NISTIR 6037

State Models for Jobs and Job Supervisors

Contents

List of Figures

List of Tables

This work was funded through the National Advanced Manufacturing Testbed (NAMT) project and the Systems Integration for Manufacturing Applications (SIMA) program.

State Models for Jobs and Job Supervisors

David Flater
Edward Barkmeyer
Evan Wallace

July 8, 1997

Abstract

Computerized supervision of discrete parts manufacturing jobs and machinery requires state models for the jobs and the supervisors. Different approaches have yielded different state models. While a dominant model is yet to emerge, the advantages and shortcomings of the existing models suggest some design principles for future efforts; most importantly, the demarcations between the shop level of control and the workcell level, and between the job, the lot, the supervisor, and the machine, should be well-defined.

Keywords: manufacturing, control, automata

1 Introduction

In the manufacturing domain, there are certain concepts that are common to all supervisory systems. Jobs are entered, dispatched, and tracked. If something goes wrong with a job, then there are various ways to stop it, ranging from "pause at the next convenient checkpoint" to "drop what you are doing, immediately." Similarly, if something goes wrong with a machine, options range from "pause at the next convenient checkpoint" to "emergency stop, *now*."

Given this basic commonality among supervisory applications, it is surprising to discover that there can be differences between the state models of different supervisors that render them incompatible with one another in significant ways. This report is an attempt to survey a range of state models, compare the costs and benefits, and identify the design principles that should guide future development or standardization efforts.

Within this document, the names of states (e.g., Pausing, Stopping, Aborting) will be capitalized to distinguish them from mere adjectives. Furthermore, in order to show the commonalities between one model and the next despite differing terminologies, the following global state concepts will be indicated by capitalized, italicized names:

- *Queued:* a job that is *Queued* has been accepted by a supervisor but not yet started.
- *Executing:* the job or supervisor is running normally.
- *Pausing:* the job or supervisor will pause at the next convenient checkpoint. A *Pausing* job will pause after the currently running task is completed. A *Pausing* supervisor will continue to accept and queue jobs but will not start any of them; jobs already started are permitted to complete.
- *Paused:* the job or supervisor has reached a checkpoint and paused. All allocated resources are retained and schedules are left intact in anticipation of a quick resumption. A supervisor in the *Paused* state will continue to accept and queue jobs but not start them.
- *Held:* the job has reached a checkpoint and paused, its remaining tasks have been unscheduled, and any allocated resources have been temporarily freed up. *Hold* is merely a more severe form of *Pause*. *Hold* is not applicable to supervisors.

- *Released:* this is the analog of *Queued* for *Held* jobs that have been resumed. While in this state, the job's remaining tasks are rescheduled and its resources are re-allocated; then it is again *Executing*.

- *Stopping:* the job or supervisor will stop at the next convenient checkpoint. All future tasks of a *Stopping* job are scrubbed and any allocated resources are freed up for other use. *Stop* is more severe than *Hold* in that once the job is *Stopped* the supervisor considers it done. A supervisor that is *Stopping* will not accept or queue new jobs.

- *Stopped:* the job or supervisor has reached a checkpoint and stopped. For a job, this is a terminal state; resuming a *Stopped* job can only be achieved by *Queue*ing the remaining tasks as a new job. For a supervisor, *Stopped* is merely an inactive state from which it may be revived.

- *Aborting:* the job or supervisor will stop as soon as possible without damaging equipment. Any workpieces or jobs that were being worked on will be left in an indeterminate state.

- *Aborted:* this is the terminal state of a job or supervisor that was told to abort. A supervisor in the *Aborted* state has gone off-line.

- *E-stopped:* this is only applicable to supervisors, and represents an immediate emergency shutdown, no questions asked. Equipment might be damaged or left in an undesirable state by an inopportune e-stop.

- *Fault:* a job or supervisor in the *Fault* state is suffering from a problem that cannot be resolved without human intervention. It may end up in any state at the discretion of the operator.

Similarly, the verbs *Queue, Pause, Stop,* and *Abort* will be used to indicate the actions or commands that place a job or supervisor into the aforementioned states.

2 Survey of State Models

In order to explain the state models in following sections, two concepts are needed: *concurrent states* and *derived states*. These concepts are based on the work of David Harel[1].

Concurrent states are particular states modeled as the composition of two or more independent state variables. For example, if a cabinet has two doors, the state of the cabinet can be modeled by composing the states of the two doors, each of which is either Open or Closed. A diagram representing this concurrent state is shown in Figure 1. One could describe the state of the cabinet as Open-Open, Open-Closed, Closed-Open, or Closed-Closed. The concurrent representation is equivalent to a "flattened" representation containing those four states.

Derived states are particular states that are not directly implemented, but are inferred from the value of a state variable. For example, a climate control system may have the states Off, Heat, and Cool, but we can add a derived state, On. A diagram representing this model is shown in Figure 2.

While the usage of derived states does not impact the underlying structure of a state diagram, the usage of concurrent states has more significant costs and benefits that must be considered. Concurrent state diagrams greatly simplify the modeling of systems with many independent components, but they fail to capture the interactions between components that are not completely independent. For example, consider a cabinet whose doors are designed to interlock when closed, and the two doors must be closed together in order for them to interlock properly. A flattened state model showing the behavior of this cabinet is shown in Figure 3. In order to capture this new restriction in the concurrent model, it would be necessary to add informal annotations to the diagram.

Flattened state models have their own drawbacks; they are a poor way of modeling systems with many independent components, and the number of states and transitions that are needed to model complex systems makes the task very complicated and error-prone.

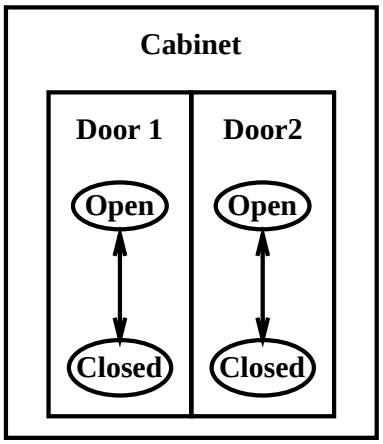

Figure 1: Concurrent states example

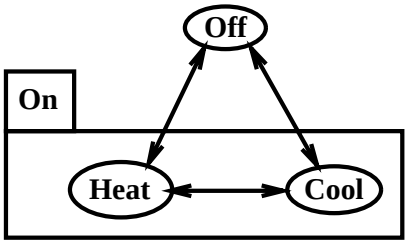

Figure 2: Derived state example

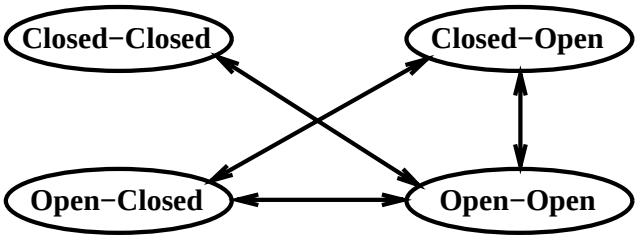

Figure 3: Flattened states example

2.1 MSI

MSI[2] (Manufacturing Systems Integration) is a NIST-developed architecture for hierarchical control and planning.

Jobs

The state model for MSI jobs (called tasks) is a concurrent state including two state variables, the task-state and the task-management-state. The task-state is what the task is actually *doing now*, while the task-management-state is what the task has been *told to do*.

Task-states include Active, Paused, Completed, Deferred, Terminated, Aborted, Waiting for Guardian, and Waiting for Planner. Deferred is merely a specialization of Aborted in which it is known that no damage has been done to the workpiece. Waiting for Guardian and Waiting for Planner are two different kinds of *Fault* states. Terminated is *Stopped*.

The MSI Pause concept is unique in allowing the operator to specify a deadline for when the pause should be completed, or to specify "as soon as possible." This allows the supervisor to optimize the "cleanness" of the pause and to choose the best of several possible checkpoints at which to pause.

Task-management-states include Normal, Pausing, Deferring, Terminating, and Aborting. The combined diagram is shown in Figure 4.

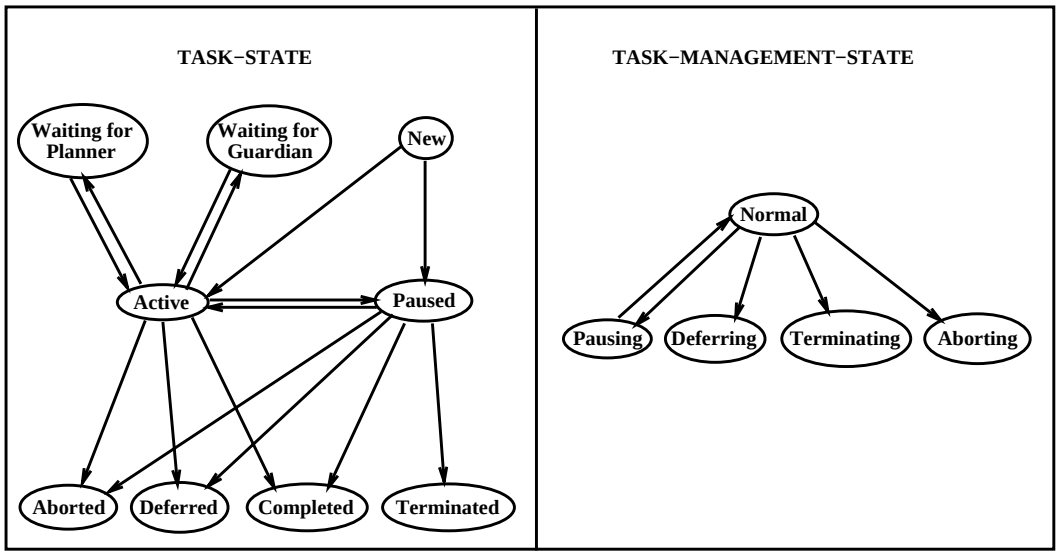

Figure 4: MSI job state model

Supervisors

MSI job supervisors have "administrative states" that describe what the supervisor is doing. These states include Available, Active, Pausing, Paused, Terminating, Terminated, and E-stopped. Figure 5 shows the state model.

The Pausing state is in response to a Pause All Tasks message. The supervisor cascades the pause down to all active tasks and enters the Paused state when all tasks have reached a checkpoint. While Paused, it will accept (*Queue*) new task requests but will not initiate (dispatch) new tasks. Terminating is analogous to Pausing, with all tasks being terminated (*Stopped*), but the supervisor does not accept new task requests.

MSI does not support a clean separation of the supervisor from all of its tasks; it is not possible to just *Stop* the supervisor without terminating all of the active tasks as well. (Instead of propagating the *Stop* downwards, the supervisor could simply quit dispatching new tasks and wait for the active tasks to complete normally.)

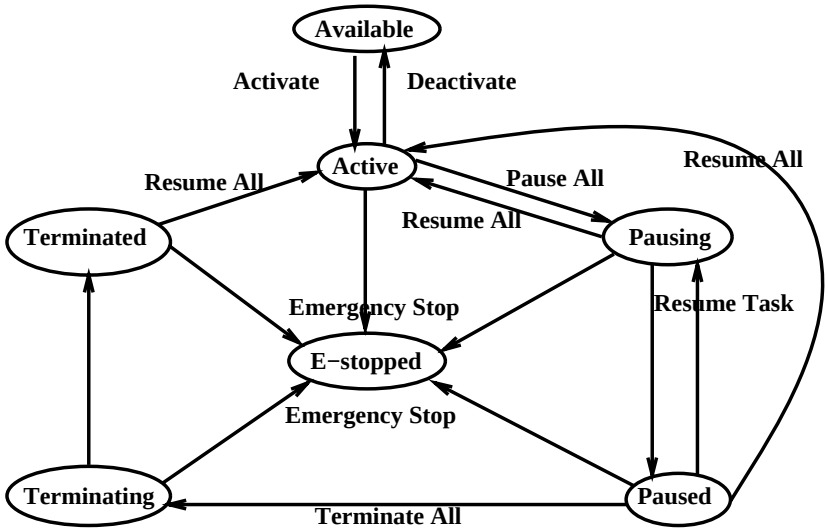

Figure 5: MSI supervisor state model

2.2 CIM Framework (CIMF)

The CIM Framework[3][1] is a specification published by SEMATECH, a consortium of U.S. semiconductor manufacturers. We were able to review an early work-in-progress version of the CIMF in collaboration with SEMATECH, to whom we are grateful for the opportunity to be part of the CIMF's evolution. Since the version that we reviewed was effectively only a draft, the issues that we describe below should not be interpreted as problems with the CIMF; our intent was merely to identify potential sources of misunderstanding that could slow the eventual deployment of the specification.

Jobs

The CIMF state model for ManagedJobs (the CIMF class for jobs) has three major parts. The first part pertains to jobs that have not yet been activated, and contains the states Created, Queued, and Cancelled. The second part pertains to active jobs, and encloses a four-way concurrent state combination with the derived state Active. The four aspects of Active are: Executing; Not Paused, Pausing, Paused; Not Stopping, Stopping; and Not Aborting, Aborting. The final part encloses the states Completed, Stopped, and Aborted with the derived state Finished. The CIMF states are compatible with *Pausing, Stopping,* etc., although they are defined more loosely. The state diagram is shown in Figure 6.

After reading the draft CIMF, we do not yet understand the rationale for permitting concurrent state combinations such as Executing-Paused-Stopping-Aborting. The text of the CIMF motivates the concurrent model by saying that it is flexible enough to handle such things as an Abort in the midst of a Stopping operation. However, this is also possible in models that do not have concurrent states, simply by transitioning from the Stopping state to the Aborting state. One member of SEMATECH posited that the concurrent states were an attempt to model the jobs of multi-faceted machines whose separate components may be in different states.

Supervisors

The CIMF supervisor class which matches the ManagedJob class is JobManager. Unfortunately, JobManager does not yet have a defined state model.

[1] Certain products and specifications are identified in this paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products and specifications identified are necessarily the best available for the purpose.

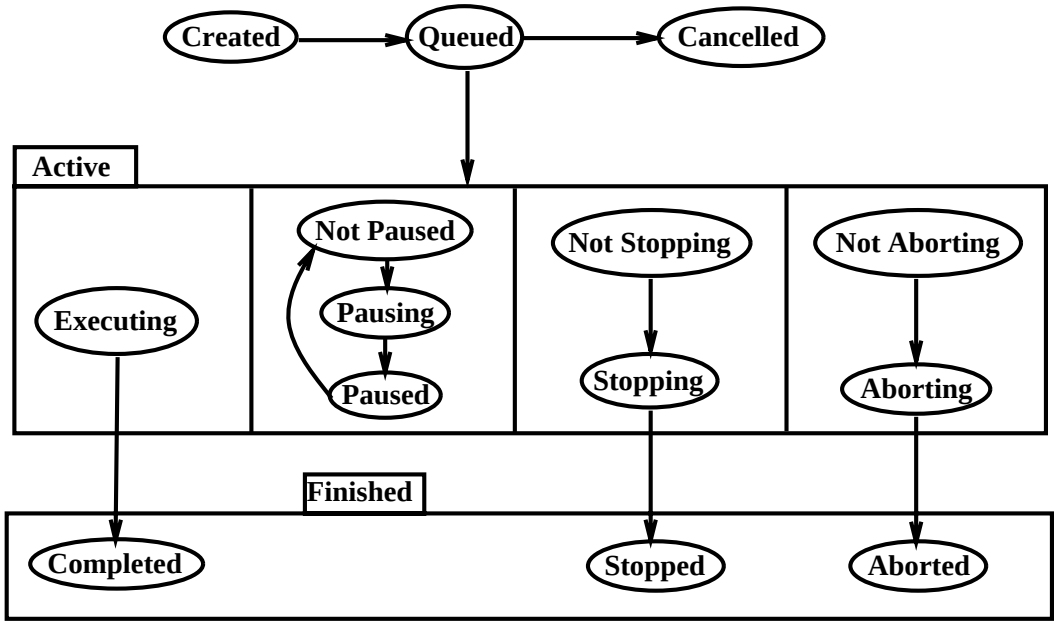

Figure 6: CIMF ManagedJob state model

The MachineResource class offers a state model that might give some indication of what the supervisor's model would look like. It provides Initializing, Waiting, and then a derived Active state consisting of the same four aspects as ManagedJob. This time, all arrows out of Active point back to Waiting; there are no terminal states. It is not possible to compare these states with the global *Pausing, Stopping,* etc. because the MachineResource is at the machine level, with no subtasks. The diagram is shown in Figure 7.

2.3 NAMT revision of CIMF

The National Advanced Manufacturing Testbed (NAMT) Framework project[4] seeks to support the industry-driven development of standards for manufacturing systems integration. One of the means to this end is building testbeds and trial implementations for emerging industry-developed specifications such as the CIMF. During the construction of the testbed for the CIMF, we suffered from controversy and misunderstanding about its concurrent states model. We believe that the concurrency as implemented in the existing testbed is not needed, and that another form of concurrency – a fault condition indicator – is needed but lacking.

The combinations of Pausing-Stopping, Pausing-Aborting, Stopping-Aborting, and Pausing-Stopping-Aborting are not meaningful in the NAMT context. Aborts take precedence over stops, and stops take precedence over pauses. The notion that a job can pause in the middle of a stop or abort – to "freeze" immediately, then resume and continue on its way to the stop point or abort point – is not valid. If a machine is in the process of cutting metal, any kind of "freeze" is equivalent to an emergency stop, with ramifications even more serious than those of an abort.

Instead of this invalid concurrency, what is needed is a way to represent fault states. For example, if a workcell has been told to stop, but before it can reach a stop-point a machine gets stuck and cannot proceed, the state of the task is modeled by Stopping-Fault. The Executing, Pausing, Paused, Stopping, and Aborting states should all be mutually exclusive, but they may each be concurrent with the Fault state. That way, it is clear when human intervention is needed, and a task that is in a fault state can be recovered and continued after an operator has intervened.

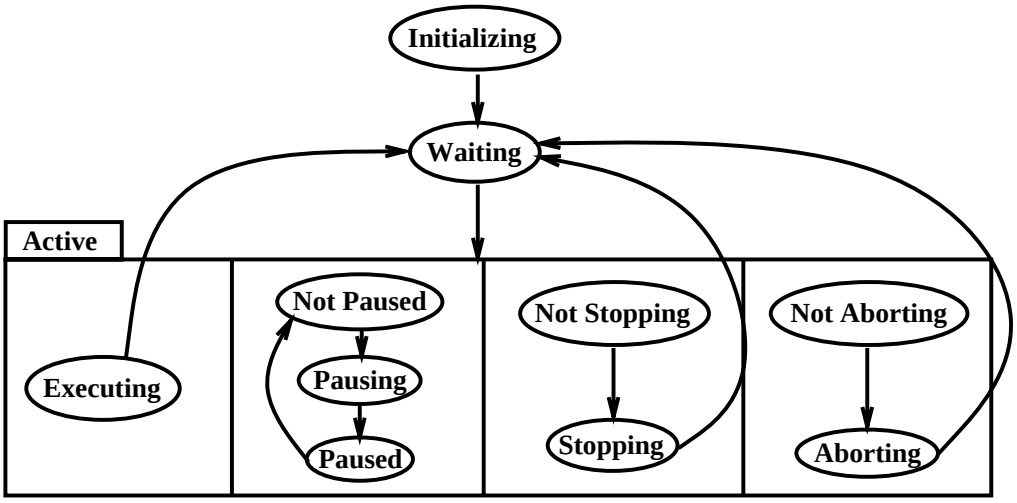

Figure 7: CIMF MachineResource state model

Jobs

Figure 8 shows the original NAMT revision of the CIMF ManagedJob state model. It is designed so that the Fault indicator can only be set when the job is Running because, in theory, there can be no failures when nothing is happening.

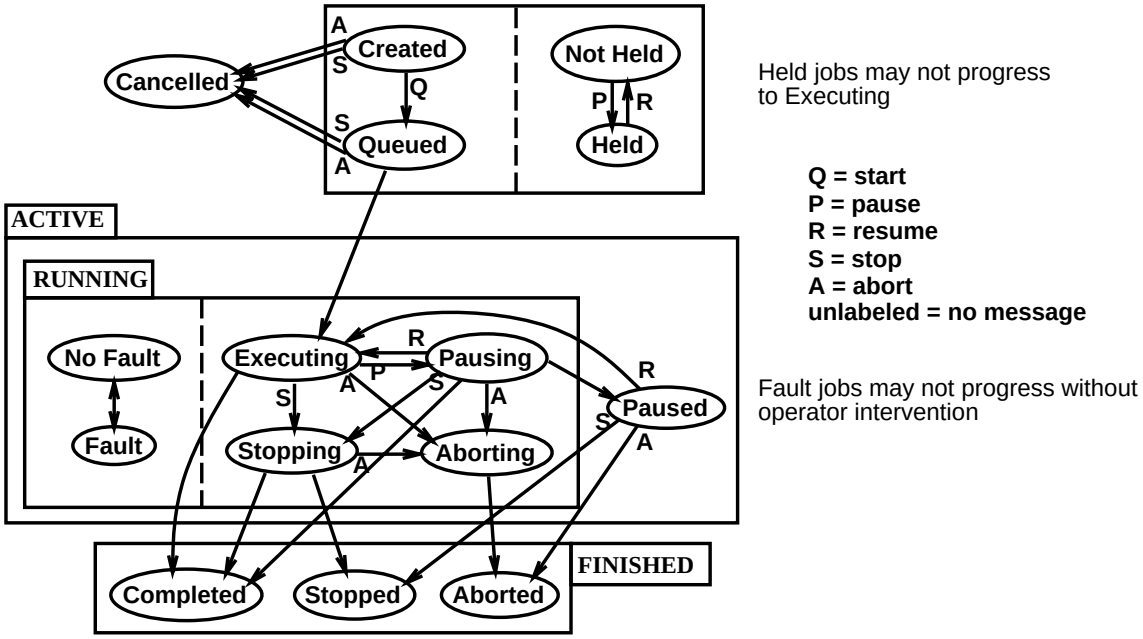

Figure 8: NAMT revision of CIMF ManagedJob state model

Simulation experiments conducted after the original revisions revealed that it may be desirable to permit Created-Fault and Queued-Fault as well. The scenario that raises the question is when the first task of the shop job is Queued at the workcell, and is unexpectedly Cancelled by the workcell operator. If the separation between levels of control is strict, the shop job will be in the Executing state at this point, and there is no problem in making it Executing-Fault. But it may be desirable

in practice for the shop job to remain in the Queued state until it is too late to Cancel the job. That way, if the job is killed at the shop level before any work is done in the workcell, the record will accurately show that it was Cancelled with no work done, rather than Aborted in an unknown state. Under this modified state model, the job that was Executing-Fault under a strict interpretation is now Queued-Fault. An analogous sequence of events leads to Created-Fault if the workcell task is unexpectedly Cancelled before it reaches the Queued state.

Supervisors

The Framework project is currently using a modified version of the CIMF MachineResource state model to serve as a JobManager state model. However, this was only an emergency fix for the lack of supervisor states, and does not represent an attempt at a valuable revision. The modifications included adding a Stopped state and changing its Abort concept.

2.4 R

The R model (an abbreviation for "Frank Riddick's model") is used by the Integration of Real-Time Scheduling and Shop Floor Data Collection project at NIST[5].

Jobs

R does not distinguish the task of processing a lot (load) from the lot object itself. Instead of having a job state model, it has load status. The status of a load can be Not_Started, Started, Ended, Interrupted, Restarted, Held, or Released. The mapping between these and the global state definitions is shown in Table 1, along with the descriptions from reference [5]:

Table 1: Load status in the R model

Load Status	R Definition	Global Mapping
Not_Started	Load has never been in started state	*Queued*
Started	Load is processing normally	*Executing*
Ended	All processing for the load has completed	*Completed*
Interrupted	A previously started load's processing has been stopped for unknown reasons	*Fault*
Restarted	A previously interrupted or held load is now processing normally	*Executing*
Held	A previously started load's processing has been manually stopped	*Held*
Released	A previously held load is ready to be restarted	*Released*

Supervisors

Being primarily a scheduling project, R does not include extensive definition of the control model. The state of resources is tracked, but these resources are at the machine level, with states like Available, Busy, and Broken but not Pausing, Stopping, etc.

2.5 Industry Practice

Currently installed industrial control systems do not subscribe to a consistent hierarchical architecture, but are closely linked to the capabilities and limitations of the workcells. Frequently, a monolithic supervisor with a proprietary architecture controls the entire shop.

Jobs

In practice, there is no shop-level Pause; if it is sent to the shop, it is merely propagated to the workcell level. The industry use of Pause means "stop on a dime, but safely," i.e., propagate this

down to the lowest level of control at which the equipment can be safely stopped. This Pause is momentary, and is usually sent to the shop dispatcher only as a means of getting it to the workcell supervisor.

Industry also uses a shop-level concept Hold, which means "stop processing at the next checkpoint and don't schedule the rest of the job until I tell you to." This is equivalent to *Hold*. On the other hand, Hold at the workcell level is usually related to some kind of manually initiated fault recovery and indicates a Pause that might turn into an Abort.

The concept Stop is used almost exclusively at the shop level and means "terminate the Job in a known state in which the lot can be pulled out of the workcell." This is equivalent to *Stop*. However, there is some concern as to whether a Stop can "propagate" to the extent of asking the workcell to stop at the end of processing one piece in a multi-piece lot. This does not correspond to *Stop* since it confuses the two levels of control.

Supervisors

Most machine controllers are custom built and provided by the vendors of the machines, with the only standard features being the serial interfaces. The typical industrial shop is built upwards from these; levels of control are created out of whole cloth to deal with the tasks that need to be done and the machines that need to be used. There is therefore little commonality among the state models of supervisors, even within the same shop.

Various PC-based "open" controllers are now being sold as retrofits for old machinery by third party vendors, but it is not clear whether any *de facto* standards will emerge for the control model, and no attempts were made at *de jure* standards. Work being done in the Manufacturing Execution System (MES)[6] arena tends to concentrate on the interface to the workcell level, following the assumption that the shop level will continue to be proprietary and MES-specific.

3 Comparison

3.1 States for Supervisors

MSI provides a clear state model for supervisors. All the others treat this as a detail and concentrate instead on jobs and machines.

The tendency to underspecify the supervisor derives from the industry practice of building custom controllers for each application. Since everything is custom built, there is no need to decide on a state model for supervisors that can apply at any level of control. R and the CIMF followed industry practice, while MSI advocated a more general approach.

Possibly this is an area where industry practice can benefit greatly from the research that has been done; however, it is also possible that a more complex supervisor is not wanted or needed.

3.2 Separation of Shop Level and Workcell Level

The separation of the shop level from the workcell level goes hand in hand with having a clear state model for supervisors. When every supervisor is custom built, there is less advantage in modularity, so there is less reason to separate the levels. The MSI model makes a clear separation necessary, to maintain a scalable and reusable architecture.

If a shop-level Pause or Stop is propagated into workcell-level Pauses or Stops, then the shop-level job is left in an inconsistent state, because it was not Paused or Stopped at a shop-level checkpoint (i.e., between workcell tasks). This makes it necessary to do more manual cleanup and recovery than might have been needed.

3.3 Fault Recovery

Being able to deal intelligently with unexpected failures is among the most important design considerations for a manufacturing system. When failures occur, a human operator must be notified of

the condition, and the supervisor must cooperate with the operator to make the recovery as efficient as possible, minimizing the amount of scrap and schedule slippage.

Common industry practice is to build a "watchdog" into the control system. The watchdog watches for errors and notifies an operator when they occur. At a minimum, this involves one or more big red warning lights on a control panel. The operator then uses whatever interface is available to perform the recovery.

MSI uses Waiting for Guardian and Waiting for Planner for fault conditions. These states effectively map to two separate warning lights. NAMT includes a Fault indicator that is concurrent with the active states. R provides the Interrupted state, meaning "stopped for unknown reasons," as distinct from Held, meaning "manually stopped." The CIMF does not explicitly model fault states.

The NAMT approach is advantageous in that all of the information about what the job or supervisor was doing when it fell into a fault state is retained. If the problem is something as simple as a machine requiring a nudge, the fault state can be cleared and the job can progress without any re-initialization. If the fault state is not concurrent with the active states then the operator must reset the state of the job every time. MSI achieves a similar effect by preserving the task-management-state while the task-state changes to Waiting for Guardian or Waiting for Planner. The use of two distinct fault task-states provides additional information on the nature of the fault condition.

It is especially important to distinguish faults from other kinds of stoppages in a hierarchical control system, so that higher levels of control do not abort unnecessarily.

3.4 Hold versus Stop

The *Paused, Held, Stopped, Aborted,* and *E-stopped* states all model different levels of severity of stoppage. In practice, it is unlikely that any given system will implement all five levels of severity. Instead, it will implement just those levels that are most useful to the application at hand.

Of particular interest is the *Held* versus *Stopped* distinction. While industry makes the most use of *Held*, CIMF and MSI have preferred to use only *Stopped*. Future standardization efforts should examine whether this incompatibility can be removed simply by adopting the industrial definition of *Held*, or whether both concepts are needed.

4 Conclusion

The preceding survey of state models yields the following suggestions to guide future work in this area.

1. Start out with a clear definition of all of the entities – supervisors, jobs, lots, machines – and decide on the responsibilities and jurisdiction of each. Clearly specify the component interactions between levels of control. Without these specifications, the semantics of the state model will be in doubt.

2. Build explicit mechanisms for fault recovery into each entity that can fail. These mechanisms should make it clear when there has been a failure, preserve as much information as possible about what led up to the failure, and be sufficiently flexible to allow the operator to avert unnecessary waste.

3. Do not mix subtly different concepts into one state. Use as many of *Paused, Held, Stopped, Aborted,* and *E-stopped* as are appropriate, and be sure that their definitions and usage can be distinguished from one another.

References

[1] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, pages 231–274, July 1987.

[2] Sarah Wallace, M. K. Senehi, Ed Barkmeyer, Steven Ray, and Evan K. Wallace. *Manufacturing Systems Integration: Control Entity Interface Specification*. National Institute of Standards and Technology, Interagency Report 5272, 1993. Available from the National Technical Information Service, Springfield, VA 22161 U.S.A.

[3] Lawrence Eng, Ken Freed, Jim Hollister, Carla Jobe, Paul McGuire, Alan Moser, Vinayak Parikh, Margaret Pratt, Fred Waskiewicz, and Frank Yeager. *Computer Integrated Manufacturing (CIM) Application Framework Specification 1.3*. SEMATECH, 2706 Montopolis Drive, Austin, TX 78741 U.S.A., 1996.

[4] Howard M. Bloom and Neil Christopher. A framework for distributed and virtual discrete part manufacturing. In *Proceedings of the CALS EXPO '96*, Long Beach, CA, October 1996.

[5] Albert Jones, Frank Riddick, and Luis Rabelo. Development of a predictive-reactive scheduler using genetic algorithms and simulation-based scheduling software. In *Advanced Manufacturing Processes, Systems, and Technologies Conference Proceedings (AMPST96)*, pages 589–598, 1996.

[6] John A. Leibert. MES and the shift toward a work flow environment. *Industrial Engineering Solutions*, 29(1):30–33, January 1997.