Anytime Rescheduling", *Proceedings 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, (ed.) K.P. Sycara, Morgan Kaufmann Publishers, 1990.