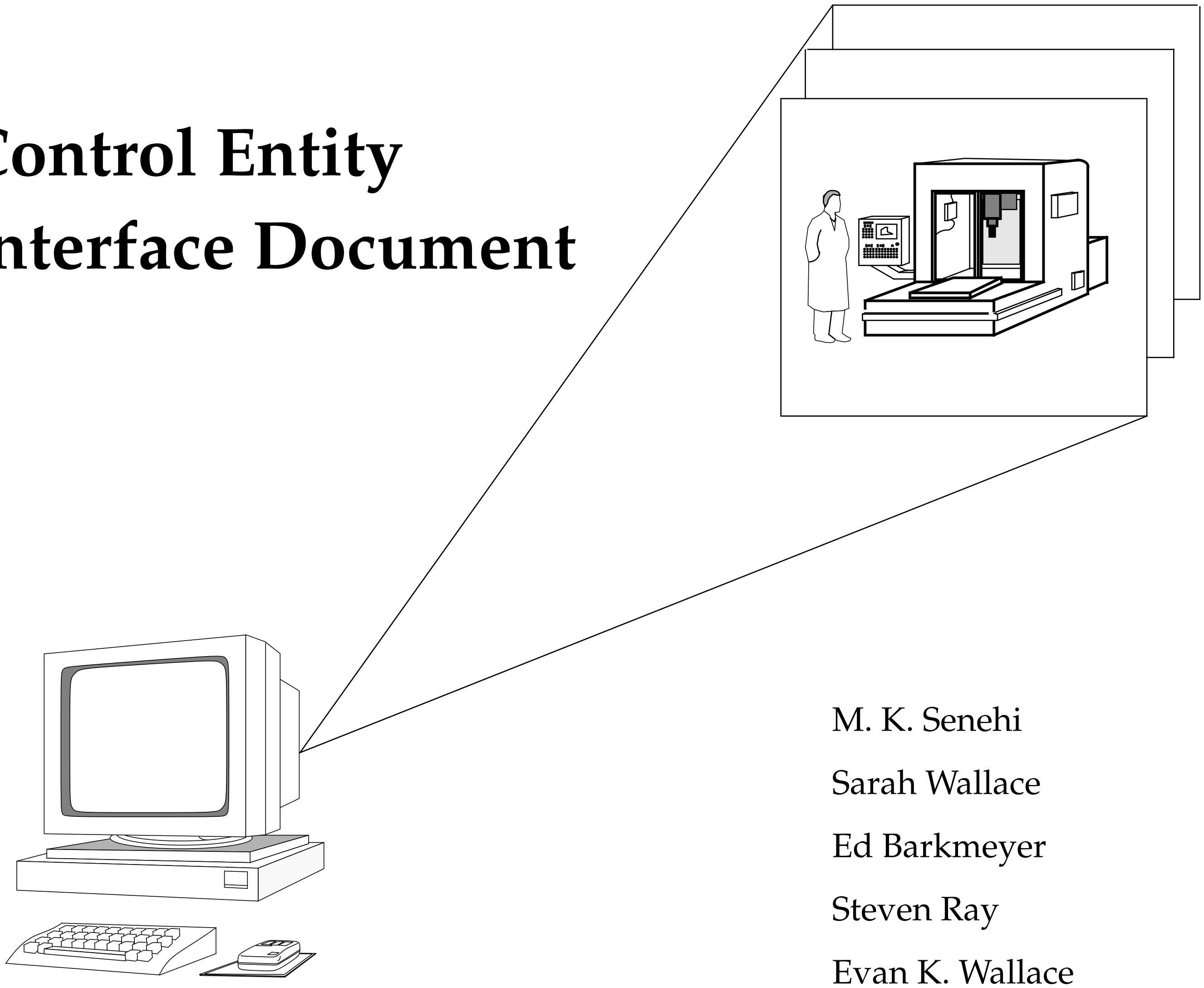
NISTIR 4626
June, 1991

Manufacturing Systems Integration

Control Entity Interface Document



M. K. Senehi
Sarah Wallace
Ed Barkmeyer
Steven Ray
Evan K. Wallace

United States Department of Commerce
National Institute of Standards and Technology
Manufacturing Engineering Laboratory
Gaithersburg, MD 20899

NISTIR 4626
June, 1991

Manufacturing Systems Integration

Control Entity Interface Document

M. K. Senehi
Sarah Wallace
Ed Barkmeyer
Steven Ray
Evan K. Wallace

United States Department of Commerce
Robert A. Mosbacher,
Secretary of Commerce

National Institute of Standards and Technology
John W. Lyons, Director

Table Of Contents

List Of Figures

List Of Tables

1. Introduction

A major activity of the National Institute of Standards and Technology (NIST) Manufacturing Systems Integration (MSI) project[1] is the development of a system architecture that incorporates an integrated production planning and control environment. This document is concerned with defining the details of interfaces for controllers which are incorporated into an integrated system that conforms to the NIST MSI architectural model. A description of the NIST MSI architectural model can be found in "Manufacturing Systems Integration Initial Architecture Document" [Senehi, forthcoming]. The purpose of this document is two fold: to document the progress and current status of the MSI Architecture's Control Entity Interfaces and implementation, and to provide designers and implementors with specifications for an MSI Architecture compliant controller. This document was developed by the MSI Architecture Committee[2] and incorporates the consensus of that committee as of July 1990. This work is partially supported by funding from the Navy Manufacturing Technology Program and NIST's Standards Technology and Research Services (STRS).

The first section of this document gives information on pertinent aspects of the MSI architecture and on the assumptions which have been made concerning the environment in which the system is operating. The balance of the document contains a detailed description of the state tables for controllers; commands and exchange conventions for the shutting down, starting up, emergency shutdown, etc. of controllers; commands and protocols for the initiation, execution and termination of tasks by controllers and details of required status reports. In the summer of 1990, the Factory Automation Systems Division at NIST developed a prototype MSI Architecture compliant controller using the control entity interfaces defined in this document and the specific message formats found in Appendices A–B on pages 23–33. Appendix C on page 34 presents an example scenerio of commands and responses between control entities. The final section of this document discusses areas to be addressed as a result of this implementation. Appendix D on page 38 contains a glossary of relevant terms.

1.1 Project Direction and Goals

The goal of the MSI project is to attack the problem of incompatible data and control processes within a manufacturing enterprise. Despite years of development of excellent products, this problem still prevents American industry from easily building truly integrated manufacturing systems. In particular, the sharing of information between engineering, production management and control systems is still difficult, and there are no standards specifying the interactions among control, planning and scheduling systems. To address these problems of integration, the MSI team is developing a testbed environment which allows experimentation with integrated production management and control systems. A critical feature of this testbed is the specification of an architecture and interfaces which allow the incorporation of commercial products and academic systems supporting production engineering and control. The validation of this architecture will be via a demonstration of the production of selected parts, using either actual or emulated shop floor equipment or any combination of both. Ultimately, the architecture and interface specifications will be submitted as candidate standards to the relevant national and international standards orga-

[1]This document was prepared by United States Government employees as part of their official duties and is, therefore, a work of the U.S. Government and not subject to copyright.
[2]In 1990, the Architecture Committee members were Ed Barkmeyer, Mark Luce, Steven Ray, M.K. Senehi, Evan Wallace, and Sarah Wallace.

nizations.

2. Background

2.1 Common Memory Environment

This interface specification assumes a pure common memory [Libes, 1990] interface between supervisor and subordinate. This means that the common memory command and status mailboxes are persistent across process restarts. Processes have no direct means of determining the death or restart of the supervisor or subordinate: the status of the supervisor or subordinate must be inferred from the messages in the common memory mailboxes.

Common memory is an imperfect communication mechanism, meaning that not necessarily will every mailgram inserted in a mailbox be seen by all intended recipients [Rybczynski, 1988]. What is guaranteed is that a process reading the mailbox will see the most recently deposited mailgram. Therefore, incremental status reporting will not work reliably when using common memory for communication. The status report of an arbitrary manufacturing control process (controller) must be complete, i.e. it must contain all the information pertinent to the current state of the controlled subsystem, so that the supervisory process can miss any number of status reports and still get the complete picture when it finally reads one. Similarly, the supervisor must be assured that the subordinate has seen the command last written in the command mailbox before it issues a new one.

As a general principle, any mailgram must contain:

- the name of the subsystem which wrote it,
- a timestamp, reflecting the time at which the mailgram was written,
- a distinct serial number (possibly provided by the Common Memory service).

Timestamps are included as tools for aiding the human operators in monitoring and debugging the system. The synchronization of clocks across multiple systems to finer accuracy than that required by shop level scheduling is not addressed in the MSI architecture. Fine clock synchronization is technically difficult, and without it, a comparison of timestamps from two distinct systems may be insufficient to determine the sequence of communication events. Therefore, the mailgram timestamps are not used as a message identifier in the control entity interface protocols.

2.2 Architecture

In MSI terminology, a controller is a software entity which instructs hardware or software to perform tasks. In order to accommodate as many types of controllers as possible, the MSI project has identified a limited set of compatibility requirements for interfacing controllers. Integrating a controller into the MSI architecture requires two types of interfaces: Supervisory (command and status) and Task (service request and response). A third type of interface, the Guardian (console) interface is optional and highly recommended. The possible interfaces of a control entity are shown in Figure 1. on page 3. The following sections describe the requirements which the MSI architecture places upon controllers in broad terms.

2.3 Administrative Hierarchy

In the MSI architecture, a controller is required to exist within an administrative hierarchy. The

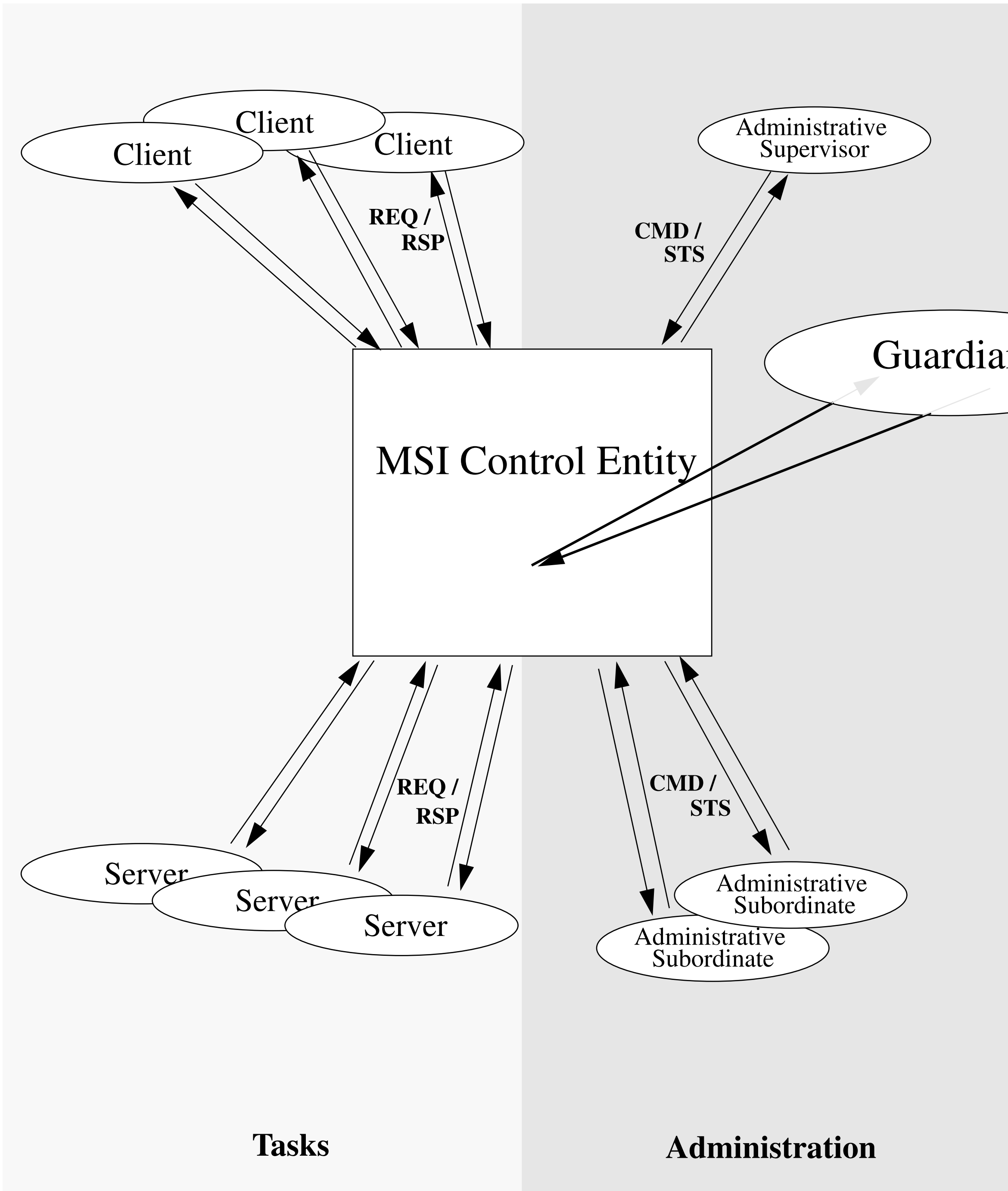


Figure 1. External Interfaces of an MSI Control Entity.

primary function of the administrative hierarchy is to provide reliable channels for directing the start up and shut down of controllers. A controller may have at most one administrative supervisor; it may have any number of administrative subordinates, or none at all. A subsystem consists of a subsystem controller and all of its administrative subordinates, if any.

Currently the Administrative hierarchy of the MSI architecture consists of three logical levels: Equipment, Workcell and Shop.

- The Equipment level is the lowest level to which the MSI architecture applies. Within this level, there is a software complex which drives the physical equipment systems. An equipment controller is responsible for the execution of tasks by an individual device or unit. It may have internal subordinate software elements which perform the primitive control tasks, but the characteristics of these internal interfaces are not specified by the MSI architecture.
- A Workcell is a subsystem consisting of a collection of equipment viewed as a functional unit. It is coordinated by a single supervisory controller designated the Workcell controller. The grouping of equipment into workcells is based upon many factors and is highly facility dependent. A workcell controller may control equipment controllers directly, or may control any number of layers of workcell level controllers.
- The top level controller is the Shop Controller. The subsystem it controls consists of a set of workcells in the manufacturing shop which are currently configured to be available. At present, the MSI project is limited to a single shop, and there is only one controller at this level.

Figure 3. on page 6 shows an example of a control hierarchy. The administrative hierarchy is defined by the dashed arrows. It should be noted that the administrative hierarchy remains unchanged during the manufacturing of tasks.

2.4 Controller Tasking

The primary function of a controller is to perform manufacturing tasks. In the MSI architecture, controller tasking is based upon a client/server model (see Figure 2. on page 5). A client requests tasks to be performed; the server may accept or reject tasks based on its administrative state and its current task capabilities. As a result of this tasking model, task control is not required to follow the administrative control paths. Figure 3. on page 6 shows an example of a control hierarchy. The task hierarchy is defined by the solid arrows. The underlying administrative hierarchy is defined by the dashed arrows. It should be noted that although the administrative hierarchy remains unchanged during the manufacturing of tasks, the task hierarchy will be continually redefined depending on which tasks are outstanding. The task interface provides for requesting new tasks and for coordinating the activities of each controller with those of the Shop as a whole. Related details and assumptions are explained in the following sections.

2.4.1 Production Plans

The tasks which controllers are directed to perform are specified in production plans. A production plan is a set of time sequenced instructions for making a specific item with specific equipment at specific times. MSI production plans are generated by the Production Planner from MSI process plans by choosing appropriate branches of the process plan, selecting proper equipment

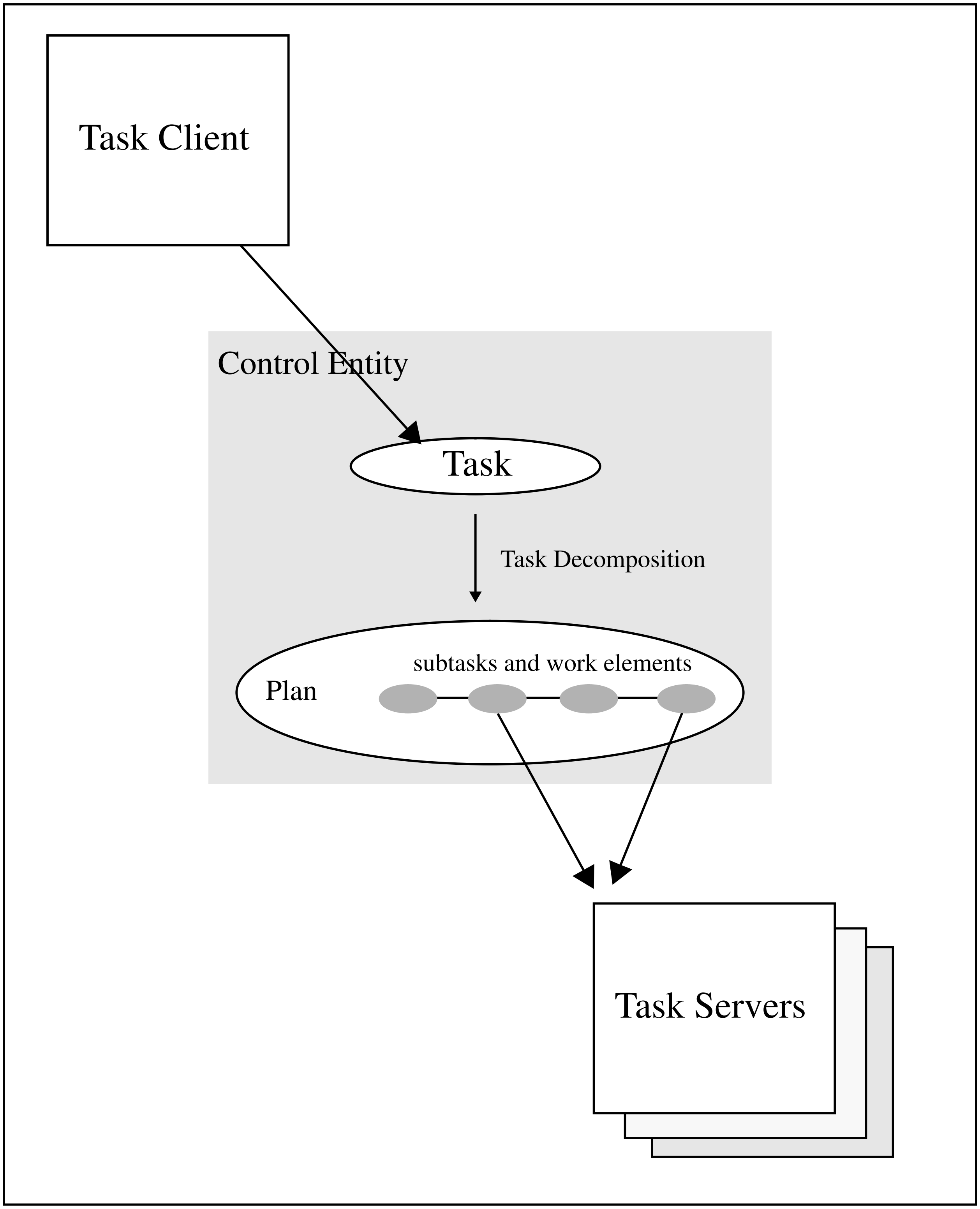


Figure 2. Task Decomposition and the Task Client/Server Model.

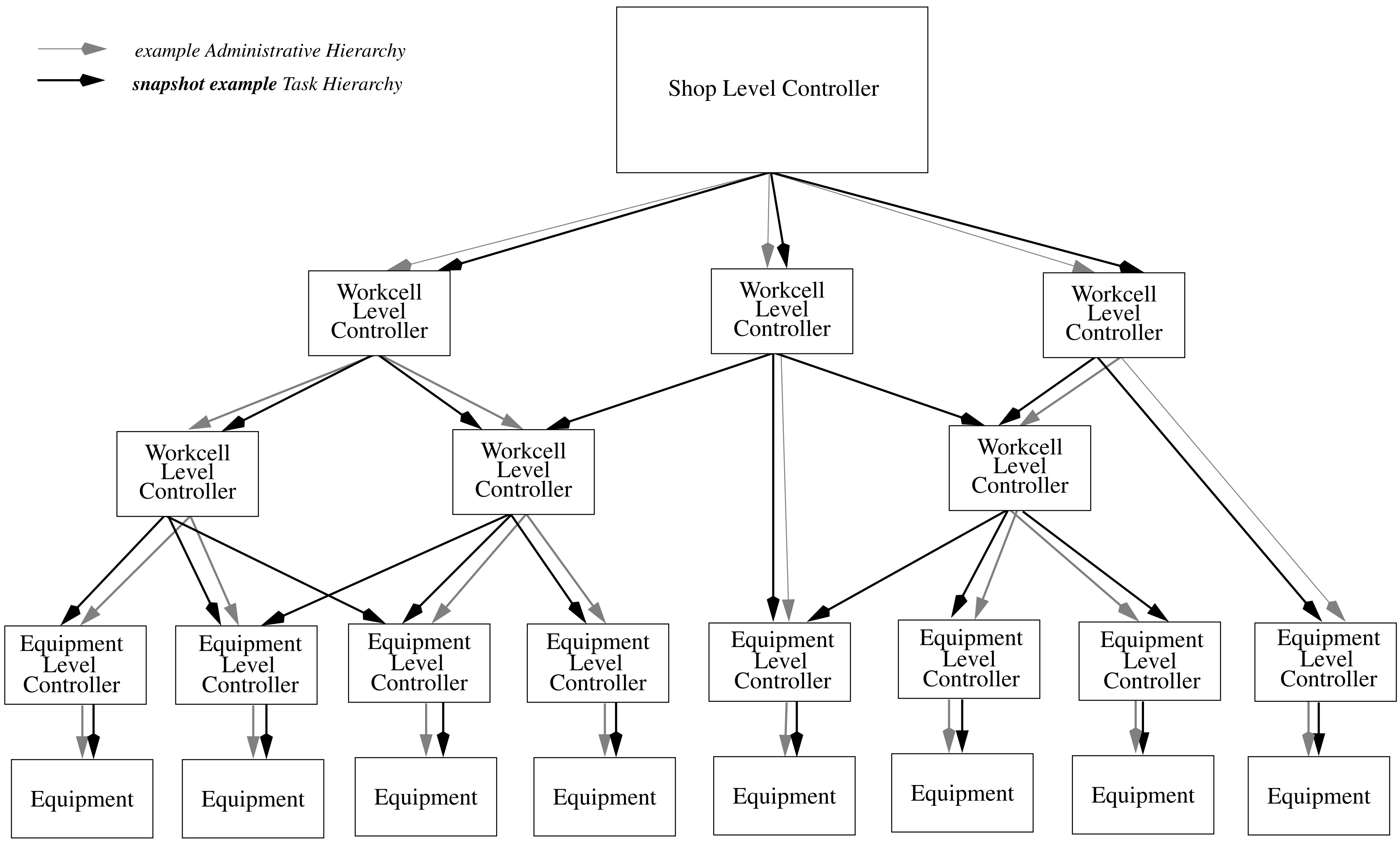


Figure 3. A Sample Control Hierarchy (Administrative and Task).

and scheduling production. Production plans are hierarchical and may exist for Shop, Workcell and Equipment levels. While the MSI architecture specifies a representation for production plans, it permits workcell and equipment level plans to be in varying formats based on the needs of the individual workcell and equipment controllers. In fact, a workcell is permitted to generate its production plans on-line.

The MSI architecture requires that the workcell controller understand the notion of checkpoint to facilitate error handling. A checkpoint is a step in the production plan where the manufacturing process may be halted in a safe manner without damaging either the equipment or the workpiece and later replaced or resumed. Obviously, the start and end of a production plan will be checkpoints. A less trivial example of a checkpoint would be a machining step which drills a hole. After the hole is completed and the spindle is withdrawn, it would be safe to stop the plan at this step and resume at a later time.

2.4.2 Task Execution

The execution of Shop level production plans is directed and monitored by independent software agents, called Shop Level Executives (SLE). A Shop Level Executive requests tasks of workcell controllers in accordance with its production plan. The SLE's are responsible for ensuring that the production plan is performed. It is not the responsibility of an SLE to monitor the health of the controllers executing its production plan.

Lower level production plans (i.e. workcell and equipment) are executed by lower level controllers. The internal structure of lower level controllers is not mandated by the architecture. In order to be included in the MSI architecture, a controller is characterized as being in one of three categories. The categories are based upon the controller's ability to queue and schedule tasks:

(1) The subsystem accepts only one task at a time, and initiates the task as soon after receipt as possible. It reports status only on the last task received.

(2) The subsystem accepts and queues tasks as received, and treats a task as eligible for initiation on receipt, but actually initiates tasks according to its own schedule. The subsystem does not understand MSI production plans and schedules per se. For each client, it reports status on all of that client's task requests whenever an event relevant to that client occurs.

(3) The subsystem accepts and queues tasks as received, and understands tasks as described by MSI production plans and schedules. For each client, it reports status on all of that client's task requests whenever an event relevant to that client occurs, including violation of the schedule.

2.5 Error Recovery

Although no attempt will be made here to describe error handling extensively, the MSI architecture has considered the impact of errors upon the operation of the system. Two main types of errors have been addressed: task failure and controller impairment.

Task failure occurs when a task fails to complete. If the failure cannot be handled locally, it is reported in the task status report to the task client. Ultimately, it will be brought to the attention of the Shop Level Controller. When appropriate, the Shop Level Controller alerts the Production Planner to the need for replanning.The Production Planner is responsible for adjusting the schedule. If contingency plans are required, new production plans may be generated from existing alternative process plans or from new process plans generated on-line by the process planner.

When a controller becomes impaired (i.e., its capabilities are diminished) it may be necessary to intervene in ways which are not provided by the administrative supervisory path. Either the internals of the equipment or controller need to be accessed, or a subordinate of a controller needs to be removed and replaced. The MSI architecture provides for a special interface called the Guardian which accommodates these needs. From this interface it is possible to remove a (dysfunctional) subordinate from the system or to add a new subordinate. Upon the option of the implementor, a Guardian may provide system specific access to the internals of the controller. This interface replaces the front panel or console interface, with the advantage that the operator may be human, but is not required to be, and may be an automated expert system, or a human with computer assistance.

3. Supervisory Interface

The supervisory interface is administrative: it deals exclusively with subsystem states, transition control and task service control, using an enhanced form of the UVA model described below (see Figure 4. on page 9) [O'Halloran, 1986].

3.1 System States

Table 1. on page 10 enumerates and defines the possible subsystem states for the purpose of the administrative supervisory interface The initial state of a subsystem, when the highest level controller has been started, and the command mailbox has not yet been examined, is Down.The transitions permitted by the administrative control protocol are indicated in the corresponding state diagram, Figure 4. on page 9. It should be noted that the administrative state diagram does not fully describe all possible causes for state transitions. In particular, external events may cause changes in the administrative state.

In addition, a subsystem will have a *capability index*, which indicates whether the subsystem has all capabilities to be expected from the baseline (initial) configuration or some different set of capabilities. The administrative supervisor does not understand the capabilities of the subsystem: it interprets a change in the capability index to mean that the subsystem capabilities have changed in such a way as to require replanning for its use. The capability index has no other intrinsic semantics. Because the supervisor may miss individual changes in the subsystem capability index, it is necessary for each value of the capability index to be distinguishable from all previous values which have occurred since the initial state of the controller. For convenience, an integer value will be used for this index. The capability index is initialized to zero when the subsystem is Synchronized (goes from Down to Idle) and is increased by one each time any of the following occurs:

(1) a configured subordinate becomes unavailable (is deconfigured by a Detach or Ignore command from the Guardian, or by a controller decision after a failure),

(2) a new subordinate becomes available (is Attached by command, and has successfully started, reaching a Ready or Active state),

(3) the capability index of a subordinate changes and the subsystem controller determines that the capabilities of the whole subsystem are thereby affected.

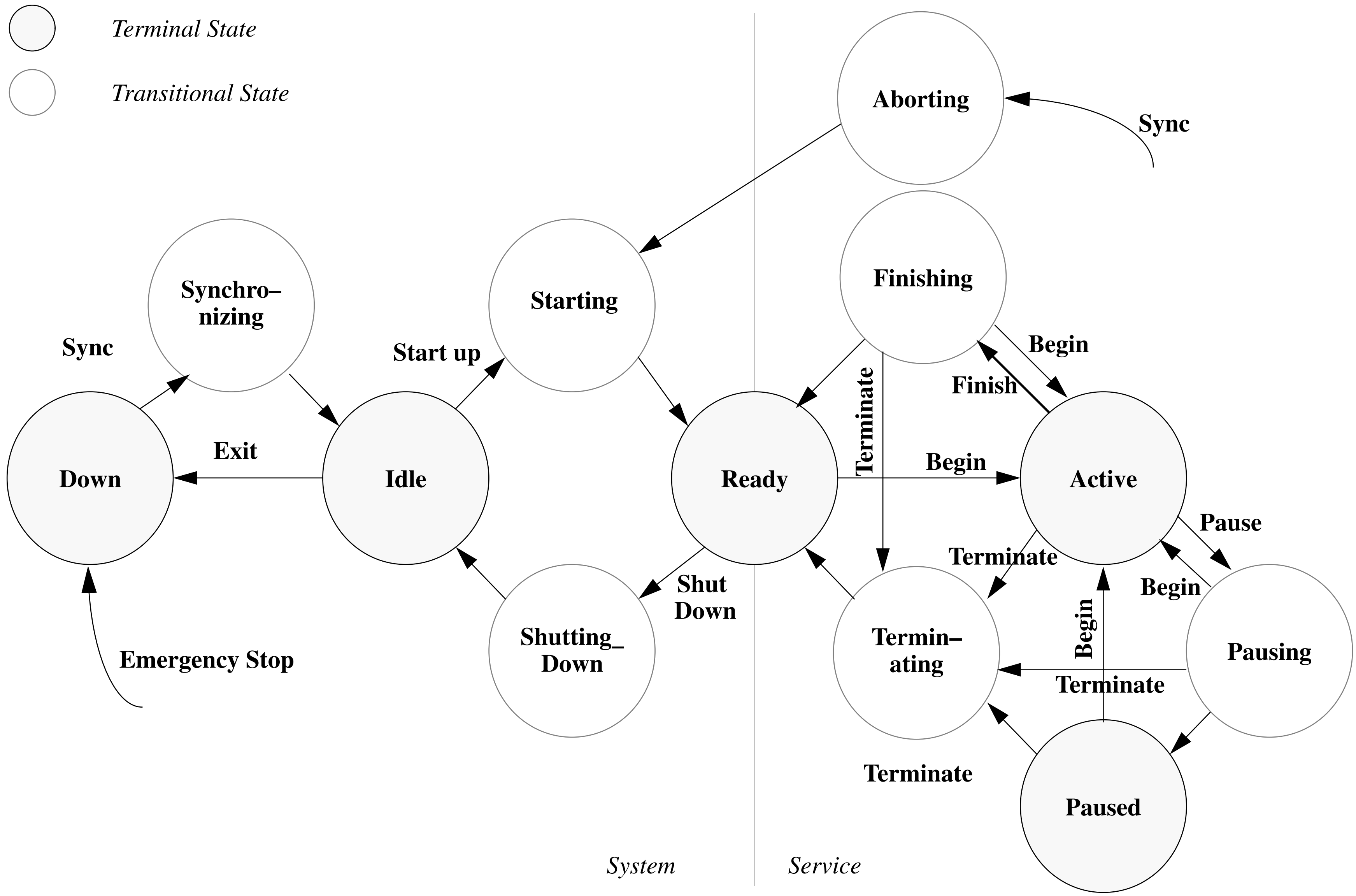


Figure 4. Administrative State Diagram for an MSI Control Entity.

.

State	Definition
Down	The subsystem controller has exited or has just restarted cold.
Synchronizing	The subsystem controller has received a Sync command, but has not yet synchronized with its subordinates. This is a transition state.
Idle	The subsystem is synchronized as configured, but inactive.
Starting	The subsystem was Idle, has received a Start_up command, but has not yet started all of its configured subordinates. This is a transition state.
Ready	The subsystem is up: the controller and all of its configured subordinates have successfully executed the Startup, but it is not accepting or processing any task requests, and it has no tasks outstanding.
Shutting_down	The subsystem has received a Sync or Shutdown command, but has not yet shutdown all of its subordinates. This is a transition state.
Active	The subsystem is up, accepting and processing task requests.
Pausing	The subsystem is up and accepting tasks, but it is not initiating new tasks, only bringing running tasks to the next checkpoint. Note that for tasks which have not started, the initial node is the checkpoint, hence they will not be permitted to start. This is a transition state.
Paused	The subsystem is up. All tasks are stalled at a checkpoint and cannot resume until the subsystem is again Active.
Finishing	The subsystem is up, but it is not accepting any new tasks. It is completing all tasks previously issued. This is a transition state.
Terminating	The subsystem is up, but it is not accepting any new tasks, and any outstanding tasks will be terminated at the next checkpoint. This is a transition state.
Aborting	The subsystem is going to the Idle state. It is not accepting any new tasks, and any outstanding tasks are being aborted. This is a transition state.

Table 1. Administrative State Definitions.

There will be other circumstances, unrelated to administrative control per se, which will also require the subsystem controller to change the capability index, e.g. cutter breakage. In any case, a change in the capability index is always preceded by updates to the facility database which reflect the actual capability changes.

3.2 Supervisory Interface Commands

The administrative command set provides for two step start up and shutdown of subsystems (to prevent accidental misalignment due to persistent information in the common memory). Table 2. on page 11 describes the administrative commands. Table 3. on page 12 defines, for each administrative state, the valid commands that a controller can receive, the action to be performed when that command is received, the next state for that command, and the final state the controller should be in at the completion of that command. It should be noted that '—' designates no change of state, and a final state is only specified when there is an intermediate transition state necessary to perform the required action associated with a command.

Command	Definition
Report	Acknowledge this command and report status. This command is a No–Op used by the supervisor to set the last command number, and in non common memory environments to force a status report.
Sync	Synchronize all configured subordinates and attempt to return the subsystem to an Idle state. This command is used by the supervisor to initialize a Down subsystem, and to re-initialize a running system. A controller which has finished cold start does not attempt to synchronize wit its configured subordinates until it receives a Sync.
Start_up	Bring the subsystem up to a Ready state, startup all configured subordinates.
Shutdown	Bring the subsystem down to an Idle state, shutdown all configured subordinates, bring equipment to a safe state.
Exit	Exit the control process in an orderly fashion.
Emergency_Stop	Save the equipment and get out fast. Emergency shutdown all subordinates, move equipment to save and power down if possible, (optionally) go to Down, and exit.
Begin	Begin/resume accepting task requests.
Finish	Complete all existing tasks, but stop accepting new task requests.
Pause	Continue accepting Task requests, but don't initiate new tasks, and pause all outstanding tasks at the next checkpoint. Then pause all configured subordinates. The object is to get activity in the subsystem to cease temporarily
Terminate	Stop accepting new tasks, stop each outstanding task at the next (or current) checkpoint, and treat it as terminated.

Table 2. Administrative Command Definitions.

An MSI Architecture compliant controller must support all administrative commands except Pause. Appendix A on page 23 contains the description of the command message formats for the administrative interface used in the 1990 MSI Architecture prototype controller.

Down

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Sync	Synchronizing	Issue Sync to subordinates, and go to Idle when all subordinates are Idle.	Idle
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Synchronizing

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Sync	—	Acknowledge Only.	
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Idle

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Sync	—	Acknowledge Only.	
Start_up	Starting	Issue Start_up to subordinates, and go to Ready when all subordinates are Ready.	Ready
Shutdown	—	Acknowledge Only.	
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	
Exit	Down	Go to Down optionally, and exit.	

Starting

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Start_up	—	Acknowledge Only.	
Sync	Shutting_Down	Issue Sync to all subordinates, and go to Idle when all subordinates are Idle.	Idle
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Table 3. Administrative State Tables.

Ready

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Start_up	—	Acknowledge Only.	
Begin	Active	Do not remove any tasks in the task queue.	
Finish	—	Acknowledge Only.	
Terminate	—	Acknowledge Only.	
Shutdown	Shutting_Down	Shutdown all subordinates, and go to Idle when all subordinates are Idle.	Idle
Sync	Shutting_Down	Shutdown all subordinates, and go to Idle when all subordinates are Idle.	Idle
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Active

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Begin	—	Acknowledge Only.	
Terminate	Terminating	Stop accepting new tasks, stop each outstanding task at the next checkpoint. Go to Ready when there are no more outstanding tasks.	Ready
Finish	Finishing	Complete all outstanding tasks if possible and then go to Ready when there are no more outstanding tasks.	Ready
Pause	Pausing	Continue accepting task requests, but don't initiate new tasks. Pause all outstanding tasks at the next checkpoint. Go to Paused. Then pause all configured subordinates.	Paused
Sync	Aborting	Abort all tasks, then go to Shutting_Down, shutdown all subordinates, go to Idle when all subordinates are Idle.	Idle
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Finishing

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Begin	Active	Do not remove any tasks in t he task queue.	
Terminate	Terminating	Stop accepting new tasks, stop each outstanding task at the next checkpoint. Go to Ready when there are no more outstanding tasks.	Ready
Finish	Finishing	Acknowledge Only.	Ready
Sync	Aborting	Abort all tasks, then go to Shutting_Down, shutdown all subordinates, go to Idle when all subordinates are Idle.	Idle
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Table 3. continued.

Terminating

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Terminate	—	Acknowledge Only.	
Sync	Aborting	Abort all tasks, then go to Shutting_Down, shutdown all subordinates, go to Idle when all subordinates are Idle.	Idle
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Pausing

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Begin	Active	Do not remove any tasks in the task queue.	
Terminate	Terminating	Stop accepting new tasks, stop each outstanding task at the next checkpoint. Go to Ready when there are no more outstanding tasks.	Ready
Pause	—	Acknowledge Only.	
Sync	Aborting	Abort all tasks, then go to Shutting_Down, shutdown all subordinates, go to Idle when all subordinates are Idle.	Idle
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Paused

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Begin	Active	Do not remove any tasks in the task queue.	
Terminate	Terminating	Stop accepting new tasks, stop each outstanding task at the next checkpoint. Go to Ready when there are no more outstanding tasks.	Ready
Pause	—	Acknowledge Only.	
Sync	Aborting	Abort all tasks, then go to Shutting_Down, shutdown all subordinates, go to Idle when all subordinates are Idle.	Idle
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Table 3. continued.

Aborting

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Sync	—	Acknowledge Only.	
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Shutting_Down

Command	Next State	Comments	Final State
Report	—	No change, always acknowledge.	
Sync	—	Acknowledge Only.	
Shutdown	—	Acknowledge Only.	
Emergency_Stop	Down	Emergency shutdown all subordinates, move equipment to save and power down if possible. Optionally go to Down, and exit.	

Table 3. continued.

3.3 Supervisory Interface Exchange Conventions

The supervisory interface is operated one command at a time, that is, the supervisor does not issue the next command until the subordinate's system status report shows that it has received the current command (by having the correct *last command identifier*). The subordinate responds to each command on receipt (i.e., accepts or rejects the command after examining the state tables). The transition states are provided to enable the subordinate to report receipt of and reaction to a command without having reached the desired final state (completion of the command). Once an administrative command is accepted, it is not logically possible for it to fail subsequently, although it may never complete.

Example: A subsystem in an Idle state receives and accepts a Start_up command and enters the Starting state. It remains in the Starting state until all of its configured subordinates reach Ready. If one of the subordinates fails to reach Ready, the subsystem stays in the Starting state indefinitely. If the subsystem controller is told or determines that the still un–Ready subsystem should be treated as Unavailable, it deconfigures the offending subordinate and thereby becomes Ready, it updates the database to reflect the loss of the subordinate, and changes the capability index. Neither the endless Starting state, nor the transition to Ready with a change in the capability index, is a failure of the Start_up command from the point of view of the subordinate. The supervisor may make some qualitative judgement of these situations, but there is no formal failure of the command.

The supervisor does not interpret the state of the subsystem as it appears in the status report until the last command identifier reflects the last command issued by the supervisor. This avoids misinterpretation of intermediate states which result from asynchronous generation of commands and status reports.

There are three exceptions to the rule that the supervisor never sends a new command until the previous command has been acknowledged:

(1) The initial command from the supervisor may be, and usually is, sent without any concern for prior commands; the initial command should be Report or Sync.

(2) A Sync command, the function of which is to force the subordinate into an Idle state regardless of its current state, is not in any way dependent on the supervisor correctly understanding the previous state of the subordinate, and may be issued at any time, whether or not the subordinate has acknowledged the previous command.

(3) An Emergency_stop command should never be delayed for any reason.

4. Guardian Interface

The Guardian interface provides the console interface to a control entity, allowing external monitoring and intervention in the otherwise automatic operation of the control entity.

The Guardian interface includes all capabilities of the supervisory interface, including all administrative commands, and the Guardian exchange conventions are identical to that of the supervisory interface. The Guardian has priority over the supervisory interface, to allow external override of the automatic transition control. This capability should never by exercised while the supervisor is still functioning.

The Guardian interface also includes two additional command sets: the configuration control command set, given below, and a private command set to change parameters and modes of operation for a particular kind of subsystem. The status portion of the Guardian interface is described in section 6.3 on page 21. Appendix B on page 28 contains the description of the command and status message formats for the Guardian interface used in the 1990 MSI Architecture prototype controller.

Configuration commands:

The configuration command set permits the Guardian to direct the subsystem to add and delete subordinate control entities, in order to facilitate recovery of failing components and continuation without them. When a configuration change takes place in response to a Guardian command, the capability index is incremented and this is reported through the supervisory interface.

Ignore (subordinate, location) — deconfigure an existing subordinate and disconnect from it immediately. This command allows the Guardian to direct a subsystem to recover from a hung state caused by a defunct subordinate.

- From Down, reject: protocol violation.
- From any other state, if the named subordinate is not configured, reject: no such subordinate.
- From Synchronizing, deconfigure the named subordinate, update the capability index, disconnect the command path and remain Synchronizing until all re-

maining configured subordinates are Idle.

- From Idle or Ready, deconfigure the named subordinate, update the capability index, disconnect the command path and do not change state.
- From Starting, deconfigure the named subordinate, disconnect the command path, update the capability index and remain Starting until all remaining configured subordinates are Ready.
- From Shutting_Down, deconfigure the named subordinate, update the capability index, disconnect the command path and remain Shutting_Down until all remaining configured subordinates are Idle.
- From any other state, abort any tasks which involve the named subordinate, deconfigure it, update the capability index and disconnect the command path, and do not change state.

Attach (subordinate, location) — configure a new subordinate.

- From Down, Synchronizing or Shutting_Down, reject: protocol violation.
- From any other state, if the named subordinate is already configured, acknowledge only.
- From Idle, if the named subordinate is new, configure the named subordinate, and synchronize with it, and remain Idle until the newly configured subordinate is Idle.
- From Starting, if the named subordinate is new, configure the named subordinate, synchronize with it, and attempt to Start it when it is Idle, remain Starting until all configured subordinates, including the new one, are Ready.
- From any other state, if the named subordinate is new, configure the named subordinate, synchronize with it, and attempt to Start it when Idle, and increment the capability index when all configured subordinates, including the new one, are Ready.

Detach (subordinate, location) — deconfigure an existing subordinate and disconnect from it in an orderly way.

- From Idle or Ready, if the named subordinate is not configured, reject: no such subordinate.
- From Idle, deconfigure the named subordinate, disconnect the command path, update the capability index, and remain Idle.
- From Ready, deconfigure the named subordinate, disconnect the command path, update the capability index, and remain Ready.
- From any other state, reject: protocol violation.

5. Task Interface

The task interface deals with the delivery and execution of actual subsystem tasks, or what was called manufacturing control in the AMRF [Albus, 1981]. The associated aspect of the status report is the Task status. An MSI controller considers all task requests to be instances of the single Execute command below. However, it abides by the protocol type selected for the particular subordinate, and interprets the plan accordingly. The execution of tasks may be serial or parallel, depending on the ability of the controller to schedule and execute tasks. See section 2.4.2 on page 7 for details of the protocols. Appendix A on page 23 contains the description of the command and status message formats for the task interface used in the 1990 MSI Architecture prototype controller.

5.1 Task Commands

Report

Acknowledge this command and report task status for the requesting client. This command is a No–Op used by the client to force a task status report.

Execute (task–id, task–activity, node–name, parameter–list)

Used to initiate any new task. The parameters are:

- task–id = an identifier for the task invented by the supervisor, guaranteed unique within the context of this session of client/server communication, and used to refer to the task in status reports and subsequent commands.
- task–activity = either a work element or an ordered pair consisting of plan–id and plan–version.
  A work element is a type of task which is meaningful to the subordinate, and possibly to the supervisor. The ordered pair {plan–id, plan–version} consists of the database keys for the detailed task description, nominally an ALPS (A Language for Process Specification) production plan, but could be any form used by the subsystem, including a process plan for a system which does dynamic production planning [Catron, 1991].
- node–name = a descriptive name attached to the task by the Production Planner and used by human operators for monitoring and debugging.
- parameter–list = names and values of information units which are known only at execution time. In the case of dynamic production planning this would include quantities, workpiece ids, tooling parameters, etc. In the case of previously defined production plans, parameters might include material handling information.

Drop_report (task–id)

Remove the designated task from the status report. This should only be used for plans which have reached a terminal state.

Pause (task–id)

Used to stall operations, for example during replanning. The task should be paused at the next Checkpoint.

Resume (task–id)

Used to resume a previously suspended task.

Terminate (task–id)

Stop and treat the plan as finished at the next Checkpoint.

Abort (task–id)

Stop the plan as soon as possible without damaging the equipment; damaging the workpiece is acceptable. To be used to recover from some anomalous situations, in particular hung tasks.

5.2 Conventions

When a subordinate receives an Execute command, it always creates a new task, and adds it to the task status report. If the task is unacceptable to the subsystem, the state of the task is rejected, which is a terminal state. Otherwise, the task goes into one of the nonterminal states while the subordinate begins planning and executing the task. Thereafter, the subordinate autonomously changes the state of the task as it executes it, until the task reaches a terminal state.

The combination (source, task–id) must always be unique in the subordinate. Once created, a task is current in the subordinate, regardless of state, until the subordinate receives a Drop_ report command for that task.

All other task management commands —Pause, Resume, Terminate, Abort, Drop_report —affect only tasks which have been previously created by an Execute command from the same source. Each of these commands only affects the state of the named task and the further handling of that task in the subsystem. If a subordinate receives a task management command for an unknown task–id, it must add the task to the task status report with a task status of rejected, and rewrites its current task status in the status mailbox.

Unlike supervisory commands, task commands can (potentially) come from more than one source, and task commands do not get explicit responses. The effect of a task command is to change some element of the subordinate's task status report (see Section 6.2). The task source therefore deduces receipt of the command by the change in the status report. In particular,

- the source determines receipt of Execute by appearance of the task in the status report.
- the source determines receipt of Drop_report by disappearance of the task from the status report.
- the source determines receipt of Pause, Resume, Terminate, Abort by a change in the value of the task management state.

6. Status Reports

The complete status of a subsystem consists of three logically distinct classes of information: system status elements, task status elements, and operational status elements. Every subsystem generates:

- a Guardian status report, comprising all of the above
- an administrative status report, consisting only of the system status elements
- one task status report for each client, comprising the task status elements for all tasks originated by that client.

The remainder of this section describes the individual status elements by type. Appendices A and B contain a description of the command and status message formats for the administrative, task and Guardian interfaces used in the 1990 MSI Architecture prototype controller.

6.1 System Status

These elements describe the administrative state of the subsystem. They include:

Subsystem state (see Section 3.1)

Capability index: number of changes made to the capabilities since subsystem start up

Last command: for last administrative command received from the supervisor/Guardian
— command identifier (sequence number)
— response code (accepted, or rejection code).

The last command information in the administrative status report refers to the last command from the supervisor, while the last command in the Guardian status report refers to the last command from the Guardian.

6.2 Task Status

These elements describe the tasks which have been given to the subsystem and their current state of execution. For each task in the subsystem, the information units include:

Task source, name of system supplying the task

Task identifier, as given by the source

Management state of task, one of:

- Normal — last management command for this task was Execute or Resume task
- Pausing — last management command for this task was Pause task
- Terminating — last management command for this task was Terminate task
- Aborting — last management command for this task was Abort task

On schedule (yes/no/unknown)

Nominal/actual initiation time

Nominal/actual completion time

Last checkpoint reached, if applicable

Current state of task, one of:

- rejected = task is unacceptable to the subsystem (terminal).
- activated = task is currently executing (nonterminal).
- suspended = task is waiting at a checkpoint for a resource, an external event, or a Resume task (nonterminal).
- completed = task is finished and no errors occurred (terminal).
- terminated = task is terminated at a checkpoint, stopped by supervisor command or a problem detected in the subsystem, even though it did not finish (terminal).
- aborted = task was terminated as expediently as possible by direction of supervisor, or unrecoverable problem in the subsystem (terminal).

Output parameters, if applicable.

6.3 Guardian Status

The Guardian status contains all the elements of the administrative status and task status report. The task status component contains three additional fields:

Detail Codes: one or more codes for the detailed status of the task, include failure codes, holding codes, event/resource identification codes, etc.

Messages: one or more optional console display messages corresponding to the codes

Assigned resources: names of immediate subordinates involved in the task.

Additionally, the Guardian status contains information describing the status of elements of the particular subsystem which are only meaningful in the context of this (kind of) subsystem and require knowledge of the internal functions of the subsystem to be useful. (This type of information is known as operational status and is often useful for console display.) There are two kinds of elements in an operational status report: configuration elements and specialized elements.

Configuration elements include for each configured subordinate:

- the subsystem name,
- the subsystem state, as perceived by the supervisor, and
- the timestamp of the last received status report.

Specialized elements (sometimes called front panel information) are defined by the subsystem, although some MSI Architecture standards may be developed for elements to be included for a given subsystem type. For a machine tool, this includes such information as coolant levels, current spindle speed, feed rate, tool in the spindle, tools in the drum/carriage, etc. For a robot, this includes joint positions, gripper configuration, current end of arm coordinates, current goal coordinates, etc. It may also include detailed information about position and operation in the local form of process plan, e.g. numerical code, robot programs, etc.

7. Notes

1. A subsystem cannot do anything about a failed supervisor. It is therefore pointless for a subsystem to attempt to diagnose the health of the supervisor, whether by meaningless commands, protocol violations, or time-outs. The subsystem's job is to keep its own house in order, no matter what it gets, or doesn't get, in the command mailbox.

2. A subsystem can and should diagnose the health of its configured subordinates, and it is appropriate to time-out responses when running in real time. Unfortunately, this creates some problems when running in rapid simulated time. This problem still needs to be researched.

8. Issues to Be Addressed in the Future

In 1990, the Factory Automation Systems Division at the National Institute of Standards and Technology developed a prototype MSI Architecture compliant controller using the control entity interfaces defined in this document. As a result of this prototype, several issues with the interfaces were discovered. The resolution of these issues is a primary focus of continuing work within the MSI project.

- Within the task command interface, there is no way for a task server to reject an invalid request pertaining to an active task. There is no way for a task server to reject a Resume request for a task that is not currently in the Suspended state. There is also no way for a task server to reject a Drop_report request for a task that is not currently in a terminal state.
- There does not currently exist an automated way to normally or abnormally shutdown a control hierarchy (or subsystem). This is so because neither the Terminate nor the Finish commands propagate to subordinate controllers, yet those are the only valid transitions from the Active to the Ready state. As a result, each controller must be individually and explicitly issued a Terminate or Finish command to transition to Ready. Only then can a Shutdown command (which does propagate) be issued to bring the control hierarchy to the Idle state. It should be noted that allowing the Terminate and Finish commands to propagate to subordinates is not an adequate solution: consider the case where in order to Finish or Terminate a task at one level in the hierarchy, a task request must be issued to a subordinate that has already Finished or Terminated its outstanding tasks and is no longer accepting new ones.
- The notion of checkpoint is not well defined. A checkpoint is defined as a step in the production plan where the manufacturing process may be halted in a safe manner without damaging either the equipment or the workpiece and later replaced or resumed. A task server is required to checkpoint a task upon receiving the task commands Pause or Terminate or upon receiving the administrative commands Pause or Terminate. Checkpointing a task at a lower level in the control hierarchy necessarily means that a higher level controller (namely the task client) is in the middle of a manufacturing step, and is not considered checkpointed on that task. The current meaning of checkpoint is not well defined for any level other than the Equipment Level.

Appendix A
Administrative And Task Message Formats Used in 1990

The format of an MSI message in a mailbox is an ASCII string where curly braces surround message objects and elements within these objects are delimited with commas. These elements may themselves be compound objects with their own elements. Spaces may be used for readability only directly following commas.

All messages have the form:

{originating subsystem name, timestamp, mailgram sequence number, {data}}

Where:

> ***originating subsystem name*** is the MSI name for the source controller,
>
> ***timestamp*** is a string in the following form:
>
> 4 digit year, 2 digit month, 2 digit day, 2 digit hour, 2 digit minute, and 2 digit second, all concatenated together in one string (all digits are decimal).
>
> ***mailgram sequence number*** is an ascii representation of an unsigned hexadecimal number less than 2^{32} or 100000000_{16}.

The following sections describe the ***data*** portion of the above message format.

Administrative commands

Administrative commands have the following format:

> ***{command id, command}***

Where:

> ***command id*** is an ascii representation of an unsigned hexadecimal number less than 2^{32} or 100000000_{16},
>
> ***command*** is one of the following:
>
> REPORT
>
> SYNC
>
> START_UP
>
> BEGIN
>
> PAUSE
>
> FINISH
>
> TERMINATE

SHUT_DOWN

EXIT

ESTOP

An example of an administrative command is {45, SYNC}.

Administrative status

Administrative status messages have the following format:

{admin state, last cmd id, response code, capability index}

Where:

admin state is one of the following:

DOWN

SYNCHRONIZING

IDLE

STARTING

READY

ACTIVE

PAUSING

PAUSED

TERMINATING

FINISHING

SHUTTING_DOWN

ABORTING

last command id is an ascii representation of an unsigned hexadecimal number less than 2^{32} or 100000000_{16}.

response code is zero when the last command is accepted, and some nonzero 8 byte hexadecimal number response code otherwise.

capability index is an unsigned 8 byte hexadecimal value representing a number of less than 2^{32}.

An example of an administrative status message is {SYNCHRONIZING, 0, 45, 0}.

Task requests

Task requests have the following format:

{task command, task id, list of parameters which vary with command}

Where:

task command is one of the following:

REPORT

EXECUTE

DROP_REPORT

PAUSE

RESUME

TERMINATE

ABORT

task id is an unsigned hexadecimal value representing a number of less than 2^{32}. It is unique only for a given task requestor.

parameter lists for their respective task commands are as follows:

REPORT— NULL

EXECUTE— ***{task activity, node name, parameter list}***

Where:

task activity is either a *work element* or a compound object of the form *{plan–id, plan–version}*, where *work element* is a primitive command, only specified when the task server is a primitive equipment controller. *plan–id* is a string which identifies a process or production plan to be used by the task server, and *plan–version* is a number indicating the version of the plan identified by *plan–id*. The compound element *{plan–id, plan–version}* is specified when the task server is a category 2 or category 3 controller (i.e., understands the MSI notion of process plan).

node name is a string passed by task requester for human task tracking purposes.

parameter list is a variable length data object supplying parameters for the specified plan or work element. This list is an alternating sequence of parameter name, parameter object where parameter object could be a list.

DROP_REPORT— NULL

PAUSE— NULL

RESUME— NULL

TERMINATE— NULL

ABORT— NULL

Examples of task requests are:

{EXECUTE, 1, {{shuttle_plan,1}, Make Shuttle, {PARTS, {8,27}}}}

{EXECUTE, 2, {{block_plan,1}, Make Block, {PARTS, {2,38}}}}

{REPORT,1,NULL}

{DROP_REPORT,1,NULL}

{PAUSE,1,NULL}

{RESUME,1,NULL}

{TERMINATE,2,NULL}

{ABORT,1,NULL}

Task responses

Task response messages have the following format:

> ***{{task source, task id, task state, management state, on schedule flag, {nom. starting time, actual starting time, nom. completion time, actual completion time}, last checkpoint, additional parameters}, {next task report in same form}}***

Where:

task source is the name of the task requester from the originating subsystem name in the basic mailgram format.

task id is an unsigned hexadecimal value representing a number of less than 2^{32}. It is unique only for a given task requestor.

task state is one of the following:

REJECTED

ACTIVATED

SUSPENDED

COMPLETED

TERMINATED

ABORTED

task management state is one of the following:

NORMAL

PAUSING

TERMINATING

ABORTING

on schedule flag is encoded as a 0 for yes, a 1 for no, and NULL for unknown, and designates whether the task is currently on schedule or not.

nominal starting time, ***actual starting time***, ***nominal completion time***, ***actual completion time*** are of the following format:

4 digit year, 2 digit month, 2 digit day, 2 digit hour, 2 digit minute, and 2 digit second, all concatenated together in one string (all digits are decimal). Any time field can be NULL if that field is currently unknown. If all fields are unknown, then the entire compound time object is NULL.

last checkpoint is the string representation of the node number of the last completed node in the process or production plan that is a valid checkpoint node. The node numbers within the process or production plan are specified by the process planner.

additional parameters is an optional compound object which contains a parameter list in the alternating name, value form, providing any return values associated with a plan as well as messages for debugging at the next level. This field is NULL if there are no parameters.

Additional reports in the above response form would be included in a task report when there is more than one task being serviced for a given requestor.

An example of a ONE response report is: {SLE3, 1, ACTIVATED, NORMAL, 0,{19900721130000, 19900721130500, 19900721131000, NULL}, 3, NULL}.

An example of a TWO response report is: {{VWS01, 1, ACTIVATED, NORMAL, 0, {19900721130000, 19900721130500, 19900721131000, NULL}, NULL}, {VWS01, 3, ACTIVATED, NORMAL, 0, {19900721130500, NULL, 19900721131500, NULL}, 0, NULL}}.

Appendix B
Guardian Message Formats Used in 1990

The format of an MSI message in a mailbox is an ASCII string where curly braces surround message objects and elements within these objects are delimited with commas. These elements may themselves be compound objects with their own elements. Spaces may be used for readability only directly following commas.

All messages have the form:

{originating subsystem name, time stamp, mailgram sequence number, {data}}

Where:

> ***originating subsystem name*** is the MSI name for the source controller,
>
> ***time stamp*** is a string in the following form:
>
> > 4 digit year, 2 digit month, 2 digit day, 2 digit hour, 2 digit minute, and 2 digit second, all concatenated together in one string (all digits are decimal).
>
> ***mailgram sequence number*** is an ascii representation of an unsigned hexadecimal number less than 2^{32} or 100000000_{16}.

The following sections describe the ***data*** portion of the above message format.

Guardian commands

Guardian commands have the following format:

> ***{command id, guardian command, parameter list}***

Where:

> ***command id*** is an ascii representation of an unsigned hexadecimal number less than 2^{32} or 100000000_{16}
>
> ***command*** is one of the following:
>
> REPORT
>
> SYNC
>
> START_UP
>
> BEGIN
>
> PAUSE
>
> FINISH
>
> TERMINATE

SHUT_DOWN

EXIT

ESTOP

IGNORE <configuration command>

ATTACH <configuration command>

DETACH <configuration command>

parameter list is used for configuration commands only and is a list of subordinates by subsystem name, otherwise ***parameter list*** is NULL.

Examples of Guardian commands are:

{45, IGNORE, {VWS01, MHS01}}

{8e, ATTACH, {VWS02, MHS02}}

{13, SYNC, NULL}

Guardian Status

Guardian status messages have the following format:

{admin state, last guardian command id, response code, {subordinate list}, {task response list}, {operational status list}}

Where:

admin state is one of the following:

DOWN

SYNCHRONIZING

IDLE

STARTING

READY

ACTIVE

PAUSING

PAUSED

TERMINATING

FINISHING

SHUTTING_DOWN

ABORTING

last guardian command id is an ascii representation of an unsigned hexadecimal number less than 2^{32} or 100000000_{16}.

response code is zero when the last command is accepted, and some nonzero 8 byte hexadecimal number response code otherwise.

subordinate list is a compound element containing the following information for each subordinate: ***{subordinate name, subordinate state, time stamp of last status}***

Where:

> ***subordinate name*** is the MSI name for the subordinate.
>
> ***subordinate state*** is one of the same list as admin state above.
>
> ***timestamp*** is of the following format:
>
> 4 digit year, 2 digit month, 2 digit day, 2 digit hour, 2 digit minute, and 2 digit second, all concatenated together in one string (all digits are decimal).

task response list is a compound element having the following format: ***{{task source, task id, task state, management state, on schedule flag, {nom. starting time, actual starting time, nom. completion time, actual completion time}, last checkpoint, additional parameters}, {next task report in same form}}***

Where:

> ***task source*** is the name of the task requester from the originating subsystem name in the basic mailgram format.
>
> ***task id*** is an unsigned hexadecimal value representing a number of less than 2^{32}. It is unique only for a given task requestor.
>
> ***task state*** is one of the following:
>
> REJECTED
>
> ACTIVATED
>
> SUSPENDED
>
> COMPLETED
>
> TERMINATED
>
> ABORTED
>
> ***task management state*** is one of the following:
>
> NORMAL
>
> PAUSING
>
> TERMINATING
>
> ABORTING
>
> ***on schedule flag*** is encoded as a 0 for yes, a 1 for no, and NULL for unknown, and designates whether the task is currently on schedule or not.
>
> ***nominal starting time***, ***actual starting time***, ***nominal completion time***, ***actual***

completion time are of the following format:

4 digit year, 2 digit month, 2 digit day, 2 digit hour, 2 digit minute, and 2 digit second, all concatenated together in one string (all digits are decimal). Any time field can be NULL if that field is currently unknown. If all fields are unknown, then the entire compound time object is NULL.

last checkpoint is the string representation of the node number of the last completed node in the process or production plan that is a valid checkpoint node. The node numbers within the process or production plan are specified by the process planner.

additional parameters is an optional compound object which contains a parameter list in the alternating name, value form, providing any return values associated with a plan as well as messages for debugging at the next level. This field is NULL if there are no parameters.

Additional reports in the above response form would be included in a task report when there is more than one task being serviced for a given requestor.

operational status list is a compound element containing an alternating sequence of type and value, reporting details of the internals of a subsystem. This is the means by which front panels are exposed to a human operator.

The one known operational status type is a "task_list". The elements in a task_list are formatted as follows: ***{task source, task id, task activity, {node list}}***.

Where:

task source is the task client associated with this task.

task id is the identifier used by the client for this task.

task activity is the "task" requested by the client with the EXECUTE command. Its value is the node name element from the EXECUTE command. It is a human readable/understandable element.

node list is a compound element describing the status of each currently active step in the process or production plan. It has the following format: ***{node number, node name, node status, {subtask}}***

Where:

node number is decimal number uniquely identifying a node (step) in the process or production plan being executed by the subsystem.

node name is a name for the node just enumerated.

node status is a character string indicating the state of the activity associated with the node. Its value is one of the following:

NEW

SLEEPING

WAITING

BUSY

DONE

subtask is a compound element describing a subordinate task if there is one associated with this node. It has the following format: ***{task server, task id, task status, management status}***.

Where:

task server is the MSI name of the subordinate executing the task.

task id is the identifier used by the controller executing the node for this task requested from its subordinate.

task status is the status returned by the subordinate, and is one of the following:

REJECTED

ACTIVATED

COMPLETED

TERMINATED

PAUSED

ABORTED

management status is the management status returned by the subordinate for the task, and is one of the following:

NORMAL

PAUSING

TERMINATING

ABORTING

Examples of Guardian status reports are:

{ACTIVE, 0, 0, {{MHS01, ACTIVE, 19901002131412}, {CDWS01, ACTIVE, 19901002131413}, {VWS01, ACTIVE, 19901002131415}}, {{shop–oversight, 110, COMPLETED, NORMAL, 0, {19901002090000, 19901002090000, 19901002131600, 19901002131905}, 18, NULL}, {shop–oversight, 112, COMPLETED, NORMAL, 1, {19901002110000, 19901002110000, 19901002151500, 19901002144959}, 12, NULL}, {shop–oversight, 120, COMPLETED, NORMAL, 1, {19901002130000, 19901002131520, 19901002162000, 19901002161642}, 10, NULL}, {shop–oversight, 122, ACTIVATED, NORMAL, 1, {19901002015000, 19901002150107, 199010020170000, NULL}, 6, NULL}}, {task_list, {{shop–oversight, 122, {make–shuttle}, {{10, machine–second–cut, NEW, NULL}}}}}}

{ACTIVE, 0, 0, {{VMILLC01, ACTIVE, 19901004170855},{VWSU4K01, ACTIVE, 19901004180000}}, {{VWSU4K01, 197, COMPLETED, NORMAL, 0, NULL, 3,

NULL}}}

{ACTIVE, 0, 0, {{VMILLC01, ACTIVE, 19901004170855},{VWSU4K01, ACTIVE, 19901004180000}}, NULL, NULL}

{DOWN, 0, 0, NULL, NULL, NULL}}

{ACTIVE, 0, 0, {{VWS01, ACTIVE, 19901004170329}, {MHS01, ACTIVE, 19901004171256}, {CDWS01, ACTIVE, 19901004171100}}, {{shop–oversight, 110, ACTIVATED, NORMAL, 0, {19901004142000, 19901004142210, 19901004200000, NULL}, 14, NULL}, {shop–oversight, 112, ACTIVATED, NORMAL, 1, {19901004170000, 19901004170000, 19901004191600, NULL}, 7, NULL}, {shop–oversight, 114, ACTIVATED, NORMAL, 1, {19901004182000, 19901004181815, 19901004195600, NULL}, 2, NULL}}, {task_list, {{shop–oversight, 110, {make–shuttle, 1}, {{6, machine–first–cut, BUSY, {VWS01, 197, ACTIVATED, N ORMAL}}, {13, deliver–lru, WAITING, {MHS01, 225, SUSPENDED, NORMAL}}, {14, Join, BUSY, NULL}}}, {shop–oversight, 112, {make–block, 1}, {{4, Deburr–block, BUSY, {CDWS01, 158, ACTIVATED, NORMAL}}}}, {shop–oversight, 114, {make–shuttle, 1}, {{2, machine–second–cut, WAITING, {VWS01, 118, SUSPENDED, NORMAL}}}}}}

Appendix C
Scenario of Commands to Controllers

The following example illustrates the normal initialization procedure.

Shop Level Administrative Supervisor	**Workcell Controller**
	System_status message format: ***{admin state, last cmd id, response code, capability index}***
	{DOWN, 0,0,0}
{100, SYNC}	
	{SYNCHRONIZING,100,0,0}
	{IDLE,100,0,0}
{101, START_UP}	
	{STARTING, 101,0,0}
	{READY, 101,0,0}
{102, BEGIN}	
	{ACTIVE, 102,0,0}

The following example illustrates the normal tasking procedure.

Shop Level Executive	**Workcell Controller**
Task requests have the following format: ***{task command, task id, list of parameters which vary with command}***	
{EXECUTE, 34, {{Get–shuttle–1–data, 01}, get–shuttle–data, {PARTS, {AC443, AQ972, A42176}}}}	
	Task response messages have the following format:

{{task source, task id, task state, management state, on schedule flag, {nom. starting time, actual starting time, nom. completion time, actual completion time}, last checkpoint, additional parameters}, {next task report in same form}}

{{SLE1,34,ACTIVATED,NORMAL,0,
{19901101120000,19901101120100,
19901101120500,NULL},1,NULL}}

This task status report acknowledges the receipt of the get_data command for the shuttle.

At this point, other execute_task commands are eligible to be issued by the SLE, assuming a type 2 or type 3 controller.

{EXECUTE, 35, {{Get–anc–101–data, 01},
get–anc–data, {PARTS, {AC4Q43, AQ972,
A42176}}}}

{{SLE1,34,COMPLETED,NORMAL,0,
{19901101120000,19901101120100,
19901101120500,19901101120300},2,
NULL},
{SLE1,35,ACTIVATED,NORMAL,0,
{19901101120400,NULL,
19901101120600,NULL},1,NULL}}

This status report communicates the completion of the get_data command for the shuttle and the receipt of the get_data command for anc.

{EXECUTE, 37, {{Make–shuttle–1, 01},
make–shuttle, {PARTS, {AC4Q43, AQ972,
A42176}}}}

{{SLE1,34,COMPLETED,NORMAL,0,
{19901101120000,19901101120100,
19901101120500,19901101120300},2,
NULL},
{SLE1,35,ACTIVATED,NORMAL,0,
{19901101120400,19901101120400,
19901101120600,NULL},1,NULL},
{SLE1,37,ACTIVATED,NORMAL,0,
{19901101120500,NULL,
19901101120700,NULL},1,NULL}}

{DROP_REPORT,34,NULL}

This command allows the subordinate to stop reporting on the finished get_data task for the shuttle.

{{SLE1,35,ACTIVATED,NORMAL,0,
{19901101120400,19901101120400,
19901101120600,NULL},1,NULL},
{SLE1,37,ACTIVATED,NORMAL,0,
{19901101120500,NULL,
19901101120700,NULL},1,NULL}}

The SLE determines receipt of the DROP_REPORT request by the removal of the task information related to that task from the task status report.

Spontaneous reports are generated when task fields change.

{{SLE1,35,COMPLETED,NORMAL,1,
{19901101120400,19901101120400,
19901101120600,19901101120600},1,
NULL},
{SLE1,37,ACTIVATED,NORMAL,1,
{19901101120500,19901101120600,
19901101120700,NULL},1,NULL}}

The workcell controller will issue a status report for each command received from the task supervisor. In addition, status reports will be spontaneously generated by the workcell at the start, completion, and checkpoints of a task.

{{SLE1,35,COMPLETED,NORMAL,1,
{19901101120400,19901101120400,
19901101120600,19901101120600},1,
NULL},
{SLE1,37,ACTIVATED,NORMAL,1,
{19901101120500,19901101120600,
19901101120700,NULL},2,
NULL}}

{DROP_REPORT,35,NULL}

This command allows the subordinate to stop reporting on the finished get_data task for anc.

{{SLE1,37,ACTIVATED,NORMAL,1,
{19901101120500,19901101120600,
19901101120700,NULL},2,
NULL}}

{{SLE1,37,COMPLETED,NORMAL,0,
{19901101120500,19901101120600,
19901101120700,19901101120700},3,
NULL}}

Appendix D
Glossary

baseline configuration

1. The process of specifying the initial configuration of the system.

2. The result of the process in 1, (i.e., the specification of the initial configuration of the system.)

capability index

A value which indicates whether a subsystem has all capabilities to be expected from the baseline (initial) configuration or some different set of capabilities.

checkpoint

A step in a production plan where the manufacturing process may be halted in a safe manner without damaging either the equipment or the workpiece and later replaced or resumed.

configuration

The collection of resources in a given facility and their physical and logical relationships to one another.

control entity

A decision making agent which directs the execution of tasks or executes tasks.

controller

1. A synonym for control entity.

2. The implementation of a control entity on a particular hardware platform which controls a physical piece of equipment.

3. When prefixed by “equipment,” “workcell” or “shop”, the term refers to a particular control entity, type of control entity, or implementation of control entity which performs a precise function in a particular manner.

equipment level

The lowest level of control specified by the MSI architecture. A controller at this level has an interface to its supervisor which conforms to the MSI architecture, but may have a non-standard interface to its subordinates

executive

A component of a control entity responsible for directing the execution of assigned tasks.

guardian

That entity described in the MSI architecture which monitors and intervenes in a control activity.

process plan

A specification of tasks, or operations, to accomplish a given goal — typically the manufacture of a product. In contrast to a production plan, a process plan serves as a template, or recipe, for the accomplishment of the tasks.

production plan

A specification of tasks, or operations, to accomplish a specific goal, using specific resources at specific times. (cf. process plan)

shop level

The highest level of control in the MSI architecture.

work element

1. A discrete activity for any control level in a manufacturing environment.

2. The collection of information which describes 1.

workcell level

An intermediate level of control which is responsible for the coordination of activities by subordinates. The workcell level presents interfaces which comply with the MSI architecture toward both its supervisor and subordinate controllers.

Reference List

Albus, J., Barbera, A., Nagel, R., “Theory and Practice of Hierarchical Control”, Twenty–third IEEE Computer Society International Conference, Sept. 1981.

Catron, Bryan and Ray, Steven, “ALPS — A Language for Process Specification”, International Journal of Computer Integrated Manufacturing, Vol. 4, number 3, 1991.

Libes, Don, “NIST Network Common Memory User Manual,” NISTIR 90–4233, PB90–183260/AS, January 1990.

O’Halloran, D. R. and Reynolds, P. F.,”A Model for AMRF Initialization, Restart, Reconfiguration and Shutdown”, NBS/GCR 88–546 May 23, 1986.

Rybczynski, S., et al. “AMRF Network Communications,” NISTIR 88–3816, pp. II–1 – II–9, June 30, 1988.

Senehi, M. K., et al., “Manufacturing Systems Integration Initial Architecture Document”, NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, forthcoming.