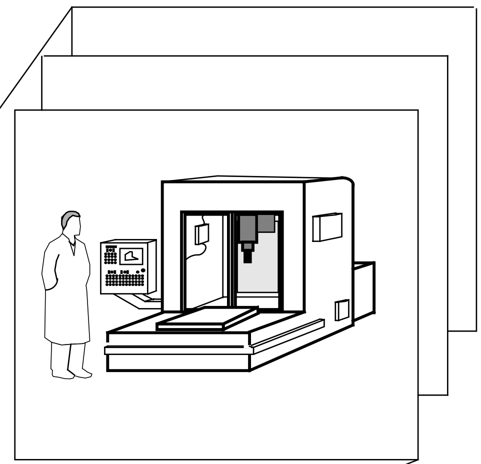
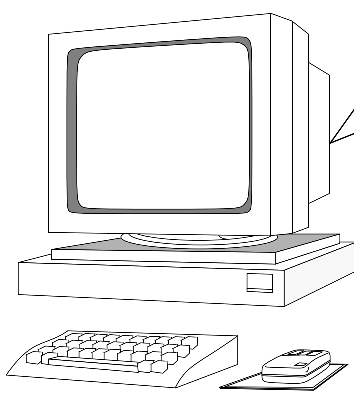
NISTIR 4682

September, 1991

Manufacturing Systems Integration

Initial Architecture Document





M. K. Senehi

Ed Barkmeyer

Mark Luce

Steven Ray

Evan K. Wallace

Sarah Wallace

United States Department of Commerce

National Institute of Standards and Technology

Manufacturing Engineering Laboratory

Gaithersburg, MD 20899

NISTIR 4682
September, 1991

Manufacturing Systems Integration

Initial Architecture Document*

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

1. Preface

This document presents the development of the Manufacturing Systems Integration project as of July, 1990. It provides a foundation for the understanding of the project and its results. It is important to note that since work on the project is ongoing, some of the results presented in this document have been superceded. This document is the initial report of a series of reports planned for this project; it serves as a reference for these future reports.

2. Introduction

A major activity of the Manufacturing Systems Integration (MSI) project is the development of a system ***architecture***[1] that incorporates an integrated ***production planning*** and control environment. This document describes the first draft of that architecture and serves as an interim report to record progress in the architecture design. The document was developed by the MSI Architecture Committee[2] and incorporates the consensus of that committee as of July 1990. The further activity of the MSI project is the cyclic implementation, testing and refinement of the proposed architecture until a complete, detailed, implemented, and thereby proven, architecture is developed.

The document is divided into three major sections. The first section gives the background and motivation for the MSI project. The second section discusses basic issues which have guided the development of the MSI architecture. The third section outlines the major components of an integrated production planning and ***control architecture***, and summarizes the functionality of each major component. Appendix 1 is a glossary of terms used with specific technical meanings within this document. Appendix 2 discusses the design of the MSI ***native controller*** used in the 1990 implementation of the architecture. Appendix 3 discusses the strengths and weaknesses of the architecture discovered in this implementation.

2.1 Project Direction and Goals

The goal of the MSI project is to develop an approach to solving 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 organizations.

---

1. The first occurrence of glossary terms appear in bold-italics (see Appendix 1 - Glossary).
2. In 1990, the Architecture Committee members were Ed Barkmeyer, Mark Luce, Steven Ray, M.K. Senehi and Evan Wallace.

2.2 Historical Background

As the United States strives to remain competitive in the world market, the need for technological improvements in the manufacturing arena is becoming acute. Improvements are anticipated within each of the four distinct aspects of the ***manufacturing process***: ***design*** engineering, manufacturing engineering, production engineering and shop floor control. However, the automation and the integrated operation of the factory will produce improvements in the quality of products and reductions in the production costs greater than advancements in any one of these areas alone. Accordingly, there is significant research on factory automation and integration.

The Automated Manufacturing Research Facility (AMRF) [1], established in 1981, is a small-batch[3] manufacturing system testbed at the National Institute of Standards and Technology (NIST) comprised of manufacturing equipment and systems designed to support research in computer integrated manufacturing. The concepts of ***hierarchical control*** for the factory and plug-compatible interfaces for discrete factory systems were developed at this facility. Because of the disparate equipment on the shop floor, implementation efforts focused on physical integration of ***workstations***, shop floor control, command and status protocols, ***communication*** and data access on the shop floor. This effort was successful in producing integrated workstations and obtained limited results in inter-workstation integration.

Subsequent work at NIST in the Manufacturing Data Preparation project (MDP) [2] continued to focus on the integration of design, manufacturing and shop floor engineering. This project, however, was primarily concerned with the representation and transfer of data through the critical junctures between these systems. The MDP project used the STEP[4] representation for parts and uncovered the need for additional work on representations which were not being supported by the STEP/PDES[5] [4] effort at that time. Primary outputs from the project were the development of a language for ***process plan*** specification, development of the means for a representation of removed volumes of material for manufacturing, and a ***controller*** which parses the process plan specification language and dynamically obtains domain-specific information from one or more independent ***database(s)***.

The MSI architecture is seen by the authors as a continuation of the AMRF work on the shop-floor control architecture and the interfaces among control systems. The hindsight gained from implementing the initial AMRF architecture[6] leads us to revise the ***hierarchical control architecture*** of the AMRF *as designed*. In many cases, the revisions are statements of the principles which were actually used in *implementing* that architecture; in other cases, they serve to resolve problems with that architecture which were discovered during the original implementation. For those who view hierarchical control as a collection of principles which guide the development of control systems and interfaces, the MSI architecture is simply the next step in the continuing evolution and refinement of those principles. Additionally, the MSI architecture contains concepts for error handling and production management.

2.3 Related Projects and Standardization Efforts

Many projects both internal and external to NIST have evolved which use concepts similar to

3. For a clarification of the term small batch see Section 3.3.1 on page 9.
4. Standard for the Exchange of Product Model Data. See [3] for additional information.
5. PDES is an acronym for Product Data Exchange using STEP.
6. By initial architecture, we mean the initial design work done in the AMRF in the years 1980-1982.

those developed in the AMRF. Although cataloging these projects and their results would be beyond the scope of this document, there are two bodies of work which relate closely to the MSI project: (1) The Next Generation Controller Program and (2) the NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM).

The Next Generation Controller Program is part of a national initiative to revitalize the American machine tool industry. Part of its mission is to develop a specification for an Open System Architecture Standard for automated workstations which perform material cutting or removal, metrology, inspection, testing, assembly, material handling, and composites fabrication or forming. The purpose of this specification is to enable manufacturers to create interchangeable and interoperable Next Generation Controller components. The project goal is to develop enabling technologies to accomplish interchangeability of automated workstations. These technologies include a neutral manufacturing language standard, product definition exchange standards and general-purpose reusable software components.

The second body of work, NASREM, specifically "defines the functional requirements and high level specification of the control system for the NASA Space Station IOC Flight Telerobot Servicer" [5]. This work is developed from the ideas of the AMRF and from other organizations including NASA, Langley Research Center, Oak Ridge National Laboratory, Johnson Spaceflight Center, Marshall Spaceflight Center, and the Ames Research Center. Like the MSI project, the NASREM work presents further refinements of the hierarchical control concepts in the AMRF. Much of the model addresses lower levels of control dealing with equipment in fine detail, which the MSI architecture does not even attempt to describe. The upper three layers of control described in the model differ from those described in the MSI architecture, although a loose correspondence may be mapped out. The fundamental reason for the discrepancy is that the MSI criterion for defining a level of control differs from that of NASREM. (See Section 3.4.2 on page 13 for a discussion of the MSI concept of control level). The authors view the work of MSI as complementary to that of NASREM, with NASREM addressing the lower levels of control and MSI addressing the higher ones. Since the nature of typical tasks, goals and ***resource allocation*** is different in the two domains, it is appropriate to have two complementary approaches.

In addition to the work stemming directly from the AMRF, several national and international standards were initiated in the past decade whose goal is to develop factory architecture and interface standards. The MSI architecture committee is aware of these ongoing standardization efforts, contributes to standards development where possible and forges ahead to develop theories and models for input to the standardization process. Although an exhaustive list of standardization work involving manufacturing is beyond the scope of this document, several of the major standardization efforts and their relationship to the MSI project are mentioned here.

Two of the broadest manufacturing architecture standardization efforts to be attempted are the ESPRIT Consortium AMICE effort [6] and the efforts of ISO TC184/ SC5. The ESPRIT Consortium's AMICE project subsumes the MSI concerns as a small part of its scope, which is integration of all aspects of a manufacturing enterprise. A primary product of this effort is the Computer-Integrated Manufacturing Open Systems Architecture (CIM-OSA). It is the intent of the MSI architecture committee to maintain an awareness of, and contribute to, such a specification.

The effort of ISO TC184/ SC5 encompasses four major areas: (1) Architecture, (2) Glossary (of manufacturing terms), (3) Communications, and (4) Manufacturing Languages. Each of these areas of inquiry is handled by a specific Working Group (WG). Within WG1, the development of a reference model for production control is underway [7]. This model identifies twelve manufacturing functions and a six-level model of manufacturing facilities. It concentrates on specifying flow of control and information throughout the facility. It specifies a number of levels of control and assigns particular functions to specific levels. In contrast, the MSI project does not specify the number of levels of control which an implementation of the architecture may have and concentrates on identifying the need for and characteristics of any level of control. It identifies three generic types of ***control entities*** based upon the level of control which they occupy in the hierarchical scheme and assigns generic functions to these types. Additionally it classifies control entities into (orthogonal) categories based upon their ability to queue tasks. We believe that abstract characterization of levels of control and formalization of their generic interfaces is likely to yield much more success in the integration of diverse controller products than any control architecture with a specific number of levels applied to diverse manufacturing enterprises. This basic aspect seems to be missing from all of the previously mentioned architectural efforts.[7]

The most mature and the most application-oriented of the manufacturing standards discussed here is the Manufacturing Automation Protocols (MAP) standard [10],[11]. MAP concentrates primarily on standardizing the formal interfaces to what MSI refers to as "Equipment Level control entities" (see Section 4.7.3 on page 44). Note that MAP specifies the manufacturing application protocol Manufacturing Messaging Specification (MMS) [12]. The MSI architecture committee regards these standards as a foundation upon which to build architectural control-entity interface specifications and is evaluating this option.

7. However, this issue is also mentioned in [8] and [9].

3. Key Issues and Concepts

This section identifies and discusses a number of key issues which impact any manufacturing architecture. The resolutions of these issues constitute the premises underlying the design of the MSI architectural components described in Section 3. Section 2 presents the resolutions agreed on by the MSI Architecture Committee and the rationale behind them. In particular, comparisons are made with concepts used in the initial AMRF architecture.

The MSI architecture committee assumes that the ultimate goal of research in automated manufacturing is to develop concepts which enable organizations to produce high quality products at a low cost. To achieve this goal it is necessary to consider both maximizing the number and quality of saleable units and minimizing the cost of manufacture. The MSI project concentrates on minimizing the cost of manufacturing items by improving the efficiency of the manufacturing process. The MSI project includes management input and variables in cases where this information directly affects the manufacturing process, but does not address business decisions. For example, the MSI architecture would support the selection of an alternative method of making a part based upon its cost, but does not address the issue of how a decision to make or buy a type of part is made.

The MSI project approach is top-down from these higher level goals of maximizing productivity rather than bottom-up from observation of performance on the ***shop*** floor. The result of this top-down view is greater integration of the shop throughout the life-cycle of the manufacturing project.

The architecture described in this document is a second generation AMRF architecture. The MSI architecture serves as a reference model for discrete machined parts manufacturing, defining the major subsystems and the interfaces that exist between them. It provides a basis for demonstrating various types of production planning architectures in similar discrete manufacturing domains and may be generalized to apply to other discrete domains[8].

3.1 Integrated Manufacturing Model

The MSI architecture committee perceives a shop as a number of discrete systems interconnected by information and control interfaces. In light of this perception, the following definition of integration is proposed:

> Integration is the process of defining the information interfaces, defining the control structure, and providing services for communications, data access and data management within an enterprise.

Manufacturing systems such as ***scheduling***, ***process planning***, ***order entry***, and design systems are integrated by interfacing them to the data representations, the control structure and the data access system. This approach is directly derived from earlier AMRF work [13].

Data and control interfaces are separated and need not use the same communications channels [14]. Figure 1 on page 6 illustrates this concept. A data interface is composed of data presentations and a data access method whereas a control interface is composed of command and status messages. It is important to note that any implementation of this architecture has considerable lat-

8. Note that at present the scope of MSI specifically excludes the domain of continuous processing, as found in the chemical manufacturing industry.

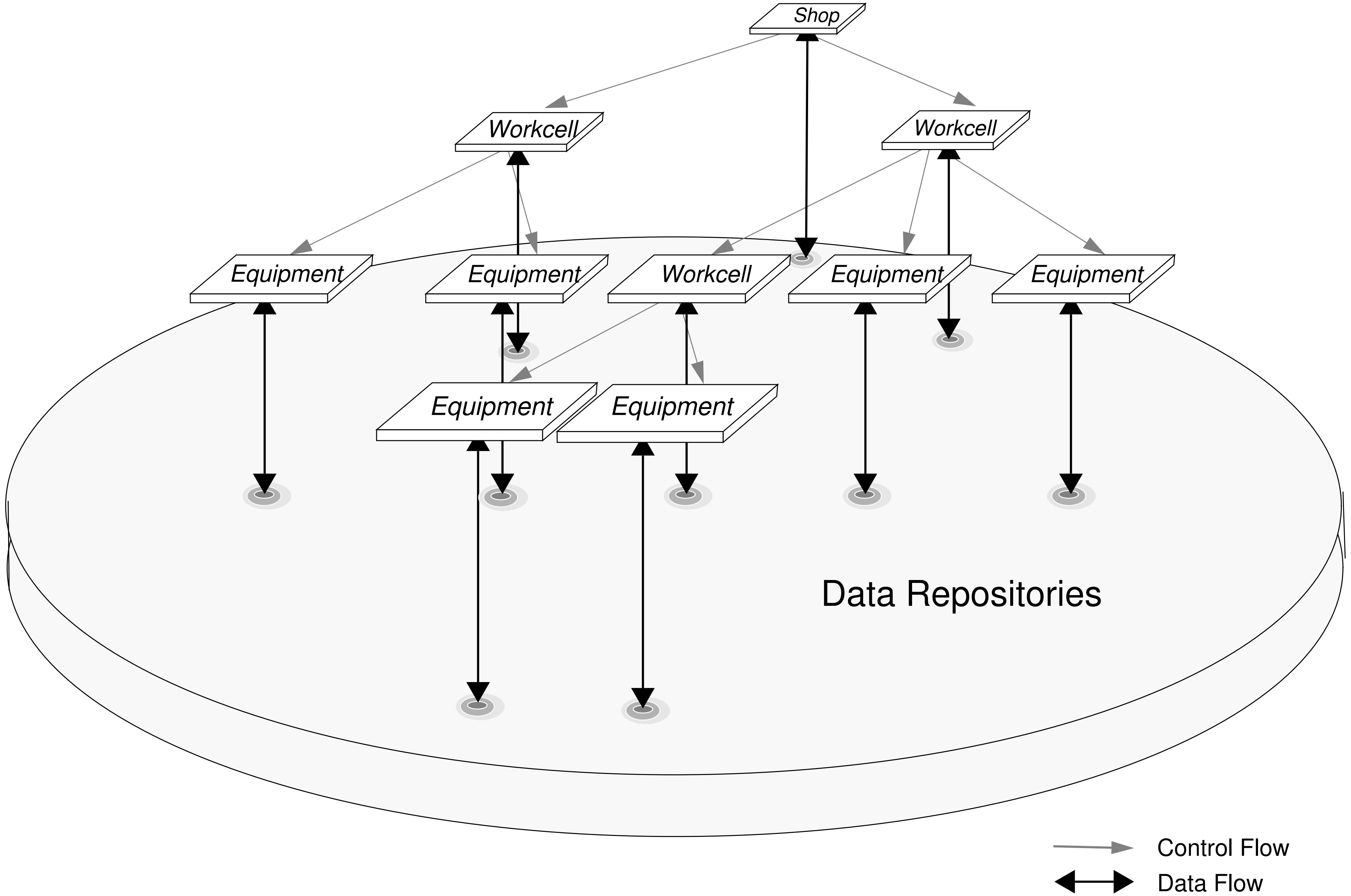


Figure 1 -Control Flow vs. Data Flow

itude in determining how information is treated by specifying whether it is transferred as commands or as data.

The MSI architecture posits the existence of a globally accessible ***information base,*** which consists of all the manufacturing data which is shared by more than one system. The information base is physically stored in ***data repositories***, e.g. files or databases, which may be either centralized or distributed.The architecture mandates the availability of a data access system which enables any system to access any of the data transparently, that is, without requiring the requesting system to have any knowledge of the location or format of the stored data.[9] This data access system makes the information base appear to be stored in a single "global" repository. Data interfaces, therefore, are between the various systems and the global data repository. Each system has only one data interface, rather than one interface to each repository or to each other system with which it shares information [15].[10]

The MSI architecture identifies a number of discrete systems which are normally part of a shop production environment. The MSI architecture provides support for the integration of a shop containing these discrete systems. In particular, information required and produced by these systems is included in the global information base. Currently supported discrete systems are:

- Process Planning Systems —which create step-by-step plans or numerical code for manufacturing the part, associated fixtures and jigs according to the part design,
- Production Planning Systems —which select batch sizes, specific machines, and scheduled times to perform the tasks specified by the process plan,
- Controllers—which perform manufacturing tasks,
- Order Entry Systems—which permit entry of orders which direct the shop as to what to make and when to make it,
- ***Configuration Management*** Systems—which identify and control shop resources and capabilities, and
- Material Handling Systems—which route and deliver material throughout the shop.

Part Design Systems, which create the designs for parts, associated fixtures and jigs, are identified as systems which must be supported to achieve full shop integration; expansion of the MSI architecture to include part design is planned for future years. As the architecture is expanded, additional systems may be identified. Figure 2 on page 8 shows the overall organization of a sample implementation of the MSI architecture.

In addition to issues mentioned thus far, there are human engineering factors. Two concerns in this area are providing for ***human interfaces*** to the various systems and providing for overriding the automatic operation of the system. These issues are discussed later in this document.

3.2 Architectural Generality

The MSI architecture is sufficiently general that it may be used in many manufacturing domains. For example, there is a mapping from the architecture as presented to its implementation for discrete metal parts fabrication. The mapping for the apparel industry, for example, may be different.

9. For performance reasons, private data and private data access is permitted for data which are used exclusively by a single system. See Local and Global Data Systems on page 33.
10. Note that the systems may well have other interfaces which are control interfaces.

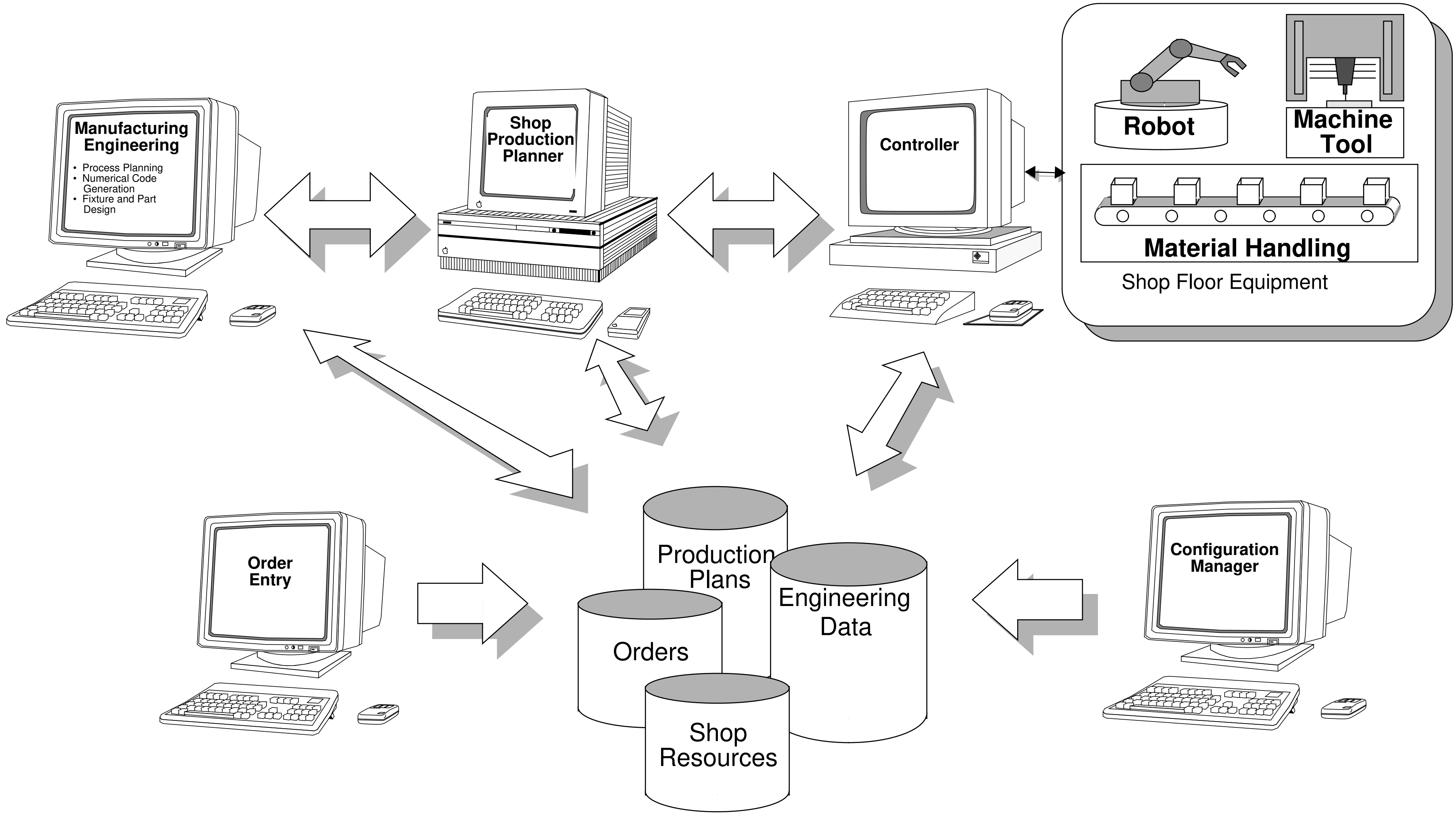


Figure 2 -A Sample Implementation of the MSI Architecture

The philosophy of the project is to identify and incorporate generic aspects of manufacturing into the basic system design while providing mechanisms for accessing domain-specific knowledge, without meshing it inextricably into the design itself.

The fundamental ***control*** and ***communication paradigms*** are expected to be universal, but the objects manipulated are different. Specifically, this means that the architecture can presume a common notion of process, a common means of representing processes, and a common means of controlling processes, while some individual process elements, processing systems and hardware are specific to a given manufacturing domain. As a consequence of this, both the systems and the process elements must be described in the global information base and manipulated as data except in those systems which are necessarily domain-specific. Typically, the lowest levels of control will be domain-specific. For example, milling machine controllers are specific to the machined parts domain.

3.3 Architectural Flexibility

The MSI architecture is designed to permit considerable flexibility for ***dynamic reconfiguration*** without becoming unwieldy. Emphasis was placed on accommodating reconfiguration for two reasons. First, past experience in the AMRF project has indicated that such flexibility is highly desirable, at least in small batch machined parts manufacturing, where changes in the shop are commonplace. Second, the flexibility of the architecture makes it more useful, by being less restrictive of the types of systems which may be integrated. Each of these perspectives is discussed below.

3.3.1 Domain Characteristics

Although the concepts developed in the project are widely applicable, the project makes a number of assumptions concerning the initial domain in which this technology is to be applied. It is assumed that the shop layout is stable and that the cost of physically rearranging the shop, either by purchasing new equipment or moving the equipment on the shop floor, is significant. Next, it is assumed that the variety of products to be manufactured is great enough, and the lot size in which these parts are made is small enough, that the manufacturing environment will be used in many different control ***configurations***. This small-batch operation mode is in contrast with an environment such as an automobile assembly line, where changes occur infrequently.

However, it is desirable that the design permits flexibility for logically adding and removing equipment without major reprogramming. Dynamic changes in the control structure will permit adjustment due to changes in the shop. Typically these changes are due to equipment failure on the shop floor and reconfiguration for routine maintenance or changes in product mix.

3.3.2 Integration of Manufacturing Systems

The MSI architecture must permit the integration of physical, commercial, and academic systems. Certainly, it is not feasible to build an architecture which can accommodate every possible manufacturing system without any limitation. In designing the MSI architecture, a goal is to minimize the number of constraints which it imposes in order to maximize the number of systems which may be integrated.

Given an existing system which is a candidate for inclusion into an implementation of the MSI architecture, its compatibility may be tested by the following algorithm. Represent a nominal shop in terms of MSI architectural elements. Then generate a control diagram of the resultant facility model. Identify the functional boxes which correspond to the functions provided by the candidate system. Consider the set of functional boxes to which the product corresponds as a single functional unit, whose internals are hidden. This unit may be regarded as a ***black box***. This unit has a set of architecturally specified interfaces to the other elements in the shop. When a system is compatible with the MSI architecture (i.e., well-behaved), it provides exactly all the functions of the black box and exactly all the specified interfaces in some fashion. A system is said to be ill-behaved if it packages pieces of architectural components and communication paths in a way which is externally inconsistent with the architecture.

A well-behaved system can be incorporated by being described in the global information base and interfaced to fit comfortably into the data and communication environment. Such interfacing consists of converting commands and information into forms expected by the system (preprocessing) and converting outputs of the system into forms expected by the MSI systems (postprocessing). See figure 3 on page 11. Ill-behaved systems can be incorporated only by circumscribing their operation, reducing their functionality to a subset which is consistent with the architecture. In some cases, ill-behaved systems simply cannot be integrated into an implementation of the MSI architecture. See figure 4 on page 12 for an illustration of a well-behaved versus an ill-behaved system.

The architecture must allow for the inclusion of a system whose internal structure is unknown, but whose external interfaces conform to the architectural specification at any given level of control. In particular, the MSI architecture will support distributed processing wherever it is feasible. The architecture will not presume distributed systems, but rather will permit a distributed implementation of a monolithic architectural component, as long as external interface requirements for that component are not violated. Implementations should never assume the availability of hidden information transfer paths, including local data repositories between systems which are not defined to be either part of the same architectural element or necessarily local.

3.4 Foundations of the MSI Project

In the following section, underlying principles are discussed which have a profound influence on the creation of the MSI architecture.

3.4.1 Hierarchical Problem-Solving Paradigm

A problem-solving *paradigm* is a method for organizing reasoning steps and knowledge to construct a solution to a problem [16]. In contrast, a problem-solving *model* provides a conceptual framework for organizing knowledge and a strategy for applying that knowledge.

The MSI project is concerned with “the manufacturing problem.” The manufacturing problem, stated earlier in this document, is to produce quality products at low cost. The MSI architecture is based upon the view that hierarchical decomposition is an appropriate strategy to use in solving this problem. Hierarchical problem decomposition is the process of dividing a problem into a number of less complex subproblems.

The manufacturing problem can be considered at different levels, each having different characteristics. At its highest levels, the manufacturing problem is seen as a management and economic problem. Choices at this level are business decisions: what product to manufacture, whether to manufacture or buy a certain part, what machines to buy and maintain to produce the product, what products to produce and so on. At these levels there are always multiple, competing goals. At the other end of the spectrum, from the point of view of the machinist, the problem is purely technical, involving the correct operation of an individual machine performing a specific ***process***

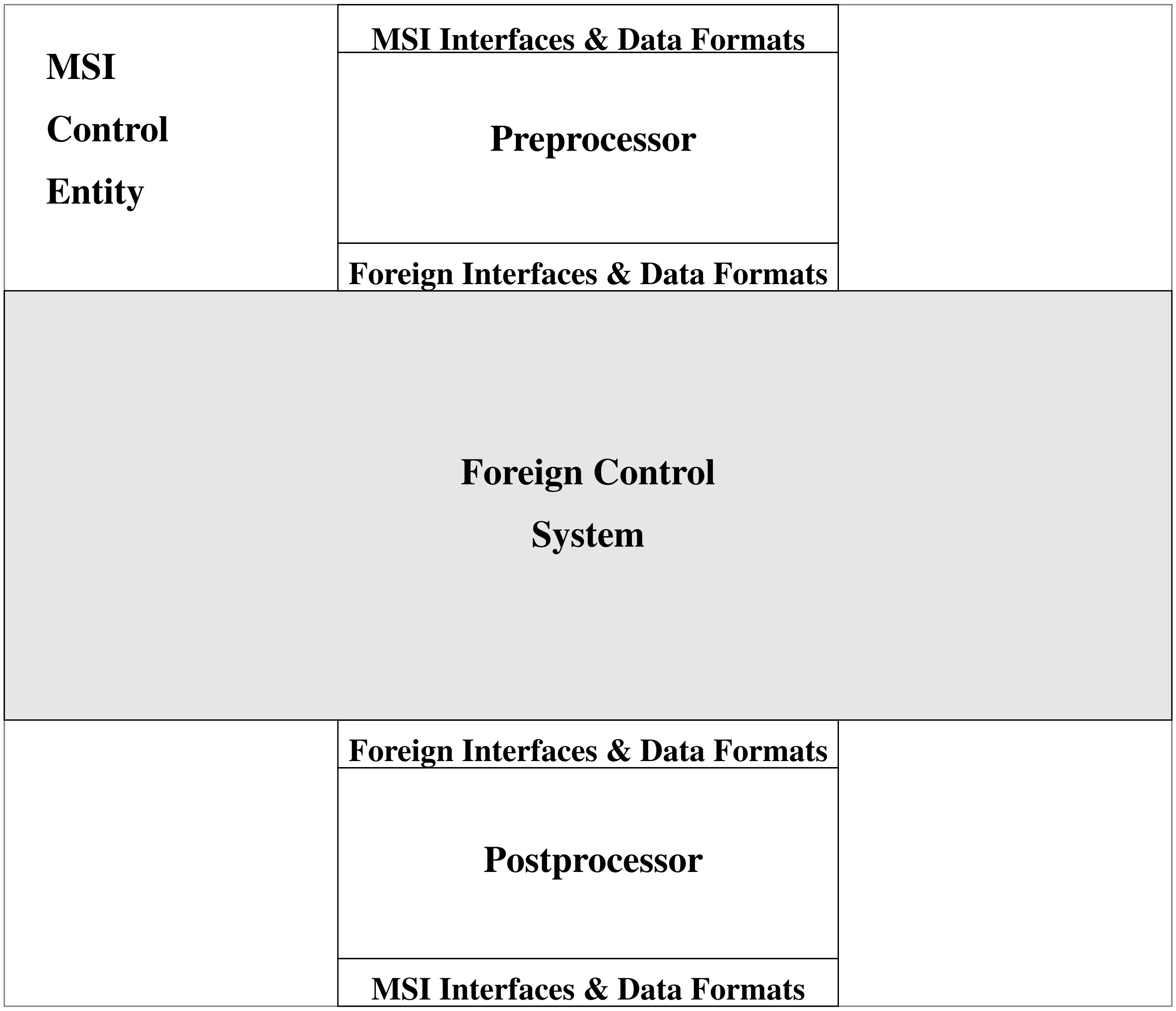


Figure 3 -Interfacing a Control Entity

on a specific workpiece. Here, at any given time, only a single goal exists. Intermediate levels involve both business and technical decisions and often involve multiple simultaneous goals. Broadly speaking, in the upper layers of the hierarchy, business considerations dominate, while at the lower layers of the hierarchy, production concerns dominate. Within each level, the internal problem-solving paradigm is not specified. Different levels may use different paradigms within the same system.

For the *control* system, the MSI architecture mandates a hierarchical control structure which corresponds to the levels of hierarchical decomposition introduced in solving the manufacturing problem. In contrast, other discrete systems, such as process planning, scheduling, and configuration management, may use distinguished problem-solving levels to simplify and limit the scope of

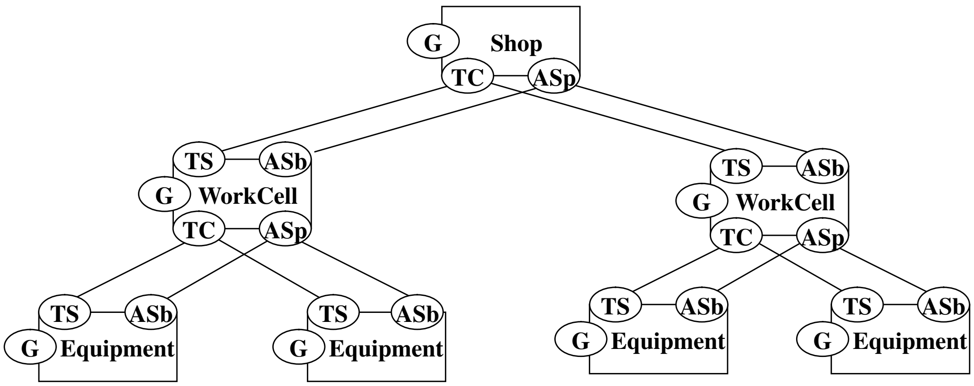


Figure 4-A. Nominal Plant Architecture

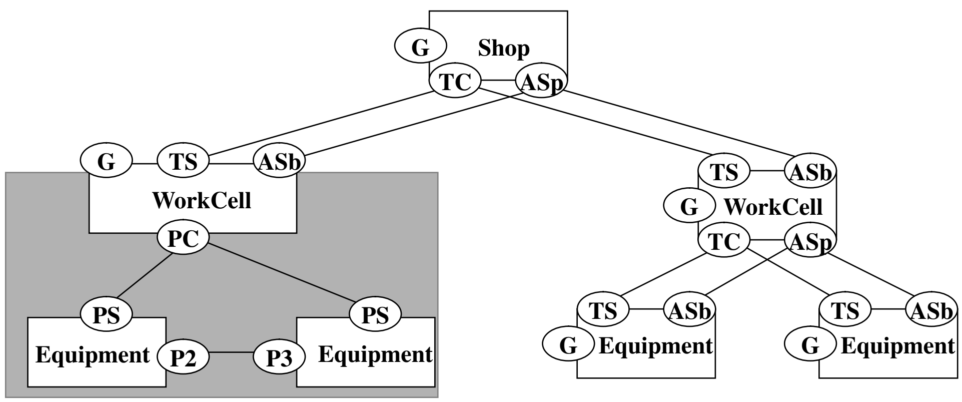


Figure 4-B. Conforming Black-Box Workcell

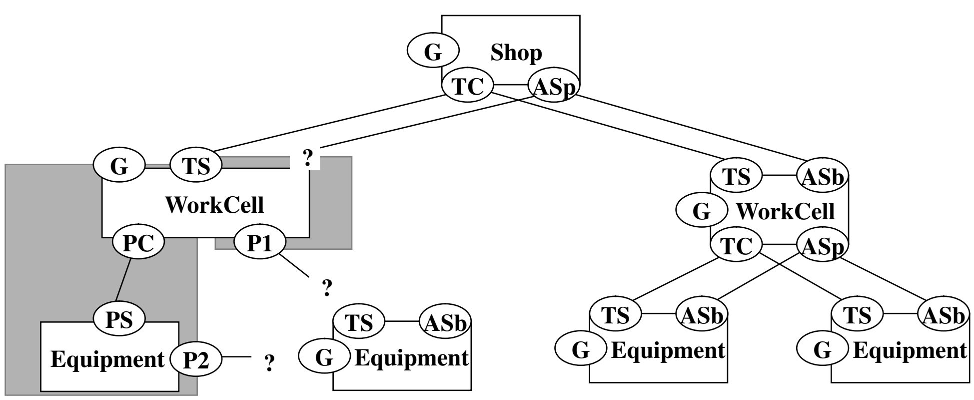


Figure 4-C. Ill-Behaved Black-Box Workcell

Legend

ASp = ***Administrative Supervisory Interface***

ASb = ***Administrative Subordinate Interface***

G = ***Guardian Interface***

PC = Private Client Interface

PS = Private Task Server Interface

Pn = Other Private Interface

TC = ***Task Client Interface***

TS = ***Task Server Interface***

Figure 4 -Black Box Workcells

the problem, but are not required to do so. It is important to recognize that the approaches chosen for process planning, production planning, and other discrete systems are independent of one another. One can very well adopt centralized process and production planning models while still supporting hierarchical control. Alternatively, one may choose to use a hierarchical process planning approach followed by a centralized scheduling cycle. The architecture described in this document makes no assumptions about which of these alternatives are chosen for either process planning, production planning, or configuration management. Whatever their internal structure, discrete systems are required to generate necessary information for any appropriate level of control.

3.4.2 Levels of Control in MSI

In the MSI architecture, control levels are arranged in a hierarchical structure. A level of control may be introduced whenever a coordinating or supervisory function is needed. This typically occurs when the decomposition of the initial problem regularly results in less complex, independent subproblems. For example, this occurs when the task decomposition of the problem often or always involves similar tasks or the same collection of systems. In particular, this means that there may be a different number of control levels, depending upon the shop configuration. Each level reflects the characteristics of the associated problem level.

This contrasts with the original concept of control in the AMRF, where a specific number of levels of control are defined and where each level is assigned specific manufacturing functions. The fundamental assumption in the initial AMRF design is that each layer of control corresponds to a level of problem decomposition. In addition, this design provides specific guidelines for the identification of levels of control for a given application. The two such guidelines are:

(1) Management of complexity: It has been shown that a human being can manage 7-plus or-minus-2 simultaneous activities. For more activities than this, the interactions between the activities become too complex for the human to make good decisions. Therefore, any complex of 7-plus-or-minus-2 controllers requires a level of control. [17]

(2) Time/Space planning horizons: When the planning horizons of the supervisor and the subordinate differ by more than two orders of magnitude, the rate of interaction cannot be appropriate to the level of detail involved in the commands. Therefore, one should expect to see a level of control with a planning horizon of *n* milliseconds, and another at *n* seconds, another at *n* minutes, another at *n* hours, etc. Each of these levels will have a proportional spatial horizon and level of task detail. [5]

The difficulty with these two criteria from the MSI point of view is that they are not based on problem decomposition or activity coordination requirements. In the case of the first criterion, note that while it is difficult for human or computer intelligence to manage 7 subordinates performing different tasks, it is not difficult to manage 20+ subordinates performing similar tasks. That is, it is the difference in the nature and complexity of the problem decomposition that determines the need for a level of control, not the number of subordinates. (It is not at all clear, for example, that a shop with 20 machining centers producing similar parts requires an intermediate level of control between the shop foreman and the machining centers.) In the case of the second criterion, it is not the planning horizon or the work volume of a subordinate which determines the needed rate of interaction, it is the complexity and duration of the tasks which can be given to the subordinate. It is to be expected that the planning horizon of a controller will be commensurate with the complexity of the tasks given it, but only at the lowest levels of control. This view applies well to robots, but does not generalize well to factories. It is our view that the spacial and temporal characteristics of the levels of control are the result of the factorization (i.e. the breaking into logically separate components) of a manufacturing control problem.[11] That is, these are not the criteria for determining the need for a level of control, but rather (typical but not necessary) hallmarks of a correct problem decomposition.

In the MSI architecture, there can be only one highest level of control. The highest level of control currently identified in the MSI architecture is called the ***Shop Level***. In the initial AMRF architecture the highest level of control is the Facility Level, which is immediately above the Shop Level. Functions performed at the Facility Level include manufacturing engineering, order handling, cost accounting, inventory management, procurement, performance monitoring and long-range scheduling [18]. In the MSI architecture, higher levels of control above the Shop Level, which would deal with business and management concerns and coordination of multiple shops, have not yet been defined.

The scope of the Shop Level includes all production activities within a single production shop – a closely related collection of manufacturing design, engineering and production processes. This level of control has general responsibility for all production processes involved in making some general category of parts, although many of the details of these processes are the responsibility of particular subordinates. The Shop Level Controller (SLC) directs the implementation of the overarching plan for fulfilling an order. The SLC is aware of all the orders for parts, scheduling constraints (due dates of orders), schedules and routing slips for the parts to be made. When there is more than one order, the SLC manages competing goals.

The lowest level of control in the MSI architecture is the ***Equipment Level***. An Equipment Level Controller (ELC) may have internal subordinates, but these subordinates are not visible to its supervisor. An ELC can execute only one task at a time. Note that this does not mean that an ELC may not decompose its task into subtasks which may be executed in parallel, only that it is restricted to processing one task from the higher level at a time. At this level, the problem is generally dealt with from the perspective of the operations within the scope of an individual piece of equipment.[12] At the Equipment Level the manufacturing problem has been decomposed into high level items such as loading a part, or moving a part to a vise or low level items such as opening a gripper, moving a robot arm, or manipulating the spindle of a machine tool, depending on the particular ELC. Within this level, there may be other problem decomposition sublevels (and control sublevels) which are concerned with translating high level goals into appropriate machine programming language. The MSI architecture recognizes these lower layers, but does not address them. Appropriate discussions of hierarchies for lower level controllers may be found in [5].

Between the Shop and Equipment control levels, there may be any number of control levels, called ***Workcell Levels***. The problem at the Workcell Level is almost exclusively engineering. There is no overall view of the manufacturing problem at this level. A Workcell Level Controller (WLC) entity coordinates two or more ELCs, WLCs or a combination of either type. There may

11. This observation is also discussed in [8] and [9].
12. The definition of Equipment Level may also include “ill-behaved” systems which are possibly multitasking, but which, because they do not expose subordinates, must be treated as much less capable than they are.

be many levels of these groupings above the individual pieces of equipment. A layer of control of this type may be inserted whenever the need for an additional level of coordination is seen. Empirical evidence for the utility of the workcell type of control may be found in the configuration of existing factories, where equipment is frequently grouped together into functional units called *workstations* or *cells* that are managed by a single controller. The notion of control in this level has similarities both to the AMRF Virtual Cell and Workstation level concepts [19], hence the term *workcell*. In fact, the lowest level workcell, which supervises a collection of equipment, corresponds to the AMRF Workstation level of control. Tasks may execute in parallel at a single workcell and tasks may be queued by the workcell, depending on its capabilities.

3.4.3 Reactive vs. Predictive Control

A control system is characterized by the set of generic tasks it can perform. When active, the intention of the system at any given time is to accomplish some particular task or list of tasks selected from its generic repertoire. The active task list may be constant or parametrized, it may be built into the system software and hardware, or passed in by command from a supervisor or perceived by communication with the real world or other control systems. In a hierarchical control scheme, a controller executes a task by decomposing it (possibly several times) into a set of primitive operations, which are the most elementary functions the controller understands, and then performing each of the primitive operations making up the task, until the whole task is complete. The primitive operations may be performed in sequence or in parallel or in some combination. Additionally, there may be optional operations whose performance is based upon task parameters or current conditions of the system and its environment.

There are two extreme approaches for going about the decomposition of a task:

(1) A ***reactive control*** system finds the best next step toward task completion on the basis of the current state of the world and initiates the corresponding operation. Upon the next significant change in the state of the world, which may be completion of the operation, it initiates the next best step, until the task is accomplished or can no longer be accomplished without external assistance.

(2) A ***predictive control*** system evaluates all known methods and options for accomplishing a task against the current state of the world before starting any action and selects a (thereafter immutable) sequence and schedule of events (a plan) and then blindly executes the plan to completion (or to abort on failure of some step).

The advantage of reactive systems is that they respond rapidly to events and therefore accommodate perturbations. For this reason, they are commonly used in low levels of control where the system only performs one task at a time, the operations to accomplish the task are mostly sequential and easily characterized, and the perturbations are frequent. It can be argued that a reactive system actually has at any given time a plan which is implicit in the choice of best next step, and it has simple plans so it can replan rapidly when there is a perturbation. An important criterion for the use of reactive control is that the best next step for one task will not negatively influence another simultaneous task.

The advantage of predictive systems is that they can efficiently manage multiple simultaneous tasks in which interactions in the use of equipment, space and materials occur. For this reason, they are commonly used in upper levels of control, where multiple simultaneous tasks and operations are the norm, interactions are frequent, and most perturbations can be handled at lower levels. Here the truly best next step toward the accomplishment of any one task is often a function of the forecasted plan for resolving the interactions with other tasks.

In a purely predictive system, certain perturbations can be accommodated by leaving flexible elements in the operating plan. In contrast, the ability to replan whenever a perturbation renders the current plan infeasible can be viewed as a reactive component of the predictive system. Historically, planning systems have never been built to be reactive because the planning problem is difficult to solve and computer resources simply could not perform this function in real time. This has led to the further difficulty that planning systems are built to start from a known state rather than to proceed incrementally from a current state.

The MSI architecture committee recognizes that both approaches are valid, and may be applicable at all levels. However, the percentage of planning performed reactively as a function of control level (and the percentage of planning performed predictively as a function of control level) should be a continuum rather than a bimodal distribution.

In the highest level systems, the primary function is planning. The execution of these plans consists largely of passing the primitive operations of each subplan as tasks to subordinates and monitoring the status of subordinates to determine when to replan. At this level, the knowledge of how to accomplish a generic task must incorporate options for choice of ***resources***, choice and sequence of operations, etc., in order to provide maximal flexibility in the development of an efficient plan.

In intermediate levels, the control systems still typically perform multiple simultaneous tasks and pass on primitive operations to subordinate controllers, but they rarely have much choice of resources and therefore few choices in how to accomplish a task. What they do have is flexibility in the sequence and timing of operations, so that their primary function is scheduling and coordination of the actions of the low level systems. At this level, the integration of the planning process for the timing and sequence of operations with the reactive response to changes in the states of subordinate systems and the environment is critical.

In lower level systems, the primary function is executing a single major operation as a task and the primitive operations are usually implemented by elements of the system itself. Here the dominant concern is reaction: constant but primitive replanning to accommodate perturbations. The replanning consists of adjustment of parameters, addition of specific corrective actions or selection of the proper next action, on the basis of embedded knowledge and sensory input.

3.4.4 Distributed Operations

Two guiding premises of the MSI project dictate support for distributed operation:

(1) an MSI manufacturing activity can be built incrementally from individual systems, each of which controls a small portion of the activities directly; and

(2) an MSI manufacturing activity should be able to incorporate separately developed off-the-shelf commercial or academic control systems.

These premises require the architecture to be modular and to permit the substitution of different implementations of a given conceptual system. To provide sufficient flexibility in choosing systems under requirement (2), it is necessary to accommodate the simultaneous use of different hardware platforms. Such a system is, at least conceptually, distributed. This does not mean that the architecture has to presume distributed systems everywhere, but rather that it has to allow for them.

Much confusion about distributed operation can be avoided if we distinguish the software component from the hardware platform. The software component is the executing program or program entity invocation while the hardware platform is the computer and operating system on which the software component runs. Some hardware platforms can run only one software component at a time. Many software components can run on only one kind of hardware platform and are sold as a software/hardware bundle, but these should be thought of as degenerate cases and not as the archetype. What is distributed in a distributed system is a single (but usually complex) logical function over a set of distinct software components. How those software components are distributed over hardware platforms is a separate consideration, irrelevant to the MSI architecture.

To distribute a function over multiple software components, each component must have a well-defined set of operations it performs and it must have a separate well-defined interface to every other software component with which it must communicate. Such interfaces must be robust enough to permit the communication to be carried by 1) a network of some kind between diverse hardware platforms, or 2) an internal mechanism between software components running on the same physical computer system. Such interfaces should never assume the availability of hidden information transfer paths, including local data repositories, between components.

An architecture must therefore define the distinguished collection of software component types, the set of operations which each software component type performs, the number and types of other software components which each software component may or must communicate with, and the nature of the corresponding interfaces. How control is achieved, whether there is a master component which controls all the other components directly or indirectly, is a principle of the architecture which determines some definitions of the functions and interfaces of the various components.

The notion *software component* can be generalized to *intelligent system*, where an intelligent system can be: *manual*, i.e. human directed and operated; *semi-automatic,* i.e. computer-operated with human direction; or *automatic*, i.e. computer-directed and operated. In the automatic and semi-automatic cases, then, the notion *system* refers to a software component, and in the semi-automatic case, the implemented component contains a direct human interface to provide the intelligent control. A manual system is one or more human beings performing a set of operations similar to those of a software component and having one or more interfaces to other software components. Such interfaces may be keyboard, screen, dials, slides, buttons, etc., or voice or paper communication with other manual systems.

It is possible to replace a collection of architectural systems with a single integrated black box system so long as each interface between a nominal architectural system inside the black box and an architectural system outside the black box is supported by the black box system. In such a case, the internal distribution of software components within the black box and the corresponding internal communication become irrelevant to systems external to the black box. Conversely, a single architectural system may be implemented as a distributed collection of software components, so long as each interface required of the architectural system is supported by some component in the collection. Note that when a single architectural system is physically distributed, each interface must be operated by a single component, i.e. the interface itself cannot be physically distributed. If multiple physical interfaces are required, these necessarily become visible to the system operating the other side of the architecturally defined interface as separate interfaces, thereby violating the architecture. Accordingly, it may be necessary in the architecture to distinguish multiple interfaces between the same two systems, if it is anticipated that implementations of either of them may be distributed in such a way as to require distributing the logical interface.

3.4.5 Provision of Error Recovery Mechanisms

A driving force behind many decisions made in the MSI architecture is the motivation to provide adequate mechanisms for error recovery. The separate hierarchical ***administrative control***[13] and ***task control***[14] provide two distinct error handling paths for distinct error types. When a task cannot be completed, or cannot be completed on schedule, the error is propagated as far up the tasking network as necessary to handle the problem. If, on the other hand, the difficulty is not with a single task, but with an entire controller, the administrative error handling mechanism is used. 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. See figure 5 on page 20. From this interface it is possible either to remove a (dysfunctional) subordinate from the system, or to add a new subordinate. Upon the option of the implementor, it may also 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, or may be an automated expert system, or a human with computer assistance. Of course, in cases where task failure is caused by a controller failure, the error would be reported via both paths. The advantage of the division is seen primarily in cases where occurrence of such errors is not simultaneous. Recovery of the task would then not provoke a controller failure, and a controller failure could be handled in such a way that a task is reassigned and does not fail.

3.5 Process Planning and Production Planning

A useful technique to describe activities within a production shop is to break them into three distinct phases: process planning, production planning (resource allocation and scheduling), and execution. These activities can be performed in a variety of combinations. Traditionally, planning, scheduling and execution are performed sequentially, and further, planning and scheduling are implemented in a centralized fashion. This need not be the case, however, and much research is underway to explore the implications of concurrent, distributed and/or hierarchical implementations of each type of task. In this document, the definitions of process and production planning given in the following sections apply.

3.5.1 Process Planning

Process planning is the activity which produces detailed instructions on how to manufacture a given product, where the product is described by a design. These detailed instructions are commonly referred to as process plans. A process plan typically describes how to make one of a given product. Exceptions would occur in cases where there is an advantage in making several of a product at the same time.

In current practice, process plans are frequently written with a specific production machine in

13. See Section 3.6.1 on page 21.
14. See Section 3.6.2 on page 22.

mind. (This often occurs when the process planner must create parts which require tight tolerances.) The steps in the process plan describe the exact sequence of steps which would be taken to produce the part. The process plans do not provide schedules for the making of the part. The scheduling of the steps in the process plan, and the allocation of resources for the steps of the process plan is performed by a production planner. Note that the production planner may be restricted in his choice of resources (and indirectly of scheduled time) by the fact that the process planner may have written a particular plan for a specific machine. In an integrated system, providing greater flexibility to the production planner will provide an increased ability to handle resource allocation problems that occur at the actual time of production.

Accordingly, our notion of process plan is slightly different from those in use today. In our usage, process plans describe how to manufacture a given product in the abstract sense, much as a cooking recipe describes how to make a given dish. In the MSI architecture, a process plan will typically refer to required resources by class reference, as in "This step requires a three-axis milling machine," rather than "This step requires machine XYZ001."[15] It may describe alternative methods for production, any of which may be selected by the production planner who knows the state of the shop at the time the part is to be manufactured. In a shop with hierarchical control, process plans would be used at many levels of abstraction, from elemental motions of equipment on a shop floor to shop-wide operations.

3.5.2 Production Planning

Production planning concerns the conversion of process plans (generic recipes) into specific instructions to produce instances of a product. All resource references are to actual resources, not classes of resources. It follows, then, that ***production plans*** must also contain scheduling information, since specific resources are called for. Without scheduling information, production plans would be of limited value, since one could never be sure if a referenced resource were available.

3.5.3 Relationships Among Process Planning, Production Planning and Control

The relationships among process planning, production planning and control in the MSI architecture is perhaps best understood by considering the evolution of the production planning concept within the AMRF.

A salient feature of controllers of the early AMRF years was state table programming [20],[21] which selects the correct next operation(s) on the basis of all significant aspects of the current state of the subsystem. Process plans were often implemented as a collection of state transitions. The state table concept, however, also implied that an operation was to be initiated as soon as the required previous process steps had been completed (the associated states had been reached). This meant that all AMRF controllers must be reactive, that is, the only means of determining what should be done next was to assess the current state of the world as perceived by the controller. The formalization of the process plan concept was introduced in the latter years of the AMRF, along with the notion of assigning specific resources to tasks appearing in the process plan. Ultimately, the AMRF even had controllers which directly executed one of two formal process plan representations. The concept of using specific equipment to accomplish specific process steps according to a schedule (production planning in MSI terms), was not supported in the AMRF.

---

15. The process plan model also supports the referencing of resources by capability, and referencing of specific resource instances.

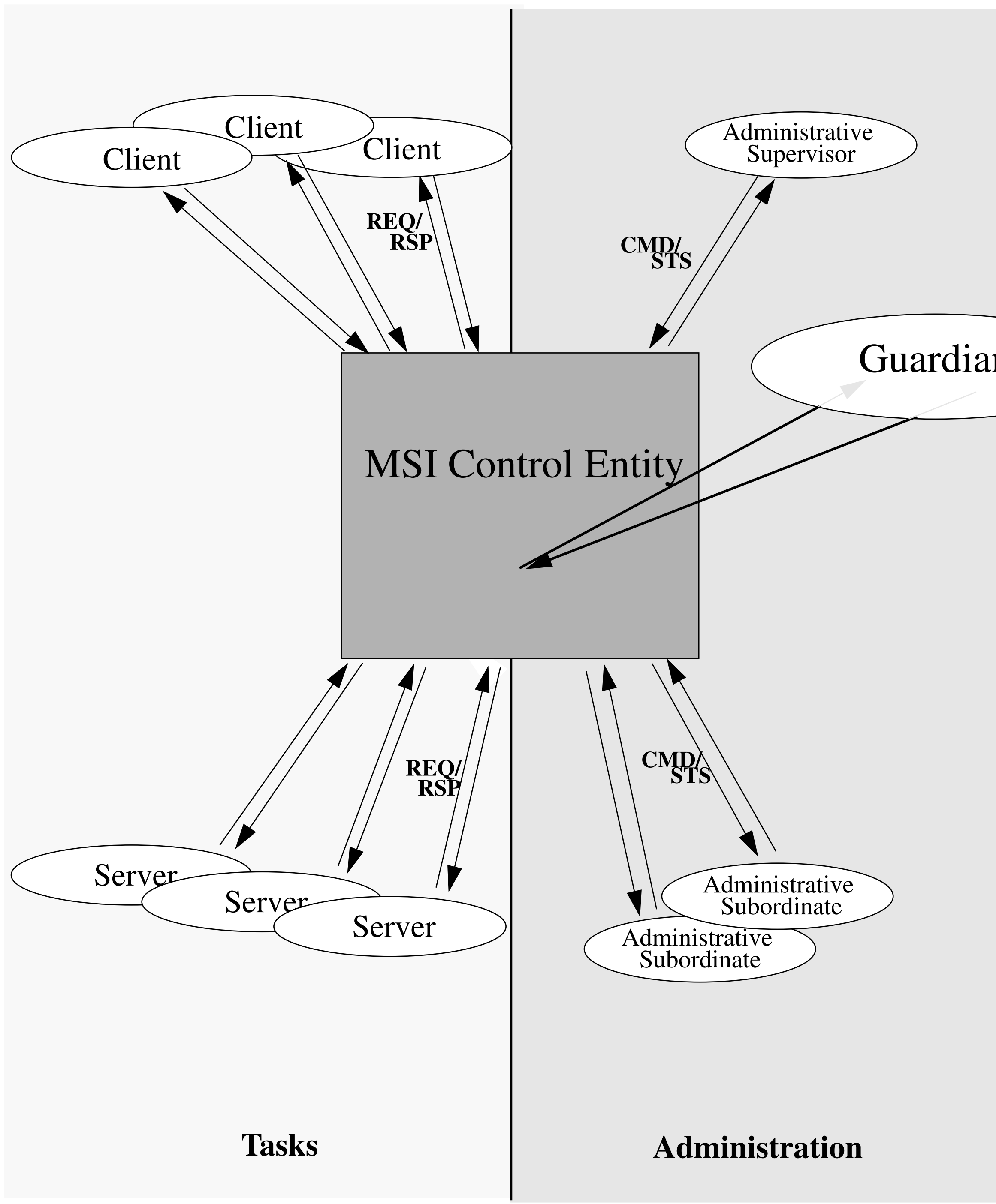


Figure 5 -External Interfaces of an MSI Control Entity

In the MSI view, the process plan is still a critical part of the definition of tasks. Important to the MSI architecture, however, is the concept of schedule – the understanding that the best operation to initiate next is not always the first eligible, but rather the operation which fits into the production plan for optimally accomplishing all outstanding tasks. Therefore in the MSI architecture, the production plan containing the chosen processing options with times and resources to be used, and not the process plan, is the critical element of task definition exchanged between control systems.

It is possible to imagine several automation scenarios where process planning, production planning and task execution interact. Considerations include: where the activities occur, how they are distributed, and when they occur, e.g. sequentially, concurrently, recursively. Below are two specific approaches which the MSI architecture seeks to support. Both assume the generation of process plan instructions for all control levels by a separate process planning system:

(1) These process plan instructions are then used by a separate, centralized production planning system to establish the schedule for the shop. This approach guarantees that requests for workcell resources are known to not conflict before any workcell controller receives them.

(2) These process plan instructions are directly operated upon by a ***hierarchical control system*** where each controller contains embedded production planning capabilities. Here, each controller schedules the use of equipment resources for the tasks it has at any one time, and allocates them accordingly. The global awareness of the total capacity of the workcells resides at the Shop Level. The scheduling system for the Shop Level controls the aggregate load placed on each workcell by the collection of tasks in production at any one time. It also manages the scheduling for the ***material handling*** system which transports materials from one workcell to another. The Shop control of the loading of workcells is reflected in the earliest start times and latest completion times of workcell tasks assigned by the scheduling system of the Shop controller.

Ultimately, it may be desirable to support a third approach as well, with a hierarchical control system containing both embedded process and production planning capabilities. However, this is not within the scope of current MSI work.

3.6 Control System Paradigms and Interfaces

In the MSI architecture, two types of control are recognized: administrative and task control. Administrative control deals with the start-up, shut-down and health of the controllers in the system. Task control deals with the monitoring and execution of tasks given to the controllers. The following sections describe both types of control in greater detail, and discusses the impact of distinquishing two types of control upon the interfaces to the controller.

3.6.1 Administrative Control

To provide reliable channels for directing the start-up and shut-down of controllers, a control hierarchy called the administrative hierarchy is established. In this scheme, 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 controller and all of its administrative subordinates, if any. A diagram of an example administrative control hierarchy is found in figure 6 on page 23.

In the MSI architecture, a controller is aware only of its immediate supervisory and subordinate controllers. In particular, it is not aware of the number and kind of subordinates which its subordinates possess. Configuration information is maintained separately from the hierarchy in a data repository. The administrative hierarchy usually remains fixed while the system is running. If the administrative hierarchy were used to reconfigure the shop, it would be necessary to bring the entire shop down, modify the information base and restart. Alternatively, the hierarchy may be modified dynamically (for example, when a subordinate fails) through a separate interface called the Guardian interface. From the Guardian interface, it is possible to reconfigure by dropping or adding a subordinate, without the requirement of shutting down the entire hierarchy. Guardian interfaces are described in Section 4.8 on page 45.

This dynamic configuration approach differs from that proposed in the initial AMRF control architecture, in which each controller above the Equipment Level was preconfigured to have certain subordinate systems and this preconfiguration was more or less built into the code.[16] This fixed and known hierarchy made it necessary to modify the controller code whenever a subordinate system was removed or a new one was added. Moreover, the state-transition tables selected for the system made it impossible for a Workstation to accept commands until all of its subordinates were ready to accept commands.[17]

Additional flexibility is required to support an operational shop in which failures and reconfigurations may occur. Controllers above the Equipment Level should, in general, have a means of determining what and where their subordinates are, and this determination should be made upon activation of the controller software. If a controller supports multiple kinds of communication paths to subordinates, then the assignment of a particular communication path to a particular subordinate should be part of this initial determination. Thus, code modification should not be necessary to extend the capability to another subordinate. This is particularly necessary at the Shop Level, where the number of instances of a particular kind of subordinate Workcell which are available does not affect the functionality of the Shop controller and should not be embedded in the code. Similarly for complex Workcells, in which not all of the subordinates may be necessary to perform many of the tasks, the Workcell controller should not require the presence of all possible subordinates in order to be ready for manufacturing tasks.

On the other hand, MSI does require that an initial configuration, describing a particular control hierarchy, exist and be accessible to the controllers when they start, and that shop start-up and shutdown (administrative control) must utilize this configuration data in the same manner as the AMRF controllers. (See section 4.2.2 on page 39 for more information on configuration management.) It must also be possible for elements of this configuration to be made unavailable for a particular start-up, and ideally for dynamic changes in the availability of systems to be supported by their supervisory controllers. The distinction between the AMRF view of the control hierarchy and the MSI view is primarily in how much knowledge of the control hierarchy is built-in to the controllers themselves.

3.6.2 Task Control

The primary function of a controller is to perform manufacturing tasks. In this document, a “task”

---

16. The top-level controller was configurable to the degree that its known subsystems could be included or excluded in a given test run which enabled the testbed to function without all of its components.
17. Exact rules for this were defined in the University of Virginia protocol [22].

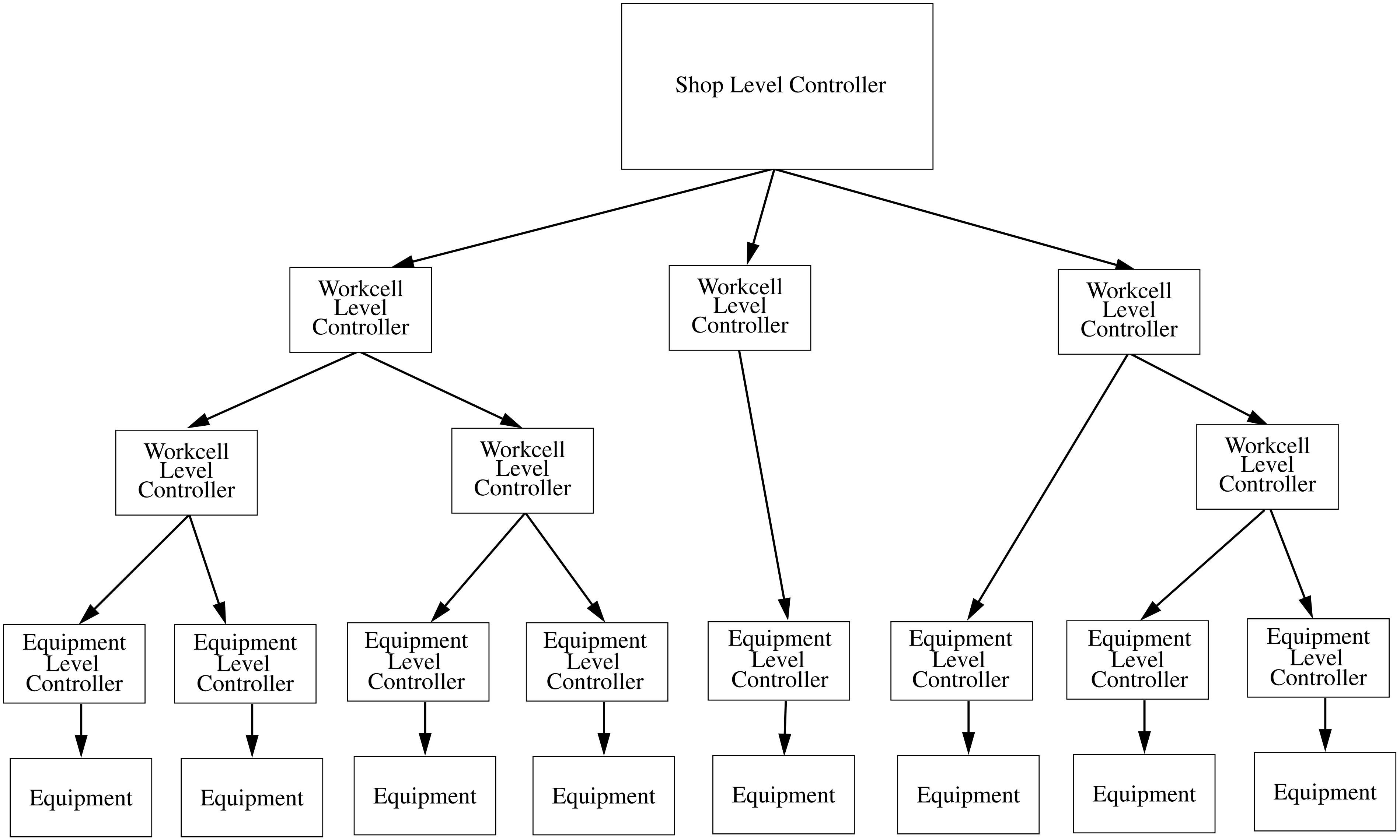


Figure 6 -An Example MSI Administrative Hierarchy

is the action which is requested of the controller; “subtasks” are the units into which the controller decomposes a requested high level task.

In the MSI architecture, controller tasking is based upon a client-server model. When a controller has a task to perform, it decomposes this task into subtasks. This process decreases the complexity of the tasks at each level. The controller performs the subtasks of which it is capable. For subtasks which it cannot execute, the controller, now playing the role of task client, must find an appropriate controller to perform its subtask. The task client would find resource allocation information in the production plan which would identify the correct controller to perform the task. The task client generates task requests for the designated controller, which is now cast in the role of task server. A task server may accept or reject tasks based on its administrative state and its current task capabilities. Figure 7 on page 25 shows the relationships among a control entity, its task decomposition, task client, and task server.

As currently envisioned, tasks will be requested only by control entities in adjacent levels and only from the higher level to the lower. In a typical situation, for example, a WLC would give tasks to an ELC of which it is the administrative supervisor. However, the controller giving a task need not be the administrative supervisor of the controller receiving the task and the possibility of intra-level tasking has explicitly not been ruled out. A snapshot of a sample task hierarchy is shown in figure 8 on page 26. The next figure, figure 9 on page 27, overlays the sample task and administrative hierarchy, illustrating that they need not be identical.

The task interface provides for coordinating the activities of each controller with those of the Shop as a whole. Task requests must be made during the time blocked out for that task by the schedule for the Shop. This prevents the over-loading of resources, because the global scheduler has knowledge of the capacity of each of the resources.

3.6.3 Five-Interface Conceptual Model

In keeping with the administrative control and task management schemes, MSI control entities have a set of external interfaces which conform to a ***five-interface conceptual model***. This model specifies five *types* of interfaces which a control entity may have. The five types of interfaces that a control entity may have are: an interface to a supervisory control entity, an interface to a subordinate control entity, a task server interface, a task client interface and an interface for internal monitoring and intervention (Guardian interface). A diagram of the interfaces of a typical control entity is found in figure 5 on page 20.

The interfaces of an MSI control entity to its supervisor and to its subordinates are administrative control interfaces. Each of these interfaces is bidirectional: each interface can send and receive. An MSI control entity may have at most one supervisory control entity and may have any number of subordinate control entities. An MSI control entity which lacks either a supervisory control entity or subordinate control entities will not exhibit the corresponding type of interface. If an MSI control entity has subordinate control entities it must have one administrative control interface for each subordinate control entity.

The task server interface and the task client interface are task management interfaces. The task server interface is the interface through which a control entity accepts requests from another control entity for a task to be performed and provides status on the performance of that task to the requesting control entity. The task client interface is the interface through which a control entity requests tasks to be performed by another control entity and receives status on the performance of

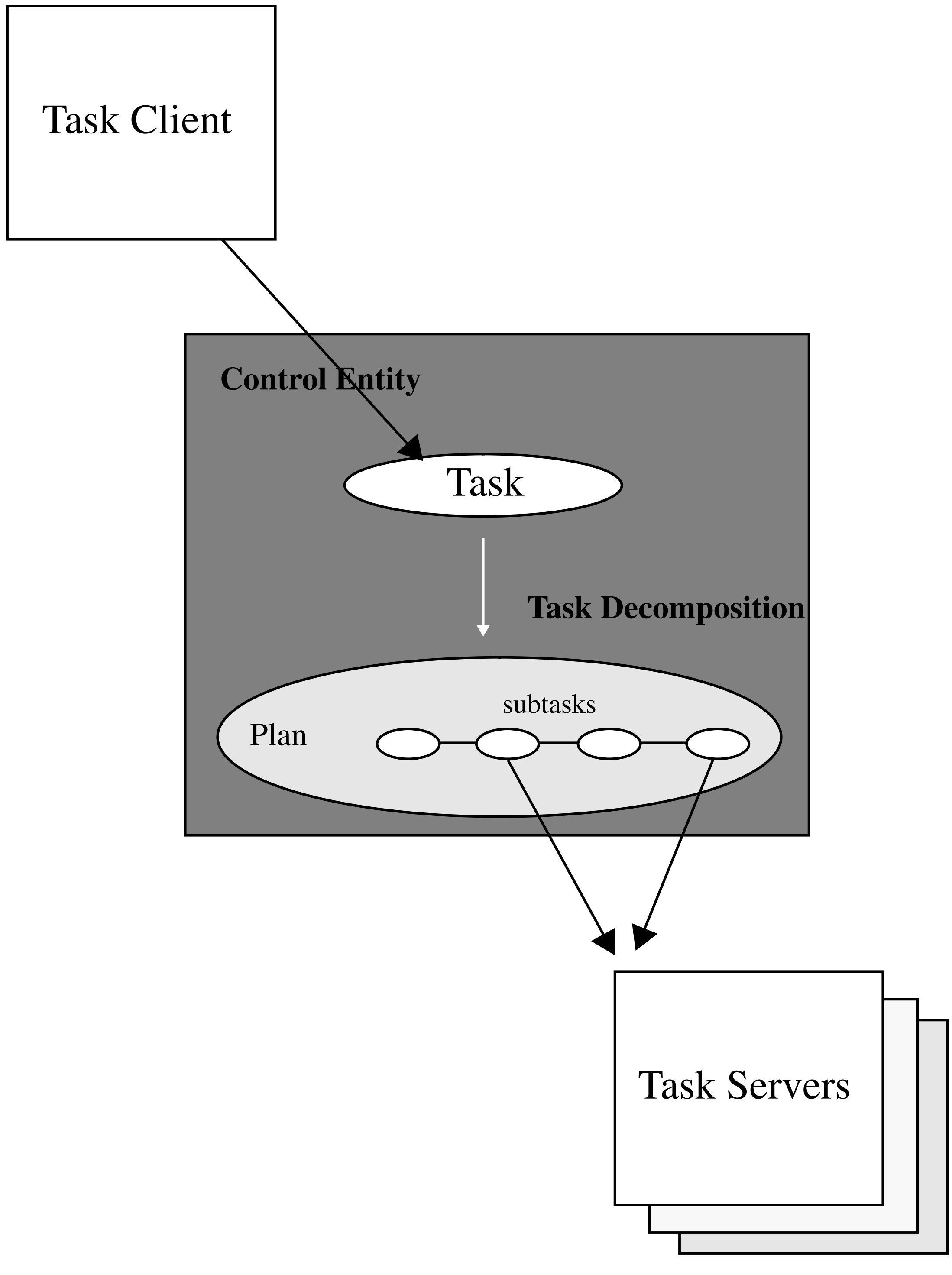


Figure 7 -Task Decomposition and the Task Client/Server Model

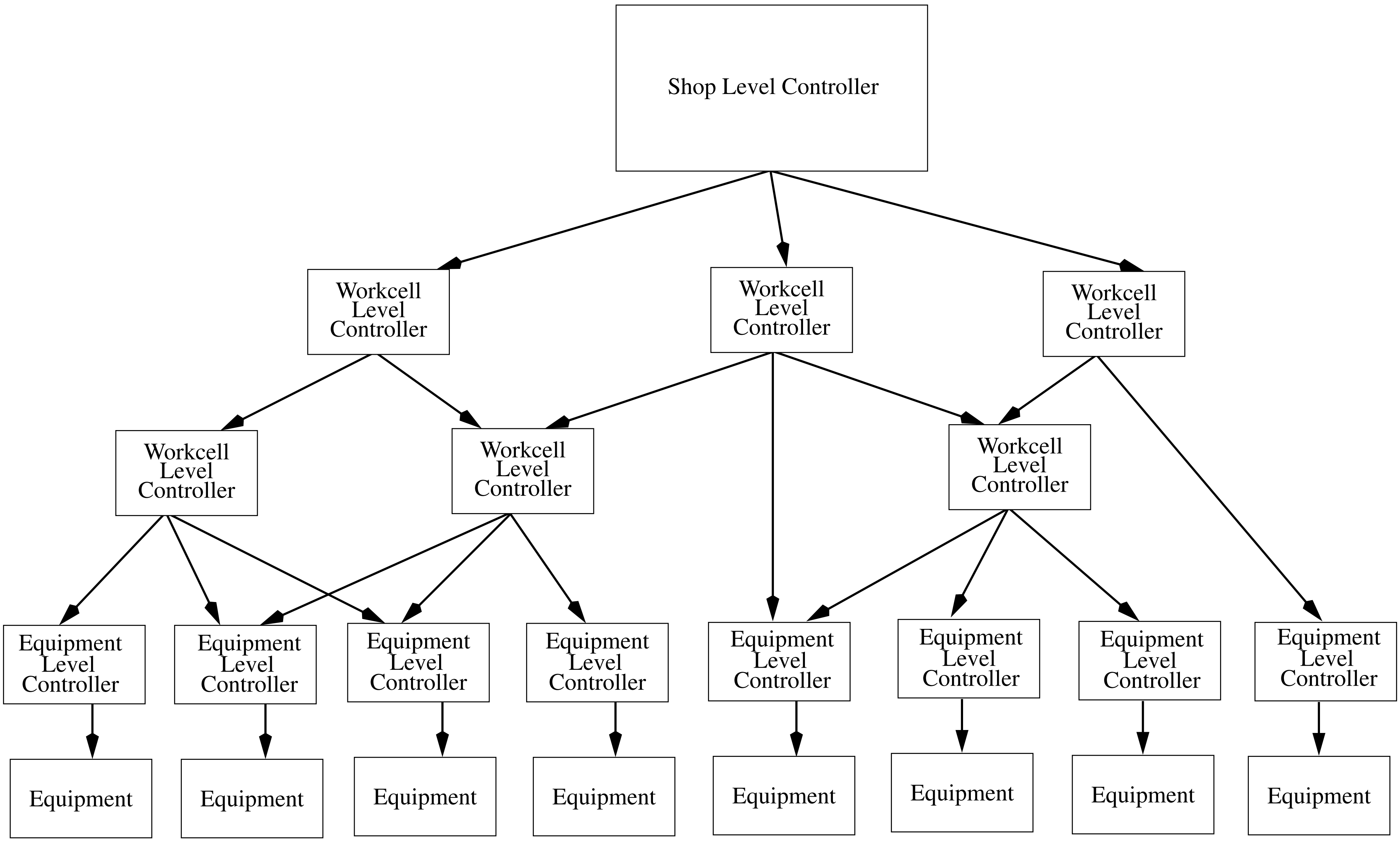


Figure 8 -A *Snapshot* Example MSI Task Hierarchy

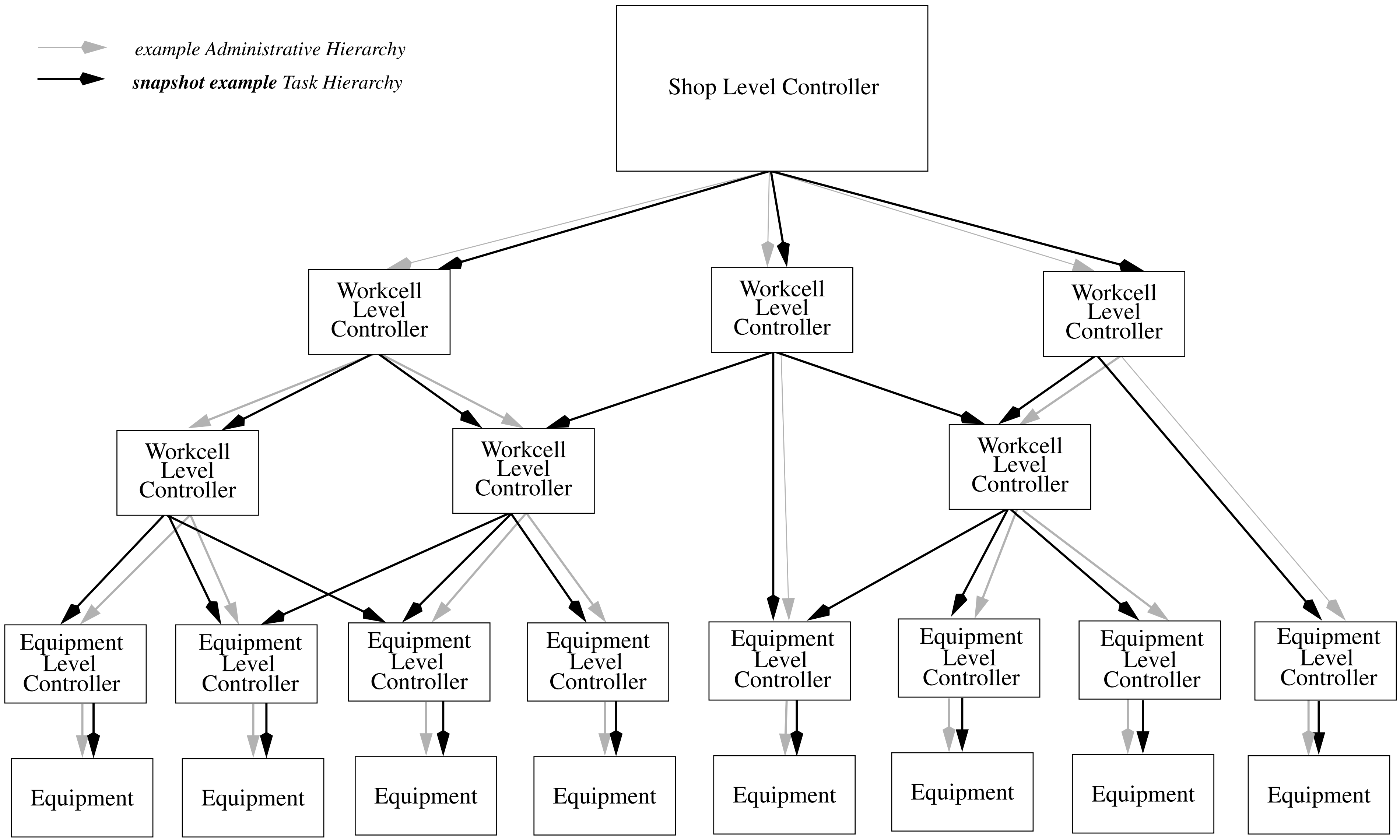


Figure 9 -A Sample Control Hierarchy (with both Task and Administrative Hierarchy)

that task from the control entity which is performing the task.

Although the administrative and task interfaces are not required to be physically separate, they are required to be logically separate. By logically separate, we mean that there is an *a priori* distinction between task and administrative commands.

For more information on the Guardian interface see Section 4.8 on page 45.

3.6.4 Multiple Control Paths

Since the supervisory control entity performs two different functions (administrative control and task control), the MSI architecture committee anticipates that the logical supervisory control entity may in fact be distributed over several distinct hardware and software components. Such a distribution makes it difficult for the supervisor to use a single control (i.e. communications) path as the substrate for all of these logically separate interfaces. The single control path effectively forces the supervisor to contain a common postprocessor for all functions involving a subordinate controller. So in the MSI architecture, each of the logically separate interfaces is permitted to operate on a separate control path.

Controllers which use modern networking software, e.g. MAP controllers, are effectively prepared to support multiple control paths as-built, but it is clear that a shop architecture must support controllers which cannot handle separate control paths. This is seen as one of the elements of the interface required for integration of such controllers.

3.7 Information Exchange

One can think of manufacturing information as being composed of elementary information units. For each information unit there is at least one system in the architecture which is capable of generating that information unit, or obtaining it from sensory or human-interactive interfaces. Such a system is said to be a “producer” of that information. Other architectural systems use the information unit to accomplish their assigned functions. They are said to be “consumers” of that information. When the only consumer of an information unit is the same system as the producer, the information unit is said to be “private” to that system. All other information units must be shared among multiple systems of the architecture. The nature of these exchanges is an important part of the architecture. Specifically, all information which passes between the producer system and a consumer system forms the *interface* between those systems. This section deals with concerns related to interfaces and information exchange specifications.

3.7.1 Interchange Mechanisms

There are three basic methods for exchanging information: direct communication, shared repository, and broadcast. The first two of these are important to MSI.

Direct communication, or *message-passing*, describes any implementation which moves data directly over a dedicated path between two systems. Such interfaces are usually bidirectional, in that interchanges in both directions may use the same path, but for each individual interchange, one system is the sender (producer) and the other is the receiver (consumer). The message-passing method requires that the sender and receiver are aware of each other, and that each interchange involves essentially simultaneous action on both sides.

A *shared repository* is any implementation in which the sender stores the information in a conceptually common physical area and the recipient retrieves the information from the same conceptual area, each in its own time frame. The most common forms of shared repository are files and databases. Unlike the message-passing techniques, the shared repository accommodates fundamental differences in the processing rates of the producer and the consumer systems. It also allows the producer to reach all consumers in a single operation and therefore to be unconcerned as to which and how many systems are actually consumers. On the other hand, it also often involves an intermediary "server" program, which provides and controls access to the actual repository.

An important feature of the AMRF was the use of "common memory" [23] as a means of interchange between architectural components. There were several implementation techniques, but all of them were shared repositories, even though none was a conventional database. The remarkable feature of the "common-memory" was that it was used as a substitute for direct communication in all cases, even command-and-status interchanges. Although the actual intersystem communications involved message-passing mechanisms, it was not necessary for the control entities to be aware of this model, except peripherally [14]. The message-passing operations were encapsulated in the repository services and invisible to the control entities themselves.

In the view of the MSI architecture committee, there are information interchanges which require a shared repository, such as those between engineering systems and production systems, and there are other interchanges which are logically direct from producer to consumer, such as the control-entity interfaces. For the latter class, the method of interchange may be important to performance of implementations, and it is clearly necessary to specify the method in a standard interface, but the method of interchange is not important to the architecture.

3.7.2 Flow of Data vs. Flow of Control

An important feature of the MSI hierarchy, derived from the AMRF, is that the location, organization, and flow of *data* does not necessarily mirror the location, organization and flow of *control*. Properly stated, the flows of data and control are related, but not aligned.

Forcing data flow to follow the control paths is an accident of some older technologies which is no longer appropriate. It carries significant limitations and performance penalties. For example, when shop floor control is a hierarchical system, and engineering and production management information is forced to enter only at the top of the hierarchy, such data must often be handled many times before it finally reaches the lower-level systems which actually use it. Similarly, data which logically flows across the hierarchy, as between production centers and shop-wide material handling systems, would have to flow upward to a common level of supervision and then downward to the "parallel" recipient. Such intermediaries in information transfer are both inefficient and unnecessary.

In the MSI view, the control path should be restricted to commands and the responses to them. The command to perform a task should contain only those information units which are "parameters" of the command itself, known to, and usually produced by, the control entity issuing the command. Other information which may be necessary to the execution of the command, but which is only meaningful to the subordinate system, such as control programs or product data sets, should be obtained by the subordinate directly from whatever shared information repositories contain it. This also applies to current information produced by other control systems which is shared *across*, rather than *along*, the control hierarchy. The shared repository mechanism is preferable to direct communication between control entities *across* the hierarchy, because the producer and consumer systems may be operating on entirely different timing and planning horizons, and direct communication between them is then subject to many constraints.

3.7.3 Universes of Discourse

An important principle of interface design is that systems can communicate only when they have a common *universe of discourse* – a collection of objects, information units, and operations whose use and meaning has been agreed to by all parties to the interface. It is a *universe* in the sense that *every* object or information unit which appears in the interface is part of this agreement.

A universe of discourse can be divided into several different specifications: the *information model*, which identifies the classes of real-world objects involved and the relationships among them, and relates the descriptive information units to the modelled object classes; and the *operational model*, which describes the allowable usages of the objects. A formal *interface specification* further relates operations to the interchange of messages and the production and consumption of information units.

Every system has its own universe of discourse, identifying the objects and information units which that system manipulates. The MSI architecture requires certain systems to support certain functions on certain generic object classes. But many of the objects and information units appearing in the universe of discourse of a given system relate to *how* it performs its function, rather than what functions it performs. Such information units are produced and consumed solely by that system and not properly a part of the architecture. On the other hand, all information units which a system uses, except for those which it generates by itself, must be part of the universe of discourse for at least one of its external interfaces. The universe of discourse (objects, information units, and operations) of every interface required by the architecture *is* a part of the architecture. Moreover, conceptually there must be a single "global" universe of discourse, which relates all of the objects, information units and operations specified by the architecture. From the architectural point-of-view, the universe of discourse of each system and interface is then a proper subset of this global universe of discourse.

Because the MSI architecture applies to multiple manufacturing domains, the object types, operations and information units it specifies will necessarily be generic. Ultimately, any instantiation of the MSI architecture for a particular manufacturing domain will require more specific object types of the generic kinds, and additional specific object types and information units, in order to define the complete universe of discourse for that application.

3.7.4 Views

The actual use of the common universe of discourse in a physical interface requires that the information requirements for specific tasks, and the organization and representation of the shared information, be specified. Formal *views*, subsets of the information model specifying the organization and representation of the shared information, are created in three places: in direct communications between control entities, in data repositories, and in control-entity access to repositories. These additional specifications are necessary to implementations and to formal interface standards, but they go beyond the concerns of the MSI architecture.

When the interchange mechanism is direct communication, both parties must agree to all details of the views involved. In order to minimize the combinatorial explosion resulting from pairwise agreements, standards for "open" interfaces, which completely specify the views, are valuable.

When the interchanges use a shared repository mechanism, multiple views for the same logical interface are possible. The repository itself defines the "conceptual" and "internal" views in which the information is logically and physically organized in the repository system. Since the same information may be part of multiple interfaces, the repository view may take into account relationships arising in the global universe of discourse, resulting in organization of the information units which is optimal for some interfaces and not for others. A specific control entity may then define a reorganization of the information units, called an "external view," for its accesses to the repository which are related to a particular architectural interface.

In the AMRF, in addition to standard interfaces, canonical external views of all repository information units were developed, based on a perception of the common information needs of all controllers performing similar tasks. Like standard interfaces, this approach greatly simplified the specification problem. But the standard views could not be the most useful representation of information for most control programs, or the AMRF data systems, because all information had to appear in a form which could be manipulated directly by *all* AMRF controllers, regardless of implementation language or platform. Therefore, for interchanges via shared repository, a common universe of discourse is necessary, but canonical views are not necessary and may not be desirable. External views, consistent with the shared universe of discourse, should be defined by each specific control system in its interface to the repositories.

3.7.5 Shared Information Repositories

The information model is a kind of abstract organizing plan for manufacturing information: the facts which are true of all objects of each kind, such as "Every workpiece has a location." The information itself, the facts which vary from object to object, such as "Workpiece xxx is in location yyy," constitutes the "information base". Note that every information base has a corresponding information model. The shared information repositories are the software/hardware components which maintain segments of the information base.

While the global information model must integrate aspects of many objects related to several manufacturing functions, ideally without operational bias, repositories tend to be organized to support particular functions optimally. Consequently, mapping from MSI information models to the views possessed by actual data repositories may be quite complex. Information about a particular type of object may be implemented in more than one repository, and one repository may contain fragments of the information about several kinds of objects. Some repositories will contain some of the information about all of the objects of a given type, and others will contain all of the information about only some of the objects of a given type. One should also expect to find repositories which maintain information which is private to a given system coupled with information which is shared.

What is important is that

- any information unit which is to be exchanged via shared repository is maintained in one or more repositories,
- any given information unit is accessible to all the systems which use it,
- all of the repositories and systems support the relationships modelled in the universe of discourse, and
- information which is maintained in multiple repositories is consistent.

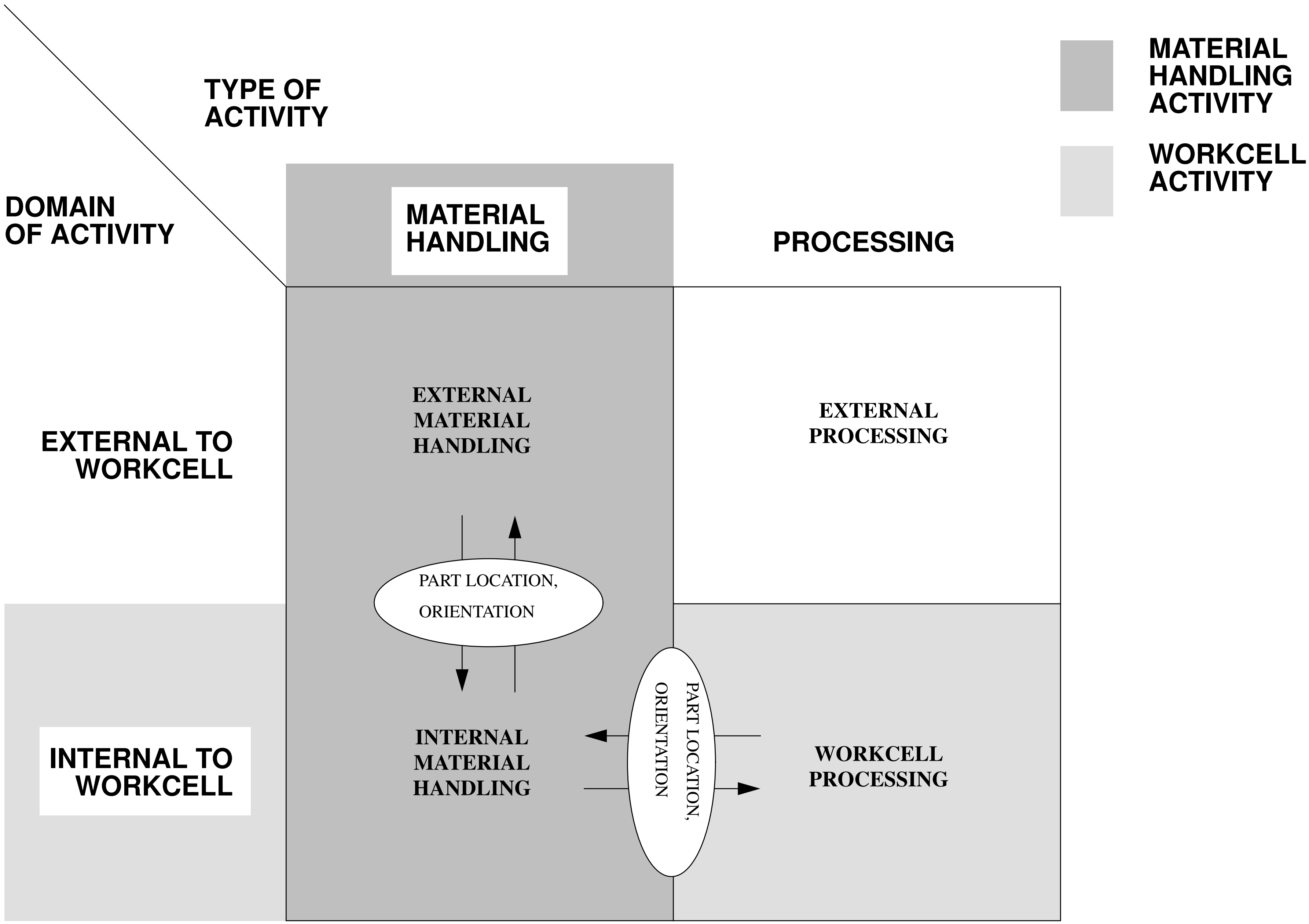


Figure 10 -Material Handling & Processing Operations

3.7.6 Local and Global Data Systems

In order to integrate independently developed control systems, and to provide efficient data services for real-time systems, it is necessary to permit control systems to embed local repository systems for locally produced information, and to permit such systems to maintain local versions of non-locally produced information.

In order to use externally developed data, stored in data systems of the developers' choice, it is desirable to provide individual control systems with a common means of access to all non-local repositories. Control systems can then be "designed for integration" by using the common access method for non-locally-produced information. Repositories (including embedded ones) can be designed for integration by providing support for the common access method for externally requested operations on the stored information. And intelligent systems can be developed to map control system information operations onto the appropriate repositories.

When a system makes extensive and frequent use of non-locally produced information, it may create a "shadow database" – a local version of non-locally produced information. In addition to performance concerns, the black-box nature of an imported system may require this. It is important to realize that when a shadow database is created, the responsibility for maintaining consistency with the shared repositories (the true source of the information) is placed on the control system and not on the data systems. The AMRF control systems universally proved unable to accept this responsibility and act on it properly, with numerous resulting test failures and system incompatibilities. Maintaining consistency between local and global versions of data remains a concern in the MSI architecture and further research will be required to resolve this issue.

3.8 Material Handling

A process plan describes one or more sequences of manufacturing tasks, each of which would produce a given product. Within these plans, material handling tasks must be treated differently from the other steps in the plan. During the generation of process plans, it is not known what specific machines and other resources will be used; resources are only referenced by class. Thus it is impossible to anticipate exactly what type of material handling operations may be required. A transfer from one manufacturing stage to the next could imply a simple robotic movement, or the transportation of material from one end of a shop to the other. Indeed, it is conceivable that two consecutive machining setups for a given product be performed either on the same machine (for small batch production) or on different machines (for large quantities if refixturing is expensive). The decision whether to use the same or different machines for these two setups is a production management one, not a process planning one. One result of this is that a process planner cannot estimate task durations for material handling tasks, although this is expected for all other manufacturing steps.

Another unique aspect of material handling tasks is that these tasks couple otherwise independent activities on a shop floor. Material handling systems are highly shared systems typically servicing transportation needs for many different workpieces simultaneously. This poses an interesting challenge to scheduling. When considering the scheduling of activities in a shop, one must ultimately decide to what degree it is appropriate to schedule. Clearly, it is not realistic to schedule every elemental movement of shop floor equipment, since the slightest unexpected event will immediately render the schedule obsolete. At the other extreme, if nothing is scheduled, one is left with an entirely reactive shop, making it vulnerable to wasteful and inefficient decisions due to the lack of foresight. At some point between these extremes lies a point where one can usefully draw the line between realistic expectations of events and unpredictability. Because the sequencing of material handling tasks depends on many otherwise uncoupled manufacturing sequences, it seems appropriate to handle material handling operations in a reactive manner. This does not imply that the durations of material handling activities should not be estimated during predictive scheduling, but only that the sequence of events, and manner in which the material handling request is serviced be determined at execution time.

In addition, the MSI architecture committee differentiates two types of material handling: that which is internal to a workcell and that which is external to the workcell and provides transport between workcells. An example of material handling which is internal to a workstation occurs when the workstation has internal storage tables: moving the materials from storage within the workstation vis a robot or conveyor belt included in the workstation is internal material handling. Moving material from one workstation to another on a conveyor belt which is not internal to any one of the workstations constitutes external material handling. The MSI model of these interfaces is shown in figure 10 on page 32. This distinction is important in allowing the process and production plan to permit flexibility in the internal material handling for intelligent workcells.

3.9 Human Interface Concepts

It is clear that people will continue to be involved in various aspects of automated manufacturing. In present day facilities, people often manually operate equipment. In the future, people will become less involved with manual operations and increasingly involved in supervisory operations instead. The MSI architecture recognizes that supervisory intervention may be required in any system. Accordingly, it defines mechanisms whereby people can be incorporated into the manufacturing system while maintaining the advantages of an integrated automated system. The two mechanisms are:

(1) An interface which decouples the operator from the process (or data) and hardware being observed and/or affected,

(2) A translator which converts between machine and human representations of data and events.

The decoupling demands that the associated data/process be physically independent from the human interface where possible. As a consequence, the associated process must be able to function without the human interface being attached.[18]

The principle of separating the interface from the underlying process with which it communicates was formulated in early AMRF design documents [24], but this principle was not generally applied in AMRF controller implementation. The resultant non-uniform approach to building console interfaces created AMRF console interfaces with diverse characteristics. These console interfaces were not portable. Consequently, it required a great number of people to run the testbed, and it was difficult for persons not involved in the development of the console interface to use the system. In order to avoid these difficulties, the MSI architecture mandates that console interfaces

18. It is recognized that there will be certain manufacturing activities which will require a human operator to be physically a part. In such cases, care should be taken both to minimize the control elements requiring physical human interaction and to contain such an operator interface within a given MSI architectural entity (e.g. an ELC).

be decoupled from any associated processes and be built according to the Guardian interface functional specification.[19]

Because MSI interfaces are physically independent from the machine on which they run and the process to which they may connect, fewer users are required to run an MSI testbed system and more efficient and effective debugging is permitted by allowing one user to inspect multiple virtual front panels at once. Moreover, if it is actually necessary for an operator to be located at a physical piece of equipment, other elements of the system can still be run and monitored by that person from that particular piece of equipment. Since access is made generally available to persons who are not designers of the system, these intermediary translator processes are less likely to have personalized elements that make them difficult to use.

Translation of data from machine readable forms to human readable forms and the reverse are minimum requirements to any automated system. With bitmapped graphics and window-based terminals it is possible to integrate the display of data or system states with the commands which affect those data or states. Using these tools, data can be represented in forms which are easier for humans to process.

This translator function has been likened to an interface one might build to integrate dissimilar systems. The difference between a translator and such an interface is that where such an interface is usually employed to add unplanned elements to a system, the translator services a common one, humans.

As a final note, the decoupling mechanism also allows for a smooth conversion from human components to automated replacement components as they are developed. This is because the interface permits a computer-based entity to be substituted for the translator without altering the manufacturing application and without forcing the computer based entity to understand human data forms.

19. See Section 4.8 on page 45 for more information on this interface and its relationship to other interfaces in the MSI architecture.

4. Architectural Components

The following section of the document describes the detailed implementation-independent MSI architecture. Functionally related components are grouped into sections and each component is discussed separately. Support services of a global nature are discussed in a separate section.

4.1 Shared Information for MSI Systems

As remarked earlier in this document, adequate data representation is seen as one of the key issues in system integration. In this section, information which is necessary in the integrated environment is discussed. Engineering data (produced by engineering systems) are of four basic types:

- Specifications of the part to be produced (shape, material, function, etc.),
- Specifications of the processes by which a part is produced,
- Specifications of the resources needed to produce the part, and
- Specifications of the capabilities of those manufacturing resources.

Production data add the following elements:

- Identification of individual parts, tools and materials, and their particular attributes during and after each manufacturing step,
- Configuration and state of the shop floor equipment,
- Production process specifications with schedules and limitations.

To integrate a collection of systems, the information which each of the systems in the collection requires must be included in the global information base.

An information model which organizes the global information base is under development. This model describes the relationships among process planning, work in-process, inventory, production planning, orders and material handling. Information models for the information required by each system in the MSI architecture are in various stages of development.

The following sections detail the *information* which is used by the systems in the MSI architecture: the information models, the related information bases, the information or any combination thereof are discussed.

4.1.1 Product Information

The ***Product model*** used within the MSI project will ultimately include: a true solid model of a part to be manufactured, a feature-based representation layered on top of the geometric part description, a digital representation of all tolerancing information, and information indicating the spatial relation of a part to other components to form an assembly.

The MSI testbed will use a part model which includes both geometrical and feature information. The full integration of this information is not within the scope of the project. However, because the manufacturing features of the part and part geometry may be needed by process planning and production planning, this information will be produced and made available to systems in the testbed.

4.1.2 Process Plan Information

Process plans play a central role in the integration process, and form one of the key vehicles by which information is shared among MSI systems. Because of this key role, the process plan representation must exhibit the following capabilities: explicit parallel and alternate sequences, multi-job synchronization, ***hierarchical task decomposition***, resource management primitives, and user extensibility.

The steps by which parts can be made are expressed in a process plan, which gives both the sequence(s) of operations by which the part is made and relative timing of these operations. The process plan expresses both alternative and parallel paths to part production. The process plan formulation envisioned in the MSI project is hierarchical; therefore the plan must also express the synchronization of the lower level plans.

In the MSI architecture, the hierarchical organization of process plans coincides with the hierarchy of control entities in the MSI architecture. For each part design, process plans must be constructed for each level of the manufacturing hierarchy. Although process plans contain similar structure at all levels, distinct types of operations are performed at each level which are unique to that level. Listed below are descriptions of the operational characteristics of process plans at levels pertinent to the MSI architecture.

(1) Shop Level.
At the Shop Level, process plans primarily address the movement of workpieces and sequencing of different types of machining operations, such as turning, milling, etching, etc.

(2) Workcell Levels.
Workcell Level process plans prescribe the coordination of controllers subordinate to a given workcell, such as the use of a robot to load a machine tool bed. Such plans can require extensive use of synchronization between process plans for subordinate controllers.

(3) Equipment Level.
Equipment Level plans describe the most detailed level of operation that a process planner would generate. In this case, the activity called process planning in MSI terminology may overlap with what is usually termed off-line programming in more general use.The steps within such a plan provide instructions which are carried out by individual pieces of equipment. Any coordination with other ELCs is specified in higher level plans. Example operations within the domain of metal cutting might include steps such as *drill hole* or *chamfer edge*. The degree of detail required in such a step depends on the capability of the controller. If the controller possesses dynamic equipment programming capabilities, higher level instructions such as those above, or even as abstract as *load part* and *fixture part*, might be sufficient. If the controller is less capable, instructions to the level of NC code may be required.

As yet, there is no standard representation for process plan information. Hence, the MSI architecture committee will continue in-house work on representations of process plans. This work solely addresses discrete part processes. Continuous process representation, such as in chemical manufacturing processes, does not fall within MSI's scope.

The MSI architecture committee regards process representation as a generic aspect of manufacturing. As in the AMRF, domain-specific descriptions of activity in processes will be represented as "work-elements" at each level of the hierarchy. A ***work-element*** is a description of a discrete activity for some class of controllers in a manufacturing environment [25]. It includes references to all supporting resources and data required for execution of the described activity. The implementation of work-elements is not fixed: work-elements may be definitions in databases or pieces of code.

The structure into which these work-elements must fit is specified in ALPS (A Language for Process Specification) [26]. An ALPS process plan describes a partial ordering for the elements to be executed, whether elements can be performed in series or in parallel and other attributes related to the relationships between the work-elements. Plans may have precedence and coordination requirements between work-elements in a single plan and between elements of related plans. A plan may also contain alternative work-elements or sequences of work-elements. Alternatives offered may represent differing shop configurations, resource availabilities, resource utilizations, materials, qualities and costs. ALPS is, in part, a grammar. Just as in English, sentence structure is independent of the vocabulary used, so in the manufacturing domain the manner in which discrete process elements are combined is independent from the process elements. Using this approach, structure of the generic elements can be viewed clearly without the distraction of domain-specific detail. Moreover, the domain-specific items are independent of the framework and may be replaced or rewritten at will, so long as certain interfaces remain compatible with the overall architecture.

Naturally, developing standards in this area will be closely monitored, and input will be provided to standardization efforts as appropriate.

4.1.3 Facility and Network Information

To support the MSI project, a ***Facility Model*** which characterizes both the form and function of resources available in the shop is necessary. The Configuration Manager creates the facility information base at run-time using this model. This information base contains not only information on shop floor configuration, but status information on shop systems. In this way, it provides a permanent repository for control information needed by the controller at the Shop Level when the shop is in continuous production.

The Network Entity Model, which is closely related to if not part of the Facility Model, describes the MSI domain: the computer systems and software processes in the shop, their network identification, current activation state, authorized and active links, etc.

4.1.4 Order Information

The orders information base maintains the collection of orders for parts including the type, quantity, engineering status and production status of the orders. Engineering status includes items such as availability of design data, process plans, control programs, etc. Production status consists of such items as the relationship to workpieces, ***jobs***, and completed inventory.

4.1.5 Other Information

Several additional models will eventually be included in the MSI project: tooling, materials, inventory and work in-process.

- The Tooling Model describes the type, characteristics, quantity, location, setting and life data for cutters, tool-holders, end-effectors, fixtures and carriers available and in use in the shop.
- The Materials Model describes the type and characteristics of raw materials and con-

sumable machining supplies.

- The Inventory Model describes the type, quantity and location of available stock consumable machining supplies and completed parts no longer in process.
- The Work In-Process Model describes the state and location of workpieces (blanks, partially completed pieces and finished parts) assigned to active jobs, lots, carriers (trays, pallets, etc.), kits, etc.

4.2 Configuration Management Systems

The following stand-alone processes provide for the maintenance of the MSI facility information base.

4.2.1 Facility Definition System (Shop Floor Configuration Manager)

This system provides for human and automated creation and maintenance of the facility information base, seen as a collection of fixed assets with detailed descriptions: controllers, equipment, capabilities, data systems, fixed control and communication paths.

4.2.2 (Baseline) Configuration Manager

This system provides for human-interactive selection of elements of the facility information base to describe an active configuration, for the purposes of production planning, controller instantiation, and communication path construction. It generates the ***baseline configuration*** which shop-floor planning and events will modify, and supports voluntary reconfiguration, i.e. human-interactive, changes to the available resources while the MSI production systems are running. As part of the human interface, the Configuration Manager provides on-line graphical displays of the active configuration, capabilities, links, etc.

The Configuration Manager alerts the Shop Level control entity to the need for replanning and/or rescheduling as a consequence of voluntary reconfigurations, and detects changes in the configuration initiated by the shop floor for incorporation into the displays used in product engineering and may be used in planning and estimating in Shop Level planning.

4.3 Support Services

The following services are provided by some distributed means (possibly libraries of software routines, data repositories or some combination of both) to support either the dynamic instantiation of control processes and communication paths or data manipulation. Figure 11 on page 40 summarizes the support services provided in the MSI architecture.

4.3.1 Single Point Start-up

Single Point Start-up is a service provided to the Configuration Manager that allows for automated creation and initiation of instances of pure software processes[20] on any of a collection of selected hosts on the MSI network.

20. Pure, in this context, is intended to mean processes which have no direct control over physical devices which would necessarily require manual involvement.

4.3.2 Data Management Services

Two kinds of data management services are used:

(1) Local data repository systems used in engineering and production planning systems to which the data organization and access is tightly coupled, to achieve satisfactory performance of the particular system.

(2) A distributed data management system to allow access, in a uniform way, from any system to any data repositories not locally maintained, including combining information from multiple data repositories. In this manner, externally developed database implementations of STEP models may be incorporated.

4.3.3 Communication Services

The services necessary for MSI are link establishment, data movement, link monitoring and directory services in support of link management.

Dynamic Link Management is a service provided to shop-floor control systems to authorize, verify and implement creation and destruction of control paths, data service paths, and related communication paths. Authorizations are hierarchically controlled, originating from the Configuration Manager, the Shop Level Planner [21]and the Workcell Level Planners.[22] Requests may originate from any controller which has dynamically assigned servers and subordinates.

4.4 Engineering Systems

While the engineering processes themselves are not within the scope of the MSI project, the information they produce is vital to the success of the MSI project.

4.4.1 Product Modeling

A product model is a description of the properties of a product, which includes the product's definition (design) and may optionally include its behavior, usage, maintenance and disposal.Various

Single Point Start-up	Automated creation and initiation of instances of pure software processes on any of a collection of selected hosts on the MSI network.
Data Management Services	1) Local data repository systems are used by systems to which the data system and access is tightly coupled. 2) A distributed data management system provides uniform access from any system to any data repository that is not locally resident.
Communication Services	Timely communications support, including link establishment, data movement, and link monitoring.

Figure 11 -Support Services in the MSI Architecture

21. See Section 2.1 on page 54.
22. See Section 3.1 on page 56.

design systems are used to produce the necessary elements of the product model. It is assumed that all incoming designs are complete and satisfy the original functional specifications for the part. All design information for parts (geometry, topology, tolerance, feature information) is imported into the MSI testbed in STEP format.

4.4.2 Process Planning

In the MSI architecture, it is assumed that, for any part which is to be made, relevant process plans exist for all control systems and functions involved. The process planners, therefore, are expected to provide not only machining plans, but also kitting, handling, fixturing, finishing, assembly and inspection plans, as necessary. As received from the engineering systems, process plans should contain a number of cost effective alternatives which take into account the resources of the local production environment, but not the status of such resources. Choosing from among these alternatives by considering scheduled availability of resources and the overall production load is the responsibility of the production planner.

The notion of feeding production considerations back into the design, which is an aspect of the concurrent engineering concept, is not within the scope of the MSI project as currently defined. However, the MSI architecture is consistent with this concept. The production information models are public and the production data repositories are expected to provide general access services, thus enabling such feedback. The engineering systems are encouraged to produce multiple cost effective plans for producing the parts, while the selection from among such alternatives is an express part of MSI production planning.

4.5 Production Planning

With the introduction of production planning into the system, the representation of scheduling information becomes critical. While schedules are often thought of independently of the originating process plan, it is important to realize that a schedule is constructed with reference to a process plan. For this reason, the scheduling and process planning representations are logically linked. Whether the scheduling information is stored with the process plans or separately, is an implementation issue, having no bearing on the architecture of the system. In the MSI project, ALPS has been extended to include the scheduling information and to support the needs of the production planning system.

4.5.1 Error Handling in Production Plans

MSI requires that controllers 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 more meaningful 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.[23] All checkpoints are identified as such in the production plan.

4.5.2 Resource Allocation

Production planning includes resource allocation, which is the selection of the equipment which

23. See Appendix 2 for limitations.

will be used to perform tasks. Resource allocation is said to be exclusive if a resource can be allocated to only one task at a time. The MSI architecture allows allocation to be nonexclusive.

In the MSI method of production planning, the production planner is given a process plan which may contain several optional methods and sequences for making a given part. The production planning system must, before beginning the scheduling of a plan, choose a method of production and assign resources appropriate for that method. This eliminates most of the alternative and optional steps in the production plan. The remaining options in the production plan are resolved by the controller at the time it executes the production plan, based upon the current data available. In order to choose the correct option, one might need information available only at run time. Sensor feedback is one example of information which cannot be obtained at scheduling time, yet may be required to choose the correct production sequence. If the controller is unable to obtain or evaluate the necessary information, a default decision is made. The possibility of alternatives within the scheduled production plan clearly creates difficulties in the scheduling process, since the plan is no longer deterministic. The only option to deal with this is to use estimates of total duration during scheduling, possibly based upon historical or stochastic data.

Usually there are simultaneous tasks at each control level, therefore care must be taken in such assignments to maintain consistency with the overall production plan for all tasks at that level. Note that the MSI approach differs from conventional production planning practice, where the production planner receives a single plan from the process planner which contains no alternative methods, and the controller receives a scheduled plan which also contains no alternative methods. Under the MSI approach, both the scheduler and the controller can expect to receive plans containing alternatives.

The steps in a production plan consist of tasks for the controller to perform. The controller may be instructed to perform each of these tasks by performing subtasks whose cumulative result is the performance of the original task.[24] In the MSI architecture, efficient use of resources is achieved by allocation of resources only during the period which they are actually being used. Such resource allocation operations may appear as steps in the plan for the task. The resource is exclusively allocated to the task, but not for its entire duration.

A system which cannot execute simultaneous tasks is called sequentially shareable. Such a resource must be allocated as a unit object, and its internal planning is reduced to assigning its component resources to each task as it receives it. When a system can execute simultaneous tasks it is more than sequentially sharable, it is simultaneously shareable. It should not be exclusively allocated, for if it is, the benefit derived from its simultaneous shareability is completely lost.

When a resource, such as Material Handling, can execute simultaneous tasks it is important to realize that such a resource has limited total capacity and the use of that capacity must be globally planned to some degree, in order to insure that the set of ongoing production plans is feasible with respect to schedule.

4.5.3 Resource Sharing

Beyond the flexible assignment of resources to tasks (the dynamic tree structure) is the concept of permitting the simultaneous sharing of a resource by two distinct tasks, and in particular by two distinct supervisors. The problem of control of sharable resources has appeared in the AMRF sev-

24. See figure 7 on page 25 for a diagram of task decomposition.

eral times, notably the Horizontal Material Buffer Controller [27], the use of Horizontal Work Station Vision [28] to perform tray identification, and the Material Handling Station in general [29]. Various solutions to the problem of control of sharable resources have been tried, including the specific creation of two command sources with a resource control protocol and the development of client/server protocols. The notion of shared resources is also supported by the Semaphore Management protocol elements of the ISO Manufacturing Messaging Services [12]. In the interest of accommodating the largest number of current and future control paradigms, therefore, there is a need to allow for simultaneously shared control of certain systems, which gives rise to a control structure which is a lattice instead of a tree. This concept has important consequences for the relationship of control paths to tasks (because a sharable system can have multiple simultaneous tasks,) and for the scheduling of simultaneously sharable resources.

It is possible, within the client-server tasking mechanism, for two simultaneous subtasks of a single plan to contend for use of the same resource. Depending on the relationship between tasks and plans, this situation would not necessarily be resolved by the high level planner. For example, consider the situation in which two or more logically parallel work-elements of a single plan involve the same subordinate. When the subordinate is a Workcell, for example, the two tasks may actually involve quite distinct equipment and therefore be executable in parallel.

The MSI architecture must tolerate two cases:

(1) The subordinate system is only capable of one task at a time, in which case the supervisor must determine the actual order of the parallel tasks and maintain a queue of one-at-a-time tasks for the subordinate, or

(2) The subordinate system is capable of multiple simultaneous tasks, in which case the supervisor can promulgate tasks to the subordinate whenever they arise, and except for control parameters imposed by the supervisor, they can be completed in any order.

It is unnecessary to assume that the choice among these two task queuing paradigms is fixed for a given level of the hierarchy, since that would restrict the importation of externally developed subsystems into an integrated system conforming to the MSI architecture.

4.6 Order Entry System

Orders are management data which drive the manufacturing operation. An order entry interface would logically be a subset of an administrative system interface. It would allow managers to enter orders, check progress of orders, and query for shop operation costs and utilization.

The order entry system for the MSI testbed comprises a human-interactive interface by which product orders can be entered, examined, modified, prioritized and deleted or deferred. The products of the order entry system are entries in an orders information base, which contains the orders pending for the shop which the controller at the Shop Level services. This information base is used by the systems which perform process and production planning for the Shop Level to determine the production requirements, i.e. the production job list, and annotated by the production planning system to show placement on the production schedule and completion.

4.7 Controllers

The MSI architecture provides a clear interface specification for ***foreign controllers*** to be plugged into an integrated system which conforms to the architecture. A foreign controller may function at any level of control, depending on its capabilities and compatibility with the architecture. Native MSI controllers are discussed in Appendix 2 on page 60.

In the MSI architecture, controllers are classified into one of three categories, based upon the controller's ability to queue and schedule tasks. The categories are:

(1) The controller accepts only one task at a time, and initiates the task as soon after receipt as possible.

(2) The controller accepts and queues tasks as received, and initiates tasks according to some internal method. The subsystem does not support external constraints like earliest start times and latest completion times, but may have a notion of its own schedule.

(3) The controller accepts and queues tasks as received, and understands externally prescribed constraints such as earliest start times, latest completion times and timing.

Controllers have additional functionality based upon the level of control at which they are integrated into the control hierarchy. A chart giving criteria for inclusion of controllers at the Workcell and Equipment Level may be found on page 46 (figure 12). Following is a brief description of the functions of a controller at each level.

4.7.1 Shop Level Controllers

The Shop Level Controller is responsible for converting orders into jobs. In the MSI architecture, a job's production is controlled by a single annotated process plan at the Shop Level. In order to make several products simultaneously, the Shop Controller allocates and schedules the use of shop resources, so that each set of parts can be made without conflicting with the others, and executes shop-level production plans.

4.7.2 Workcell Level Controllers

As discussed previously in the section on the Hierarchical Problem-Solving Paradigm on page 10, the term workcell refers to a grouping of equipment, or grouping of other workcells. A workcell controller coordinates the actions of its subordinate equipment or workcells as a unit. For example, if a workcell consists of a machining station, two robots and a roller table, it is the function of the workcell controller to ensure that all of these function in a unified way without collisions, or other timing problems. The WLC would receive its instructions from the SLC.

4.7.3 Equipment Level Controllers

The lowest level in the MSI architecture is the Equipment Level. An ***Equipment Level*** Controller (ELC) is one which can only perform one task at a time, and exposes none of its subordinates. It exposes an administrative interface to its supervisor and task interfaces to clients as required by the MSI architecture.

Typically, an ELC controls a single piece of equipment. In this case, there are a number of additional characteristics which are helpful in understanding the manufacturing problem at this level. Generally, such controllers do not have a notion of scheduling, but may have a notion of timing. Within the ELC, there may be other levels of control with other internal controllers. These levels are typically reactive and controllers at these levels have only a limited local view of the world. Sensory input is common at these lower levels. Ultimately, some controller must interface to the vendor-provided controller. Manufacturing equipment provided by vendors varies in command interface, communication interface and even control functionality. Equipment is not operated relative to an absolute clock, but rather, instructions are executed as soon as they are received. The ELC will translate commands and data into a form that the equipment expects to see. This may require various command and data exchange protocols for initiating/aborting, monitoring, feedback, start-up/shutdown/synchronization and downloading of equipment programs. The MSI architecture acknowledges lower levels, but has not attempted to describe them.

4.8 Guardian Interfaces

The five-interface model (see figure 5 on page 20) defines one port known as the Guardian Interface. This is the door through which monitoring of an MSI control entity takes place. A Guardian entity is considered to be everything on the other side of the interface, that is, any translator process plus any additional decision-making intelligence (human or otherwise). This Guardian Interface to a control entity exposes certain aspects of the internals of a control entity subsystem. A few envisioned uses for this interface are: monitoring procedures of a subsystem which may be dangerous to equipment or persons, assessing the health of a subsystem and in the case of ailment performing some remedy such as deconfiguring a subordinate, debugging and testing through direct modification of subsystem internal state data instead of dependence on external effects and substituting for an administrative supervisor when none is available at a given control level.

5. Conclusion

The MSI architecture incorporates an integrated production planning and control environment. This architecture identifies a number of discrete systems which are part of the shop production environment and the functions which each of these systems must perform. It specifies the information which is needed to permit information exchange between these discrete systems and provides mechanisms for information exchange. A method for integrating foreign (non-conforming) systems into an implementation of the MSI architecture is provided. Through this method, the architecture permits the integration of systems satisfying certain minimal functional constraints.

The MSI architecture specifies a hierarchical control structure that reflects the need to handle errors which occur on the shop floor. It provides interfaces for human intervention in error situations which are decoupled from the physical equipment. These uniform and user-friendly interfaces greatly facilitate the operation of an automated facility for the shop operators and will permit the replacement of the human by an automated system when such technology is available.

Current work on the MSI project includes the following activities:

- Elaboration of the information models for production planning and control,
- Alignment of the control entity interfaces with industry standards, notably MMS,
- Further analysis of the relationship between planning and control,
- Identification of the interface requirements for multi-level (distributed) planning and scheduling, and
- Analysis of the implications of integrating error identification, error recovery and performance analysis capabilities into the planning and control architecture.

As results are obtained from each of these activities, appropriate modifications will be made to the MSI architecture. Periodically, implementations of the revised architecture will be made and a revised architecture document will be published.

MSI level	*Task execution multiplicity [1 or Multiple (M)]*	*Administrative Unit or Complex (U/C)*	*Exposes those work-elements it can perform in parallel (Y/N)*	*Comments*
Equipment Level	*1*	*U*	*Y*	This is well-behaved, MSI conforming ELC. It is not capable of doing more than one task at a time, although it can do multiple subtasks at a time.
Equipment Level	*1*	*U*	*N*	This is a well-behaved, MSI conforming ELC. It is not capable of doing more than one task (or subtask) at a time.
Workcell Level	*1*	*C*	*Y*	This is a WLC whose control process is not sophisticated enough to control its subordinate equipment.
Workcell Level	*1*	*C*	*N*	This is a WLC which can only execute one task (or subtask) at a time. It is considered a WLC because it is reconfigurable, and it can be 'partially up'.
Workcell Level	*M*	*U*	*Y*	This is a WLC which multitasks, allowing other MSI systems to take advantage of its multitasking capabilities. The internal structure is not externally visible.
Workcell Level	*M*	*U*	*N*	This is a 'blackbody' workcell controller. It can multitask, but other MSI systems cannot take advantage of its multitasking capabilities.
Workcell Level	*M*	*C*	*Y*	This is a well-behaved, MSI conforming WLC that multitasks. Other MSI systems can take advantage of its multitasking capabilities.
Workcell Level	*M*	*C*	*N*	This is an ill-behaved WLC.

Figure 12 -Classes of Controllers Integrated at the Workcell and Equipment Levels

Appendix 1 - Glossary

administrative control

The process of ensuring the smooth start-up and shut-down of the control system, both in ordinary operations and in response to emergency conditions.

administrative subordinate interface

An MSI control entity interface from which administrative supervisory commands are received and administrative subordinate status is reported.

administrative supervisor interface

An MSI control entity interface from which administrative supervisory commands are issued and administrative subordinate status is received.

architecture

The design and structure of a system.

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.)

black box

A subsystem which is described only in terms of its inputs, outputs and functionality, but whose internal architecture is unspecified.

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 resumed.

communication paradigm

The set of operating principles characterizing a given communications approach.

communication

The transmission of information between distinct agents.

configuration

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

configuration management

Specification and control of shop resources, capabilities, and their physical and logical relationships.

control architecture

The design and structure of a control system.

control entity

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

control paradigm

The set of operating principles characterizing a given control approach.

control system

A collection of control entities which operate in a coordinated manner to perform a control function.

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.

database

A kind of data repository, often used as a synonym for data repository.

data repository

A logical system, incorporating some physical storage medium, in which elements of the information base are stored.

design

Detailed qualitative and quantitative description of an object to be manufactured.

dynamic reconfiguration

Changes to the shop configuration based upon changing conditions, such as equipment start-up or failure, while the shop is actively engaged in manufacturing. (cf. baseline configuration).

entity

A block in the MSI architecture.

equipment

A physical device within a manufacturing shop.

equipment controller

A particular kind of control entity responsible for managing tasks for a specific physical device.

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.

Facility Model

An information model describing the relevant classifications, properties, form, function, status and relationships of all components of a manufacturing facility (q.v.).

five-interface conceptual model

The model enumerating and describing the different types of interfaces to a control entity within the MSI architecture.

foreign controller

Any controller not specifically designed to conform to the MSI architecture.

Guardian

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

Guardian interface

An MSI control entity interface from which guardian commands are received and guardian status is reported.

hierarchical control

An approach to control which involves hierarchical task decomposition and a hierarchical control architecture.

hierarchical control architecture

A control architecture wherein there exists a single highest-level control entity and every other control entity has one or more supervisory control entities from which it receives commands, and zero or more subordinate control entities to which it may issue commands. The resulting structure is an acyclic lattice, with the simplest case being a tree structure.

hierarchical control system

A collection of control entities conforming to a hierarchical control architecture (q.v.). Each control entity accepts commands to execute higher level operations, performs hierarchical task decomposition (q.v.) on the operations and issues to its subordinates commands to execute the resulting lower level operations.

hierarchical task decomposition

The process of recasting a high level operation into a partially ordered collection of simpler lower level operations, such that their completion constitutes completion of the high level operation. This process is repeated through several levels until the low level operations produced are executable as atomic functions.

human interface

The mechanism through which humans can interact with the production process.

information base

A collection of sentences (facts) consistent with each other and with the information model (q.v.) expressing all facts about concrete and abstract items of interest which are not necessary propositions. The information base states the facts which are true of the individual objects of interest at a given time. It is typically represented by a collection of databases and state displays [30].

information model

A consistent collection of sentences (facts) expressing the necessary propositions that hold for all concrete and abstract items of interest (in the manufacture of products.) The information model states the facts which are true of each kind of object of interest [30].

integration

The process of defining the information interfaces, the control structure and providing services for communications, data access and data management within an enterprise.

interface

A point at which independent systems interact.

job

One lot or batch of a single product which has been scheduled for production.

manufacturing facility

The collection of human, physical and information resources dedicated to the manufacture of products.

manufacturing process

Synonym for processing operation.

material handling

Routing and delivery of material throughout the manufacturing facility (q.v.). It includes all operations on stock, parts, tooling, fixtures (etc.) and work in-process which is not a processing operation. (cf. processing operation)

native controller

A controller designed specifically to conform to the MSI architecture.

order entry

Process by which the shop is directed as to what to make and when to make it.

predictive control

An approach to control wherein a system evaluates all known methods and options for accomplishing a set of tasks against the current and anticipated state of the world before starting any action and selects a (thereafter immutable) sequence and schedule of events (a plan) and then blindly executes the plan to completion (or to abort on failure of some step). (cf. reactive control)

process

See processing operation.

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.

process planning

Creating step-by-step process plans for manufacturing a part, associated fixtures and jigs according to the design.

processing operation

A manufacturing activity which adds value to a movable resource, such as a part blank, fixture or tool. (cf. material handling)

product model

A description of the properties of a product, which includes the product's definition (design) and may optionally include its behavior, usage, maintenance and disposal.

product modeling

The act of creating a product model.

production plan

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

production planning

The act of creating process plans. Specifically, selecting batch sizes, allocating resources and assigning times to perform the tasks specified by the process plan.

reactive control

An approach to control wherein a system finds the best next step toward task completion on the basis of the current state of the world and initiates the corresponding operation. Upon the next significant change in the state of the world, which may be completion of the operation, it initiates the next best step, until the task is accomplished or can no longer be accomplished without external assistance. (cf. predictive control)

resource

An item which must be available for the accomplishment of a manufacturing task.

resource allocation

The explicit assignment of resources for special purposes according to a plan.

scheduling

The function of assigning execution times for all outstanding tasks within a production shop.

shop

(1) When followed by Level, the highest level of control in the MSI architecture.

(2) A partitioning of a facility into discrete parts by major function, e.g. paint shop, machine shop.

Shop Level

The highest level of control in the MSI architecture.

state table

A representation of a system in which the system is characterized by the specification of (1) a number of states representing all possible total conditions of the system, (2) the set of allowed transitions from one state to another.

state table programming

Developing programs according to the following model: a program is a collection of independent procedures to each of which is associated an invocation condition.The set of all variables occurring in invocation conditions are called the state variables of the program. Whenever a change occurs in any state variable, each invocation condition is examined and if it is satisfied, the corresponding procedure is executed. The term “state table programming” arises from the fact that the relationship between state variable values and procedure invocations can typically be formulated in a table.

task client interface

An MSI control entity interface from which task requests are issued and task status is received.

task control

The issuing of production tasks to the system, the monitoring of task execution and intervention in cases of necessity.

task server interface

An MSI control entity interface from which task requests are received and task status is reported.

testbed

A software and hardware platform designed specifically to support experimentation with systems, architectures or approaches.

work-element

(1) A discrete activity for some class of controllers in a manufacturing environment.

(2) The collection of information which describes (1.)

workcell

(1) An intermediate level of control which is responsible for the coordination of activities by subordinates.

(2) A control entity which coordinates the activities of a collection of equipment and/or other control entities.

Workcell Level

An intermediate level of control within the MSI architecture (q.v. workcell).

workstation

The lowest level of workcell control, immediately supervises the Equipment Level.

Appendix 2 - MSI Native Controller Architecture

This appendix describes the behavior and the internal functional structure of a controller designed as part of the MSI testbed which conforms to the interface specifications described in Section 3. A native MSI controller was designed and built to validate the specification of the role of a controller within the MSI architecture.The design of this controller should not be construed as forming part of the MSI architecture specification. This description of a specific controller design is provided as an aid to the reader in understanding the system architecture through example.

1.0 Planners and Executives

The MSI architecture recognizes that both predictive and reactive control are necessary, and necessary at all levels. Predictive and reactive control are incorporated at every level (per Albus' hierarchical control paradigms) [5][1] in a conceptual pair of modules:

(1) A Planner, which predictively generates a plan for the high level goal and a collection of plans for the individual subgoals which are consistent with the high level plan. The Planner is capable of replanning from the current status whenever any of the subplans becomes infeasible; and

(2) An ***Executive***, which implements the plan for a subgoal reactively, with whatever flexibility is left to it by the high level plan. In order to proceed toward several subgoals simultaneously, there are multiple conceptual Executives, one for each subgoal plan.

In higher level systems, the primary function is planning, and the executive function consists largely of passing the work-elements on to subordinates and determining when to replan. In lower level systems, the primary function is executing the work-elements, with constant but simplistic replanning to accommodate perturbations.

2.0 Shop Level Controller

As previously stated, the Shop Level has no automated counterpart in the shop of today. The MSI architecture includes an internal architecture for this controller, encompassing both the planning and executive elements. The object is to identify in detail all the logically distinct functions which must be performed at this level and to structure them in a meaningful and clear manner. An implementation of this architecture may make any number of choices as to whether to physically separate logically separated systems, but whatever the implementation, it must abide by the specified interface rules. A diagram of the SLC components is found in figure 13 on page 57.

2.1 Shop Level Planner

A Shop Level Planner (SLP) performs three functions: a planning function, a job control function, and an oversight function.

The planning function consists of:

(1) examination of the orders (which may be stored in a database) to determine production requirements and generate the job list;

1. The inclusion of reactive and predictive components in each level of control is also discussed in [9].

(2) examination of the process plans associated with the parts in the job list to determine resources required – workcells, equipment, tooling;

(3) examination of the shop and tooling information bases to determine resources available;

(4) algorithmic production planning, which selects from alternatives available in the process plan, determines the allocation of resources to jobs and the approximate schedule of resource utilization for each job;

(5) modifications to the plan information base to incorporate the annotations and path selections in the process plans, and modifications to the orders information base to show the scheduling of the job lots.

The job control function consists of developing a collection of directives for the creation, suspension, redirection and deletion of Shop Level Executives (SLEs) according to the production plan. The SLP implements these directives by spawning SLE instances and communicating with them. It authorizes the attachment of workcells by the SLEs. The control function recognizes the completion or failure of a SLE and updates the orders information base and the facility information base accordingly.

The Oversight function provides for both automatic and human interactive invocation of the planning cycle and the job control function. In the human interactive mode, it initiates planning at the request of the human operator, and provides for the human override of the job control function. In the automatic mode, it recognizes the need for invocation of the planning cycle, from changes in the configuration, changes in the orders information base, and failures reported by the SLEs, and automatically initiates the appropriate planning and job control functions. In all cases, it provides a description of the state of the jobs in its console interface.

2.2 Shop Level Executive

A Shop Level Executive is a software process which interprets and executes one Shop Level production plan, overseeing and coordinating the operations of several workcells and delivery systems in the manufacture of the associated product lot.[2]

The Shop Level Executive (SLE) is created by the SLP and given the process plan it is to execute. It attaches and releases workcell task servers according to the plan and requests creation and deletion of the associated control paths from the Dynamic Link Management service. In executing the process plan, the SLE dispatches tasks with subordinate plans to the appropriate workcells and coordinates their interaction according to the job plan.

It also deals with the problem of failure of a task in the assigned workcell and informs the SLP accordingly. Similarly, it responds to requests from the planner to suspend, reschedule, or substitute plans, or to abort the job, and handles the workcells accordingly. In the normal case, when the plan completes, the SLE releases any remaining task servers, and informs the SLP of the completion.

The SLE monitor interface identifies the current state of the subsystem, progress in the plan, active nodes, status of nodes, start times and estimated completion times, status of attached resources, etc. It is not clear whether any interventions are appropriate.

2. This is similar to the AMRF virtual cell concept. See [31].

3.0 Workcell Level Controller

The functions of a workcell controller are identified in this section, and the internal architecture for emulated controllers is discussed.

3.1 Workcell Level Planner

The Workcell Level Planner (WLP) provides services similar to those of the SLP. Instead of operating on sets of orders, however, the WLP operates on sets of tasks each of which is described by a process plan. The WLP schedules the set of Workcell Level process plans. It dispatches plans to and coordinates Workcell Level Executives. It authorizes the use of Workcell or Equipment Level task servers, handling conflicts between tasks which use the same resources in a given workcell. Whether scheduling of the Workcell Level resources is done at this level or at the Shop Level depends on the capabilities of the scheduling system and whether equipment resources may be shared by more than one SLE.

If each workcell can only perform one task at a time, or if it is necessary to perform all the scheduling at the Shop Level, the WLP is trivial.

3.2 Workcell Level Executive

Each Workcell Level Executive (WLE) executes one process plan at a time. This executive contracts with the dynamic link management services to make links to task servers (either equipment or lower level controllers.) The Workcell Level plan may be decomposed into a number of Equipment or (lower level) Workcell Level plans. The WLE dispatches the subtasks in the production plan to task servers. It attaches and releases task servers as specified by its production plan. It supervises the execution of the subtasks, keeping track of their completion or abortion. If a subtask is aborted, the WLE is responsible for recovering from the error or for informing its supervisor of the need to replan and reschedule.

4.0 Equipment Level Controller

The Equipment Level Controller is the simplest of the MSI controllers. At this level scheduling alternatives have been stripped from any plan given to these controllers, and real time scheduling does not occur. The ELCs are software processes which receive Equipment Level tasks from a WLC and oversees execution of these tasks on actual (or emulated) equipment such as robots or machine tools.Within this level of control is the software which interfaces with the actual physical equipment. There may be several levels of control within this level of control. Appropriate concepts for this type of control are discussed in [5].

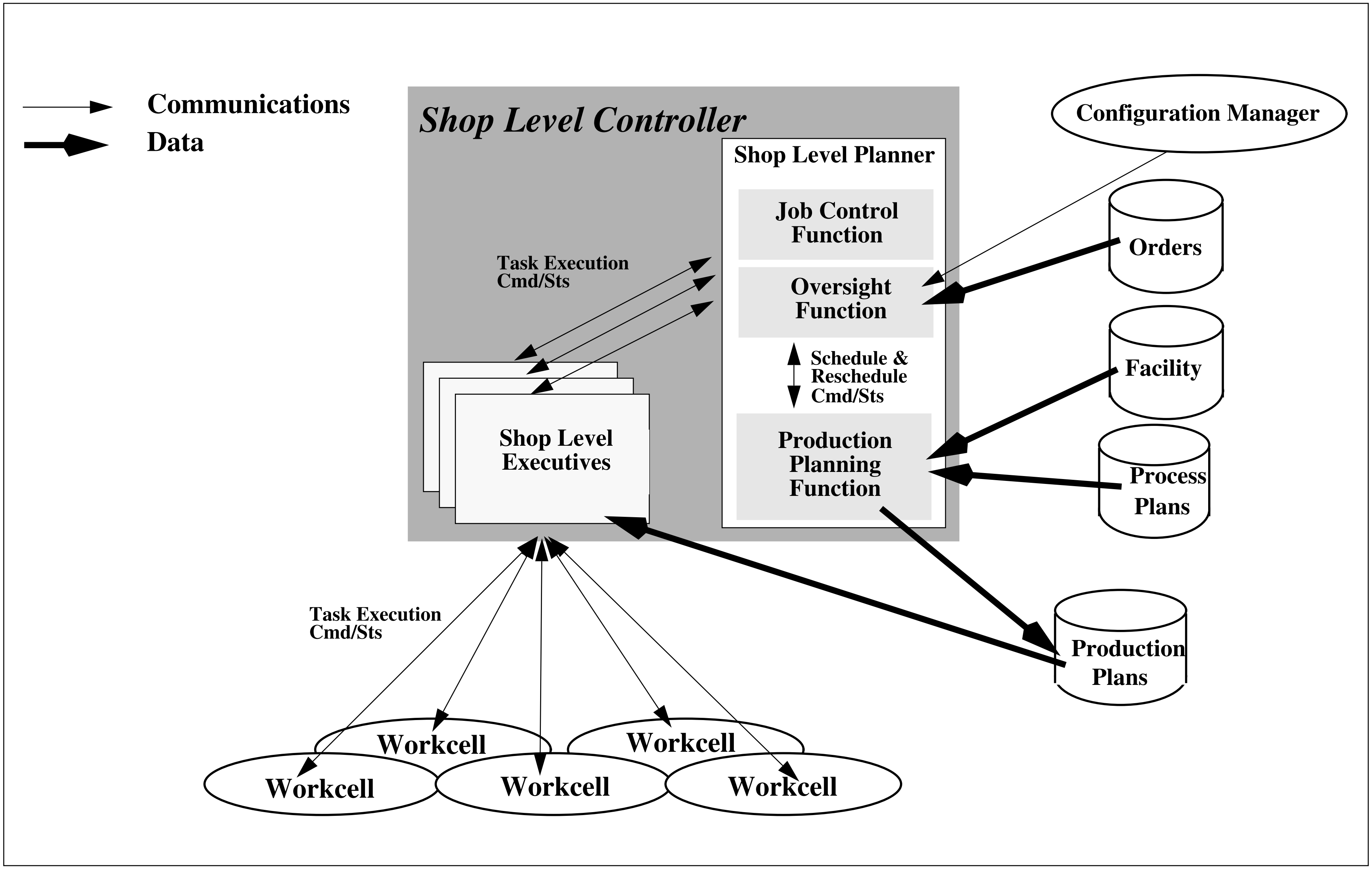


Figure 13 -The Internals of the Shop Level Controller: Internal and External Interfaces

Appendix 3 - Limitations

In the first year of the project, an initial implementation was made of the architecture described in this document. Strengths and limitations in the architecture uncovered as a result of this implementation are discussed in the sections below.

1.0 Control and Controllers

As a result of this implementation, we found that it was indeed possible to interface existing AMRF controllers to the MSI architecture. We also found that it is possible to operate with combinations of actual equipment and emulated controllers in a manner which was transparent to the system. Two controllers were used in this testbed that actually operated physical equipment. One controller accepted only one task at a time. The interfacing for this was done at the Equipment Level. The other controller queued tasks. The interfacing for this was done at the Workcell Level, as it had its own embedded scheduling capabilities.

Several conclusions were drawn from this work. First, it is always possible to integrate a controller having some basic features (such as the ability to be invoked remotely) in a trivial manner. By this we mean that it is always possible simply to send the controller commands in the format it desires and have the controller execute these commands. The second is that, in order to interface in a non-trivial manner, a controller must exhibit certain behaviors (e.g. it must understand some notion of production plan) which are not routinely found in controllers. The identification of the characteristics needed for a non-trivial integration is an avenue for continued investigation.

The separation of task and administrative interfaces exposed the very complicated interaction between the state of tasks and the state of the system. In exposing so much of the internal controller architecture, we may have greatly increased the work needed to interface to a controller. Given that the initial reason for this division was the accommodation of systems such as material handling which take requests from several sources, research is needed to determine if this is the proper view of these service systems and whether the division of the interface into task and administrative parts is the best method of accommodating them.

Related to the separation of task and administrative control was the question of whether a controller should continue to operate in the absence of its administrative controller. To date this has not been resolved.

Also apparent from implementational difficulties was that the notion of checkpointing a production plan is not as simple as it intuitively appears. The notion of checkpointing is clear for Equipment Level plans, but requires refinement for higher level plans.

During this work, some additional issues presented themselves which were related to the implementation of the architecture. More detail on the implementation and its limitations encountered may be found in [32].

2.0 Production Planning

A commercial centralized (production) planning and scheduling system was integrated into the architecture. The greatest difficulty in accomplishing this proved to be getting the commercial scheduler to understand the hierarchical model of the control levels enough to produce production plans at each level. The main observation on this aspect of the implementation is that it may not be possible to uncouple the approaches used for scheduling and control as cleanly as previously thought. Whether one has a centralized or distributed system greatly impacts the interface between the SLC and the scheduler. Deciding whether it is possible to simply plug in a new scheduling system in the same manner as plugging in a new controller requires further work.

3.0 Communications

In our implementation, we found the Common Memory [33] communication paradigm to be satisfactory for the needs of the testbed. We are aware however, of the existence of standards in this area which assume that all communication is point-to-point and based upon a messaging methodology. How to best use the standard, whether by changing paradigms or by encapsulating the communication to shield higher level systems from the new paradigm, is currently a topic of investigation.

Reference List

[1] J. Simpson, R. Hocken, and J. Albus, "The Automated Manufacturing Research Facility," *Journal of Manufacturing Systems*, Volume 1, Number 1, 1982.

[2] T. Hopp, "Proceedings of the Manufacturing Data Preparation (MDP) Workshop," unpublished report, June 1988. (Available from the Factory Automation Systems Division, National Institute of Standards and Technology, Building 220, Room A127, Gaithersburg, MD 20899.)

[3] ISO 10303, "Product Data Representation and Exchange, Part 1: Overview and Fundamental Principles," ISO TC184/SC4/Editing: Document N11 (Working Draft) (Available from the IGES/PDES/STEP Administration Office, National Institute of Standards and Technology, Building 220, Room A127, Gaithersburg, MD 20899.)

[4] C. Furlani, J. Wellington, and S. Kemmerer, "Status of PDES-Related Activities (Standards and Testing)," National Institute of Standards and Technology, Interagency Report 4432, October, 1990. (Available from the National Technical Information Service, Springfield, VA 22161.)

[5] J. Albus, H. McCain, and R. Lumia, "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)," National Institute of Standards and Technology, Technical Note 1235, April, 1989. (Available from the National Technical Information Service, Springfield, VA 22161.)

[6] P. MacConaill, "Introduction to the ESPRIT Programme," *International Journal of Computer Integrated Manufacturing*, Volume 3, Numbers 3 & 4, 1990.

[7] U. Graefe and V. Thomson, "A Reference Model for Production Control," *International Journal of Computer Integrated Manufacturing*, Volume 2, Number 2, 1989.

[8] A. Jones, E. Barkmeyer, and W. Davis, "Issues in the design and implementation of a system architecture for computer integrated manufacturing," *International Journal of Computer Integrated Manufacturing*, Volume 2, Number 2, 1989.

[9] A. Jones and A. Saleh, "A multi-level/multi-layer architecture for intelligent shop-floor control," *International Journal of Computer Integrated Manufacturing*, Volume 3, Number 1, 1990.

[10] "Manufacturing Automated Protocol Version 3.0," August 1, 1988. (Available from North American MAP/TOP Users Group, ITRC, P.O. Box 1157, Ann Arbor, MI 48106.)

[11] "Technical and Office Protocols Version 3.0" August 31, 1988. (Available from North American MAP/TOP Users Group, ITRC, P.O. Box 1157, Ann Arbor, MI 48106.)

[12] ISO 9506, "Industrial Automation Systems Manufacturing Message Specification, Part 1: Service Definition." (Available from the International Organization for Standardization, Geneva, Switzerland.)

[13] C. McLean, "Interface Concepts for Plug-Compatible Production Management Systems," Proceedings of the IFIP WG5.7 Working Conference on Information Flow in Automated Manufacturing Systems, Gaithersburg, MD, August 1987. Reprinted in "Computers in Industry", Volume 9, Elsevier Science Publishers, B.V. (North Holland), September,1987, pp. 307-318.

[14] E. Barkmeyer, "Some Interactions of Information and Control in the Automation of Materials Flow," in Advanced Information Technologies for Industrial Material Flow Systems, S. Nof and C. Moodie, eds., NATO ASI Series F, Volume 53, Springer-Verlag, New York, 1989.

[15] E. Barkmeyer, M. Mitchell, K. Mikkilineni, S. Su, and H. Lam, "An Architecture for Distributed Data Management in Computer Integrated Manufacturing," National Bureau of Standards Interagency Report 86-3312, January, 1986. (Available from the National Technical Information Service, Springfield, VA 22161.)

[16] R. Engelmore, (ed.), "Blackboard Systems," New York, NY, Addison-Wesley Publishing Company, 1988, pp. 602.

[17] A. Barbera, M. L. Fitzgerald, and J. Albus, "Concepts for a Real-Time Sensory-Interactive Control System Architecture," Proceedings of the Fourteenth Southeastern Symposium on System Theory, April, 1982.

[18] C. M. Furlani, E. W. Kent, H. M. Bloom, and C. R. McLean, "The Automated Manufacturing Research Facility of the National Bureau of Standards," Proceedings of Summer Computer Simulation Conference, Vancouver, B.C., Canada, July 11-13, 1983.

[19] A. Jones and C. McLean, "A Proposed Hierarchical Control Model for Automated Manufacturing Systems," *Journal of Manufacturing Systems*, Volume 5, Number 1, March 1986.

[20] A. Barbera, M. L. Fitzgerald, and J. Albus, "Concepts for a Real-Time Sensory-Interactive Control System Architecture," Proceedings of the Fourteenth Southeastern Symposium on System Theory, April, 1982.

[21] A. Barbera, J. Albus, M. Fitzgerald, and L. Haynes, "RCS: The NBS Real-Time Control System," *Robots 8 Conference and Exposition*, Detroit, MI, June, 1984.

[22] D. O'Halloran and P. Reynolds, "A Model for AMRF Initialization, Restart, Reconfiguration and Shutdown," National Bureau of Standards Grant/Contract Report 88-546, May 23, 1986. (Available from the National Technical Information Service, Springfield, VA 22161.)

[23] S. Rybczynski et. al., "AMRF Network Communications," National Institute of Standards and Technology, Interagency Report 88-3816, June 30, 1988. (Available from the National Technical Information Service, Springfield, VA 22161.)

[24] C. McLean, M. Mitchell, and E. Barkmeyer, "A Computing Architecture for Small Batch Manufacturing," *IEEE Spectrum*, May 1983.

[25] S. Ray, "A Knowledge Representation Scheme for Processes in an Automated Manufacturing Environment," Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Atlanta, Georgia, October, 1986.

[26] B. Catron, and S. Ray, "ALPS-A Language for Process Specification," *International Journal of Computer Integrated Manufacturing*, Volume 4, Number 2, 1991, pp. 105-113.

[27] D. Fishman, H. Scott, K. Strouse, and B. Bunch, "Integration of Material Buffering Devices in an Automated Factory," unpublished document, 1986. (Available from the Automated Production Technology Division, National Institute of Standards and Technology, Sound Building, Room B106, Gaithersburg, MD 20899.)

[28] M. Nashman and K. Chaconas, "The NBS Vision System in the AMRF," *Journal of Research of the National Bureau of Standards*, Volume 93, Number 4, July-August 1988.

[29] H. Scott and K. Strouse, "Workstation Control in an Computer Integrated Manufacturing System," Proceedings of AUTOFACT 6, Anaheim, CA, October 1984.

[30] ISO TR-9007,"Concepts and Terminology for the Conceptual Schema and the Information Base." (Available from the International Organization for Standardization, Geneva, Switzerland.)

[31] C. McLean, H. Bloom, and T. Hopp, "The Virtual Manufacturing Cell," IFAC/IFIP Conference on Information Control Problems in Manufacturing Technology, Gaithersburg, MD, October, 1982.

[32] M. Senehi, et.al., "Manufacturing Systems Integration Control Entity Interface Document" National Institute of Standards and Technology, Interagency Report 91-4626, June, 1991. (Available from the National Technical Information Service, Springfield, VA 22161.)

[33] D. Libes, "NIST Network Common Memory User Manual," National Institute of Standards and Technology, Interagency Report 90-4233, January 1990. (Available from the National Technical Information Service, Springfield, VA 22161.)