Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

A Machine Entity for a Coordinate Measurement Machine

The Generic Workcell Project

Submitted to
Department of Computer and Information Science
New Jersey Institute of Technology

in partial fulfillment
of

the requirements for the degree of
Master of Science

by
Richard C. Meyer

APPROVALS

Date Submitted: 1/8/92

Date Approved: 1/8/92

Approved by: _____

Abstract

The Center for Manufacturing Systems (CMS) department at New Jersey Institute of Technology and Siemens Corporate Research located in Princeton have agreed to jointly implement a research project in generic workcell control architectures. This paper discusses the module, called a *Machine Entity,* developed by the author that interfaces the Brown & Sharpe Coordinate Measurement Machine located on the CMS factory floor with the cell control software. The module has been designed in such a manner to simplify the development of future Machine Entities, thereby reducing the time required to integrate the CMS factory floor.

Table of Contents

1. Introduction and Background

1.1. Introduction

The author has been working at Siemens Corporate Research (SCR) for nearly two years assisting in the development phase of the *Generic Workcell Project*. The project is a collaborative effort between NJIT and SCR. Two departments at NJIT are involved: The Center for Manufacturing Systems (CMS), and the Computer & Information Sciences (CIS) department.

The CMS department originated the effort with Siemens to help meet their goals of integrating the equipment on the factory floor. The CIS department, specifically Professor Alexander Stoyenko, became involved as the author's undergraduate and graduate project advisor.

This paper concentrates on the authors latest project assignment: the development of a machine interface module, called a *Machine Entity*, for the coordinate measurement machine (CMM) on the CMS factory floor.

1.2. Problem Statement

The Center for Manufacturing Systems at NJIT has plans to integrate the machines on the factory floor, including,

- a Mazak Turning Center
- a Brown and Sharpe Coordinate Measuring Machine
- a White Automated Storage and Retrieval System
- an AT&T Flexible Workstation
- a Charmille Technologies Electronic Discharge Machine
- and a Plastar Plastic Injection Molding Machine.

Reggie Cauldill, executive director of CMS, knew of the *Generic Workcell Project* being developed at Siemens. An agreement was reached where NJIT would provide two students to Siemens to assist in the implementation phase. In return, Siemens gave NJIT unrestricted rights to the software which would be used not only for coordinating the factory floor, but for further research and a demonstration site.

In January of this year, the port of the software from VMS to Unix began in anticipation of installing the software at NJIT. Several months later, around July, the author concentrated his efforts in developing a Machine Entity, a module of the Generic Workcell that interfaces with a machine, and writing *The Functional Requirements for a CMS Generic Workcell*, a document that specifies the physical layout, products, communications, and performance objectives of the cell at NJIT[4].

1.3. Previous Work

The Generic Workcell is a research project in distributed systems. Although the architecture has been applied to factory automation for testing purposes, it is believed to be applicable to other industries[1]. The traditional structure of workcell control systems is hierarchical, that is, classifying things in a top-down manner, forming a tree structure composed of many levels[12]. The benefit gained from using a hierarchical system is fast response times due to the strong master/slave relationship between the levels. However, modifications are difficult since making a change at one level requires knowledge of the level above as well as below it. Experience has shown that fault tolerance is obtained in hierarchical systems with considerable expense and complexity[10]. Papers as recent as 1984 argue that a hierarchical workcell control system is necessary because different functions in the system are located in different geographical areas[13].

In [13], Bjorke describes his three level hierarchy motion system. The lowest level in the hierarchy pertains to the the functions executed on machines such as those specified in a machine part program. The middle level pertains to functions executed in machine groups (or workcell) such as the flow of raw materials and tools to/from machines. The highest level pertains to functions performed in a factory unit that may contain several machine groups, such as global transport of tools, raw materials and workpieces among machine groups.

Although the three level hierarchical motion model is a logical and natural way to approach the integration of manufacturing cells and their subsystems, the design introduces many complex internal dependencies. For example, the cell supervision module which feeds control data to machines, coordinates their activity, and acts as the operator interface, is comprised of six modules. Among these six modules, typically each module interfaces with three other modules.

As computer hardware costs decreased, making microcomputer based networks economically feasible, the potential benefits of applying computer networking to machinery control systems became clearer[14] and researchers began to question the relevance of hierarchically controlled systems[8]. In [14], Duffie presents a formal approach to designing a distributed, networked control system. Briefly stated, his approach is a top-down strategy that can be used to decompose a complex control system into a set of more managable processes that cooperate via a communication network.

His strategy is to first specify the requirements of the system. The second step is to partition the system into a group of quasi-independent processes. After the system has been specified and partitioned, the feasibility of communications between the processes is examined. If communication is feasible, then the partitioned processes are assigned to processors based on hardware resources and other considerations. If the allocation of processes to processors satisfies the original system requirements, then the communication network, processor hardware, and software are designed.

Duffie argues:

> *while hierarchical structures have been traditionally proposed for...process control systems, there seems to be little justification for the strictly hierarchical structure. A natural decomposition...is likely to result in a nearly hierarchical, multilevel structure where communication takes place between processes without regard to conceptual levels.... The point here is that communications, reliability, and other consideration may result in a process structure that is different from any preconceived functional structure.*

The present belief, as described in the literature [6, 8, 10, 11, 12] is that the decision making should be located at the point of information gathering rather that in a central location. This methodology has been termed *heterarchical* or non–hierarchical.

The advantages of a heterarchical system are numerous. For instance, flexibility is increased since programs are small and are local autonomous units. Complexity is reduced since information is localized. Reliability and fault tolerance is high since the systems is distributed over several processors. Global information is minimized as well as the complexity of relationships between modules. Also, the model is more readable since there are no hidden modular interdependences that is typical in a hierarchical system[8]. Furthermore, prgrammers of modules need only concern themselves with the logic of the module they are programming and the messages needed for cooperation with the rest of the system[10].

Many papers [6, 7, 8, 10, 11, 12, 13] have been written that describe workcell control architectures that are similar to the design used in the Generic Workcell. For instance, in [6], Histon describes his approach to building a distributed system infrastructure based on what he calls proxies. Proxies, which provide a uniform interface to a common set of protocols, are not unlike our Conversation Tool. The Proxy libraries which provide basic required distributed system services such as naming, message passing, security, and monitoring/control, are similar in function to some of our modules. Also, like our design, Histon uses a mulitcast system for interprocess communication. The Generic Workcell is unique, however, since it supports dynamic reconfiguration in a heterogeneous environment.

To date, two prototype cells have been implemented – a Chemical Vapor Deposition Cell in Germany and a simple robotic assembly cell in Princeton. The simple robotic assembly cell, which is the testbed for the Unix version of the software, consists of:

- a robot with 5 degrees of freedom (Scorbot)
- a pneumatic pick and place robot with 3 degrees of freedom
- a linear table
- two conveyers
- several bar code readers
- a parts presentation station
- an assembly area
- a quality control center

The first Machine Entity was developed for the Scorbot by NJIT student Jaskaran Dhaliwal, assisted by Research Scientist Paul Bruschi. His Machine Entity specification describes the interfaces, database tables, conversations and design[5].

In his design, as shown in Figure 1-1 two processes are used to implement the Entity. The WC (Workcell) process spends most of its time listening at the message bus waiting for a message to arrive. When a message does arrive it stuffs a copy of the message in a piece of memory shared by both the WC and the ME processes. The ME process spends most of its time monitoring the machine (for results or alarms). At regular intervals, the ME polls the shared memory to see if a message has arrived.

There are several flaws with the implementation of the design. First, although using shared memory as a form of interprocess communication is fast and efficient, it provides no synchronization between the processes. This must be provided separately by using semaphores, signals or the like. With the lack of synchronization in this design we can not guarantee that the ME has read shared memory. Second, since an operation at the machine

can take from several seconds to several minutes – preventing the ME process from polling – a buffering mechanism needs to be introduced to prevent loss of messages.

These deficiencies have been dealt with in a new implementation of the Scorbot Machine Entity and in the implementation of the CMM Machine Entity. The changes are discussed in detail throughout this document.

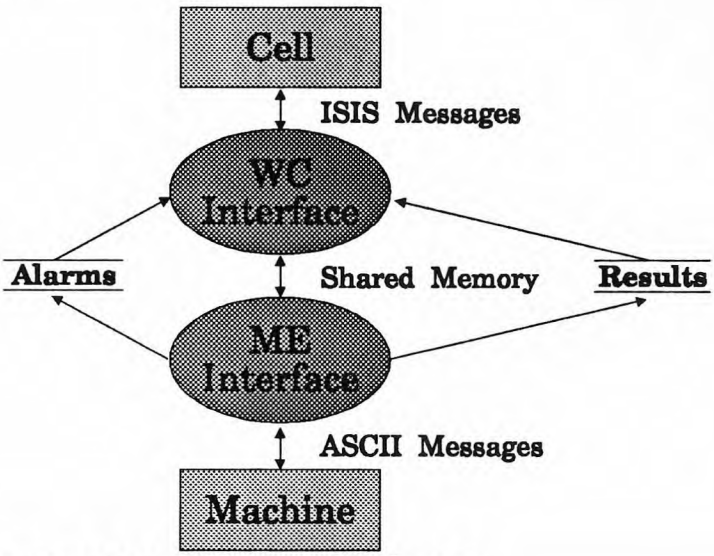


Figure 1–1
Original Machine Entity Design. Taken from *Machine Entity Specification,* Version 1.0, Jaskaran Dhaliwal.

1.4. Theoretical & Conceptual Foundations

The theoretical and conceptual foundations for the Generic Workcell Controller is based on the research done by scientists Dan Wolfson, Paul Bruschi, et. al. Their work is described here to give the reader a better understanding on the depth of the project and to help the reader understand what the author's project is and how it relates to the overall project.

D. Wolfson and P. Bruschi wrote in their *A Reconfigurable Generic Workcell Architecture:*

> *Fundamentally, we view manufacturing automation as a distributed activity in which software modules cooperate to collectively control manufacturing. This form of loosely coupled, distributed control has been termed heterarchical or non-hierarchical to contrast it with the more traditional hierarchical control architectures....*

Based on this view, they provide a conceptual model, called the *entity–server model.* Entities depict physical features such as machines or production lots, or abstractions such as manufacturing operations[1]. The behavior and goals of an entity is described by its script. For example, a lot entity script directs the processing of a physical lot while it is being manufactured in the cell. Entities cooperate with other entities directly by broadcasting messages or indirectly through the sharing of data.

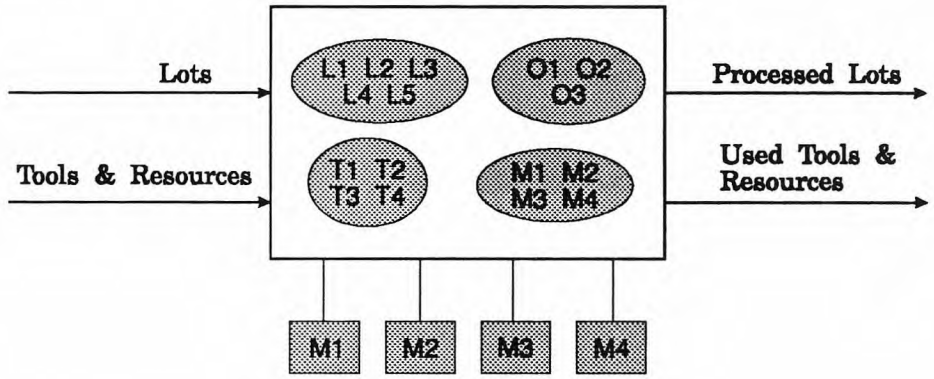
Entities representing the same type of manufacturing feature are grouped together to form an entity class[1]. All entities within a class have common properties and definition, store similar information, maintain common representations of entity state, and utilize common definitions of communication messages[1]. Some commonly found entity classes for manufacturing cell are a lot class, resource class, tool class, and machine class.

Each entity class is overseen by an entity server which provides class specific services common to that class. For example, the machine entity server provides all machine entities

with recipe (part program) management functions, machine data collection, machine alarm handling, and preventative maintenance scheduling.

Figure 1-2 shows the product(lots), tools, and resources entering the cell. Four classes are shown: lot class, operation class, tool class, and machine class. Each class contains several entities. For instance, the machine class has four machine entities, one for each physical machine M1 through M4.

Figure 1–2



1.4.1 Software Architecture

Each class in the entity-server model maps to software modules collectively called a service. Each entity in an entity class is implemented as an autonomous software module[1]. Each entity server is implemented as one or more autonomous software modules depending upon the complexity of the functions provided by the entity server[1].

Modules communicate via message bus, and store state information in a commercial database.

Figure 1-3 shows several computer tasks (shown as bubbles) that are servers and entities. The figure stresses that the system is distributed – two computers are shown, possibly one Unix and the other VMS, each running a portion of the cell software. Note that the two databases shown represent one physical distributed database. Also, the two message busses shown represent one logical bus that extends across the network via ethernet.

1.4.2 The message bus

The message bus is implemented using the ISIS distributed toolkit. The ISIS system adopts an approach based on synchronous execution, whereby every process sees the same events in the same order[2]. ISIS also provides tools for creating and managing process groups, group broadcasts, failure detection and recovery, and distributed execution and synchronization[2].

A common interface to ISIS is provided for cell modules by a tool developed at Siemens called *The Conversation Tool*. The tool not only provides a method for defining conversations, but also maintains the libraries that are linked with cell modules.

The message bus scheme is used since it supports dynamic reconfiguration of the cell. Modules set filters to specify which conversations they are interested in participating in. When a message is sent, it is delivered to all modules on the message bus. Thus, modules

can join or be removed from a message bus without notifying or effecting existing bus members.

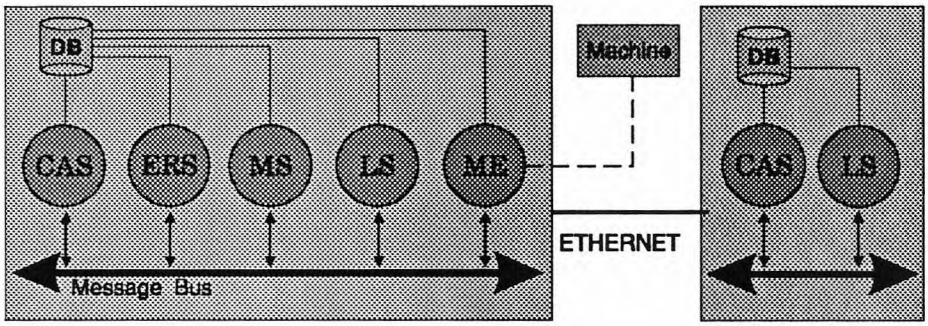


Figure 1–3
Generic Workcell Architecture.

1.4.3 The Human Interface

The Generic Workcell human interface is a single Ingres ABF application called HI. The HI architecture provides two types of frames: menu frames and transaction frames. Every frame, regardless of its type, has a unique four letter Transaction Code (TA code) associated with it. TA codes provide the user with the ability to "jump" to a particular frame, rather than traveling through menus and sub-menus, as well as a means of controlling user access privileges on an individual frame basis.

Menu frames display menus that allow the user to either browse through selections or type a TA code directly. A menu frame can take the user to another menu frame or to a transaction frame. Transaction frames are the frames that perform specific tasks (i.e. create recipes, clear cell alarms, start and shutdown cell modules, etc.). See appendix A for samples of the human interface screens.

1.4.4 Other Modules

Several servers have been implemented: the Cell Alarm Server (CAS), the Equipment Recipe Server (ERS), the Machine Server (MS), and the Lot Server (LS). There functionallity is now briefly described.

The CAS manages alarms in the cell. The user is notified of alarms via an asynchronous communications mechanism, called the *status line server*, that is a part of all HI menu frames. After appropriate action has been taken, the user clears the alarm.

The ERS manages recipes, or *part programs*, in the cell for all machines. The ERS provides functions to define recipes, upload & download recipes from/to machines, update recipe states, and update recipes[15].

The MS provides functions common to all machines. For instance the MS module initiates the startup sequence for each machine in the cell. It also provides other services like individual machine shutdown and recording the state of each machine.

The LS provides services to the lot entities, the computer tasks that represent the physical lot in the cell. In particular, the LS provides functions to create, merge and delete lot entities.

As mentioned, a Machine Entity was implemented for the Scorbot robot in the simple robotic assembly testbed cell in Princeton. Also, various Lot Entities were created which coordinated

the assembly operation performed by the robot: the building of a plane and helicopter assembled from Lego's™.

1.5. Project Description

The author's project was to develop a Machine Entity (machine interface) for the Coordinate Measurement Machine of the CMS factory floor. To achieve that end, the author coordinated with the Computer Services department for the computer hardware needed for the project, ported the software from Siemens, and became a proficient programmer/operator on the CMM.

2. System Functional Specification

2.1. Functions Performed

The cell control software, as a whole, controls the entire manufacturing process including, but not limited to, scheduling, sequencing, material handling, process control, resource and non–consumable management, and part program management. See Figure 2-1.

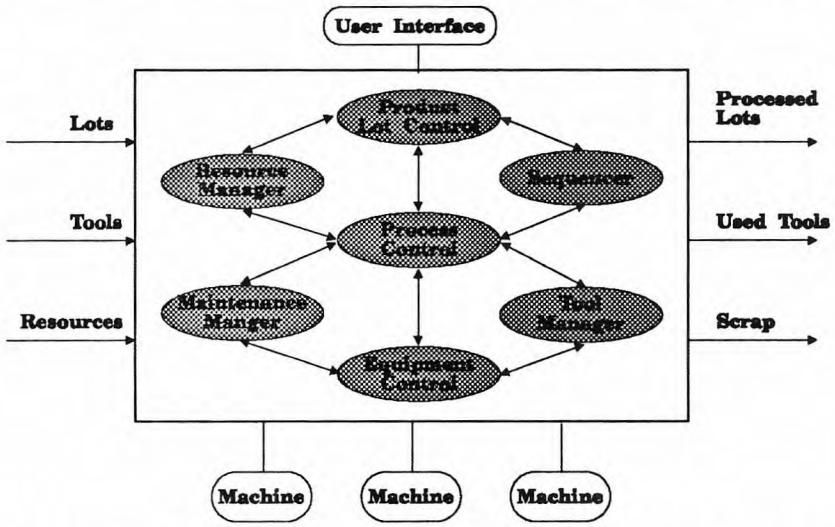


Figure 2–1
Workcell Environment. Copied from *A Reconfigurable Generic Workcell Architecture.*

Each machine entity module of the control software interfaces with one machine and is customized to utilize that machines features. The features of the CMM that are utilized by the CMM machine entity software are:

- *homing the machine* – part of the initialization process of the machine; must be done after every shutdown/power-down.
- *selecting part programs* – allows the user to select an existing measurement program.
- *execution of part programs* – executes the previously selected measurement program; the selecting and executing of part programs is synchronized by the software.
- *turning privileges on and off* – a method of providing password protection to certain file maintenance functions.
- *part program file maintenance* – including, deleting, updating, and system backup of part programs
- *shutdown of the CMM* – takes the machine off line and unavailable for use.

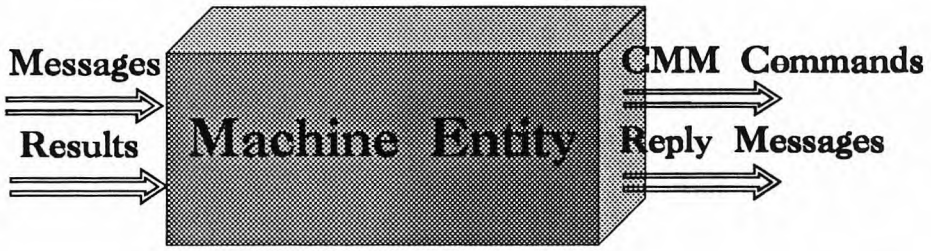
2.2. Input Preview

As shown in Figure 2–2 there are two inputs to the machine entity: messsages and results. The messages recieved by the ME process from the WC process are triggered by ISIS

messages. ISIS messages can originate from any module in the cell, but typically for the CMM Machine Entity, messages will be initiated by the Human Interface, Machine Scheduler, and Lot Entity scripts.

Since the authors project was the ME portion of the machine entity, attention is given only to the IPC mechanism used between the WC and ME in section 5–2. The following discussion about ISIS messages is done to give the reader some insight on the functionality of the WC.

Figure 2–2
The inputs and outputs of the Machine Entity module.



ISIS messages are defined and created using a software tool called *The Conversation Tool,* developed by Rick Taft of Siemens. Using the tools graphical user interface, the user defines the messages (specifically the fields of the message, their data type, and thier lengths). The definitions are used to create C++ classes which are separately compiled and eventually linked with the modules of the cell. For more information the reader is reffered to document [17] written by Rick Taft.

Results include all responses generated by the CMM including results from a measurement and alarms caused by an extraordinary event. In the current implementation, no alarms are detected, only results are passed.

The report generated by the CMM is transferred to the cell in a results file. The Lot Entity contolling the production of lot will be responsible for deciding if the part is accepted, rejected, or needs rework based on the contents of the file. Also the Lot Entity will be responsible for parsing the results file and placing the results of certain measured features in the cell database to collects statistics.

2.3. Output Preview

As showm in Figure 2–1, there are two outputs of the machine entity module: reply messages and CMM commands. Replies are messages which are used to indicate either success or failure. Most modules in the cell behave in this way; they do not just send messages, but engage in a conversation which is typically comprised of two messages: a request and a reply.

CMM commands are the actual outputs generated by the machine entity to invoke a function of the CMM, like engaging the homing sequence. To understand the CMM commands, it is neccessary to understand the software supplied by Brown & Sharpe, the manufacturer of the CMM. The software package supplied by Brown & Sharpe to create and manage recipes is

called AVAIL, the *Advanced Validator Interface Language*.[16]. Under AVAIL, a CMM part program (or recipe) consists of at least two distinct files. One is called the Probe Qualification File or QUAL file. Probe Qualification tells the computer the probe diameter, location, and the angle of the PH9 probe wrist. The computer compensates for this information when measuring parts. The other file is called the list learn file (llf). The llf is the main program that controls the CMM to measure a specific part. The llf includes one or more QUAL files. The software allows only one part program to manipulated at a time. Once the part program is selected, all functions from the AVAIL menu effect that part program. Thus by specifying the default part program, the llf and QUAL files are also specified. The CMM commands generated by the machine entity to perform the functions mentioned above are character strings that correspond to the menu selections from the AVAIL menu. For example, the sequence EY shuts down the machine (menu selection E for shutdown, and Y to confirm).

2.4. System Database/File Structure Preview

All modules have access to a common database. The database is used to store three types of data: static, dynamic, and historical. Static data includes configuration information. Dynamic data includes the status of the machines in the cell and process results. These data are selectively archived to provide a history of the cell.

The database is currently consists of 49 tables. The tables used by the ME are:

- OP_MACHINE_RECIPE
- MACHINE_RUN_RESULTS
- CONFIGURE_ACCEPT_TYPES
- MACHINE_STATUS
- ALARM_DESCRIPTION
- ALARMS_RECOGNIZED

2.5. Message Preview

In section 2–2, a brief introduction on creating and using conversations utilizing The Conversation Tool was given. The following is a list of conversations that take place between the Machine Entity and the rest of the Cell. Specifically, these conversations take place between the cell and the WC process that was implemented by Peter Murray. They are listed here, however, since they are sent by the WC to the ME (in a different form) via an IPC mechanism called message queues. The cooperation of the WC and the ME processes are discussed in deatil in a later section.

- ME_START – The ME sets its conversations appropriately and homes the machine
- ME_START_COMPLETE – Reply.
- ME_SHUT – The ME performs a shutdown of the machine.
- ME_SHUT_COMPLETE – Reply.

- SETUP_RUN – The ME performs a sequence of events to prepare for a lot, specifically, verifies that the machine is homed, downloads the recipes, and "sets" the part program.
- START_RUN – The ME executes the part program.
- RUN_COMPLETE – The ME notifies the cell that the execution of program X is complete.
- CELL_ALARM – Initiated by the ME when it detects an extraordinary event.
- UPLD_ME – Instructs the ME to upload a recipe to the cell.
- DNLD_ME – Instructs the ME to download a recipe from the cell.
- ACC_TYPE_COMPLETE – The ME notifies the cell that there was a change in one of its accept types.
- UPD_ME_STATUS – The ME notifies the cell that its state has changed.

3. System Performance Requirements

3.1. Efficiency

The workcell system host computer at NJIT is a Sun 4/210 running the Andrew File System (AFS), version 3.1 under Sun OS 4.1.1 with 32 megabytes of RAM. The computer runs the Ingres relational database, the ISIS distributed toolkit and all cell modules, including the Equipment Recipe Server (ERS), the Cell Alarm Server (CAS), the Machine Scheduler (MSCHED), the Machine Server (MS), the Lot Server (LS), and various Lot Entities (LE) and Machine Entities (WC & ME).

This platform performs satisfactorily when the modules are compiled with the optimization flag set and with the logging level set to its lowest level (i.e only fatal errors are logged). Problems arose when the modules were compiled with debug information and the logging level was set at high. In particular, ISIS conversations were timing out since the computer was overloaded.

The software also performs well on a Sun Sparc 1+ with 32 megabytes of RAM and a 200 megabyte hard disk. In this configuration, however, it is necessary to setup the station as an Ingres client and run the Ingres back end on a server. This is the preferred configuration.

3.2. Reliability

Both ISIS and Ingres play a central role in the architecture of the software. If either fails, the cell control software will effectively stop running.

Regarding ISIS, problems of this nature have been minimized due in part to the efforts of Rick Taft, the designer of *The Conversation Tool*. The tool has provided a common ISIS communications interface that has proven to be consistent and reliable.

Problems associated with the database have been almost non existent. Ingres is a reliable, commercial database.

3.2.1 Description of Reliability Measures

Several measures were taken to ensure the reliability of the software: a logging system, database recovery macros, and multiple occurrences of a module to increases concurrency.

A common logging systems is used by all modules to record fatal errors, warnings, and general notes. The logging level is set on a per module basis and can be changed while the module is running. The log files are particularly helpful during debugging phases. See appendix E for sample log file output.

A set of M4 macros are defined to provide a consistent database recovery mechanism. For instance, the macro *saveto()* allows the programmer to define an Ingres save point that is used to define the start of a database transaction. The macro *check_and_recover_to()* is placed after all database select, update and delete commands. It automatically checks for

fatal errors such as deadlock and log full errors. If the user provides the expected number of rows to be returned from the transaction, depending on the result, either the work will be committed, rolledback, or tried again.

The issue of concurrency was addressed by Bill Nell. His project allows multiple modules (i.e. several Cell Alarm Servers) to be running in the same cell. A biding technique is used to decide which module will handle the incoming request.

3.2.2 Error/Failure Detection and Recovery

In addition to the logging system and database recovery macros which provide our error detection and recover mechanism, ISIS and Ingres have their own mechanisms. For example, ISIS provides reliable ordering and message delivery; either all modules get the message or none do. Ingres uses a logging system for recovery.

3.3. Security

The Generic Workcell project is considered to be, by Siemens Corporate Research, confidential and proprietary information. NJIT has been granted rights to the software, including the patented *entity–server model* and the source code written to date. NJIT is expected to maintain the confidentiality of the software.

3.3.1 Hardware Security

The hardware for the project is maintained and secured by the Computer Services (CS) department at NJIT. The workstation on the factory floor is secured with metal wire harnesses. The main computer, Spruce, is locked in a room on the second floor of the Information Technologies building. Access to the room is controlled by the CS department.

3.3.2 Software Security

Software security is provided by password authentication procedures controlled by the Andrew File System. Copies of the software made for backup purposes are stored in locked file cabinet.

3.4. Modifiability

Modifiability can be addressed on two levels: the modifiability of the architecture as a whole, and the modifiability of a single module.

Addressing the modifiability of the architecture, the goals of the project have been to provide a cell controller that is dynamcally reconfigurable in a heterogenous environment. Theses goals have been achieved by using a non-traditional heterarchical design as explained earlier. For instance, the design allows a user to alter a module's behavior, and then replace it with the existing, running version without interfering with the normal operation of the cell.

The Machine Entity module, like the rest of the modules in the Generic Workcell project, is fully documented. The documentation is more than sufficient to allow a programmer to modify and upgrade any module of the cell. Typically, the documents include functional description, database interface, and ISIS interface sections, as well as pseudo code.

3.5. Portability

The software, with modifications, has run on a MicroVAX II running VMS 5.1, and on a Sun 4 running Sun OS 4.1.1. During the port from VMS to Unix, attempts were made not to introduce non portable code. However, several Unix specific IPC mechanism have been used in the Machine Entity to improve the performance and reliability. Modifications will be needed to port back to VMS.

As for the other modules of the cell, it has been shown that modules that were compiled under the Conversation Tool environment will port with only minor modifications. The modifications needed are a result of the inherent minor differences in the compilers available for the systems.

The systems in which the software can be ported to is limited to the systems in which ISIS and Ingres are available.

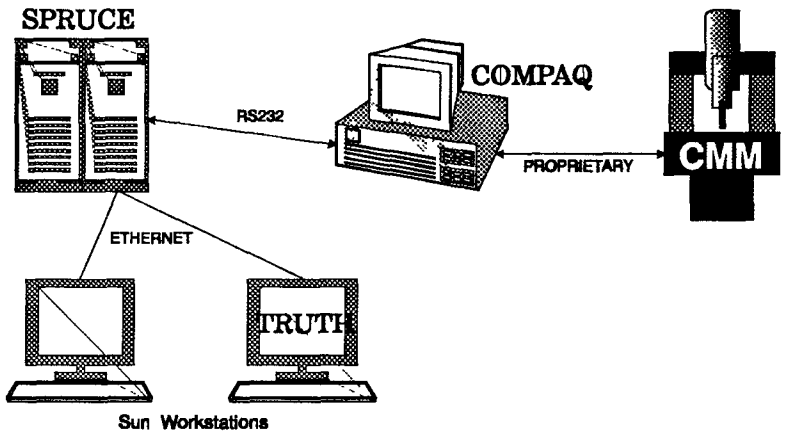
4. System Design Overview

4.1. Equipment Configuration

The implementation at NJIT uses a Sun 4/210 (called Spruce) running Sun OS 4.1.1 with 32 megabytes of memory, and 50 megabytes of disk storage for the cell software. Additional storage is required for the ISIS distributed toolkit and Ask Technologies' Ingres Relational Database version 6.3. Spruce runs all of the cell's modules as well as ISIS and Ingres.

The computer controlling the CMM is a compaq 386, with 4 megabytes of memory and a 40 megabyte hard drive running the Xenix operating system. In total, four serial ports and 2 parallel ports are available. Currently 1 parallel port is being used for an Epson LX-800 printer, and 1 serial port is connected to Spruce. The computer runs AVAIL, the measurement software provided by the manufacturer and "RS232_to_AVAIL", a process created by the author.

Figure 4–1



Two Sun workstations were provide by the Computer Services department for the factory floor to develop and test the CMM machine entity. One workstation, named TRUTH, is a Sun Sparc 1+, the other is a Sun/3. Both were connected to the network to access Spruce. See figure 4–1.

4.2. Implementation Languages

Modules are coded using the AT&T C++ compiler, version 2.1. Two preprocessors are used: M4 which is supplied with Sun OS and Embedded SQL supplied by Ingres. M4 is used so it is possible to define macros which contain ESQL calls.

The "RS232_TO_AVAIL" program was coded in non-ansi C using the compiler provided with SCO Xenix. Since the Xenix operating system installed on the compaq at NJIT does not include the development kit (i.e. a compiler), it was necessary to compile the code at Siemens on a similar machine and transfer the executable.

4.3. Required Support Software

There are three support software packages:

- ISIS, version 2.1
- Ingres Relational Database, version 6.3 or 6.4
- Buildtool, Siemens proprietary "make" utility

5. System Data Structure Specifications

5.1. User Input Specification

5.1.1 Identification of Input Data

In section 2–2, two inputs to the machine entity were identified: messages and results. This section discusses both in detail.

5.1.2 Messages via Message-Queues

The two processes that make up a machine entity, namely, the WC and the ME, communicate via the interprocess communincation facility called message-queues. Message-queues are a form of shared memory, but have several distinct added advantages over ordinary shared memory. First, it provides synchronization. Shared memory is usually used in conjuction with a synchronization method like semaphores. Second, many messages can be be sent with out loss of data. With shared memory, only one "message" could be sent at a time.

Two system level functions are used to access the IPC faciltiy: mqsnd() to send a message and mqrcv() to receive a message. One of the arguments to the functions is a pointer to the structure that defines the fields of the message. The message structure used by the WC and ME is:

- *message_id* – an integer field that contains a unique number identifying the message. For consistency, the message-queue message_id is equivalent to the message_id of the corresponding ISIS message that triggered the event.
- *filename* – a string field used in a download and upload message to specify the path and filename to be acted upon.
- *recipe_id* – a string field containing the corresponding id given to a file/recipe
- *ack_code* – an integer used in reply messages sent by the ME to the WC. The code is either an ACK or a NACK.
- *alarm_text* – a string field that contains information detailing the extraordinary event.

The following is a list of the message-queue messages that are exchanged by the WC and ME. Instead of listing the message_id's, a meaningful tag, like ME_START, is given.

- ME_START – the ME initiates the homing sequence.
- ME_START_COMPLETE – Reply.
- ME_SHUT – The ME performs a shutdown of the machine.
- ME_SHUT_COMPLETE – Reply.
- SETUP_RUN – The ME performs a sequence of events to prepare for a lot, specifically, verifies that the machine is homed, downloads the recipes, and "sets" the part program.
- SETUP_RUN_COMPLETE – Reply.
- START_RUN – The ME executes the part program. Results and alarms are collected during this phase of operation.

- RUN_COMPLETE – The ME notifies the cell that the execution of program X is complete. The results are transferred to the "cell".
- CELL_ALARM – Initiated by the ME when it detects an extraordinary event.
- UPLD_ME – Instructs the ME to upload a recipe to the cell.
- DNLD_ME – Instructs the ME to download a recipe from the cell.
- ACC_TYPE_COMPLETE – The ME notifies the cell that there was a change in one of its accept types.
- UPD_ME_STATUS – The ME notifies the cell that its state has changed.

5.1.3 Results

The results file, created by AVAIL, is transferred from the CMM computer to the cell computer where it is used for collecting long-term statistics and determining if the part is in or out of tolerance. The AVAIL programmer has several report formats to choose from. A typical report is as follows:

FEATURE	ACCEPTED	MEASURED	UPP-TOL	LOW-TOL	DEVIATION	OUT/TOL
CIRCLE1	2.00	1.996	0.005	-0.005	0.004	—*—
CIRCLE2	2.00	1.977	0.005	-0.005	0.023	0.018

Each feature of the part that is being measured is given a name by the programmer to uniquely identify it like circle1 and circle2. For each feature, the programmer specifies the accepted value and the upper and lower tolerances. The measured value and deviation are calculated. Note that the last column of the report shows that the feature called CICRLE1 is within the specified tolerance, while the feature CICRLE2 is not.

On the CMM computer, the results file is place in a directory called:

/usr/avail/part/**PART_NAME**

where **PART_NAME** is the name of the part.

After the reply message RUN_COMPLETE is sent, the file is transferred to the cell using Kermit via the RS232 connection between Spruce, the cell computer, and the CMM computer.

Once the file is transferred to the cell computer, the Lot Entity associated with the physical lot (the part being measured) parses the file and places the results of ceratin features in the cell database to collect statistics. The AVAIL programmer can speicify which features are to be used for statistics by placing a special character, such ans an asterisk, as the first character in the feature name in the llf. In a similar manner, the programmer can notify the cell that a particular feature is critical and must be within tolerance for the part to be accepted.

5.2. Output Specification

5.2.1 Identification of Output Data

In section 2–3, two outputs of the machine entity were identified: reply messages and CMM commands. This section discusses both in detail.

5.2.2 Reply Messages

A reply message is sent via message-queues from the ME to the WC in response to a message. The reply consists of a message_id and an ack_code; the other fields of the structure are not used. The valid ack_codes are ACK (a positive one) and NACK (a negative one). The following are the valid message_id's for replies:

- ME_START_COMPLETE: Notifies the WC wether the CMM startup sequence was successful or not.
- ME_SHUT_COMPLETE: Notifies the WC wether the CMM shutdown sequence was successful or not.
- SETUP_RUN_COMPLETE: Notifies the WC when the recipe download is done.
- RUN_COMPLETE: The ME notifies the cell that the execution of program X is complete and that the results are ready to be transferred to the "cell".

5.2.3 CMM Commands

CMM commands are the outputs of the machine entity that invokes the functions of the CMM. The commands generated by the machine entity are piped to AVAIL, the measurement and control software provided by the manufacturer. The commands are actually character strings that correspond to the menu selections of AVAIL.

The following are the sequence of characters generated by the machine entity upon recieving a message from the WC.

- ME_START: "AY" & "\rA*"
  - "A" Home Machine
  - "Y" answer Yes to confirm selection

The User is notified via cell alarm to Press the Machine Start button on the machine. After the alarm is cleared, the homing sequence continues as follows:

  - "\r" A carriage return to continue the homing sequence
  - "A" Select "Home all Axes"
  - "*" Return to the main menu
- ME_SHUT: "EY"
  - "E" Initiate shutdown
  - "Y" Yes to confirm shutdown
- ME_SETUP_RUN: "1filename\rY*"
  - "1" Set part program
  - filename: the name of the part program
  - "\r" A carriage return
  - "Y" Confirm selection of part program
  - "*" Return to the main menu
- ME_START_RUN: "DA"
  - "D" Enter measurement sub-menu
  - "A" Execute part program

5.3. System Database/File Structure Specification

5.3.1 Identification of Database/Files

Machine entities use six tables in the common cell database. These tables are described below.

OP_MACHINE_RECIPE:

Column Name	Type	Nulls	Defaults
op_type	varchar 12	no	no
machine_id	varchar 12	no	no
recipe_id	varchar 20	no	no

MACHINE_RUN_RESULTS:

Column Name	Type	Nulls	Defaults
machine_run_id	integer 4	no	no
param_id	varchar 12	no	no
value	varchar 25	yes	no
set_name	varchar 25	yes	no
instance	integer 2	yes	no

CONFIGURE_ACCEPT_TYPES

Column Name	Type	Nulls	Defaults
machine_id	varchar 12	no	no
op_type	varchar 12	no	no
op_function	varchar 80	no	no
op_sequence	integer 1	no	no

MACHINE_STATUS:

Column Name	Type	Nulls	Defaults
machine_id	varchar 12	no	no
machine_run_id	integer 4	yes	no
recipe_id	varchar 20	yes	no
start_time	date	yes	no
end_time	date	yes	no
me_status	varchar 16	no	no

note_id	integer 4	yes	no
dts	date	no	no

ALARM_DESCRIPTION:

Column Name	Type	Nulls	Defaults
robot_alarm	varchar 80	yes	no
alarm_id	varchar 12	yes	no
description	varchar 40	no	no

The OP_MACHINE_RECIPE table contains the operation types for every machine/recipe pair. The list of valid operation types is listed in the PARAMETER_STRING_VALUES table. For example, the recipes that are run on the CMM must have operation types *measure* and *complex*, or the cell will complain. The MACHINE_RUN_RESULTS table is used to store the parsed output of the results file. The CONFIGURE_ACCEPT_TYPES table specifies the function (process) to change accept types for each machine/operation type pair. An entry is appended to the MACHINE_STATUS table every time the machine's status changes. The ALARM_DESCRIPTION table is used to map the alarms generated by a machine to an alarm identifer.

5.3.2 Database Creation and Update Procedure

The database at NJIT is a copy of the database installed at Siemens. Since Ingres was updated to version 6.4 at Siemens and NJIT is using version 6.3, the "copydb" command at Siemens could not be used. Instead, using the Ingres network utility, netu, a virtual database named *siemens* was defined at NJIT to point to the database on *kong.siemens.com*. This allowed the 6.3 version of the software to be run against the 6.4 database at Siemens. For example, to query the releases database at Siemens from a workstation at NJIT, one would simply type:

```
sql siemens::releases
```

The first step in creating the Ingres database is to setup up the system catalogs by typing:

```
createdb releases
```

After the database is created, SQL scripts for the two owners, namely, CELL and RELEASES, are created by typing:

```
copydb -ucell -c siemens::releases
mv copy.out copy-cell.out
mv copy.in copy-cell.in
```

and,

```
copydb -ureleases -c siemens::releases
mv copy.out copy-rel.out
mv copy.in copy-rel.in
```

Next, the human interface ABF application is coppied by typing:

```
copyapp out siemens::releases hi

mv iicopyapp.tmp copyapp-hi.in
```

Now that all the SQL scripts are created, the *out* scripts are run first to copy the database to ascii text files. Then the *in* scripts are run to extract the information contained in the text files into the new database. The seqence of commands is as follows:

```
/* run out scripts */

sql siemens::releases < copy-rel.out

sql siemens::releases <copy-cell.out

/* run in scripts */

copyapp in -q releases copyapp-hi.in

sql releases <copy-rel.in > RESULTS-REL

sql releases <copy-cell.in > RESULTS-CELL
```

The output of the SQL interpreter is redirected to the files RESULTS-REL and RESULTS-CELL so they can be easily scanned for errors. All the above files are in

```
~cell/releases/db.
```

6. Module Design Specifications

A typical machine entity is comprised of two modules: a WC module and a ME module. The CMM machine entity has an additional module, named RS232_TO_AVAIL, that is specifically designed for the CMM.

This section defines the functional and operational specifications of the ME and RS232_TO_AVAIL modules. The WC module is discussed by its developer, Peter Murray, in his thesis.

6.1. ME Module Functional Specification

6.1.1 Functions Performed

The ME module performs the following functions:

- Message queue setup
- Send & receive messages
- Serial port setup
- Send & receive via serial port
- Control loop – message queue & serial port polling

6.1.2 Module Interface Specifications

Each of the above functions are implemented as one or more C language functions. Their functional interface is described here.

Message Queue Setup

A function to setup message queues has been defined and is accessible by both the WC and the ME. If successful, the function returns an integer that is the handle to the queue. If a fatal error occurs (i.e. the system calls to create a message queue failed), the functions exits and logs the error. The argument to the function is the name of the process in which the queue is to be shared. The protocol used to setup the queue is as follows:

- The WC process uses its process id as a key in the system calls to setup the queue
- The ME process is given the WC's process name on the command line which is used to get the WC's process id. By using the same key, the queue is established between them.

```
int init_mq(char *process_name)
```

Message Queue Receive

Two functions are used to read the queue; one is a blocking read, the other polls. Both return an integer that indicates either a message was received, no message was received, or an error condition. The arguments to both functions are

- msgid: the handle to the queue returned by init_mq()

- TO_ME or FROM_ME: indicates the direction of the message, and
- wc_me_message: the contents of the message.

```
int mq_read_nowait(int msgid, int TO_ME, (char *)wc_me_message)
int mq_read_wait(int msgid, int FROM_ME, (char *)wc_me_message)
```

Message Queue Send

The function used to send a message queue behaves in a similar fashion and takes the same arguments as the read functions. The prototype is as follows:

```
int mq_send(int msgid, int FROM_ME, (char *)wc_me_message)
```

Serial Port Initialization

The following function initializes the RS232 port to raw mode. It returns a void; any error condition that can occur is fatal and therefore will exit immediately. The arguments to the function are:

- channel: a pointer to an integer (a file descriptor) that is the handle to the RS232 port.
- savetty: a pointer to the termio structure that contains the port configuration information before the port was set to raw mode. Used to restore the original port configuration.

```
void termchan(int *channel, struct termio *savetty)
```

Serial Port Read

The function term_read() polls the RS232 port. Term_read() returns an integer that represents either a time-out (i.e. no data to be read), an error or a normal condition. The arguments are

- channel: a handle to the open and configured RS232 port.
- time_out: the time, in 10^{ths} of a second, to poll the port.
- buf_length: the number of bytes read
- buf: a char array containing the data read.

```
int term_read(int *channel, int time_out, int *buf_length, char *buf)
```

Serial Port Send

The function used by the ME to send data to the RS232_TO_AVAIL process is cwrite(). The function returns a void; if an error occurs it is logged and the function continues. The arguments to the function are:

- channel: the handle to the port returned by termchan()
- a literal string: the contents to be sent.

Control Loop

The control loop, which is placed in main(), polls the RS232 port and the message queue approximately every 2 seconds. Main() returns an integer in which no significance is placed. One command line argument is expected: the name of the WC process which is used for message queue initialization. When a message is received via message queue, a corresponding message is sent to the RS232 module. See section 6–2–2.

- argv[2]: the process name of the WC

int main(int argc, char *argv[]);

6.2. ME Module Operational Specification

6.2.1 Data Specification (Variable Dictionary)

- struct WC_ME_MSG wc_me_message – holds the contents of a message queue message. The fields of the structure are defined below.
- int msgid – the handle returned during message queue initialization
- int ACK – indicates a positive result
- int NACK – indicates a negative result
- int TO_ME – specifies direction of message is from WC to the ME
- int FROM_ME – specifies direction of message is from ME to WC
- int IS_NO_MESSAGE – indicates that no message was read from the queue
- int channel – the file descriptor corresponding to the RS232 port
- int msg_id – a field of wc_me_message; valid id's are discussed in a later section.
- char* file_name – a field of wc_me_message
- char* recipe_id – a field of wc_me_message
- int ack_code – a field of wc_me_message
- char* alarm_text – a field of wc_me_message
- long status – the return value of term_read(); either NORMAL, TIMEOUT, or ERROR.
- char* result_file – specifies where the result file is placed
- FILE** result_fp – the file pointer to the results file
- char ch – used by read() to contain the one character read in.
- char* buf_temp – buffer that is built from concatenating the characters read in one at a time
- int length_buf – length of the temporary buffer buf_temp.

6.2.2 Algorithm Specification (Pseudocode)

ME Control Loop

```
msgid = init_mq(process_name);  /* message queues */
setup_rs232_port();             /* calls termchan() */
forever
{
  while (message_queue_read_nowait() == NOMESSAGE)
    poll_rs232_port();

  /* got a message from message queue */
  switch(message_id)
  {
```

```
    case STARTUP:
      /* startup procedure */
    case SHUTDOWN:
      /* shutdown procedure */
    case SETUP_RUN:
      /* download recipe */
    case START_RUN:
      /* execute recipe */
    default:
      /* error, log message received */
    } /* end of switch */
  }/* end of forever */
```

6.3. RS232–TO–AVAIL Functional Specification

6.3.1 Functions Performed

The RS232_TO_AVAIL module performs the following functions:

- Launches AVAIL as a child process and establishes a pipe in which commands are passed
- Initializes the serial port
- Control Loop – listens at port and sends commands

6.3.2 Module Interface Specifications

Each of the above functions are implemented as one or more C language functions. Their functional interface is described here.

Launch AVAIL

The function set_pipe() in RS232_TO_AVAIL, launches AVAIL as a child process and redirects its standard input to a Unix pipe. The pipe is used to transfer the commands received from the serial port to AVAIL. The function has no arguments and returns void.

```
void set_pipe(void)
```

Serial Port Initialization

The function set_terminal() initializes serial port 2A to raw mode. Read()'s are blocking and are satisfied after 1 character is read. The arguments are:

- size: the MIN number of characters to be read; currently set to 1
- time: the intercharacter timer; currently set to 0

```
void set_terminal(int size, int time)
```

Serial Port Send

One function is used to send data through the pipe to AVAIL and to the ME via the serial port. The arguments are:

- direction: the handle to either the pipe or the serial port
- buffer: the contents to be sent

```
void output(int direction, char *buffer)
```

Control Loop

The control loop is placed in a function called listen(). Listen() performs a blocking read on the serial port. When a command is read, the corresponding AVAIL menu string is generated and sent.

6.4. RS232–TO–AVAIL Operational Specification

6.4.1 Data Specification (Variable Dictionary)

- int DATA_OUT – alias for the handle to the pipe
- int PORT_OUT – alias for the handle to the serial port
- int ERR_OUT – specifies the device in which error messages are written
- int TRUE – used for loops
- int FALSE – used for loops
- int fd – the file descriptor (handle) for the serial port
- char buffer[] – buffer to hold data from serial port
- char err_buf[] – the error buffer
- int fd[2] – the file descriptor (handle) for the pipe to AVAIL

6.4.2 Algorithm Specification (Pseudocode)

Listen()

```
forever
{
  read(fd, buffer, 1); /* read rs232 port, blocking */
  if (!strcmp(buffer, "DOWNLOAD"))
    /* invoke kermit for file transfer */
  else if (!strcmp(buffer, "UPLOAD"))
    /* invoke kermit for file transfer */
  else if (!strcmp(buffer, "DOWNLOAD"))
    /* invoke kermit for file transfer */
  else if (!strcmp(buffer, "SETUPRUN"))
    output(DATA_OUT, buffer);
  else if (!strcmp(buffer, "STARTRUN"))
    {
    output(DATA_OUT, buffer);
    /* determine if part is good or bad */
    if (AVAIL report file exists)
      output(PORT_OUT, "FAIL");
    else
      output(PORT_OUT, "PASS");
    }
} /* end of forever */
```

7. System Demonstration

7.1. Functions to be Demonstrated

In section 2–1 the following functions of the CMM were described:

- *homing the machine* – part of the initialization process of the machine; must be done after every shutdown/power-down.
- *selecting part programs* – allows the user to select an existing measurement program.
- *execution of part programs* – executes the previously selected measurement program; the selecting and executing of part programs is synchronized by the software.
- *turning privileges on and off* – a method of providing password protection to certain file maintenance functions.
- *part program file maintenance* – including, deleting, updating, and system backup of part programs
- *shutdown of the CMM* – takes the machine off line and unavailable for use.

All of the above functions have been implemented and tested as of December 12, 1991, in anticipation of the demonstration for the Chief Executive Officer of Siemens Corporate Research in January of 1992. The next sections describe and justify the test cases performed in preparation for the demonstration.

7.2. Demonstration Setup

The cell software startup procedure begins by verifying that both ISIS and Ingres are running on Spruce. ISIS can be started by any user who as access to the cell account. Ingres, however, can only be started by the superuser.

After Ingres and ISIS are running, the cell software is started using a Unix C-shell script[1] called *start.csh*. The start script starts the cell modules and sets their logging level. Each module upon startup joins a common ISIS group (message bus), initializes its communication filters for the *startup* conversation, and connects to the cell database. The last module started is the Human Interface (HI) module.

To shutdown the cell software, a C-shell script called *stop.csh* is provided. The script sends a user-defined signal to each of the cell modules. In response to the signal, the modules run a signal handler that will allow the process to die gracefully (i.e. delete shared memory that would otherwise stay allocated until explicitly deleted or until the system was rebooted).

1 See Appendix D for the listings of shell scripts.

Another script of interest is called *dbclean.csh*. This script deletes remnant data from the database after, for instance, the cell software malfunctions. The script is intended to be a debugging tool only.

7.3. Description of Test Cases

The first function to be implemented and tested was the homing sequence. The reason is two fold. First, the machine must be homed before any other function will work. Second, the homing sequence is the most complicated: it uses the Cell Alarm Server, Human Interface (including the status line server mechanism), and the Machine Entity Modules, making it an attractive, thorough test case.

The function was tested by starting the cell as described above, and then initiating the sequence at the Human Interface[2] as follows. At the opening HI screen, the TA code STRT was entered to jump to the "Start Cell Modules" frame. The startup sequence was started by pressing key R5. The entire cell was started, including the CAS, ERS, LSR, MSCHED, and MS modules.

The MS (Machine Server) module, upon receiving a startup message, sends a secondary startup messages to the Machine Entities listed in the "Machine Entity Startup" frame. The WC, upon receiving a startup message from the MS, initiates the homing sequence.

During the CMM homing sequence, an alarm is forced to notify the user to press the machine start button located on the CMM console. The user is notified via the status line server, an asynchronous communications mechanism, located at the bottom of most HI frames. When the alarm is cleared by the user at the HI frame "Clear Alarm", a message is broadcast by the CAS. The WC responds to the message and sends the final set of commands to the ME. The success of the test is obvious; the machine moves to home position.

The functions select part program, turn privileges on, and execute part program were tested using a Lot Entity script. The procedure went as follows. A work order was defined using the "Work Order Entry" screen. The product called "test" was selected since that was previously set up to run the CMM Lot Entity script. The script was introduced to the cell by selecting Run(L2) in the "Work Order Status" frame. When any Lot Entity is first introduced to the cell it registers with LSR, the Lot Server, and then waits for what is called an Import. The Import request is displayed on the status line server and is also listed in the "Dispatch List".

When the lot was imported using the "Import Lot" frame, the lot script continued its execution. The script successfully completed, which includes being scheduled by the Machine Scheduler module for the machine, selecting a part program (hard coded in the script for testing purposes), and executing the part program.

2 The Human Interface screens referred to are in Appendix A, and are placed in the order in which they are referenced.

7.4. Test Run Results

The results show that it is possible to develop a Machine Entity for the cell in a relatively short period of time. Further, it has been shown that a sizable portion of the Machine Entity code is reusable even for unrelated machines, thus making it feasible to provide a generic RS232 based Machine Entity.

The results also reveal a problem with the cell software. In several of the tests conducted, the log files reported that conversations were being aborted by ISIS because either all the modules did not receive it, or because all of the modules that were interested in the conversation were not able to reply in the specified amount of time. As a result of the time-outs, the database contained data inconsistent with the actual state of the machine. The actual cause of the problem has not been found. We suspect its either a database deadlock (which would effectively stop the module from receiving or replying to messages) or system overload.

8. Conclusions

8.1. Summary

The installation of the Generic Workcell software at New Jersey Institute of Technology signifies the end of the author's two year participation in the project. The project will continue, however, under the direction of Professor Alexander Stoyenko of the Computer Science department and Reggie Cauldill, director of the Center for Manufacturing Systems.

The implementations of the Generic Workcell at Siemens, NJIT, and Germany have proven that the entity server model and the software architecture used is generic. It is hoped by the Siemens research scientists to have the opportunity to evaluate the model and software architecture in a variety of non-factory automation related areas such as Patient Monitoring and Medical Systems.

8.2. Problems Encountered and Solved

Two types of problems were encountered: those that were related to the computer system hardware and those that were related to the Generic Workcell software.

Computer system problems occurred because several of our requests were either unable to be fulfilled, delayed, or they were done incorrectly. For example, the AT+T version 2.2 of the C++ compiler was requested, but a different version was installed. The installation of Ingres 6.3 was delayed since only version 6.1 had been installed at NJIT to date. Hardware malfunctions (specifically repeaters) made the network unreachable for several days. The installation of the Andrew File System made the system unavailable for two weeks, and in the process, the Generic Workcell platform was moved to a different computer. Also, the original goal was to have a workstation on the factory floor that would be the cell computer. This was not provided since a workstation could not be found with the needed resources.

The solutions (or "patches") to the above problems sometimes cascaded into larger, unforseeable problems. For example, since Spruce, rather than Truth had to be used as the cell computer, the serial connection for the Machine Entity was 100 feet as opposed to 5 feet.

Only minor problems were encountered with the cell software. For instance, there was a compatibility problem in three of the Ingres ABF frames. The programmer used constructs that were introduced in version 6.4 and not available in 6.3. The code was modified to use only version 6.3 constructs. The Ingres version difference also caused problems during the porting of the database. The solution is explained in detail in section 5–3.

These problems were eventually overcome, and the platform used to develop and test the CMM Machine Entity was stabilized.

8.3. Suggestions for Future Extensions to Project

The original goals[1] of the Generic Workcell Project were to

- provide a generic system architecture and software toolbox to develop specific workcells rapidly
- Allow for independent or cooperative operation among workcells
- Design a flexible and exstensible workcell system:
  - support the reconfiguration (addition and removal) of equipment
  - support changes and additions to the manufacturing function with few modifications to software
  - Enhance the quality of the product

The Generic Workcell software installed at NJIT provides the CMS department with the opportunity to meet these goals in a relatively short period of time; several years ahead of their original schedule to integrate the factory floor. To achieve full integration of the factory floor, the following needs to be accomplished:

- Machine Entities for the Automatic Storage and Retrieval System, the Mazak Turning Center, the Charmille Technologies EDM, and the AT+T Flexible Workstation need to be written. The current implementation of the CMM Machine Entity will assist in the development since it was designed in a modular fashion (i.e. the CMM specific code is easily separated, leaving a generic RS232 machine interface).
- A document specifying the purpose of the Cell, including transport systems, identification systems, operator work areas, descriptions of the products and manufacturing process plans.
- The development of the Lot Entity Scripts to perform the tasks outlined in the functional requirements document.

By tending to these immediate needs, the factory floor will be integrated in a short period of time, and as a side effect, the persons responsible will become proficient in the design of the workcell and would (should) ultimately be responsible for continuing the research aspects of the project.

8.3.1 Enhancement Suggestions

The benefits of C++ is not being used in the modules. Unfortunately, C++ has merely been used as a better C, no objects other than those provided by the Conversation Tool exists. Clearly, the nature of the project and the software architecture will benefit using the Object Oriented paradigm.

The current implementations of the project use the Ingres Relational Database. This may not be practical since the cost of Ingres is high. Research needs to be done to provide an cost effective alternative to Ingres, making the project more marketable.

1 Taken from presentation slides authored by Paul J. Bruschi & Dan Wolfson

9. Key Words, Phrases, and Acronyms

9.0.1 Modules of the Generic Workcell

- *HI* – Human Interface to the Generic Workcell
- *ERS* – Equipment Recipe Sever
- *LS* – Lot Server
- *MSCHED* – Machine Scheduler
- *CAS* – Cell Alarm Server
- *MS* – Machine Server
- *WC & ME* – Machine Entity

9.0.2 Software Tools Developed

- *The Conversation Tool* – (Ctool) a software tool that provides a common ISIS interface across cell modules
- *The Conversation Monitor* – (CMON) a software tool that captures ISIS messages; used during debugging of modules

9.0.3 Machines

- *CMM* – Coordinate Measurement Machine
- *MAZAK* – Mazak Turning Center
- *FWS* – AT&T Flexible Work Station

9.0.4 Other Project Related Terms

- *GWC* – Generic Workcell
- *SCR* – Siemens Corporate Research, Inc., Princeton
- *CMS* – Center for Manufacturing Systems at NJIT
- *Recipe* – a synonym for part program
- *Ingres* – A commercial relational database
- *ISIS* – Message bus communications software, public domain
- *Lot Script* – a computer task that coordinates the manufacturing processes of a lot
- *Lot Entity* – the computer representation of a lot; associated with one or more lot scripts.
- *Machine Entity* – an interface between a machine and the cell control software
- *Hierarchical Control Architecture* – the traditional method for control architectures; centralized control with complex interrelationships explicitly programmed into the system
- *Heterarchical Contol Architecture (non-hierarchical)* – the architecture used in this project; distributed control achieved using independent processes

Works Cited

1) Wolfson, D. and P. Bruschi. A *Reconfigurable Generic Workcell Architecture.* Siemens Corporate Research. July 9, 1990.

2) Mullender, S. *Distributed Systems.* ACM Press, New York, New York. 1989.

3) Wolfson, D. and P. Bruschi. *A Generic Workcell Controller.* New Jersey Institute of Technology Symposium on Advanced Manufacturing. May 1990.

4) Stoyenko, A. and R. Meyer, et.al. *Functional Requirements for a CMS Generic Workcell.* New Jersey Institute of Technology, department of Computer and Information Sciences. June, 1991.

5) Dhaliwal, J and P. Bruschi. Generic *Workcell Machine Entity Specification.* Siemens Corporate Research. Version 1. January, 1991.

6) Histon, B. *Distributed System Infrastructure for a Prolific Manufacturing Enterprise.* IEEE Publications, Nov., 1991.

7) Rana, S. and S.K. Taneja. *A Distributed Architecture for Automated Manufacturing Systems.* The International Journal of Advanced Manufacturing Technology, Vol. 3, 1988.

8) Duffie, N. and R. Piper. *Non-hierarchical Control of a Flexible Manufacturing Cell.* Robotics & Computer Integrated Manufacturing, Vol. 3, No. 2, pp. 175–179, 1987.

9) Ingres Corporation, ABF and Windows 4GL Reference Manual. 1991

10) Duffie, N., Chitturi, R., and J. Mou. *Fault-tolerant Heterarchical Control of Heterogeneous Manufacturing System Entities.* Volume 7, No. 4., pp. 325–327, 1988.

11) Duffie, N. A., R.S. Piper, B.J. Humphrey and J.P Hartwick Jr. *Hierarchical and Non-Hierarchical Manufacturing Cell control with Dynamic Part-Oriented Scheduling.* North American Manufacturing Research Conference, pp. 504–507, 1986.

12) Hatvany, J. *Intelligence and Cooperation in Heterarchic Manufacturing Systems.* 16th CIRP International Seminar on Manufacturing Systems. Tokyo 1984.

13) Bjorke, O. *Towards Integrated Manufacturing Systems – Manufacturing Cells and their Subsystems.* Robotics and Computer Integrated Manufacturing. Vol. 1, No. 1, pp. 3–19, 1984.

14) Duffie, N. A. *An Approach to the Design of Distributed Machinery Control Systems.* IEEE Transactions on Industry Applications, Vol. IA-18, No. 4, July/August 1982.

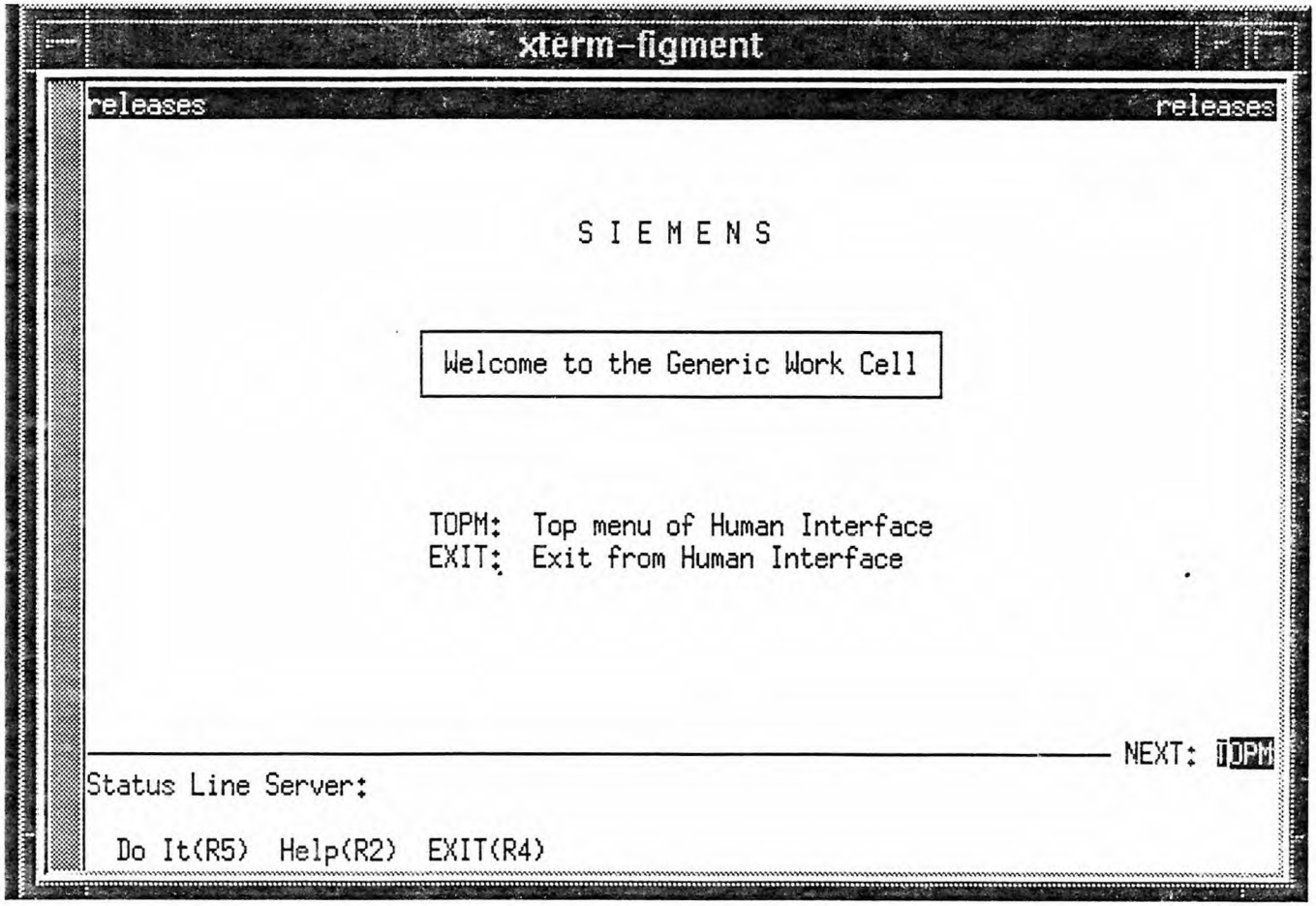
15) Meyer, R. and P. Bruschi. *Generic Workcell Cell Equipment Recipe Server Specification.* Siemens Corporate Research. Version 2.0.

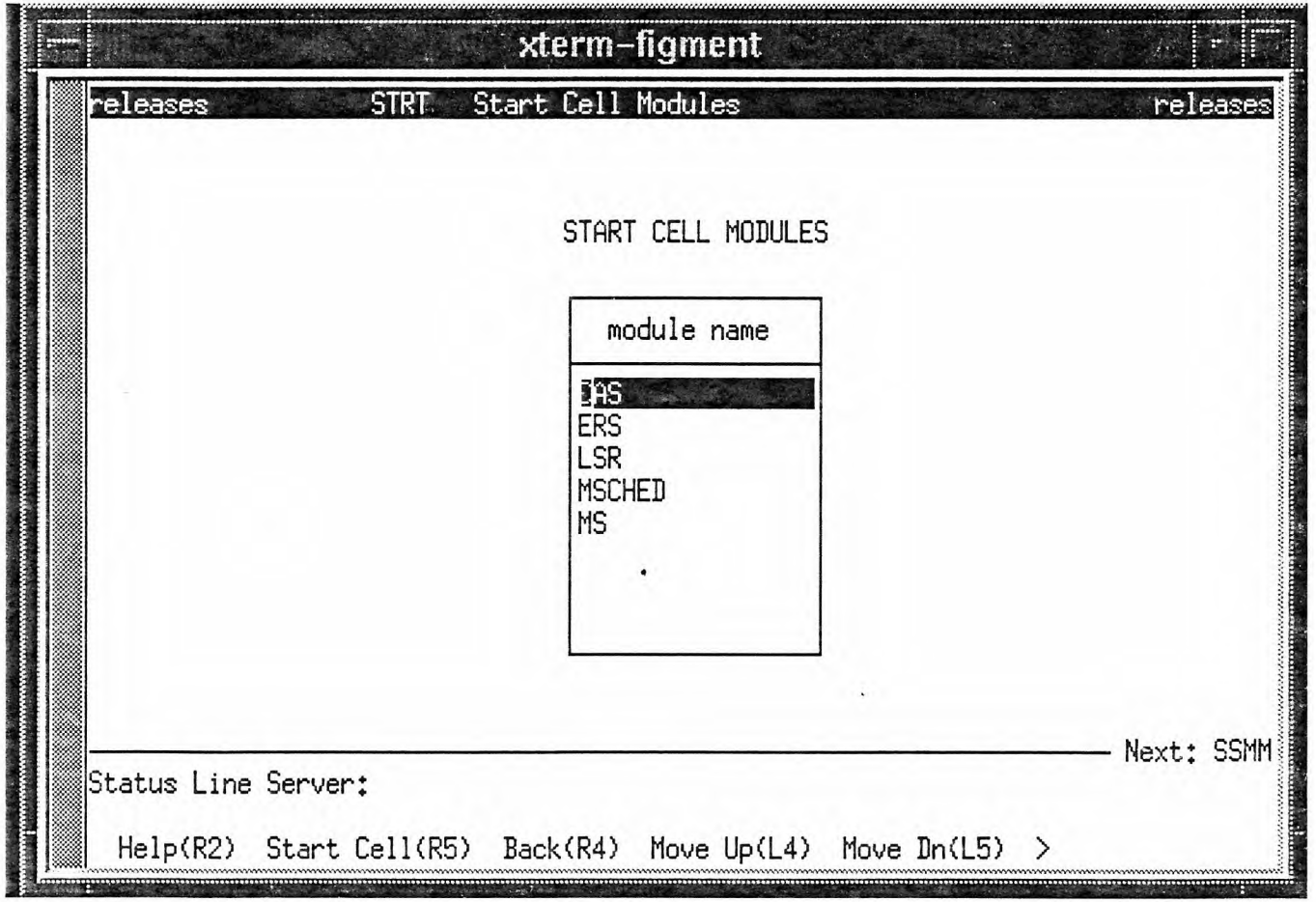
16) Brown and Sharpe, Inc. *Advanced Validator Interface Language Reference Manual.*

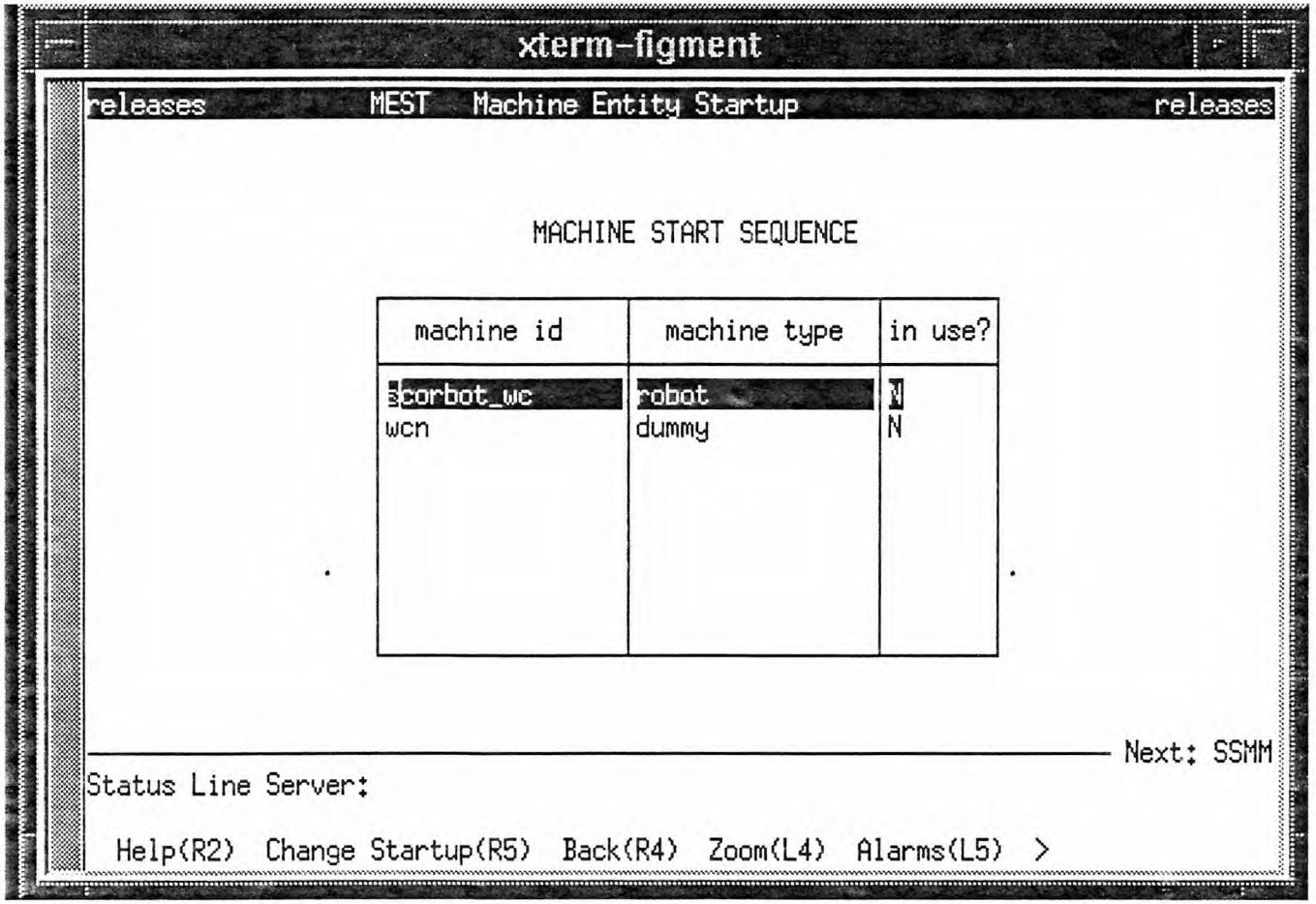
17) Taft, Rick. *Generic Work Cell Conversation Tool Specification.* Siemens Corporate Research. Version 2.0.

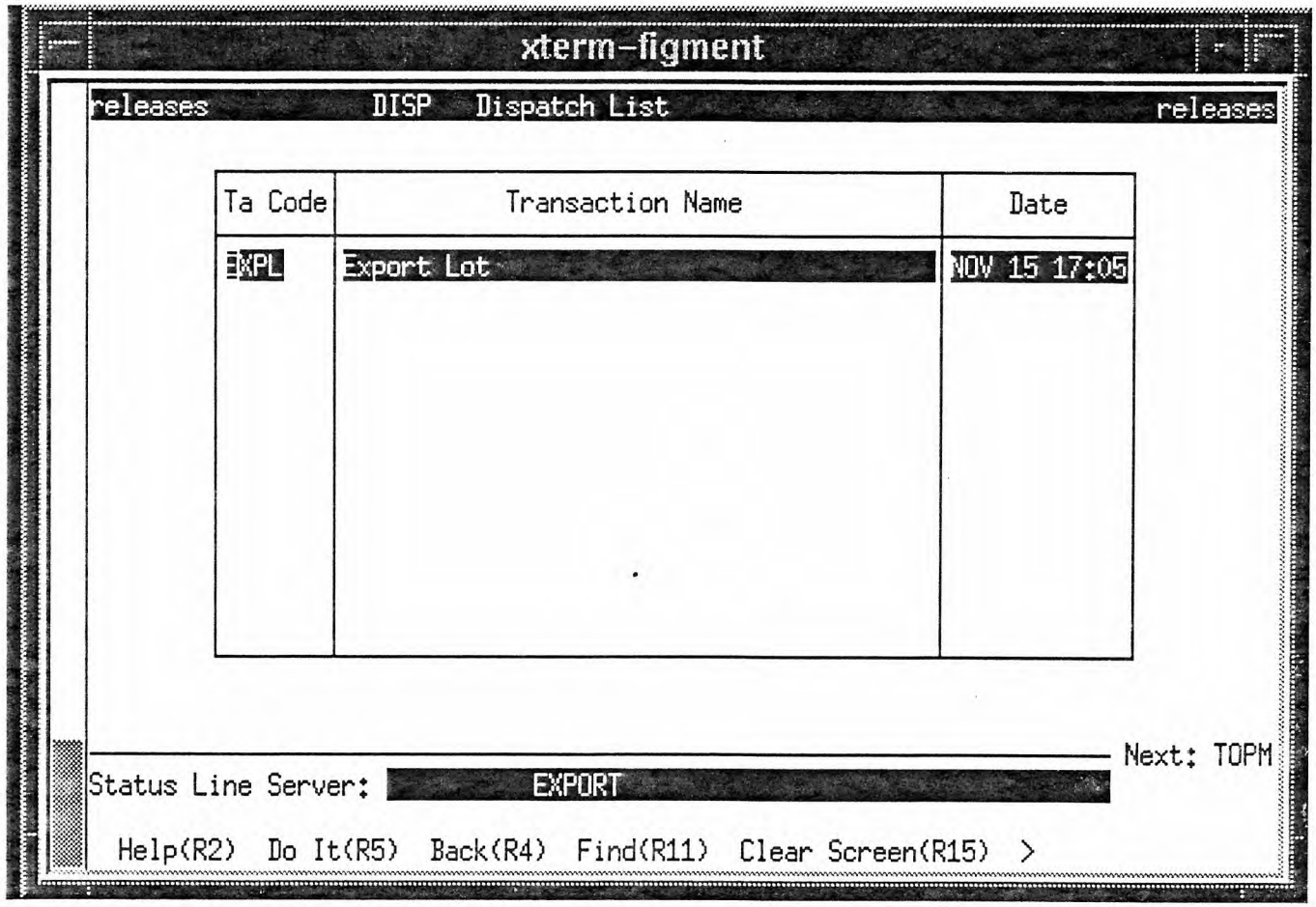
A. HI Screens

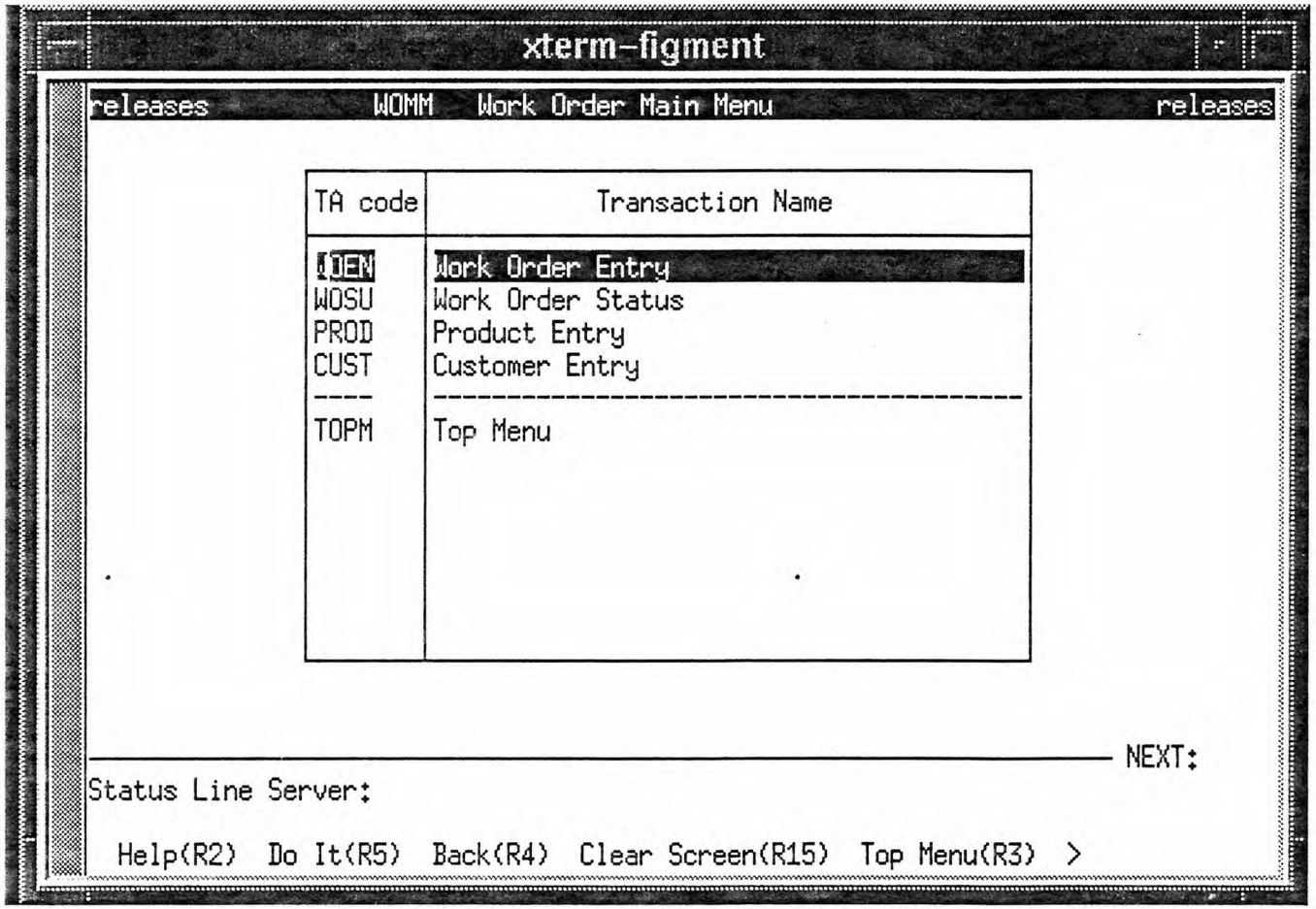
- Introduction Screen
- Start Cell Modules Screen
- Machine Entity Startup Screen
- Dispatch List
- Work Order Main Menu
- Work Order Entry Screen
- Work Order Status Screen
- Dispatch List (showing a lot waiting for import)
- Import Lot Screen
- Transaction Code Entry Screen
- Transaction Code Menu Definition Screen
- Machine Status Screen

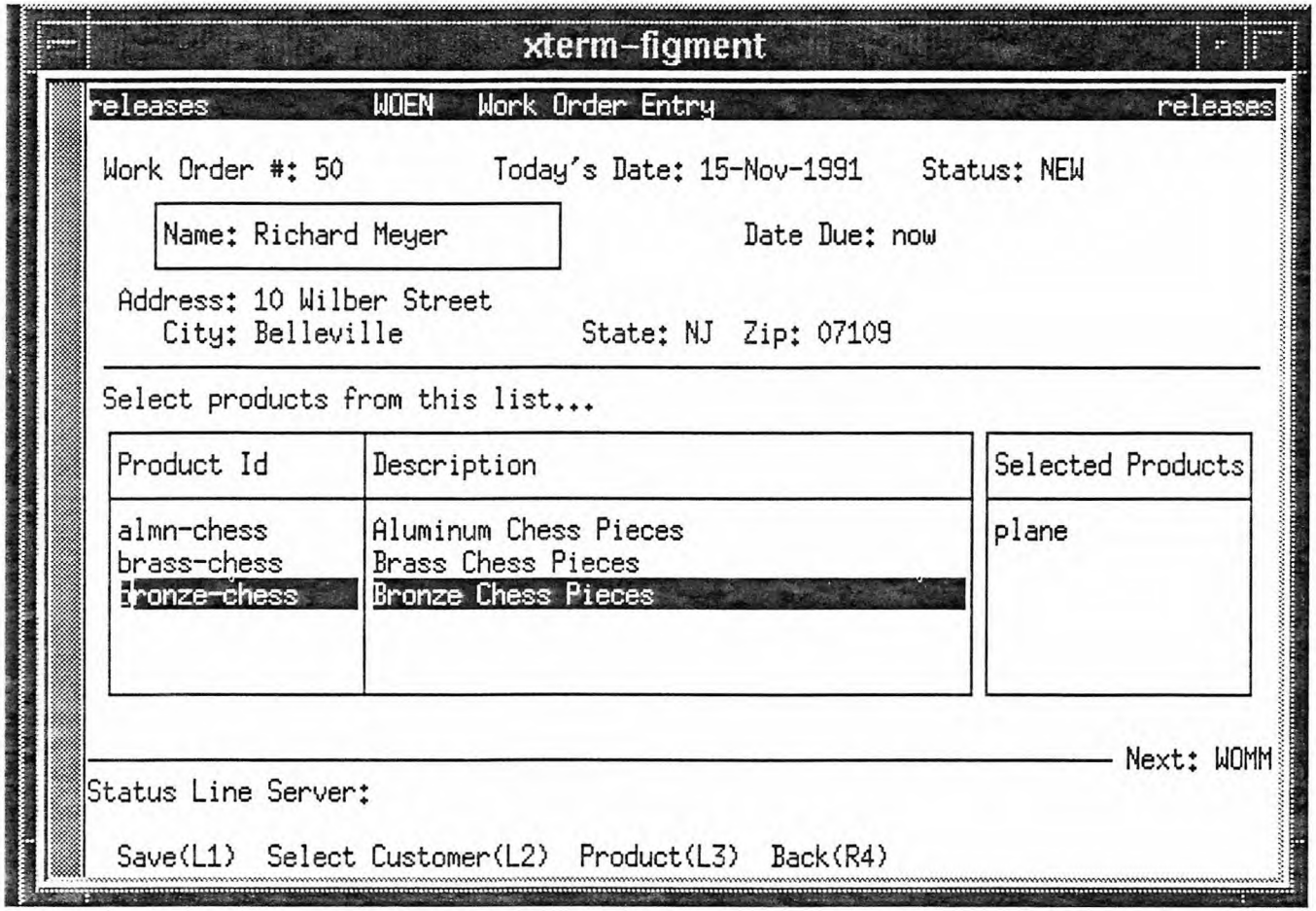

xterm-figment
releases
releases
S I E M E N S
Welcome to the Generic Work Cell
TOPM: Top menu of Human Interface
EXIT: Exit from Human Interface
NEXT: TOPM
Status Line Server:
Do It(R5) Help(R2) EXIT(R4)

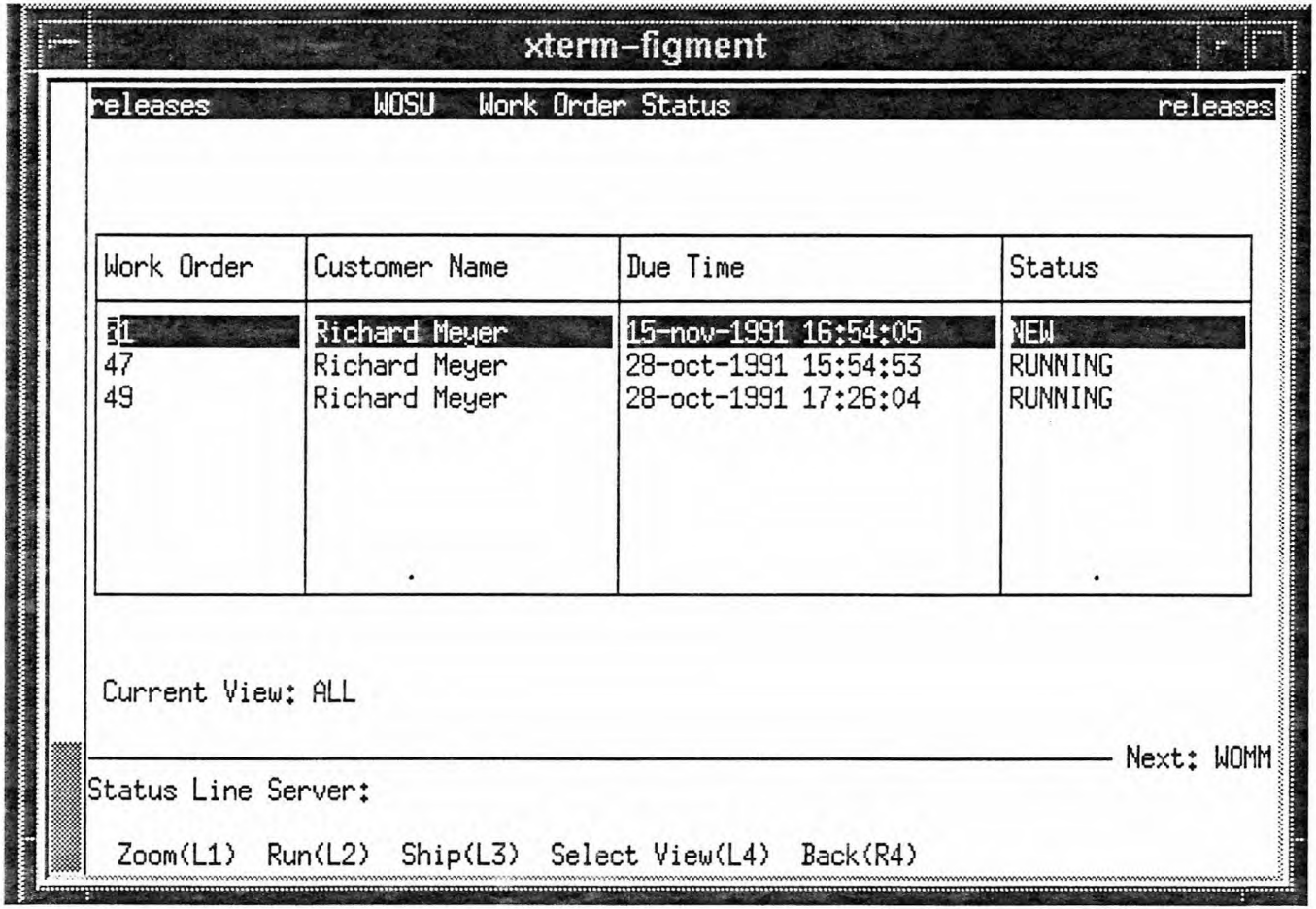

xterm-figment
releases STRT Start Cell Modules releases
START CELL MODULES
module name
AS
ERS
LSR
MSCHED
MS
Next: SSMM
Status Line Server:
Help(R2) Start Cell(R5) Back(R4) Move Up(L4) Move Dn(L5) >

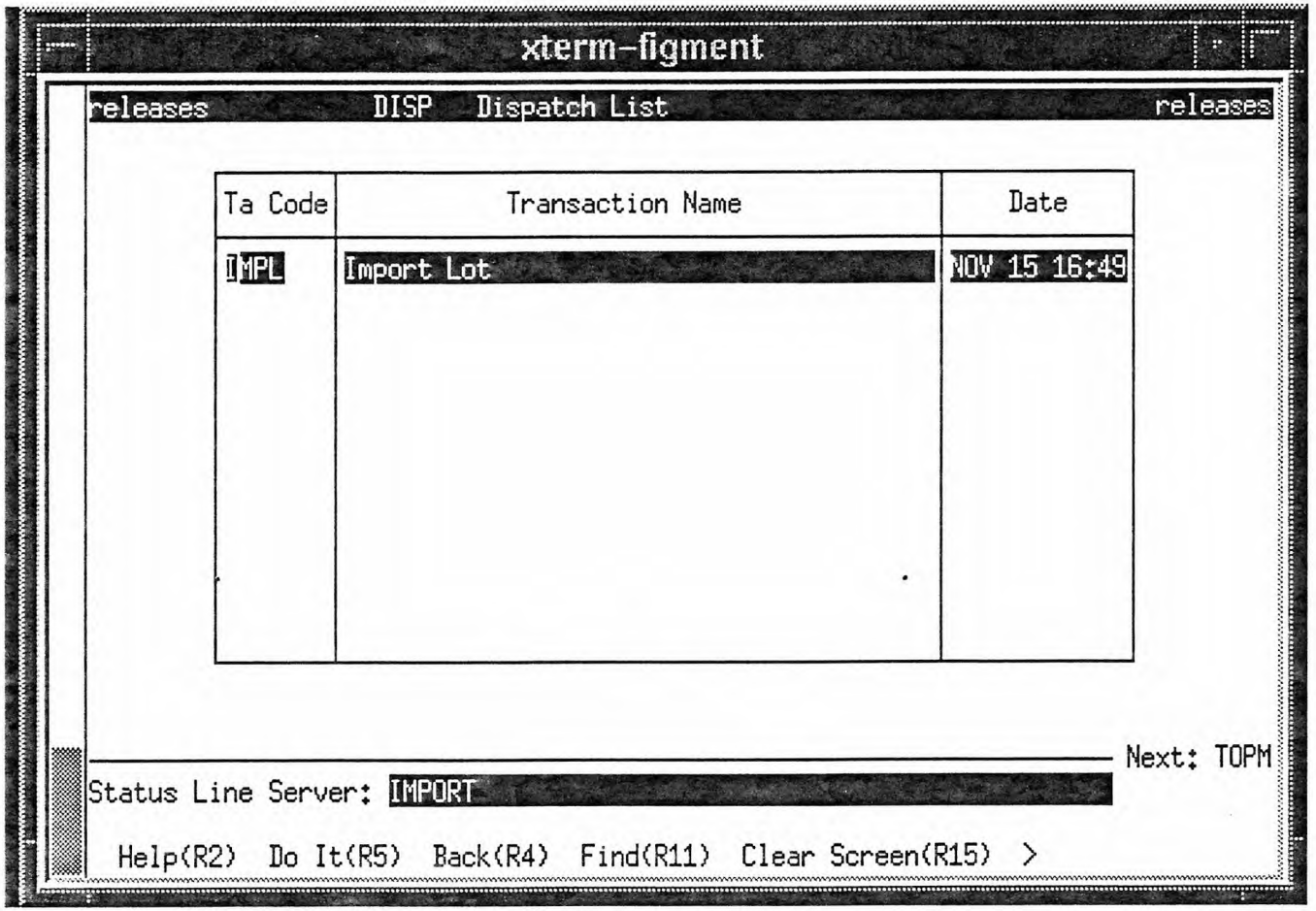

xterm-figment
releases MEST Machine Entity Startup releases
MACHINE START SEQUENCE
machine id
machine type
in use?
scorbot_wc
robot
Y
wcn
dummy
N
Next: SSMM
Status Line Server:
Help(R2) Change Startup(R5) Back(R4) Zoom(L4) Alarms(L5) >

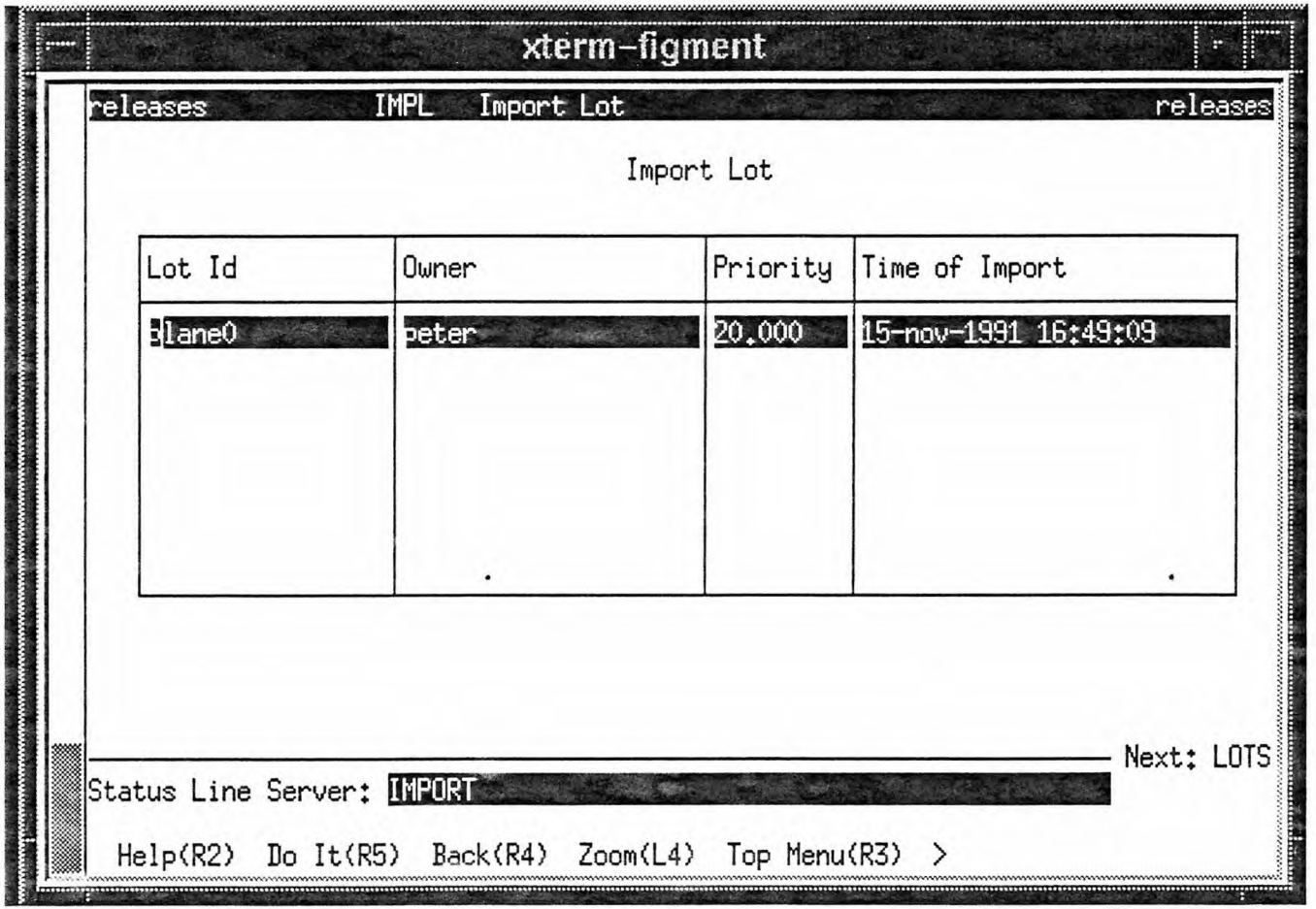

xterm-figment
releases DISP Dispatch List releases
Ta Code
Transaction Name
Date
EXPL
Export Lot
NOV 15 17:05
Next: TOPM
Status Line Server:
EXPORT
Help(R2) Do It(R5) Back(R4) Find(R11) Clear Screen(R15) >

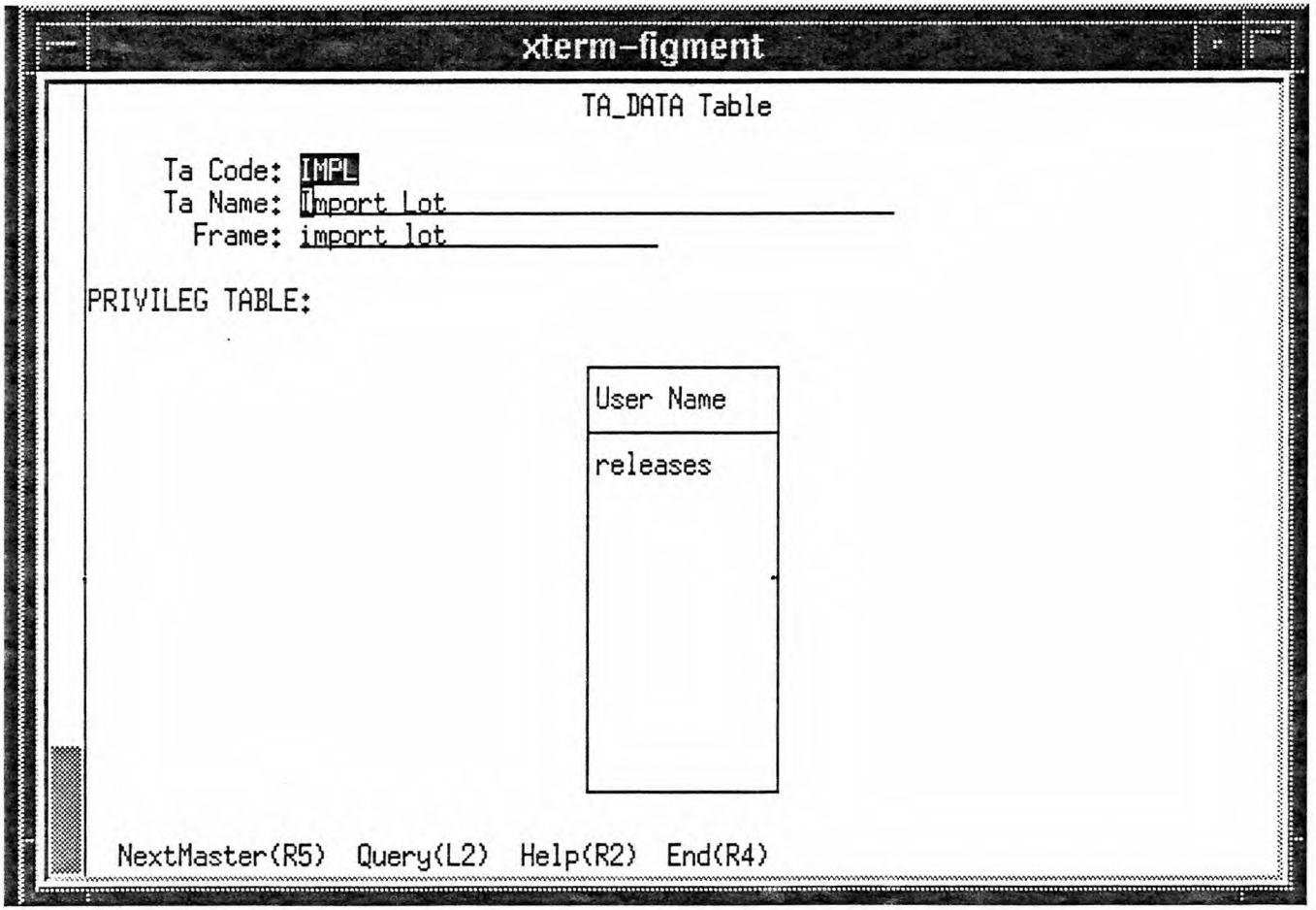

xterm-figment
releases WOMM Work Order Main Menu releases
TA code
Transaction Name
WOEN
Work Order Entry
WOSU
Work Order Status
PROD
Product Entry
CUST
Customer Entry
TOPM
Top Menu
NEXT:
Status Line Server:
Help(R2) Do It(R5) Back(R4) Clear Screen(R15) Top Menu(R3) >

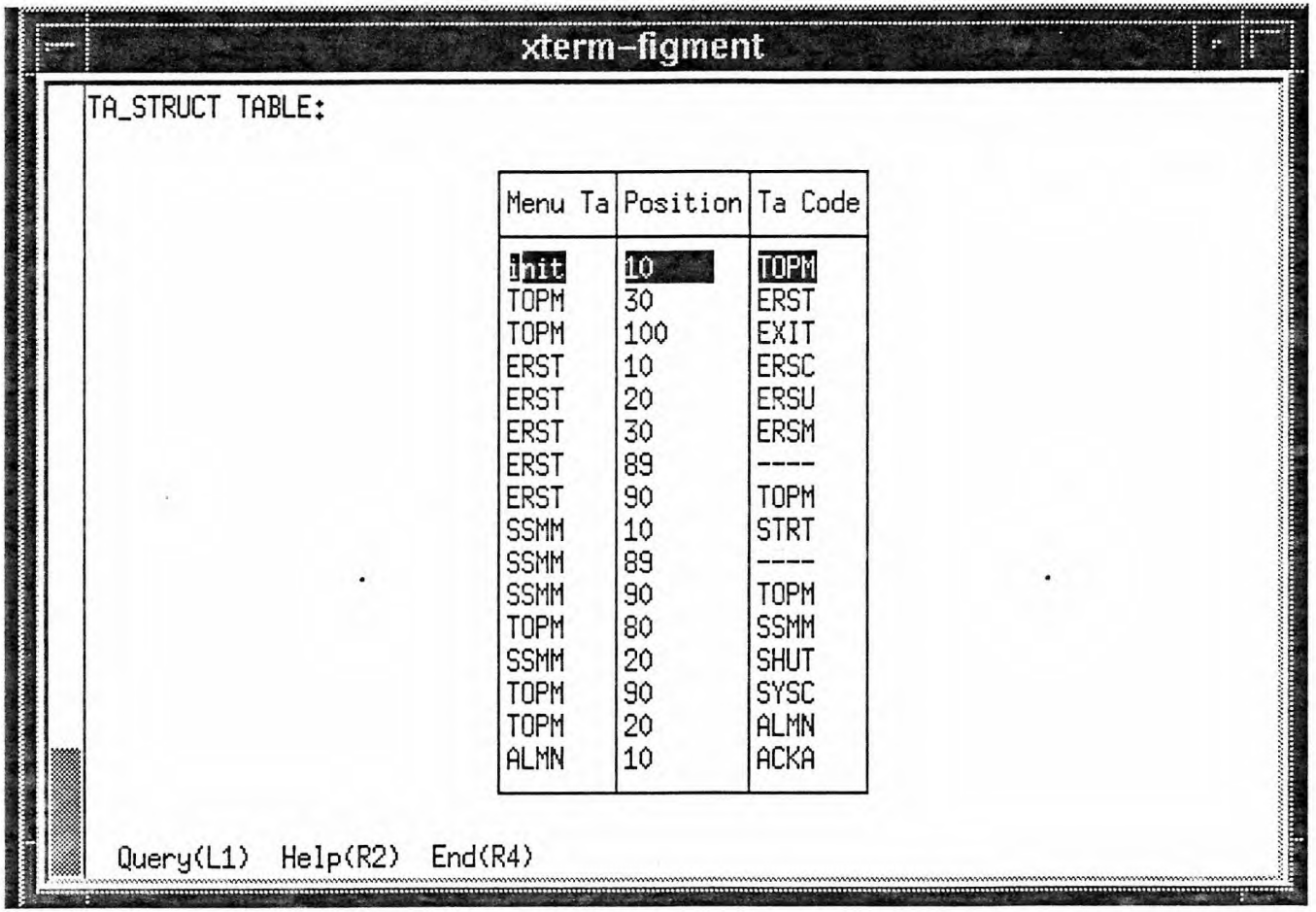

xterm-figment
releases WOEN Work Order Entry releases
Work Order #: 50 Today's Date: 15-Nov-1991 Status: NEW
Name: Richard Meyer
Date Due: now
Address: 10 Wilber Street
City: Belleville State: NJ Zip: 07109
Select products from this list...
Product Id
Description
Selected Products
almn-chess
brass-chess
bronze-chess
Aluminum Chess Pieces
Brass Chess Pieces
Bronze Chess Pieces
plane
Next: WOMM
Status Line Server:
Save(L1) Select Customer(L2) Product(L3) Back(R4)


xterm-figment
releases WOSU Work Order Status releases
Work Order
Customer Name
Due Time
Status
31
Richard Meyer
15-nov-1991 16:54:05
NEW
47
Richard Meyer
28-oct-1991 15:54:53
RUNNING
49
Richard Meyer
28-oct-1991 17:26:04
RUNNING
Current View: ALL
Next: WOMM
Status Line Server:
Zoom(L1) Run(L2) Ship(L3) Select View(L4) Back(R4)


xterm-figment
releases DISP Dispatch List releases
Ta Code
Transaction Name
Date
IMPL
Import Lot
NOV 15 16:49
Next: TOPM
Status Line Server: IMPORT
Help(R2) Do It(R5) Back(R4) Find(R11) Clear Screen(R15) >

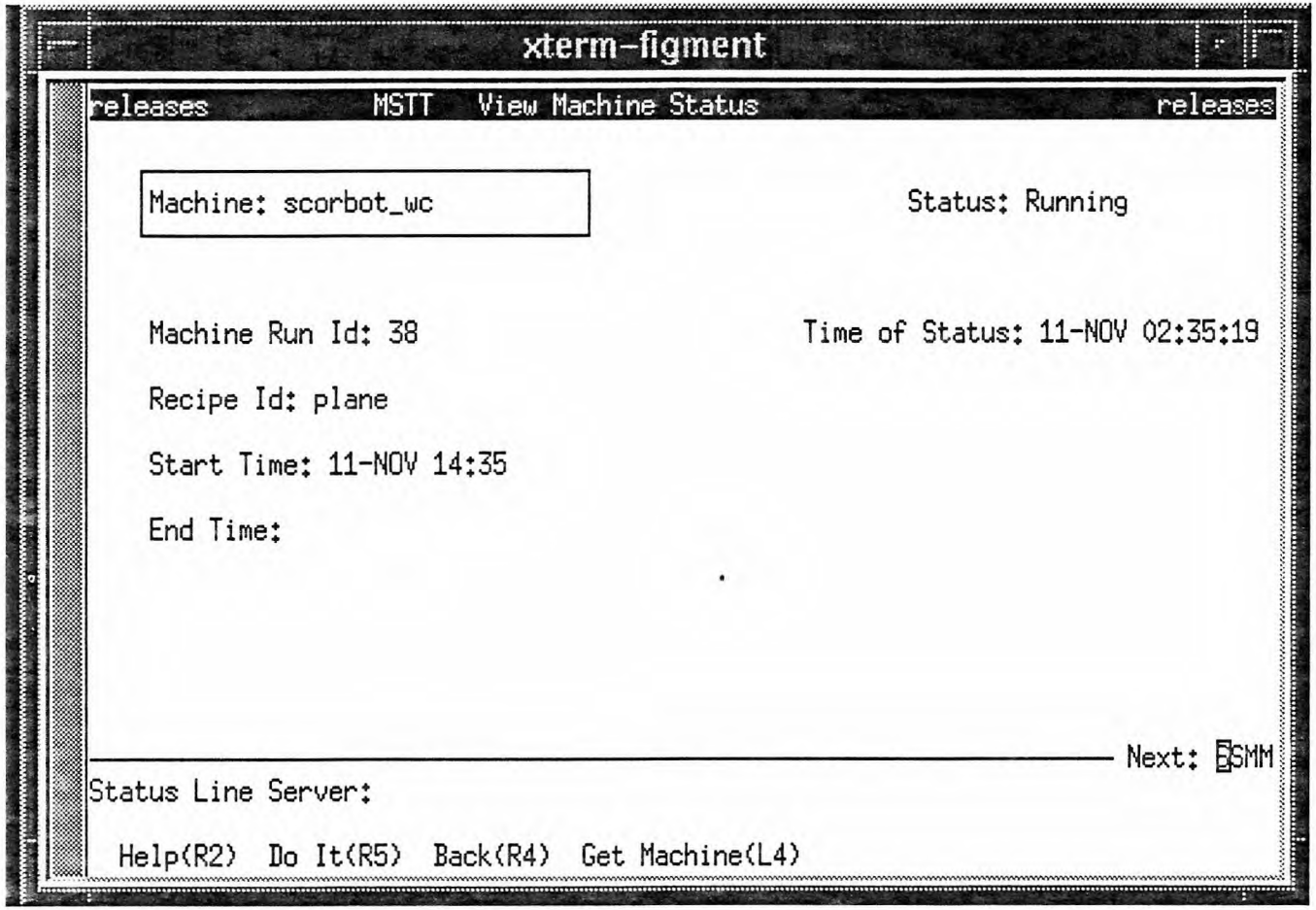

xterm-figment
releases IMPL Import Lot releases
Import Lot
Lot Id
Owner
Priority
Time of Import
Blane0
peter
20.000
15-nov-1991 16:49:09
Next: LOTS
Status Line Server: IMPORT
Help(R2) Do It(R5) Back(R4) Zoom(L4) Top Menu(R3) >


xterm-figment
TA_DATA Table
Ta Code: IMPL
Ta Name: Import Lot
Frame: import lot
PRIVILEG TABLE:
User Name
releases
NextMaster(R5) Query(L2) Help(R2) End(R4)


xterm-figment
TA_STRUCT TABLE:
Menu Ta
Position
Ta Code
init
10
TOPM
TOPM
30
ERST
TOPM
100
EXIT
ERST
10
ERSC
ERST
20
ERSU
ERST
30
ERSM
ERST
89

ERST
90
TOPM
SSMM
10
STRT
SSMM
89

SSMM
90
TOPM
TOPM
80
SSMM
SSMM
20
SHUT
TOPM
90
SYSC
TOPM
20
ALMN
ALMN
10
ACKA
Query(L1) Help(R2) End(R4)


xterm-figment
releases MSTT View Machine Status releases
Machine: scorbot_wc
Status: Running
Machine Run Id: 38
Time of Status: 11-NOV 02:35:19
Recipe Id: plane
Start Time: 11-NOV 14:35
End Time:
Next: SMM
Status Line Server:
Help(R2) Do It(R5) Back(R4) Get Machine(L4)

B. HI ABF Code

- Work Order Status
- Work Order Entry
- Product Entry

```
/*
   Copyright 1990, Siemens Corporate Research, Inc.
   All Rights Reserved
*/
/*
   Form: work_order_status

   Purpose: Allows the user to view and change the status
        of a work_order.
        When the work_order is changed from Waiting to Running,
        the lot script(s) is/are spawnded off and executed.
*/

initialize(
   err = integer,
   change = integer,
   reply = char(1),
   m = vchar(80),                    /* message */
   selected_status = varchar(100),
   datarows = integer,
   on_table = integer,
   lot_id  = integer,
/* Status Line Server variables */
   tmp = varchar(49),
   subj = varchar,
   what_to_do = varchar
   ) =
{
/* Start the Status Line Server */
   set_forms frs (timeout = 0);
   err = callproc read_sls(byref(:tmp));
   if err = -1 then
      m = 'Error reading status line info. Check error log.';
      callproc messg(m = :m);
   elseif err = 0 then
     set_forms field work_order_status (reverse (sls_data) = 0,
                                        blink (sls_data) = 0);
     sls_data := null;
   elseif err = 1 then
     set_forms field work_order_status (reverse (sls_data) = 1,
                                        blink (sls_data) = 1);
     sls_data := tmp;
   endif;
/* End of Status Line Server */

   inittable work_list read;

   work_list = select work_order_key, cust_name, due_time, status
        from work_order_detail;

   err = callproc ing_err();
   if err != 0 then return;endif;

   commit;

   current_view = 'ALL';

   resume field work_list.work_order_key;

}
```

```
on timeout = {
   /* Status Line Server update every 10 seconds */
   set_forms frs (timeout = 0);
   err = callproc read_sls(byref(:tmp));
   if err = -1 then
      m = 'Error reading status line info. Check error log.';
      callproc messg(m = :m);
   elseif err = 0 then
     set_forms field work_order_status (reverse (sls_data) = 0,
                                        blink (sls_data) = 0);
     sls_data := null;
   elseif err = 1 then
     set_forms field work_order_status (reverse (sls_data) = 1,
                                        blink (sls_data) = 1);
     sls_data := tmp;
   endif;
   set_forms frs (timeout = 10);
   /* End of Status Line Server */

   /* refresh listing of work orders*/
   work_list = select work_order_key, cust_name, due_time, status
        from work_order_detail;

   err = callproc ing_err();
   if err != 0 then return;endif;

   commit;

}


'Zoom ' = {
        /* display cust info, and products ordered ...*/
        callframe work_order_entry (work_order_entry.p_wo_key =
                :work_list.work_order_key) with style = fullscreen;

        /* redisplay data in table -- (could have deleted an order) */
        work_list = select work_order_key, cust_name, due_time, status
          from work_order_detail;

        err = callproc ing_err();
        if err != 0 then return;endif;

        commit;
}


'Run ' = {
        if :work_list.status != 'NEW' then
          message 'Work order must have status "NEW" to "RUN"';
          sleep 2;
        else
          /* start scripts */
          err = callproc execute_lot(:work_list.work_order_key);
          if err = -1 then
            message 'Lot execution failed';
            sleep 2;
            exit;
          endif;
```

```
        /* update db and screen */
        update work_order_detail
              set status = 'RUNNING'
              where work_order_key = :work_list.work_order_key;

        err = callproc ing_err();
        if err != 0 then return;endif;

        commit;

        work_list.status = 'RUNNING';

        message 'The lot(s) are now RUNNING and awaiting IMPORT'
              with style = popup;

      endif;
}


'Ship ' = {
      if :work_list.status != 'COMPLETED' then
        message 'Work order must have status "COMPLETED" to "SHIP"';
        sleep 2;
      else
        /* update db and screen */
        update work_order_detail
          set status = 'SHIPPED'
          where work_oder_key = :work_list.work_order_key;

        err = callproc ing_err();
        if err != 0 then return;endif;

        commit;

        work_list.status = 'SHIPPED';

        /* TO DO: set up a rule to transfer data to
              hist table and remove from primary tables */

      endif;
}


'Select View ' =
{
      /* user gets to view work_orders by status selected */

      /* clear table */
      inquire_forms field work_order_status (on_table = table);
      if :on_table = 0 then
        message 'Cursor must be in table field';
        resume;
      endif;

      inquire_forms table work_order_status(datarows = datarows);

      while datarows != 0 do
        deleterow work_list[datarows];
        datarows = datarows - 1;
```

```
        endwhile;

        /* display menu and update table appropriately */
/*      display submenu begin
        'New ' = {
                selected_status = 'status = ''NEW''';
                current_view = 'NEW';
                endloop;
          }

          'Running ' = {
                selected_status = 'status = ''RUNNING''';
                current_view = 'RUNNING';
                endloop;
          }

          'Completed ' = {
                selected_status = 'status = ''COMPLETED''';
                current_view = 'COMPLETED';
                endloop;
          }

          'Shiped ' = {
                selected_status = 'status = ''SHIPPED''';
                current_view = 'SHIPPED';
                endloop;
          }

          'ALL ' = {
                selected_status = 'status like ''%''';
                current_view = 'ALL';
                endloop;
          }
        end;
*/

        work_list = select work_order_key, cust_name, due_time, status
              from work_order_detail
              where :selected_status;

        err = callproc ing_err();
        if err != 0 then return;endif;

        commit;

}


'Back ', key frskey3 =  {
   set_forms frs(timeout = 0);
   inquire_forms form(change=change);
   if :change = 0 then
      return;
   else
      reply := prompt 'Really quit(y/n)? ';
      if :reply = 'y' or :reply = 'Y' then
         return;
      else
         set_forms frs(timeout = 10);
         resume;
```

```
      endif;
   endif;
}
```

```
/*
   Copyright 1990, Siemens Corporate Research, Inc.
   All Rights Reserved
*/
/*
   Form: work_order_entry

   Purpose: Creates a work order that consists of customer info and
            products ordered by the customer. The work order initiates
            lot entities (specified by the products ordered).
*/

initialize(
   err = integer,
   change = integer,
   reply = char(1),
   m = vchar(80),                  /* message */
   datarows = integer,
   table_name = varchar(20),
   on_table = integer,
   item_to_move = varchar(16) not null,
   wo_key = integer not null,
   p_wo_key = integer not null,   /* passed in for Zoom */

/* Status Line Server variables */
   tmp = varchar(49),
   subj = varchar,
   what_to_do = varchar
   ) =
{
/* Start the Status Line Server */
   set_forms frs (timeout = 0);
   err = callproc read_sls(byref(:tmp));
   if err = -1 then
      m = 'Error reading status line info. Check error log.';
      callproc messg(m = :m);
   elseif err = 0 then
     set_forms field work_order_entry (reverse (sls_data) = 0,
                                       blink (sls_data) = 0);
     sls_data := null;
   elseif err = 1 then
     set_forms field work_order_entry (reverse (sls_data) = 1,
                                       blink (sls_data) = 1);
     sls_data := tmp;
   endif;
/* End of Status Line Server */

   inittable product_list read;
   inittable selected_list read;

   if p_wo_key !=0 then
        /* ******** ZOOM mode ********* */
        /* frame called with work_order_key */
        /* display approprite data (not editable) */

        set_forms form (mode = 'read');   /* no editing allowed */
        work_order_key = :p_wo_key;

        work_order_entry = select cust_name, start_time, status, due_time
          from work_order_detail
```

```
        where work_order_key = :work_order_key;

      work_order_entry = select address, city, state, zip
        from customers
        where cust_name = :cust_name;

      selected_list = select item = product_id
        from work_order
        where work_order_key = :work_order_key;

      err = callproc ing_err();
      if err != 0 then return;endif;
      commit;
/*
      display submenu begin
        'Delete ' = {
           reply = prompt 'Really delete entire Work Order? ' with
              style = popup;

            if :reply = 'Y' or :reply = 'y' then
              delete from work_order_detail
                where work_order_key = :work_order_key;

              err = callproc ing_err();
              if err != 0 then return;endif;

              repeated delete from work_order
                where work_order_key = :work_order_key;

              err = callproc ing_err();
              if err != 0 then return;endif;

              commit;
              message 'Work Order deleted';
              sleep 2;

            endif;

          return;
        }

        'Back ', key frskey3 =  {
          return;
        }
      end;
*/

   else
      /* get ready for user to enter new work order */
      product_list = select product_id, description
        from products;

      err = callproc ing_err();
      if err != 0 then return;endif;

      commit;

      start_time = date('now');

      wo_key = get_new_max_value(type = 'WORK_ORDER');
```

```
        if :wo_key < 0 then
          message 'Fatal Error: unable to get work_order key';
          sleep 2;
          exit;
        else
          work_order_key = wo_key;
        endif;

   endif;

   set_forms frs (timeout = 10);
   resume field cust_name;
}

on timeout = {
   /* Status Line Server update every 10 seconds */
   set_forms frs (timeout = 0);
   err = callproc read_sls(byref(:tmp));
   if err = -1 then
      m = 'Error reading status line info. Check error log.';
      callproc messg(m = :m);
   elseif err = 0 then
     set_forms field work_order_entry (reverse (sls_data) = 0,
                                       blink (sls_data) = 0);
     sls_data := null;
   elseif err = 1 then
     set_forms field work_order_entry (reverse (sls_data) = 1,
                                       blink (sls_data) = 1);
     sls_data := tmp;
   endif;
   set_forms frs (timeout = 10);
   /* End of Status Line Server */
}


'Save ' = {
        validate;

        unloadtable selected_list
        {
          if selected_list._state = 2 then
            insert into work_order
                (work_order_key, product_id)
                values (:work_order_key, :selected_list.item)
          endif
        };

        insert into work_order_detail
        (work_order_key, cust_name, start_time, due_time, status)
        values (:work_order_key, :cust_name, :start_time, :due_time, :status);

        message 'Work Order Saved';
        sleep 2;

        return;

}
```

```
'Select Customer' = {
        cust_name := callframe customer_list with style = popup;

        work_order_entry = select address, city, state, zip
        from customers
        where cust_name = :cust_name;

        err = callproc ing_err();
        if err != 0 then return;endif;

        commit;
}


'Product ' = {
        inquire_forms field work_order_entry(on_table = table);
        if :on_table = 1 then
          inquire_forms table work_order_entry
                (table_name = name, datarows = datarows);
          if :datarows = 0 then
            message 'No more products available';
            sleep 2;
          elseif :table_name = 'product_list' then
            item_to_move = product_list.product_id;
            deleterow product_list;
            insertrow selected_list[0](item = :item_to_move);
          elseif :table_name = 'selected_list' then
            item_to_move = selected_list.item;
            deleterow selected_list;
            insertrow product_list[0](product_id = :item_to_move);
          endif;
        endif;
}


'Back ', key frskey3 =  {
   set_forms frs(timeout = 0);
   inquire_forms form(change=change);
   if :change = 0 then
      return;
   else
      reply := prompt 'Really quit(y/n)? ';
      if :reply = 'y' or :reply = 'Y' then
         return;
      else
         set_forms frs(timeout = 10);
         resume;
      endif;
   endif;
}
```

```
/*
   Copyright 1990, Siemens Corporate Research, Inc.
   All Rights Reserved
*/
/*
   Form: product_entry


*/

initialize(
   err = integer,
   change = integer,
   on_table = integer,
   reply = char(1),
   m = vchar(80),                  /* message */

/* Status Line Server variables */
   tmp = varchar(49),
   subj = varchar,
   what_to_do = varchar
   ) =
{
/* Start the Status Line Server */
   set_forms frs (timeout = 0);
   err = callproc read_sls(byref(:tmp));
   if err = -1 then
      m = 'Error reading status line info. Check error log.';
      callproc messg(m = :m);
   elseif err = 0 then
     set_forms field product_entry (reverse (sls_data) = 0,
                                    blink (sls_data) = 0);
     sls_data := null;
   elseif err = 1 then
     set_forms field product_entry (reverse (sls_data) = 1,
                                    blink (sls_data) = 1);
     sls_data := tmp;
   endif;
/* End of Status Line Server */

   inittable product_list read;

   product_list = select product_id
        from products;

   err = callproc ing_err();
   if err != 0 then return;endif;

   commit;

   set_forms frs (timeout = 10);

   resume field product_list.product_id;

}

on timeout = {
   /* Status Line Server update every 10 seconds */
   set_forms frs (timeout = 0);
   err = callproc read_sls(byref(:tmp));
```

```
   if err = -1 then
      m = 'Error reading status line info. Check error log.';
      callproc messg(m = :m);
   elseif err = 0 then
     set_forms field product_entry (reverse (sls_data) = 0,
                                    blink (sls_data) = 0);
     sls_data := null;
   elseif err = 1 then
     set_forms field product_entry (reverse (sls_data) = 1,
                                    blink (sls_data) = 1);
     sls_data := tmp;
   endif;
   set_forms frs (timeout = 10);
   /* End of Status Line Server */

   product_list = select product_id
        from products;

   err = callproc ing_err();
   if err != 0 then return;endif;

   commit;

}

'Zoom' =
{
  set_forms frs (timeout = 0);
  inquire_forms field product_entry (on_table = table);
  if :on_table = 0 then
    message 'Cursor must be in table field to Zoom';
    sleep 2;
  else
    product_entry = select product_id, description, function
      from products
      where product_id = :product_list.product_id;

    err = callproc ing_err();
    if err != 0 then return;endif;

    commit;

    set_forms field product_entry (displayonly(product_id) = 1);

/*    display submenu begin
      'Update ' = {
          validate;

          update products
            set product_id = :product_id, description = :description,
                function = :function
            where product_id = :product_id;

          err = callproc ing_err();
          if err != 0 then return;endif;
          commit;

          endloop;
       }
```

```
      'Delete ' = {
          reply = prompt 'Really delete? ' with style = popup;
          if :reply = 'Y' or :reply = 'y' then
            delete from products where product_id = :product_id;

            err = callproc ing_err();
            if err != 0 then return;endif;
            commit;

            product_list = select product_id
              from products;

            err = callproc ing_err();
            if err != 0 then return;endif;
            commit;

          endif;
          endloop;
        }

      'Back ', key frskey3 = {
        endloop;
      }
    end;
*/


/*    clear field product_id, description, function;     */
    set_forms field product_entry (displayonly(product_id) = 0);
    set_forms frs (timeout = 10);
    resume field product_list.product_id;

  endif;

}


'Add' = {
        validate;

        inquire_forms field product_entry (on_table = table);
        if :on_table = 1 then
          message 'Enter data in fields, then select Add';
          sleep 2;
        else
          insert into products
            (product_id, description, function)
            values(:product_id, :description, :function);

          err = callproc ing_err();
          if err != 0 then return;endif;

          commit;

          product_list = select product_id
            from products;

          err = callproc ing_err();
          if err != 0 then return;endif;
```

```
            commit;

            clear field product_id, description, function;
            resume field product_list.product_id;

         endif;
}


'Back ', key frskey3 =  {
   set_forms frs(timeout = 0);
   inquire_forms form(change=change);
   if :change = 0 then
      return;
   else
      reply := prompt 'Really quit(y/n)? ';
      if :reply = 'y' or :reply = 'Y' then
         return;
      else
         set_forms frs(timeout = 10);
         resume;
      endif;
   endif;
}
```

C. Machine Entity C++ Code

- ME_INTERFACE.C – Main Program
- FILE.C – Supportive Routines
- TERMINAL.C – RS232 Specific Routines
- RS232_TO_AVAIL – CMM Specific Process

```
/*************************************************************
*                                                            *
*     MACHINE ENITITY PROGRAMM FOR COMUNICATION TO ROBOT     *
*                      "me_interface.c"                      *
*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "me_messages.h"
#include "shared_mem_struct.h"
#include "logs.h"
#include "cmm.h"
#include "vms.h"

#define  TRUE       1
#define  NOT_TRUE  -1
#define  ACK        0
#define  NACK      -1

//
// prototypes
//
void cwrite(int, char *);
void stuff_msg(int, char *, char *, int, char *);
extern "C" write(int, char *, int);
extern void kill_mq_sm(int = 0);
extern int init_mq(char*);
extern int mq_read_nowait(int msgid, long msg_type, char *msg_text);
extern int mq_read_wait(int msgid, long msg_type, char *msg_text);
extern int mq_send(int msgid, long msg_type, char *msg_text);
extern int setup_port(int *channel,char *buf);
extern void LOGS(char*,char*,char* = "%s",LOG_MSG_TYPE = note);
extern void robot_response(int msgid, int *channel_ptr, char *buf, FILE **result
extern int term_read(int *, int, int *, char *);

WC_ME_BUFFER wc_me_message;

/* the name of the WC process is passed as a command line argument
** so message queues can be set up
*/
main(int argc, char *argv[])
{
  int channel;
  int num;
  static int msgid;
  int tstat = SS$_NORMAL;              /* status set by reading from the termi
  int response_length;                 /* length of response */
  int response_ack;                    /* converted reply from machine */
  char response_buf[125];              /* reply from machine */
  char buffer[BUF_SIZE];               /* msg buffer */

  FILE *result_fp;

  char process[20];
  strcpy(process, argv[1]);

  msgid = init_mq((char *)&process);

  /* setup port for communication with machine */
```

```
/* note: param buffer not currently used */
if (ACK != setup_port(&channel,(char *)&buffer))
  kill_mq_sm();

while (TRUE)
  {
    while (mq_read_nowait(msgid,TO_ME,(char *)&wc_me_message) == IS_NOMESSAGE)
      {
        robot_response(msgid, &channel, (char *)&buffer, &result_fp);
      }

    switch(wc_me_message.message_id)
      {
      case ME_START:

        cwrite(channel, "AY!");

        /* notify user to Press Machine Start button */
        stuff_msg(CELL_ALAR, "NULL", "NULL", ACK, "Press MS");

        if (mq_send(msgid, FROM_ME, (char *)&wc_me_message) == ERROR_MSG)
          {
            LOGS("cmm_me", "mq_send failed while sending alarm");
            exit(1);
          }

        /* wait for response from user */
        if (mq_read_wait(msgid, TO_ME, (char *)&wc_me_message) == ERROR_MSG)
          {
            LOGS("cmm_me", "mq_read failed: was waiting for ME_START_WAIT\n");
            exit(1);
          }

        if (wc_me_message.message_id == ME_START_WAIT)
          {
            /* got user response -- finish sending commands to CMM */
            cwrite(channel, "\rAC!");
            cwrite(channel, "*!");

            /* Send ACK */
            stuff_msg(ME_START,"NULL", "NULL", ACK, "NULL");

            if (mq_send(msgid, FROM_ME, (char *)&wc_me_message) == ERROR_MSG)
              {
                LOGS("cmm_me", "mq_send failed while sending alarm");
                exit(1);
              }
          }
        else
          {
            /* got something we didn't expect */
            sprintf(line, "Got an unexpected message: id = %d", wc_me_message.
            LOGS("cmm_me", line);
          }

        break;
      case ME_SHUT:

        cwrite(channel, "EY!");
```

```
        /* send shutdown complete message */
        stuff_msg(ME_SHUT, "NULL", "NULL", ACK, "NULL");
        if (mq_send(msgid, FROM_ME, (char *)&wc_me_message) == ERROR_MSG)
          {
            LOGS("cmm_me", "mq_send failed while sending shutdown complete");
            exit(1);
          }

        break;
      case SETUP_RUN:
        /* select part */
        /* filename field contains part name */
        cwrite(channel, "_SETUP_RUN_!");
        cwrite(channel, "1!");
        cwrite(channel, wc_me_message.file_name);
        cwrite(channel, "\r!");
        cwrite(channel, "Y!");

        stuff_msg(SETUP_RUN, "NULL", "NULL", ACK, "NULL");
        if (mq_send(msgid, FROM_ME, (char *)&wc_me_message) == ERROR_MSG)
          {
            LOGS("cmm_me", "mq_send failed while sending setup_run  complete")
            exit(1);
          }

        break;
      case START_RUN:
        /* execute recipe */
        /* note: order is important! CMM command first, then
        ** send start_run_ flag
        */
        cwrite(channel, "DA!");

        cwrite(channel, "_START_RUN_!");

        stuff_msg(START_RUN, "NULL", "NULL", ACK, "NULL");
        if (mq_send(msgid, FROM_ME, (char *)&wc_me_message) == ERROR_MSG)
          {
            LOGS("cmm_me", "mq_send failed while sending start complete");
            exit(1);
          }

        break;
      default:
        sprintf(line, "Unkown message from cmm_wc #%d", wc_me_message.message_
        LOGS("cmm_me", line, "%s", warning);
        break;
      }

    } /* eo while */
}


void cwrite(int channel, char *command)
{
  int i,lgth,num;

  lgth = strlen(command);

  for (i=0 ; i < lgth ; i++)
```

```
    {
      if ((num = write(channel, command, 1)) < 0)
        {
          sprintf(line, "Write to port failed. Num written = %d", num);
          LOGS("cmm_me", line);
        }
      command++;
    } /* eo for */
}

/* stuff message structure with data to be sent
** if sending a simple ack, only the message_id need be supplied
*/
void stuff_msg(int msg_id,
          char *file_name,
          char *recipe_id,
          int ack_code,
          char *alarm_text)
{
  wc_me_message.message_id = msg_id;
  strcpy(wc_me_message.file_name, file_name);
  strcpy(wc_me_message.recipe_id, recipe_id);
  wc_me_message.ack_code = ack_code;
  strcpy(wc_me_message.alarm_text, alarm_text);
}
```

```
/*****************************************************
*                                                    *
*                COMMON SUBROUTINES                  *
*                  "FILE.C"                          *
*****************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "me_messages.h"
#include "shared_mem_struct.h"
#include "vms.h"
#include "logs.h"
#include "cmm.h"


#define TRUE      1
#define FALSE     0
#define ACK       0
#define NACK     -1

extern WC_ME_BUFFER wc_me_message;
extern void LOGS(char*,char*,char* = "%s",LOG_MSG_TYPE = note);
extern int mq_send(int msgid,long msg_type,char *msg_text);
extern int term_read(int *channel_ptr,int time_out,int *buf_length,char *buf);
extern void stuff_msg(int, char *, char *, int, char *);


/* buf is the msg FROM the robot */
void robot_response(int msgid, int *channel_ptr, char *buf, FILE **result_fp)
{
  int buf_length;       /* out var */
  int time_out = 20;    /* in */
  long status;

  char *result_file = "/afs/cad/usr/class/cell/releases/results/cmm_results.res"

  status = term_read(channel_ptr, time_out, &buf_length, buf);

  /* printf("Buffer from CMM: (%s)", buf); */

  if ((status == SS$_TIMEOUT) || (status == SS$_NORMAL))
    {
      if (buf_length == 0)
        {
          /* no message from CMM */
          return;
        }

      /* the determination if the part passed or failed was determined by the so
      ** on the compq. We simply translate it here to a parameter (PASS) and a v
      ** (either YES or NO) which will be put in the DB by the WC.
      */

      if (((*result_fp) = fopen(result_file, "w")) == NULL)
        {
          LOGS("cmm_me, file.m4", "Error opening result file", "%s", error);
        }
      else
        {
          if ((strcmp(buf, "PASS")) == 0)
```

```
            {
              LOGS("cmm_me", "Got PASS from CMM");
              fprintf(*result_fp, "%s %s\n", "PASS", "YES");

              /* notify WC that run is complete */
              stuff_msg(RUN_COMPL, result_file, "NULL", ACK, "NULL");
              if (mq_send(msgid, FROM_ME, (char *)&wc_me_message) == ERROR_MSG)
                {
                  LOGS("cmm_me, file.m4",
                       "mq_send failed while sending RUN_COMPLETE", "%s", error)
                  exit(1);
                }

            }
          else
            if ((strcmp(buf, "FAIL")) == 0)
              {
                LOGS("cmm_me", "Got FAIL from CMM");
                fprintf(*result_fp, "%s %s\n", "PASS", "NO");

                /* notify WC that run is complete */
                stuff_msg(RUN_COMPL, result_file, "NULL", ACK, "NULL");
                if (mq_send(msgid, FROM_ME, (char *)&wc_me_message) == ERROR_MSG
                  {
                    LOGS("cmm_me, file.m4",
                         "mq_send failed while sending RUN_COMPLETE", "%s", erro
                    exit(1);
                  }

              }
            else
              LOGS("cmm_me, file.m4",
                   "Got something from CMM but it wasn't a PASS or FAIL!");

          fclose(*result_fp);
          return;
        }
    }
  else
    {
      sprintf(line,"Error, status returned is not TIMEOUT or NORMAL");
      LOGS("robot_response",line,"%s",error);
      return;
    }
}
```

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <termio.h>
#include <signal.h>
#include <sys/fcntlcom.h>
#include "vms.h"
#include "cmm.h"

// LOW_BOUND is the ascii char one less than a space.
#define LOW_BOUND       31
#define FLUSH_Q         1  /* flush output queue only */

int return_value;

extern "C" int ioctl(int, int, struct termio *);

extern void kill_mq_sm(int = 0);

/* uses ioctl() for compatibility with Xenix. */
int term_read(int  *channel_ptr,
              int  time_out,
              int  *buf_length,
              char *buf)
{
  int chan;
  chan = *channel_ptr;

  char ch = '\0';

  char buf_temp[BUF_SIZE];
  buf_temp[0] = '\0';
  int length_buf = 0;

  int err = 0;
  return_value = SS$_NORMAL;
  static struct termio tty;


  /* set the timeout */
  if (ioctl(chan, TCGETA, &tty) == -1)
    {
      perror("ioctl, TCGETA:");
      kill_mq_sm();
    }

  tty.c_cc[5] = time_out;

  if (ioctl(chan, TCSETA, &tty) == -1)
    {
      perror("ioctl, TCSETA:");
      kill_mq_sm();
    }

  do
    {
      err = read(chan, &ch, 1);
      /* printf("err: %d, buf_temp: (%s)\n",err, buf_temp); */
      if (err < 0)
        {
```

```
          return_value = SS$_ERROR;
          perror("terminal: read:");
        }
      if (err == 0)
        return_value = SS$_TIMEOUT;
      else
        {
          if ((int)ch > LOW_BOUND)
            buf_temp[length_buf++] = ch;
        }
    } while ((return_value == SS$_NORMAL) && (length_buf < BUF_SIZE));
  /* printf("length_buf: %d, buf_temp: (%s)\n",length_buf, buf_temp); */

  strncpy(buf, buf_temp, strlen(buf_temp));


  (*buf_length) = length_buf;
  return(return_value);
}


/*
   deassign_chan routine is here so that all VMS system calls
   are in this one file.
*/
void deassign_chan(int *channel_ptr, struct termio *savtty)
{
  int chan;

  chan = *channel_ptr;

  if (ioctl(chan, TCFLSH, (struct termio *)FLUSH_Q) == -1)
    perror("ioctl, flush");

  close(chan);
  return;
}


/* assign a channel to tta1 */
void termchan(int *channel_ptr, struct termio *savtty)
{
  char dst[10];
  strcpy (dst, "/dev/ttya");
  int chan;
  static struct termio tty;

  chan = open(dst, O_RDWR);
  if (-1 == chan)
    {
      perror(dst);
      kill_mq_sm();
    }

  /* set terminal to raw mode */
  if (ioctl(chan , TCGETA, &tty) == -1)
    {
      perror("ioctl, tcgeta");
      kill_mq_sm();
    }
```

```
  (*savtty) = tty;

  tty.c_iflag &= ~(BRKINT | ISTRIP | INLCR | ICRNL | IUCLC | IXON );
  tty.c_oflag &= ~(OPOST | ONLCR );
  /* tty.c_oflag |= (CR0 | NL0 | TAB0 | BS0 | FF0); */
  tty.c_lflag &= ~(ISIG | ICANON | ECHO );
  tty.c_cflag |= CS8;
  /* tty.c_cflag &= ~(LOBLK); */
  tty.c_cc[4] = 0;              /* set up for NON-BLOCKING reads */
  tty.c_cc[5] = 20;            /* time out = 2 secs */

  if (ioctl(chan, TCSETA, &tty) == -1)
    {
      perror("ioctl, tcseta");
      kill_mq_sm();
    }
  (*channel_ptr) = chan;
}

void send(int *channel_ptr, char *buff)
{
  /* removed */
}
```

```
/*
   This module is for communicating with the CMM and the ME on the cell side.
   It was done in the quick and dirty manner so it is not prity.

   Written on or about Dec 3 1991
   by
        Peter Murray & Rich Meyer
   or   Rich Meyer & Peter Murray
   depending on you point of view...

   The module reads in the serial port checking for the Up/Down load
   commands.  It it is the U/D command it takes the appropiat action.
   Everything else is passed on to SUP/Avail...

   This program is written to be run on the Xenix side.
*/
#include <stdio.h>
#include <stdlib.h>
#include <termio.h>
#include <fcntl.h>

/*
   Description of the file descriptors:
   DATA_OUT  = rs232_to_sup commands to avail (usually 1, stdout)
   PORT_OUT  = line to Machine Entity (usually fd)
   PORT_IN   = line to Machine Entity (usually fd)
   ERR_OUT   = errors incurred by this program (usually 2, stderr)
*/
#define DATA_OUT   1
#define PORT_OUT   fd
#define PORT_IN    fd
#define ERR_OUT    stderr


/* following must match size in ME interface */
#define BUF_SIZE  125
#define ERR_SIZE  100

#define TRUE    1
#define FALSE   !TRUE

/* GLOBAL's */
int fd;                     /* the fd for tty2a */
char buffer[BUF_SIZE];
char err_buf[ERR_SIZE];
int flag;

main()
{
  set_pipe();

  if ( (fd = open("/dev/tty2a", O_RDWR)) < 0)
    {
      perror("open /dev/tty2a:");
      exit(1);
    }

  set_terminal(1, 0, TRUE); /* 10 = 1 sec */

  listen();
```

```
}


set_pipe()
{
  int fd[2];
  int pid;
  int fd_out;


  if (pipe(fd) == -1)
    {
      perror("pipe: ");
      exit(1);
    }

  if ((pid = fork()) >0)
    {
      /* the parent */
      /* redirect std out */
      close(1);
      dup(fd[1]);
      close(fd[0]);
      close(fd[1]);
    }
  else if (pid == 0)
    {
      /* the child */
      /* redirect std input */
      close(0);
      dup(fd[0]);
      close(1);

      if ( (fd_out = open("outfile", O_WRONLY | O_CREAT | O_TRUNC)) < 0)
        {
          perror("open outfile:");
          exit(1);
        }

      dup(fd_out);
      close(fd[0]);
      close(fd[1]);

      /* execl("/afs/cad/usr/class/cell/releases/src/cmm/get", "get",
        (char *) 0); */

      if (execl("/usr/super/xeq/sup", "sup", (char *) 0) <0)
        perror("execl:");

    }
  else if (pid < 0)
    {
      perror("fork() error:");
      exit(1);
    }

}

set_terminal(size,  time, flag)
```

```
int size;
int time;
int flag;
{
  struct termio tty;

  fprintf(ERR_OUT, "Terminal set, min = %d, time = %d\n", size, time);

  /* modify tty structure/settings to raw mode*/
  if ( ioctl(fd, TCGETA, &tty) <0 )
    {
      perror("ioctl, TCGETA:");
      exit(1);
    }

  tty.c_iflag &= ~(IGNBRK|INLCR|IGNPAR|PARMRK|INPCK|ICRNL|IUCLC|ISTRIP|IGNCR|IXO
  tty.c_oflag &= ~(OPOST|OLCUC|ONLCR|OCRNL|ONOCR|ONLRET|OFILL|OFDEL);
  tty.c_cflag &= ~(PARODD|CSTOPB);
  tty.c_cflag |= (PARENB|CS8|HUPCL|CREAD|CLOCAL) ;
  tty.c_lflag &= ~(ISIG|ICANON|ECHO|XCASE|ECHOE|ECHOK|ECHONL|NOFLSH);

  tty.c_cc[4] = size;
  tty.c_cc[5] = time;  /* 10 = 1 sec */

  if (ioctl(fd, TCSETA, &tty) <0)
    {
      perror("ioctl, TCSETAF:");
      exit(1);
    }
}

/*
    This function listens to the RS232 port...It assumes all entries are
    finished with a ! char.
*/
listen()
{
char tbuf[BUF_SIZE + 30];
char tname[BUF_SIZE];
char tpath[BUF_SIZE];
FILE *fp;

  do /* forever */
  {
      file_input(buffer);  /* read rs232 port */
      fprintf(ERR_OUT, "do:buffer read: (%s)\n", buffer);

   if (!strcmp(buffer,"_DOWNLD_!"))
   {
      /* download */
     system("kermit ilbr /dev/tty2a 9600");
     system("tar -xvf /usr/avail/part/*.tar");
   }
   else if (!strcmp(buffer,"_UPLD_!"))
   {
      /* upload */
      strcpy(tname,(buffer+6));
     strcpy(tpath,"/usr/avail/part/");

/* tar -cvf /usr/avail/part/tname/tname.tar /usr/avail/part/tname/STAR */
```

```
      strcat(tbuf,"tar -cvf ");
      strcat(tbuf,tpath);
      strcat(tbuf,tname);
      strcat(tbuf,"/");
      strcat(tbuf,tname);
      strcat(tbuf,".tar /usr/avail/part/");
      strcat(tbuf,tname);
      strcat(tbuf,"/*");
      fprintf(ERR_OUT, "tbuf tar: %s\n", tbuf);
      system(tbuf);

/* kermit ilbs /dev/tty2a 9600 /usr/avail/part/tname/tname.tar */
      strcpy(tbuf,"kermit ilbs /dev/tty2a 9600");
      strcat(tbuf,tpath);
      strcat(tbuf,tname);
      strcat(tbuf,".tar");
      fprintf(ERR_OUT, "tbuf kermit. %s\n", tbuf);
      system(tbuf);
   }
   else if (!strcmp(buffer,"_SETUP_RUN_!"))
   {
      file_input(buffer);
      output(DATA_OUT, buffer);
      if (strcmp(buffer,"1!"))
          fprintf(ERR_OUT,"unexpected input...");
      file_input(buffer);
      output(DATA_OUT, buffer);
      strncpy(tname,buffer,(strlen(buffer)-1));    /* save the part name */
      fprintf(ERR_OUT,"Part name: %s\n",tname);
   }
   else if (!strcmp(buffer,"_START_RUN_!"))
   {
      strcpy(tpath,"/usr/avail/part/");
      strncat(tpath, tname, (strlen(tname)-1));
      strcat(tpath,"/");
      strcat(tpath,"RESULTS");
      fprintf(ERR_OUT,"Looking for: %s\n",tpath);
      while(1)
      {
         if ((fp = fopen(tpath, "r")) != NULL)
         {
            if (fread(buffer,80,1,fp) > 1)
               output(PORT_OUT,"FAIL!");  /* to the workcell */
            else
               output(PORT_OUT,"PASS!");  /* to the workcell */
            break;
         }
         sleep(1);
         fprintf(ERR_OUT,"Wating for: %s   file to be created.\n",tpath);
      }
   }
   else
     output(DATA_OUT, buffer);
  } while(1);
}
file_input(buffer)
char *buffer;
{
char rbuf[4];
int num;
```

```
int buf_count = 0;

strcpy(buffer, "");
   do
   {
      if ((num =  read(PORT_IN, rbuf, 1)) < 0)
      {
        fprintf(ERR_OUT, "num read: %d\n", num);
      }
      rbuf[1] = '\0';
      strcat(buffer,rbuf);
      fprintf(ERR_OUT, "buffer read: (%s)\n", buffer);
      buf_count++;
   } while ((rbuf[0] != '!') && (buf_count < BUF_SIZE));
}

output(direction, buffer)
int direction;
char *buffer;
{
int num;

      /* pass it on to the sup */
      if ((num =  write(direction, buffer, (strlen(buffer)-1))) < 0)
      {
        fprintf(ERR_OUT, "num write: %d\n", num);
        return -1;
      }
      fprintf(ERR_OUT, "num write: %d, (%s)\n", num, buffer);
}
```

D. Unix Shell Scripts

- Generic Workcell Module Start Script
- CMM Start Script
- ERS Start Script
- Generic Workcell Module Shutdown Script

```
#
echo "Starting the cell."
~/bin/cas.csh
echo "Started cas"
~/bin/ers.csh
echo "Started ers"
~/bin/ms.csh
echo "Started ms"
~/bin/lsr.csh
echo "Started lsr"
~/bin/msched.csh
echo "Started msched"
#~/bin/scorbot.csh
#echo "Started SCORBOT"
~/bin/dummy1.csh
echo "Started DUMMY1"
~/bin/cmm.csh
echo "Started CMM"
~/bin/hi.csh
```

```
#
~/bin/setport.csh&
setenv ERROR_LOG ~/logs/cmm.log
~/bin/cmm_wc&
~/bin/cmm_me cmm_wc&
(sleep 10;loglev "cmm_wc" "note")&
(sleep 10;loglev "cmm_me" "note")&
```

```
#
setenv ERROR_LOG ~/logs/ers.log
~/bin/ers&
(sleep 10;loglev "ers" "also_warnings")&
```

```
#
kall.csh -15 cmm_dummy_wc cmm_wc cmm_me cat lsr msched ers cas ms wcn men wc_int
ipcs | awk '{if (($1 == "q" || $1 == "m" || $1 == "s") && $5 == "cell") \
          printf("ipcrm -%s %d\n",$1,$2)}' > .clean.up.shmem
chmod 777 .clean.up.shmem
.clean.up.shmem
/bin/rm .clean.up.shmem
```

E. Log File Example

- Sample CMM Log File

```
==========================================
From    mq_send
Date    Wed Dec 11 17:38:26 1991
User    cell
Term    /dev/tty
PID     16625
--------
Note:
 msg_type = 2 message_id = 222
file_name = (gwc1)
recipe_id = (gwc1) ack_code = 0
alarm_text = (NULL)
--------

==========================================
From    int Conversation:: send (Message& msg, int last)

Date    Wed Dec 11 17:38:58 1991
User    cell
Term    /dev/tty
PID     16625
--------
Note:
 Group:   UPD_ME_ST (239)
Message: Update_ME_Status (407)
Delay:   0 secs
Sender:  3/0:16625.0

  msg_name       = "Update_ME_Status"
  machine_id     = "cmm_wc"
  machine_run_id         = 76
  recipe_id      = "gwc1"
  start_time     = " "
  end_time       = " "
  me_status      = "Setup"
--------
==========================================
From    Conversation tool
Date    Wed Dec 11 17:49:18 1991
User    cell
Term    /dev/tty
PID     16625
--------
Note:
 NOTE: -------receive------- User cmm_wc is signed on
Buffer:
{ 0x177de8, 0x177de8, qlength = 1 }
        { CELL, 254, 440, <11, 553653023, 16625> }

Filters:
CELL: 217 234 241 244 245

Current Convs:



----------
==========================================
```