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

Dynamic Machine Scheduling, Expansion, and Control of a Generic WorkCell

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

Peter A. Murray

Approval Page

Date Submitted: 1/13/92

Date Approved: 1/13/92

Approved By (Faculty Advisor): _____

0.3 Abstract

Factory automation has come a long way since the invention of the automatic flour mill. A workcell is a group of machines (robots) working together to produce a product. In the past workcells have been hard wired using methods such as Programmable Logic Controllers (PLCs). To change a part of this system would require reprogramming the entire system.

The Generic Work Cell (GWC) is a dynamic architecture which allows a workcell to be modified on the fly. The architecture enables the generation of new cells with minimal effort, and the modification of the system without reprogramming the entire system. The concept behind the GWC is that in every workcell there are generic parts which are the same for all workcells. For instance, the machine scheduler is an example of a generic piece which is used by all systems. The GWC is also a reactive system This implies that the system is capable of adjusting itself to environmental disturbances, such as machine shutdown or startup.

0.4 Key Words and Phrases

When discussing programs which are run on a robot, the word *recipe* is substituted for the word *program.*

Programmable Logic Controller(PLC) - A hard wired workcell controller.

GWC - Generic WorkCell.

MSCHED - Machine Scheduler.

HI - Human Interface.

LS - Lot Server.

MS - Machine Server.

ERS - Equipment Recipe Server.

CAS - Cell Alarm Server.

UNIX - An AT&T/Bell Labs multitasking computer operating system.

SUN - Sun Micro Systems work station computer.

X-Window - A Graphical user interface for UNIX.

Workcell - A collection of robots working together to produce a product.

SCR - Siemens Corporate Research.

CMM - Coordinate Measuring Machine.

MAZAK - A computer controlled milling machine.

Recipe - A program which runs on a robot to control the robot.

Lot Scripts - A program which controls the operations of a workcell.

Lot Entities - A Lot Script recognized by the workcell.

OE - Operation Entity.

Hierarchal Control - A multi level controller.

Non-Hierarchical - A single level controller.

Alarm - A signal sent to the HI from another module to notify the HI of an event.

Resource - This can be machines, robots, or stock used in the cell such as

aluminum or brass.

Message Buss - A communications architecture to pass messages from one module to another with out knowing where the receiver is.

ISIS - A communications package which is used to implement the Message Buss.

P2P - A German communications package which was used but dropped from its lack of reliability.

Machine Interface - The module which communicates between the Robots and the workcell.

WC - Work Cell side of the Machine Interface.

ME - Machine side of the Machine Interface.

Feed Back - A way of adjusting a Recipe by using past data.

Feed Forward - A way of adjusting a Recipe so the current product is manufactured correctly from the current point.

Ingres - A commercial database package.

VMS - Digital Equipment's proprietary operating system.

Logs/Log files - Used for debugging problems in the GWC software.

Modules - Independent executable programs in the GWC.

DBT - Data Base Tool.

CTool - Conversation Tool.

CMON - Conversation Monitor.

C, C++ - Computer programming languages.

Machine - A reference to a robot.

IMPS - Intelligent Moving Processes.

Home - The position a robot goes to align its self.

MEN - Dummy ME half of a Machine Interface, N of them.

WCN - Dummy WC half of a Machine Interface, N of them.

KERMIT - A communications package to move data between two computers.

0.5 Table of Contents

1.0 Introduction

A workcell is a collection of machine(s) which work in tandem to perform a desired task. In a car manufacturing plant for example, a workcell could be used to paint cars. This may require 20 robots working together to achieve the desired results. The Siemens-NJIT workcell is a prototype to examine the Generic Workcell (GWC) feasibility.

This paper will briefly discuss the entire GWC architecture and design so that the reader may more fully appreciate and understand the work done by the author of this paper. The work done by the author of this paper is split into three parts of the GWC. First is the Machine Scheduler (MSCHED). Second is the expansion of a GWC, the machine interface. Lastly, there is the control of the GWC, which is done through Lot Scripts.

1.1 Motivation

Consider a hypothetical car painting workcell. Assume that it takes nine months to set up a new workcell, and the workcell is needed more that three months a year to be cost effective. If the market demanded that the latest equipment be used every year. The workcell would have to be rebuilt every year. The factory could not operate effectively and would be closed.

The basic steps required to build a workcell are as follows: First, the cell must be defined by a process engineer. Then it must be implemented (programmed); a task which can take hundreds of man months depending on the size of the cell. The machines must be physically brought in. Each machine in the cell must be programmed to work correctly with the controlling computer. Each robot must be programmed how to preform its task, and finally, everything must work together to produce the desired result.

Many factories today use programmable logic controllers (PLC). This is a way of controlling the workcell by hard-wiring the robots to the controller. Making changes to this system is very difficult since it requires that the entire PLC software be rewritten.

The controlling computer must preform many tasks. One of these tasks is to choose a paint station at which a car should be painted. If the workcell has five painting stations, each station providing a different color, and two stations are for model A and

the other three for model T, then how does the controlling computer know where to send the next car waiting to be painted? What if one of the stations is turned off? This decision has to be made on the fly (dynamically). The decision of what color the car should be is defined by the lot script, but the scheduling of the car to a particular robot must be done dynamically by a machine scheduler.

Another task the controlling computer should be able to handle is when a robot breaks. Will the controlling system allow an operator to dynamically replace the broken robot without effecting the rest of the workcell? Can the broken robot be replace by a newer model robot dynamically with out effecting production? How can the production demand be met if it takes nine months to redesign the workcell to replace a robot?

Finding an acceptable solution to theses questions and implementing this solution, has motivated me to do the work presented in this paper.

1.2 Previous and Current Work

The traditional way to create a new workcell is to start from scratch, but this takes to much time. In the past, the work required to expand or modify a workcell involved stopping the workcell and reprogramming it. This process could take months and cost the company thousands of dollars, while production is halted for the modifications. If a robot in a traditional workcell fails or has an unpredictable maintenance schedule, the workcell must be shutdown in order to service the robot, since it is statically configured.

To be able to expand a workcell dynamically, the controller cannot be static such as the Programmable Logic Controller (PLC). The workcell must be able to adapt to a changing environment quickly. The control of a workcell is critical to the proper operation of a factory. It should, therefore have a high degree of fault tolerance. It should also be structured so that maintenance, and development costs are low. This implies that a non-hierarchical architecture should be used for the workcell controller. Non-hierarchical control of a workcell is not a new concept. In fact, many papers have been published addressing non-hierarchical vs hierarchical control of a workcell.[1,2,3,4,5,7] Dynamic scheduling should also be part of such a cells design, which is also easier to implement in a non-hierarchical architecture. To achieve all this, the Entity-Server Model[8] is used in conjunction with a message bus implemented using the Conversation Tool developed at Siemens. The Conversation Tool creates the

message bus and helps programmers create conversations to use on the message bus. The current implementation uses ISIS, a communications package developed at Cornell University.

The GWC is a collaboration of many ideas brought together. Such ideas are non-hierarchical distributed architecture, modularity, dynamic control of a workcell, and dynamic expansion/reduction of a workcell. Each concept is, in itself, good and has been used before, but not all of the concepts have ever been brought together into one system.

Research is being done on many different approaches to solving manufacturing problems. Neil A. Duffie has written several papers addressing problems in manufacturing. He has designed and implemented a workcell which has a non-hierarchical control with a high degree of fault-tolerance. This system does not allow dynamic scheduling lowering its degree of flexibility. The cell implements a master-slave binding mechanism which also limits it flexibility, but the cell does maintains a good level of control over its processes.[4,5]

Oyvind Orke of the Production Engineering Laboratory, NTH-SINTEF, in Norway, describes the projects where he is working in his paper. They have developed a large system which is designed to create a more flexible manufacturing systems. The system encompasses the design phase to production, but does not address dynamic scheduling problem. It is not possible to have a flexible workcell unless you have dynamic scheduling.[15]

Michael J. Shaw has a dynamic scheduling schema which is essentially the same as the GWC's. The system broadcasts a request and waits for responses. Modules on the net return a bid with an estimated processing time, estimated wait time, and estimated travel time. The bids are gathered and the lowest bit gets the job. The GWC does essentially the same but over a message bus which can also cross over the network.[14]

At Siemens Corporate Research (SCR) in Princeton NJ, there have been three implementations of the GWC which makes the current one built at NJIT the fourth. Two were implemented in Germany at a Siemens micro-chip manufacturing plant. One is still in operation and is being used to control a Chemical Vapor Deposition (CVD) cell[8]. The second two are using manufacturing robots The first of the latter two is a small cell at SCR Princeton which implemented a Scorbot Robot to build Lego planes

and helicopters. The forth implementation is at NJIT using a Coordinate Measuring Machine (CMM) and later a Mazak milling machine to produce chess pieces to a precise tolerance.[16]

1.3 A Brief Description of the Siemens-NJIT Generic Workcell

For the first phase of the Siemens-NJIT workcell, the implementation will consist of one machine; CMM. This workcell will hopefully be expanded to include other machines such as the Mazak milling machine. The plan is that the Mazak will mill chess pieces and the CMM will measure them to see if the pieces are within an acceptable tolerance. If not, they will either be sent back to be re-milled or discarded as scrap.[16]

1.4 Modules of the GWC

The following is a description of modules in the GWC which are important as background information. In order for the reader to understand the work presented here, he should have a general knowledge of the entire project

1.4.1 Equipment Recipe Server (ERS)

The Equipment Recipe Server keeps track of recipes on the system. It maintains new and old recipes so original recipes can be kept unmodified. It also ensures that the recipe being used is the proper one and not a modified one. It can be thought of as a librarian of recipes.[13]

1.4.2 Machine Server (MS)

The Machine Server provides the common services required for all machines. This includes maintaining the status of the machines in the database, keeping track of scraps, and keeping track of maintenance schedules. The Machine Server communicates through the communications bus as do all the other modules.[11]

1.4.3 Human Interface (HI)

The Human Interface provides a clean way for people to communicate with the GWC. It provides the ability to control all functions of the GWC through easy to use menus.

1.4.4 Lot Server (LS)

The Lot Server provides services to lots. It controls the creation of entities, division of entities, merger of entities, and the destruction of entities Any service related to lot entities are provided through the LS.[8]

1.4.5 Cell Alarm Server (CAS)

The Cell Alarm Server(CAS) is used to handle alarms which occur in the workcell. The functions performed by the CAS are reporting alarms to the HI, queuing pending alarms, keeping a history of alarms, and taking the appropriate actions necessary for the given alarm.[12]

1.5 Purpose/Objectives/Justification of the Project.

This project has the potential of improving the state of the art of automated manufacturing. Using lot scripts the GWC has the ability to dynamically change what a workcell produces, check to see if the workcell is producing what it should, and see if the product is to the desired specifications. By reusing the generic parts of the GWC, the startup time from conception of the cell to implementation is dramatically reduced, and all that need be written are the machine interfaces.

The proposed purpose of this project is to integrate the factory floor at NJIT in the Center for Manufacturing System (CMS). It is hoped that the GWC soft real-time system could provide the services necessary to allow the equipment on the factory floor function together. This would provide a state of the art learning tool for the school to use. The system is a reactive system; meaning, it reacts to changes in its environment. It tries to adjust its self so it can continue to function correctly in a changing environment. The work done by the author was primarily in the three following areas: The Machine Scheduler MSCHED, the NJIT lot script, and the WC half of the Machine Interface.

Without dynamic scheduling capabilities offered by MSCHED, the cell would not operate as efficiently or effectively. The ability to take machines on and off line dynamically without affecting the rest of the cell is a tremendous timesaving capability Controlling how the workcell behaves and operates is important to flexible manufacturing by its definition. The quick expansion of a GWC is also desirable for a workcell to be called flexible. This is demonstrated by adding the CMM to the NJIT

GWC in a relatively short period of time.

1.6 Thesis Organization

The organization of this paper is as follows: Section 2 will discuss the functional requirements of the MSCHED, WC, and Lot Script for the NJIT workcell. Section 3 will discuss the performance requirements for the above mentioned. Section 4 will describe the overall design of the above mentioned. Section 5 will discuss the data structures of the implementation at NJIT. Section 6 will discuss the detailed specifics of the implementation for each module. Section 7 will discuss the demonstration of the work done, and Section 8 will discuss the problems encountered while developing the work.

2.0 System Functional Specifications of the Work Done

There are three areas in the GWC on which I worked on: The Lot Script, Machine Scheduler (MSCHED), and the workcell side of the Machine Interface (WC).

The function of the Lot Script is to control the operations of the NJIT workcell. The Lot Script uses the MSCHED module, which controls access to the machines in the workcell. The GWC communicates with these machines through the Machine Interface, which is split into two section WC and ME.

The Lot Script performs the following functions: requesting machines, down loading recipes to the machine, up loading a recipes from the machine, and running a recipes on the machine.

2.1 Functional Requirements

The Machine Scheduler (MSCHED, Listing 2) should control access to all the machines in the workcell by deciding which request gets a machines and which does not. Built into MSCHED should be the logic of what requests are filled and not filled. Priority scheduling can also be implemented using any fair algorithm. Preemptive scheduling cannot be used for the reason that some processes on some machine can not be interrupted.

Communication between the GWC and the machines in the workcell must happen or the workcell is non-functional. The workcell side of the Machine Interface WC, should be able to handle all the communications between the GWC and the machine side of the Machine Interface ME. These two modules must cooperate in order for the GWC to communicate with machines in the workcell. Information regarding the ME is not presented in this paper since this work was done by my colleague Richard Meyer. The function of the ME is to communicate between the WC and the machine.

Product flow should be directly controlled by lot scripts The scripts are executed, and then requests services provided by the cell through servers. This script will determine how a manufacturing process is done and by which machines; it is the router for a production.

2.2 User Input Preview

The user may input requests through the Human Interface (HI). In addition to ordering a chess set and importing a lot into the cell, there are many more operations a user may perform through the HI. The user should be allowed to write new Lot Scripts to control the functions of the GWC.

The WC half of the Machine Interface gets its input through the message bus and shared memory. It should listen for conversations on the bus and respond only to specific messages in effect, ignoring all other conversations. The WC should also watch the message queue in shared memory for incoming messages from the ME half of the Machine Interface.

The input to the MSCHED is a request over the message bus to reserve a machine or cancel a request. It also watches for a change of status and attempts to schedule a request when it sees that a machine has changed it status.

2.3 User Output Preview

The output of the WC should be a response to any conversations for which it listened. It should also send requests and commands to the other half of the Machine Interface, ME. The only conversation it should ever initiate is an alarm, which it receives from the ME half of the module.

The output of the MSCHED module is to bus, and is an acknowledgment responding to the requester. This response tells the requester that the request hasbeen received or not. It should also update the database if the request was successful.

2.4 System Data Base/File Structure Preview

The database provides a high degree of data integrity with an easy way to store information about the workcell. The database is logically organized by services which correspond to modules or groups of modules. It can be assumed that all relevant information ranging from the cell organization to run results from the WC are all kept in the database. MSCHED should also use the database to store scheduling information about the machines in the workcell.

There are several files used in the system to keep track of information. One

important purpose of files are to keep logs of warnings, errors, and messages from each module. MSCHED, WC, and the Lot Script should all use log files for error tracking. If there is a system failure, these logs are invaluable when searching for the problem.

The shared memory between the two halves of the Machine Interface, WC and ME, also contain data for a short period of time. However, this is flushed clean quickly.

The message bus lets MSCHED receive requests from the Lot Script, and lets the WC converse with other modules. This has been implemented using the Conversation Tool, which buffers messages until they are read, and guarantees the order of their arrival.

2.5 External and internal Limitations and Restrictions

The access to the data base should be tightly controlled to ensure persistent storage. The servers control how the database is updated, and if a lot wants to reserve a machine, it must be required to do so through the MSCHED. No module can reserve a machine(s) besides MSCHED. This ensures that data is stored consistently and persistently.

Restrictions of the Lot Script are limited by the capabilities of the resources in the workcell. The cell can be changed dynamically so that it may, at any time, produce any product. The workcell is limited by its hardware capabilities, which can be expanded at any time by using MSCHED, the Machine Interface, and the Lot Script

3.0 System Performance Requirements

The GWC was conceived for the purpose of quality control. It was designed so that it could be dynamically reconfigured. It has not been optimized for speed, and is only in its third rewrite. All of the modules have not been implemented. The GWC performance requirements are only that it runs sufficiently fast so the computer is not the bottleneck. As long as the system is running faster than the actual machines in the workcell, then the system is performing adequately.

The requirements of the Machine scheduler is only that it schedules machines predictably and follows the guidelines stated above. The requirement of the Lot Script is that it functions as expected. The requirement of the WC half of the Machine Interface is that it works consistently, correctly with the ME, and the rest of the system.

3.1 Efficiency

The GWC is very efficient at keeping track information about the workcell. It is not suggested that using a GWC will speed production. The MSCHED should be efficient enough so it does not slow the system down. The Lot Script should be designed so it does not bottleneck the system by making poor requests, and the WC should operate without slowing the rest of the system excessively either.

3.2 Reliability

The design of the architecture of the GWC should be reliable. The GWC is designed as a non-hierarchical distributed system This means that redundancy is easy to implement, and, since it is distributed, it should have a higher degree of fault tolerance. Since the cell is dynamically configured, if a part of the cell starts to fail, then the failing part could be replaced with little or no down time to the workcell.

The MSCHED should be able to be replaced at any time However, since it is an integral part the workcell, the workcell would have to be halted for a short time to start the new MSCHED. The WC failing would affect the ME half of the Machine Interface; but if the Machine Interface fails, MSCHED should not try to schedule requests to that machine. If the Lot Script fails at any point, it could leave database tables corrupted. So the Lot Script should be written carefully so as not to leave the database corrupted if it fails.

3.2.1 Description of Reliability Measures

The integrity of the database is kept by using servers which control access to the database for consistency. When a module wishes to get information from the database or put information in it, it should do so through the proper server. Since the data is kept on a commercial database it is preserved and one should be able to consider it consistent.

MSCHED should store information about the status of each machine in the database. The WC should also be able to retrieve information from the database, as should the Lot Script.

3.2.2 Error/Failure Detection and Recovery

Error detection of unusual events are handled through alarms. Alarms are used to alert the human operator that an event has happened which requires human intervention. Essentially, if a failure occurs in the system which cannot be directly handled by the cell software, an alarm should be sent out. If the cell should fail for any reason, then the fault should be traceable by looking at the log files, which are kept on the file system. The log file should contain information such as what procedure logged the entry, and what conversations were occurring (Fig 2). This information is invaluable when trying to trace a problem in the GWC software. Once the fault is found, and if the module has not died on its own accord, the MSCHED should allow for the shutdown of the module, and the startup of a revised version to replace the faulty module. This should be able to take place while the cell is still in operation, causing no down time, except to the machine module in question.

If a request for a machine startup is made, and the machine is turned off. An alarm could be sent asking an operator to turn on the machine. Alarms should be listened for from the ME by the WC and, then passed to the Cell Alarm Server (CAS).

Database failures should be handled by retrying the request which failed. If the error is fatal, the module should kill itself in a clean and orderly manner. MSCHED, WC, and the Lot Script should all operate in this way.

3.2.3 Allowable/Acceptable Error/Failure Rate

For many situations, the allowable or acceptable failure rate is not determined

statically. There should be limitations in the modules which, if not met, would cause the module to fail. If one of the three modules is waiting for a response to a conversation or a continuation of a conversation, and the response takes longer than a predefined time period, the conversation should fail and the module should log the error.

Lot Scripts should have the ability to determine if a product produced by a machine is within the allowable tolerances. The script should be able to do this by investigating the run results returned from the machine. This should allow the Lot Script to determine if the lot is good or bad. If it is determined the lot is bad, the script could be given the ability to take action to correct the problem while the production is still running.

3.3 Security

The security of this system should be determined the purpose of the system. The security could be adjusted for any environment, but since it is a research project. It would only slow down the researchers trying to develop the system. It should therefore be of no concern.

3.3.1 Hardware Security

The hardware platform at this point can be either UNIX based or VMS based Considering the large number of hardware vendors that makes UNIX systems, it is conceivable that any level of hardware security could be obtained for use of the GWC

3.3.2 Software Security

The architecture of the GWC should allow modules to be added or removed at arbitrary times. This would imply that one could write a software module which would take care of security as well. This module could be updated whenever needed, since it would be as dynamic as the rest of the workcell. The Human Interface (HI) could incorporate the security module requiring passwords to execute commands of significance such as shutting down, starting up, or changing the configuration of the cell. The software security can be as stringent as necessary

3.3.3 Data Security

The security of the data is as secure as the operating system, since the results of a

run on the workcell are stored in a file on the system. Data security is not very high in the UNIX operating system, since UNIX is not a secure system. Hence, the data should be more secure if it were on VMS, but this is not a concern at this stage of the project.

3.4 Maintainability

The GWC architecture was designed to facilitate ease of maintenance. With such a large software system, maintenance can be very expensive. Several tools have been built around the GWC project to support the GWC. All of the modules were written in C++, which is inherently easier to maintain than many other languages. The use of logs is intended to help a person maintain the existing modules as well as develop new ones. A developer can test his modules on the cell, while the cell is running without shutting the cell down. This ability is a significant maintenance capability

There are three tools to help maintain the cell: The Conversation Tool, Database Tool, and Conversation Monitor. The Conversation Tool (CTOOL) supplies an easy way to create new or modify existing conversations, which are sent and received between modules. The Database Tool is used to assist the programmer when interacting with the database. This is done by making another level of abstraction between the modules and the database. Many of the tedious embedded SQL tasks required by the programmer can now be eliminated by using the Database Tool (DBT). The Conversation Monitor (CMON) allows a programmer to look at the conversations taking place between modules on the bus. Using these tools a programmer should be able to maintain the cell software efficiently and effectively.

3.5 Modifiability

The cell should be extremely modifiable as the architecture would suggest. The addition of modules while the cell is running, if works as described above, would allow the cell to be dynamically reconfigured. Then ability to modify existing modules dynamically would also be possible.

Through the Lot Script, recipes used by the cell could be modified while the cell is running. The cell should have the ability to tweak the recipe to improve the product quality

The MSCHED was designed with modifiability in mind There is only one function which determines how MSCHED schedules, and this one function could be

changed to use any fair scheduling algorithm.

3.6 Portability

The software which describes the GWC is written in AT&T C++ so it should be compatible on any AT&T C++ compiler on any system. The cell software has been built on both VMS and UNIX operating systems successfully. Essentially, the cell could be built on any multi-tasking operating system.

Porting the system from one factory to another with a completely different setup should not be as difficult as one could imagine. The largest amount of work required would should be writing the machine interfaces. Since each robot would require its specific communications protocol, each different robot would require an ME_INTERFACE and some modification to the WC_INTERFACE module (FIG 3). The making of an interface to communicate with a controller needs to be done no matter what method of controlling the workcell is used. Naturally, the recipes (programs) for controlling what the robots do must also be created

4.0 System Design Overview

As previously mentioned the GWC has been designed in a very modular format. Each module is a separate executable entity on the system allowing it to be isolated from the other modules. This ensures that if one module has a problem it will not adversely effect the other modules, other than not providing services to the other modules. Each module can be considered a black box which accepts input and gives results.

4.1 System Internal Data Structure Preview

Since most of the internal data structure are stored in Ingres, it is also controlled by Ingres. This is done because the database is consistent and reliable. The small amounts of data which are used in a module and are not stored in the database are only trivial data stores for temporary use.

All conversations which happen over the bus follow a specific format. The header of a conversation has information in it such as what conversation it is, and how many more parts of the conversation there are. Most conversations require that they are acknowledged so the sender knows that the conversation has been received.

4.2 Description of System Operations

MSCHED controls which machines will be used when a lot requests a machine for a service. If all the requested machines are busy, then MSCHED will queue the request. Once the machine is made available, MSCHED will look at the queue for past requests. After a lot is finished with the machine(s), MSCHED unreserves the machine allowing other lots to use the machine.

The WC operations are limited to communications between the ME and the GWC. It handles all interaction between the two modules and is built specifically for each ME. Most WCs should be similar; however, not this is not necessary since this is only a convention.

The Lot Scripts operations control the operations in the GWC. The Script describes the characteristics of the GWC and how it will function. The Lot Script is not static and can be changed whenever a new machine is added to the workcell. The Lot Scripts operations are limited by the machines in the workcell

4.3 Equipment Configuration

The equipment used in the Siemens-NJIT GWC project consists of a Coordinate Measuring Machine (CMM) and a Sun (Sparc) workstation. The CMM has a Compac PC connected to it running the Xenix operating system, which is connected to the Sun via serial port. The physical layout is shown in Fig 4. This layout is not the only way it can be set up. The Sun was placed, physically, next to the CMM to ensure that if anything were to go wrong the panic button could be pressed before anything serious happened. It is anticipated that once the cell is running, the Sun would reside in the operator control booth above the factory floor, and a dumb terminal, like a VT-100, would reside next to the CMM for operator interaction.[16]

As was described in section 3.6, Portability, the hardware platform for the cell could be almost any vendor, and the operating system could be almost any multitasking system.

4.4 Implementation Languages

The GWC was written using C++, C, ABF, 4GL, M4 macros, and embedded SQL. The majority of the code was written in C++. The ability to build classes and structure in an object oriented environment is highly desirable. C++ promotes the sharing of code and expedites the development of new code. An object oriented approach for this project seems to be the best way to proceed. Its benefits outweigh any restrictions it might have and since C++ is widely used and available, it is the logical choice. It is also strongly supported and is constantly being upgraded by its creators at Bell Labs. C was used in conjunction with C++, since it is a subset of C++.

ABF/4GL is used since these are the database application languages supported and supplied by Ingres. ABF is used where ASCII terminals are required and 4GL is used when a graphical interface is possible, as it is on a Sun Workstation. Ingres was chosen simply because it is a good reliable distributed relational database. It supports all the functionality which the project requires, and is a widely used database package. MSCHED uses ABF since it has an HI screen for canceling requests.

M4 macros are used for database queries. If a failure in a query occurs, the query can be tried until it succeeds. The macros also provide error checking for other failures and provides a standardized way for error handling. The M4 macros are being phased out, and the Database Tool (DBT) is being phased in to provide all the necessary error

handling abilities when accessing the database. The DBT also provides another level of abstraction making it easier and faster to write applications. It is, essentially, the same as the Conversation Tool, which provides an abstraction away from communicating between the modules. Embedded SQL is used to provide an easy standard way of querying the data base. SQL is provided with Ingres and is a standardized query language. The DBT, as mentioned, above will reduce the amount of embedded SQL required in GWC modules and eliminate the need for M4 macros.

4.5 Required Support Software

As has been mentioned, several other software packages are required to operate the GWC. Ingres, for consistent data control, UNIX as an operating system, ISIS for communicating between modules of the GWC, and any specific software packages to control the machines (robots). Any and all of the software packages could be substituted by other packages, but this combination works sufficiently

5.0 System Data Structure and Communications Specifications

This section describes the conversations in which MSCHED, WC, and the Lot Script modules participate. A description of the database tables used by the modules MSCHED, WC, and the Lot Script is also in this section. Finally, there is a discussion of the communications used by theses modules.

5.1 System Data Base/Conversation Structure Specification

Each of the three modules communicates using the message bus. Conversations are classes which are essentially structures with a private section. Conversations over this message bus follow a strict format. When a conversation is initiated, it is broadcasted over the entire message bus. Any module which is interested in the conversation will receive it. All other modules will ignore it. The conversation header contains information about the conversation such as: a unique identifier, the number of messages in the conversation, and data to be passed from the sending module to the receiving module. These conversations are classes created using the Conversation Tool. Most conversations require a return acknowledgment, confirming the reception of the conversation and returning the results of the request made by the conversation. Each module has its own conversations as well as some common conversation. The conversations of the three modules are explained in this section. The database tables used by each of the three modules are also listed in this section with a short explanation of their use. Each module has access to a common database. This allows the sharing of information as easy as accessing the database. There are some conventions used when accessing the database. If the table is controlled by another module, then to update that table one must initiate a conversation with the controlling module, asking that module to update the table. If this convention is broken, then the integrity of the system is sacrificed.

5.1.1 MSCHED Conversations

Machine_Request: Request one or more machines to be bound to the requesting lot script.

Cancel_Request: Cancels a request for one or more machines which were requested by the lot script earlier.

Update_Machine_Status: If the machine status is changed to 'Shutdown' MSCHED removes all requests of that machine from the REQUEST_MACHINES database table. MSCHED

then sends the machine_request_complete acknowledgment to all of the lot entities which had outstanding requests with the shutdown machine. The requests cannot be filled since the status of the machine is now 'Shutdown'.

Unreserved_Machine: Is used to unreserved a machine which was reserved by the lot script earlier.

Accept_Type_Change: This conversation is only listened for. When MSCHED hears this conversation it runs the scheduler function. A machine may now be able to accept a request that it could not before. So if any entities can now be scheduled they will be.

5.1.2 MSCHED Database Tables

REQUEST_MACHINES: This table keeps a list of machines requested by lot entities, on the bus of what entity, and what combinations of machines have been requested.

RESERVED_MACHINES: This table keeps a list of machine which have been reserved by lot entities through MSCHED, and cannot be used by any other lot entities until the machines have been unreserved.

ME_CONFIG: This table is used to see if the requested machine exists or not.

MACHINE_STATUS: This table keeps track of the status of all the machines in the workcell. This table tells us if a machine is 'Shutdown', 'Idle', 'Running', etc.

5.1.3 WC Conversations

ME_START: This conversation is used to start the module and get it ready to receive requests.

ME_SHUT: This tells the WC to shut its self down, finish what it is doing, and not to accept any conversations accept ME_START.

SETUP_RUN: Downloads the recipe to the machine and get the machine ready to run the recipe on the machine.

START_RUN: Tells the WC to tell the ME to start running the recipe that was just sent to the machine from the SETUP_RUN message.

DNLD_ME: Downloads a recipe to the machine from the GWC.

UPLD_ME: Uploads a recipe from the machine to the GWC.

ALARM_SEND: This conversation is initiated by the ME. The machine will send the ME and

alarm. The ME will pass the alarm to the WC who will send the alarm to the Cell Alarm Server. This alarm is used to tell the operator to "Push the start button on the CMM".

ALARM_CLEAR: This conversation is listened for so the WC knows when the alarm to the operator "Push the start button" was acknowledged. When this message is seen the WC knows it may continue.

5.1.4 WC Database Tables

OP_MACHINE_RECIPE: Table contains information about the recipes. The information in this table is: machine_id, recipe_id, and op_type. This information is used to check to see if the recipe_id and op_type are valid for the machine.

MACHINE_RUN_RESULT: Is used to store the results returned from a run on a machine. These results are retrieved from a flat file created by the ME.

ALARM_DESCRIPTION: Is used to cross reference an alarm with known alarms in the database. This is used to tell the operator to push the start button on the CMM.

5.1.5 Lot Script Conversations

SETUP_RUN: Downloads the recipe to the machine and get everything ready to run the recipe.

START_RUN: Tells the WC to tell the ME to start running the recipe that was just sent to the machine from the SETUP_RUN message.

DNLD_ME: Downloads a recipe to the machine from the GWC.

UPLD_ME: Uploads a recipe from the machine to the GWC.

UNRESERVE_MACHINE: Unreserve a machine previously reserved by the script.

RESERVE_MACHINE: Request that MSCHED reserve a machine with specific abilities and specifications.

IMPORT_LOT: Ask to be imported into the workcell.

EXPORT_LOT: Ask to be exported from the workcell.

5.1.6 Lot Script Database Tables

The Lot Script does not directly manipulate the database at this time. It can however obtain information from the database if desired.

5.2 System Internal Data Structure Specification

The WC uses message queues to communicate with the ME half of the Machine Interface. Each queue has the following fields in it: file_name of the recipe to use, recipe_id to uniquely identify each recipe, message_id to identify what message it is, ack_code to ensure that the ME receives the message correctly, and alarm_text to identify the alarm. The CMM cannot directly send alarms such as fire or collision. Alarms are only used to tell an operator what to do, like push the start button. These alarms are actually generated by the ME since the CMM cannot generate them.

The other form of data structures are conversations, which are passed via message bus. A typical conversation consists of an initial message, multiple pieces of data, and an acknowledgment to the message as described earlier.

The Lot Script and MSCHED do not have any other significant data structure that have not already been discussed, but the Lot Script could have other structures if required.

5.3 Evaluation of Communications

The ME and WC are two separate programs which cooperatively coordinate communications between the machines (robots) and the GWC The two halves use UNIX message queues to achieve asynchronous communications. The message queues are used so that if WC sends ME several messages with little or no delay in between, the messages are not lost, instead they are queued to be processed when possible. The same is true going in the other direction. If the robot sends several messages to the cell with little or no delay, the messages are queued till they can be processed by the WC. This scheme seems to work well.

The Lot Script and MSCHED use the message bus as their communications interface. As an example, if the Lot Script built for NJIT workcell needed machines M1 and M2, the script would send a request to MSCHED for M1 and M2. MSCHED would respond with an acknowledgment telling the Lot Script that its request has been filled and the machines are reserved for it. If the machines were busy then MSCHED would put the request on MSCHEDs queue until the machine could service the request. If the request fails, it would mean the requested machines are not available. The Lot Script

could decide to try again or to try another machine. When making the request, the Lot can also make requests for a combination of machines: M1 and M2, or M1 and M3, or M4 and M5. If any of the combinations are valid, MSCHED will reserve that combination of machines. This method of communication between the two modules seems appropriate for our requirements.

6.0 Module Design Specifications

All modules have been designed in a analogous way so that the maintenance and additions to the modules require a minimal amount of effort. As described in section 5 the Lot Script, MSCHED and WC communicate through a common message bus with predefined messages. These modules operate using and Entity Server model[8], where entities request a service and a server provides the service to fill that request.

6.1 Operation Entity(scripts) Functions

An Operation Entity controls and monitors a workcell's manufacturing process during every step of the process. The Operation Entity (OE) defines which processes should be performed, how they should be performed, and which machines should perform the processes[8]. During the life of an operation entity, it controls how the workcell behaves. In a given workcell many operation entities may exist at one time[8]. OEs are created from a Lot Script. A single Lot Script may produce more than one OE.

The tasks that the Operation Entities must perform are:"

- Control a manufacturing process according to process and production rules.
- Determine which machines and machine recipes are qualified to perform the process.
- Coordinate the rendezvous of all resources required for a manufacturing step to be performed(e.g. product lots, tooling, machines, and dummy wafers).
- Initiate machine scheduling and initiate process runs according to schedule
- Monitor and report process quality.
- Determine process maintenance intervals and initiate maintenance.

"[8].

The operations described above are not all necessary nor are required for every operation entity. There may be no need to monitor the quality of the process, or maintenance a machine. This depends on the specific process the workcell performs, and on the sophistication of the machines in the workcell. If an intelligent machine such as the Mazak is used, the process quality can be monitored carefully. On the other hand, if a simple robot like the SCORBOT is used which at best can only determine that it collided into an immovable object. Hence, to ensure the quality of a product, the OE would be required to use another machine. A machine such as the Coordinate Measuring Machine(CMM) could be used to check the quality of the process. An

operation entity can evolve and learn from its history if written with the ability to learn. For example, the operation entity could learn the results from machine M1 are better than those from machine M2. It could also be written to tweek a recipe if it perceives that the recipe is off by a small amount. If a tool wears as it is used, the OE could modify the recipe so that the effect of the dulling tool would be minimized by using feedback. The intelligence described have not been applied to the NJIT workcell, but could be if desired.

In the future, Operation Entities will be served by the Operation Server(OS), but since the OS module has not been written yet, the OE must manage on its own. The OE is created through the HI where an order can be placed. To fulfill the order the appropriate Lot Script is started which becomes the OE. An OE can actually start other OEs if it requires the operation of those OEs. The OE is as flexible as the system it is running under, and the language it is written in. The OE in the Siemens-NJIT workcell is written in C/C++, but could be written in any language including LISP.

The Operation Entity for our workcell is in listing 1. This listing is the script which is executed by the HI when a chess set is ordered. The HI uses a UNIX fork/exec to start the script.

6.1.1 Logic of the Lot Script

The script first retrieves the lot_id from the HI upon execution. At this time most of the information needed by the entity is built into it. This design is acceptable, since the script has been defined to perform a specific function. If programmed, the script could also be dynamic. The NJIT script has the following information built into it:

---

op_type - "alum_chess", this entity is for measuring the aluminum chess pieces
machine_list - a list of machines to schedule, WC_CMM
num_machine=1 - number of machines to schedule
machine_run_id = 26551; This could be any unique number.
recipe_name - Name of the recipes to run, "alum_chess_set"
file_name - of the recipe on the disk, with the full path
lot_type - not really used yet...,"CMM_LOT"
priority = 20 - not used yet...sets the priority of the lot
param1,param2,param3- Result parameters...what a successful running of this script should return.
script_id - ID for this script non-changing
entity_id - Created by the log server each time the script is run
lot_id - assigned, valid tell the lot is export...If an entity is exported and then imported the lot_id will be different from the previous time

---

Once the Lot Script is started it will ask the Lot Server(LS) to become an entity

These steps are described in the test of the software in section 7. Once the script has become an entity, the entity must wait to be imported by the HI; this process is also described in section 7. After the lot has been imported, it sends a request for the needed machine(s). This is done by sending a Machine_Request to MSCHED (described in section 6.2). Each entity then waits for an acknowledgment from MSCHED. This is to confirm that MSCHED received the request and also to obtain the results of the request. If the request was filled the entity is given a machine_run_id, otherwise the machines were not available. The entity sends a SETUP_RUN message to the WC which homes the machine and downloads the recipe to the CMM. To home the CMM the ME sends an alarm to the WC in turn which is sent to the Cell Alarm Server(CAS). The CAS then sends the alarm to the Status Line Server. This displays the alarm on the HI, asking the operator to press the start button on the CMM Once the alarm is acknowledged, the WC continues. After the SETUP_RUN is accomplished the entity sends a START_RUN message which tells the CMM to run the recipe. The CMM carries this out and returns its results. The entity decides what to do next depending on the results of the run. Once the results are returned, the entity sends a request to HI to be exported from the workcell.

6.2 Machine Scheduler, MSCHED

The machine scheduler schedules or binds operation entities to machine(s) it requests. If the machine is not available then the MSCHED can reserve the machine for that operation entity's use when the machine is unreserved[11].

6.2.3 HI Screens

There is only one HI screen for MSCHED. The screen is used for canceling the requests and observing what requests are currently scheduled. The interface allows an operator to easily scan through the scheduled requests. Once the operator finds the request he wishes to remove, he needs only to highlight the request and press a key to remove the request. The screen follows all the same look and feel conventions used by all the other screens in the HI. This operation calls the MSCHED Cancel_Request function.

6.2.4 MSCHED Functions

The Machine Scheduler can perform several functions. The functions performed by the MSCHED can only be done only by MSCHED. A diagram of MSCHEDs

operations is shown in Figure 5. As seen in Figure 5, cell modules interact with MSCHED to acquire services from MSCHED.

6.2.4.1 Startup

The startup function used in MSCHED is modeled after the startup function in other modules such as the Cell Alarm Server (CAS), Entity Recipe Server (ERS), and Machine Server (MS).

The Startup function is received by MSCHED and an Acknowledgment is returned to the initiating module. MSCHED makes sure that the REQUEST_MACHINES, and RESERVED_MACHINES tables are empty. MSCHED will change its conversation filter from STARTUP to SHUTDOWN, MACH_REQUEST, CANCEL_REQUEST, UPLOAD_ME, ACCEPT_TYPE_CHANGE, and UNRESERVE. Since the module is running, it will not listen for a startup message until it is shutdown first. Once the startup is complete MSCHED sends a Startup_Complete message to the initiator of the conversation, and MSCHED idles until it receives a request.

6.2.4.2 Shutdown

As with the Startup function, this Shutdown function was modeled after other Shutdown functions. The first job of this function is to send a shutdown acknowledgment to the sending module. The filters is set to Startup. This module cannot offer any services until it is started up again. Shutdown goes through the REQUEST_MACHINES table and sends a Machine_Req_Complete(NO_MACHINE) for each request that it finds. For each Machine_Req_Complete message sent, the MSCHED listens for an acknowledgment message to ensure it was received. The module is shutdown at this point. So MSCHED sends a Shutdown_Complete message to the module which sent the shutdown request. MSCHED will now idle until a startup message is received.

6.2.4.3 Machine Request

The machine request is the most complex part of MSCHED. It must keep track of all the requests for every machine. The first function is to receive the multiple second messages which provide the machine id(s) to be requested. The machine request is a multiple message conversation, where part of the first message is how many messages are to follow. The machine requests are stored in a linked list class The convention

used for requesting machines is: If a lot entity wants machine M1 & M2, or M1 & M3, it will make two requests. The first request will have two messages, one for M1 and the second for M2. The second request will be the same as the first but for M1 and M3. This allows us to request machines using both logical expressions AND and OR. The AND is done when a request is made like M1 AND M2, and the OR is between different requests; (M1 AND M2), OR (M1 AND M3).

After all the requests have been put on the linked list, MSCHED makes sure that none of the requested machine are shutdown. If a machine is shutdown the request is canceled and removed from the linked list. If all of the requests of a lot entity are removed from the list, the lot entity is sent a message telling it that the request cannot be satisfied.

If the request is valid, it is given a unique request_key number to identify it, and the request is put in the database table REQUEST_MACHINES, and a Machine_Req_Ack is sent to the requesting lot entity.

6.2.4.4 Cancel_Request

The cancel request conversation takes a (request_key, entity_id) and removes all occurrences of that (request_key, entity_id) from the REQUEST_MACHINES table. If no request were removed from the table, the Cancel_Req_Ack tells the lot entity there were no requests to be removed.

6.2.4.5 Update_ME_Status

This function is to monitor any changes in the status of a machine. It does nothing unless it sees that the status of a machine has been changed to 'Shutdown'. Once it has determined that a machine has been shutdown, it takes action to cancel all requests for that machine. This is done in the following manner: First MSCHED goes through the MACHINE_REQUESTS table to find all occurrences of the shutdown machine. If any entities are waiting for the machine, the entities must be told that the machine is no longer available, but only if there are no alternate requests which can still satisfy the request. As an example, an entity could request machines M1 or M2. If M1 is shutdown, the request could still be satisfied by M2. There would be no reason in this situation to tell the entity anything. If the request cannot be satisfied then MSCHED must send a Machine_Req_Comp(NO_MACHINE) message to the entity(s) requesting the machine.

6.2.4.6 Accept_Type_Change

This conversation is listened for by MSCHED to trigger the scheduling function to be run. The scheduling algorithm which is the heart of this module is a simple First In First Out (FIFO) algorithm. FIFO was chosen because of its simplicity and fair scheduling scheme. MSCHED listens for this message, and once it is heard the scheduler goes through the REQUEST_MACHINES table looking for the oldest request in the table. Once the request is found, the number of machines requested is tallied. The scheduler checks to see that each and every machine requested is available using the AND and OR method described earlier in section 6.2.4.3. The machines must not only be available but they must have the proper ACCEPT_TYPES for the request. If all this is satisfied by at least one of the requests, all the requests of the given request_key are removed from the REQUEST_MACHINES table. The machines which satisfy the request are then reserved by putting them on the RESERVED_MACHINES table, and a new machine_run_id is created. The requesting lot entity is sent a Machine_Req_Comp(Success,...) telling the entity that its request has been satisfied along with information about how it was satisfied.

6.2.4.7 Unreserve_Machine

This allows a machine which was previously reserved to be unreserved This is used when an operator determines that he does not want to perform the task which the machine was reserved for. The lot entity may have crashed after it reserved a machine. In this case the operator can clean up the database quickly and easily through the HI. This must be done so that the scheduler does not try to schedule a machine for a lot entity which does not exist.

6.3 Machine_Interface (WC/ME)

The Machine Interface is the module which lets the machine communicate with the rest of the cell. This module is broken into two sections to help modularize the code as much as possible. Of the two sections one communicates with the workcell (WC), and the other communicates with the machine (ME). The two small modules communicate with each other through message queues, which on UNIX is part of the shared memory.

6.3.1 Machine Interface Workcell Side (WC)

The WC half of the Machine Interface provides the communication between the GWC and the ME half of the Machine Interface. This module is critical in the operation of the workcell. Without this module the workcell would not be able to communicate with external machines.

6.3.2 WC Functions

The WC provides several functions which allow the GWC to communicate with machines in the workcell and vice versa. The functions provided are: STARTUP, SHUTDOWN, SETUP_RUN, START_RUN, DOWNLOAD, UPLOAD, RUN_COMPLETE_ACTION, and sending alarm for the machine.

6.3.2.1 Startup

The startup function is to get the module ready for use by the workcell. When a startup message is received, the WC first confirms that the machine is shutdown, and if the WC is shutdown it responds with a startup acknowledgment. Otherwise it returns an acknowledgment that the startup has failed.

If the startup is acknowledged the filters are changed to SHUTDOWN and ALARM_CLEARED. The WC sends the ME a ME_START message over the queue and waits for an acknowledgment. The first response from the ME should be an alarm telling the operator to press the start button on the CMM so the CMM can be homed. The WC passes the alarm on and waits until it sees an ALARM_CLEARED conversation. This conversation indicates to WC that the operator responded to the alarm. WC sends ME the news and waits for ME to send it an acknowledgment back This acknowledgment tells WC that ME is ready for further instructions. WC's interests are changed to listen for SHUTDOWN, SETUP_RUN, START_RUN, DOWNLOAD, and UPLOAD.

6.3.2.2 SHUTDOWN

The shutdown conversation will change the filters to accept nothing. WC does not want to hear any conversations while shutting down because this is critical to ensure the database is left in a proper state WC then confirms that its status is not

shutdown and changes the status of the machine to shutdown. If the change of status failed then the shutdown request fails and returns an error to the caller.

If there is not an error, WC will send a shutdown message to ME half telling it to shutdown. WC listens for an acknowledgment and changes its filters to listen for startup conversations.

6.3.2.3 SETUP_RUN

The setup run conversation is to get the machine ready for use by an operation entity. The WC confirms that the machine is "idle", and if it is idle the WC will continue.

The WC checks the OP_MACHINE_RECIPE table to confirm that the operation_type and the recipe_id are valid for the machine_id The WC then proceeds to download the recipe to the machine. WC waits for an acknowledgment from ME and then changes its status to "setup". WC sends an acknowledgment over the bus to the calling module telling the module that it is finished.

6.3.2.4 Start_Run

The start_run message first confirms that the machine is in a "Setup" status which will allow the recipe to run. After this is confirmed, it sends a start_run message to the machine and waits for an acknowledgment. When the acknowledgment is received, the status is changed to "Running". The last operation done by this function is to send an acknowledgment back to the calling module telling it that start_run failed or succeeded.

6.3.2.5 Upload

The upload is used to send a recipe to the machine from the GWC. The status is checked to see if the machine is "idle", and if it is, the WC sends the request to ME. After the upload is finished, an upload acknowledgment is sent to the calling module over the bus.

6.3.2.6 Download

The download is the exact opposite of the upload. It transfers a recipe from the machine to the GWC. It functions in the same manner as upload.

6.3.2.7 Run_Complete_Action

This occurs whenever the WC get a run complete message from the ME. The WC updates the database table with the time when the run complete occurred, and changes the status to "now". The results file is opened and the contents of the file are put into the MACHINE_RUN_RESULTS table in the database.

A Run_Complete message is sent out on the bus and the status of the machine is changed to "idle". The accept types for the machine are set to "all operations" and the processes finished.

6.3.2.8 Alarm_Action

Alarms are sent from ME to WC where they are passed on to the Cell Alarm Server (CAS). When an alarm is received the WC checks the ALARM_DESCRIPTION table to confirm that the alarm is valid. It then sends the alarm and waits for an operator response. This is signalled by an ALARM_CLEAR conversation over the bus. Once this is received, the WC is allowed to continue what it was doing before the alarm occurred.

6.4 Module Design Summary

The three modules described all function together closely to make the GWC operate correctly. The conversations described allow the services of a module to be utilized by other modules. The modules as described earlier make requests of other modules through the conversations. These conversations map directly to the functions in each module. For example, the Machine Request conversation is a function of MSCHED allowing any other module to request a machine. The Operation Entity module will use conversations to request services from MSCHED, Machine Interface (WC half), and other modules in the GWC. This is how the Entity Server model works: nd entity requests a service and a server services the request.

7.0 Demonstration of the System

To demonstrate that the modules which I wrote work correctly, the entire system must be demonstrated. It is not feasible to demonstrate the Lot Scripts without the MSCHED, which is not feasible to demonstrate without the WC, which cannot be demonstrated without the ME..

The system cannot be proven to be better than other systems without several detailed case studies. Due to timing constraints no case studies were done.

7.1 Items/Functions to be Demonstrated

The Lot Script, MSCHED, and WC are demonstrated to show that they function as expected.

7.2 Description of Demonstration

The demonstration involves running lot entities on the workcell as if the workcell is statically configured. The demonstration will also involve starting and stopping machine modules while the workcell is running to show the dynamic abilities of the GWC. Multiple lot entities will be started to show that MSCHED does handle the scheduling of multiple lots correctly.

7.3 Justification of Demonstration

To show that the Lot Script, MSCHED, and WC function correctly a demonstration is done. The demonstration only needs to show that the modules work, not that they are unbreakable.

7.4 Demonstration Run Procedures and Results

The first step is to confirm that both ISIS and Ingres are running on the workstation. Once this has been confirmed the GWC is started by running a CSH script start (listing 4), or if the cell was running, we stop it by using a CSH script stop (listing 5) to ensure a consistent state in the cell. The startup script sets the log level for each module. After the last module is started, the script runs the HI module (Figure6). From the HI, we go into the system service menu where we can start the modules of the GWC (Figure 6.1). The GWC will report that the cell was started successfully if there were no problems. Once the cell is running the operator goes to the Work Order Main Menu

(WOMM) (Figure 6.2) where he can make a Work Order Entry (WOEN) (Figure 6.3). The operator fills out the appropriate information: name, address, city, state, zip, and selects the product to be built from the product list. At this point the HI will start the Lot Script to carry out the order made in the HI. The Lot Script will take control of its destiny as described in section 6, thus turning itself into an operation entity. The entity will request to be imported into the workcell so it can use the workcell's resources. The Status Line Server tells the operator that a lot entity is waiting to be imported. The operator can then go to the Dispatch List (DISP) (Figure 6.4) and import the lot entity (Figure 6.5). Several lot entities may have been created by one order in the HI, hence several lot entities my request to be imported. If there are several lots in the workcell an operator can check to see the status of a particular work order by looking at the Work Order Status screen (Figure 6.6). This screen displays the status of the order, customer name, due date, and the work order number. The work order number indicates the requests in which the orders will be serviced. Information about what is currently being serviced by a machine can be obtained by looking at the View Machine Status screen (Figure 6.7), which tells the operator what Recipe is being run at the moment along with the start time of the recipe, the machine run id, and the status of the machine. If the lot in which the operator is interested is not running on a machine at that time, and it has been imported, it should be waiting for a machine. The operator can check on the lot entity by looking at the Delete Machine Requests screen (Figure 7.8). This allows an operator to remove a Lot entity request from the scheduling queue. After the lot entity is finished using the workcell, it requests the GWC to be exported. This causes the Status Line Server to notify the operator of the request made by the lot entity. The operator can then export the lot in the same way the lot was imported (Figure 6.9). The lot entity could go to other workcells to do more work, but this has not been implemented with our implementation since we only have one workcell operating at this point.

The lot entity as discussed in section 6, operates independently of the HI, except when it requires importation or exportation to the workcell The Machine Scheduler selects the lot entity to be serviced by the requested machine. To check if the workcell can operate with multiple machines, a dummy machine was built. The interface to the dummy machine is exactly the same as to any other machine The module is split into two parts: machine side MEN, and the workcell side WCN. The 'N' stands for N dummy machines. In addition to the above, machine modules were also started and stopped during the test to see if MSCHED handled dynamic removal and insertion of

machines to the workcell.

7.5 Discussion of Demonstration

All the modules of the workcell cooperated to control the operation of the workcell in a smooth and predictable manner. MSCHED did not try to schedule lot entities to run on a machine after it was shutdown, and when a new machine was added, MSCHED scheduled lot entities to the new machine. The WC functioned correctly as did the Lot Script. The demonstration was successful in showing that the modules functioned correctly as described in section 6. Some operations of WC were not demonstrated, upload and download functions, due to time constraints (see section 8). These are easily implemented using KERMIT to transmit the files from the SUN to the Compac.

The demonstration shows that MSCHED does startup, shutdown, machine_request, cancel_request, update_me_status, accept_type_change, and unreserve_machines. This is shown by the fact that the CMM runs a recipe and when done the CMM is unreserved and ready to be requested again.

In the demonstration the Operation Entity (OE) has worked correctly since. Specifically, the OE has control of the workcell telling it how to function. Had the OE not worked nothing would have happened.

The Machine Interface (WC half) has worked correctly since the CMM ran the recipe and returned a result. This shows that the startup, shutdown, setup_run, and start_run of the WC were functioning correctly. If they did not, the run result would not be in the database, which it is. This also demonstrated that the communications between the WC and ME work correctly.

8.0 Conclusions

This sections will summarize the project by discussing the problems encountered while developing the system. It will also suggest ways to approach the problems encountered and solve them. There are also suggestions for future extensions to the current project. Last credit is given where deserved.

8.1 Summary

The GWC project has been a success as far as providing to the manufacturing world a true generic workcell. The ability to dynamically expand and reduce a workcell was achieved. The startup time of a completely new workcell has been reduced by a considerable amount, and control of the GWC at NJIT was successful. A study done by N. A. Duffie comparingCentralized, Hierarchical, and heterarchical systems are as follows:"

	Centralized Controller	Hierarchical Controller	Heterarchical Controller
Line of source code	680	2450	256
Software development cost(1)	$17,000	$61,250	$6,475
Expansion software cost(2)	$17,000(6)	$960	0
Machine utilization(3)	-	64%	60%
Memory requirements(4)	-	50,680	7,104
Average CPU utilization(5)	-	20%	60%
Complexity	low	highest	lowest
Flexibility	lowest	high	highest
Modifiability	lowest	high	highest Fault
tolerance	lowest	high	highest
Intelligent parts	no	no	yes

1-At $25 per line of source code
2-24 hours per machine added at $40 per hour
3-From simulated fault-free cell operation, random part mix
4-Scheduler bytes plus stack and data segments
5-Four cell processors, scheduling 50%, machine and part 10%
6-Complete redevelopment required due to explicit sequencing
"[4]

This study shows that the heterarchical system outpreforms the other systems considerably in all of the comparisons. Since the GWC is also a Heterarchical system, it should have similar results.

8.2 Problems Encountered and Solved

There were two types of problems encountered during the development of this

project, technical and non-technical.

8.2.1 Technical Problems Encountered and Solved

One of the technical problems was to help port the entire system from VMS to UNIX. Due to different manufactures of the compiler, and different implementations of C++ handled destructors, many problems needed to be resolved. In addition, due to different file systems all references to the file system had to be changed. The communications between the two halves of the machine interface required much thought and effort to find and develop a better interface between the two halves of the module.

The system set up at NJIT caused another major problem. The different Ingres installations, required some work. Richard Meyer and I were able to setup the dummy machines, and then we started work on the communications between the GWC and the CMM. To setup the communications between the CMM and the SUN, a serial cable had to be run from the first floor to the second. The hardest problem was yet to come. The communications between Xenix and SUN OS/UNIX was not as simple as one would expect. Each system has a specific serial port(RS232) settings which brings to attention the intersect differences between the two operating systems. After this was solved several smaller problems were tackled and the cell was successfully implemented. There is a design flaw in the message bus which became apparent at NJIT. Conversations can collide and cause a deadlock if the modules are not programmed carefully. The patch to circumvent this problem was to put to sleep in the problem causing sections. This seem to allow the other conversation to get through and prevent the deadlock.

8.2.2 Non-Technical Problems Encountered

The project checkpoints, dates due, and dates done are listed in the following table:

Project Checkpoints	Date Due	Date Done
Finish porting system to UNIX	June 20	June 20
Build a new WC interface	July 10	July 10
Build a Dummy WC/ME	July 20	July 20
Port the GWC to NJIT	September 2	November 15
Build the CMM WC	August 15	December 4
Build the CMM ME	August 15	December 12
Demonstrate the GWC with the CMM	October 1	December 12

The dates are not exact, but an approximation of the original schedule laid out for the project by both NJIT and Siemens management.

Richard Meyer and myself found that we unexpectedly had to deal with problems at NJIT involving hardware. We found faults faults in serial cables and get them replaced ourselves. The system changed in the middle of the summer from National File System(NFS) to Andrew File System(AFS), and the two system are not fully compatible. The database at NJIT was an older version than at Siemens, and was not setup as a distributed system. The C++ compiler was an old version and it took around two weeks to get the new compiler installed. Passwords were changed during the week so when we came to work on the weekends we could not proceed any further. The configuration of the system seemed to change almost weekly making it hard to find faults in our system. This resulted in not knowing if it was our software or the system causing the faults. Many of the problems would have never occurred if the original hardware request was filled exactly as we asked. These problems caused a time delay in the expected completion.

8.3 Suggestions for better Approaches to Problem/Project

The maturity of the project has gotten to a stage where it does not seem to be any basic design flaws except the problem with conversations colliding occasionally causing a deadlock. This problem was amplified at NJIT since the SUN we are using is not as fast as the machines at SCR, and the entire Ingres package is running on the machine At SCR we only run the front end on the local machine, and the backend is on a server. Most probably there are other implementation problems, but they are not apparent yet due to incomplete testing of the system potentials.

8.4 Suggestions for Future Extensions to Project

The GWC has not been fully implemented. New modules can be used to expand the performance of the existing GWC system. The design of all the crucial modules has been done, but there are probably some design problems which are not apparent yet. The GWC is designed so any number of modules can be added to the system. The implementation of the GWC at more production sights would expose faults in the system. Once the GWC modules are understood, rewriting them using more advantages of C++ is feasible.

The future should be very promising for the GWC. The GWC could have a

graphical user interface with a graphical language to control the workcell. So even a manager with no technical abilities could program the workcell to produce a new product. Also the GWC could be designed as a system and not just an application on a system. This could lead to the redesign of the GWC to work as a hard, real time system instead of a soft system[17]. It could be implemented on a hard real time system, and a Real Time Server(RTS) could be designed to ensure hard deadline preserving. This would open up many areas for the GWCs use not available now. The education of people as to the abilities of a GWC should be done so it is accepted and used. It is too valuable to neglect.

8.5 Acknowledgments

The GWC project has been a long term project which was started several years before I became involved. The project over the last four years, has grown to what it is today, but was not possible without the contributions of many people. The principal investigators Dan Wolfson and Paul Bruschi, with the contributions of other members of the project: Fred Brehm, Frank Maslar, Rick Taft, Ellen Voorhees, and last but not least Richard Meyer were also critical to the development of the project

I would also like to thank my advisor and principal investigator at NJIT Dr. Alexander D. Stoyenko of the CIS Dept. Still at NJIT there were several other people instrumental to the success of the project: Reggie Caudill of the CMS Dept., David Perrel of the CSD Dept., and Allen Bondhus of the CMS Dept.

I am exceedingly grateful to Siemens Corporate Research for providing the capital and financial support which made this project possible. I am also very grateful to SCR for giving me the opportunity to gain experience in an exciting area of computer science.

9.0 Bibliography

1. Dan Wolfson & Paul Bruschi, A Reconfigurable Generic Workcell Architecture, Siemens Corporate Research, Princeton NJ., July 9 1991, Internal Document.

2. Dan Wolfson & Paul Bruschi, A Generic Workcell Architecture, Siemens Corporate Research, Princeton NJ., 1990,Internal Document.

3. Dan Wolfson & Paul bruschi, A Generic Workcell Controller, Siemens Corporate Research, Princeton NJ., 1990,Internal Document.

4. Neil A. Duffie And Rex S. Piper, Non-Hierarchical Control of a Flexible Manufacturing Cell, Robotics & Computer-integrated Manufacturing, Vol. 3, No. 2, pp. 175-179, 1987.

5. N. A. Duffie, Hierarchical And Non-Hierarchical Manufacturing Cell control with Dynamic Part-Oriented Scheduling, North American Manufacturing Research Conference. 14th: 1986

6. S,P. Rana & S. K. Taneja, A Distributed Architecture for Automated Manufacturing Systems, The International journal of Advanced Manufacturing Technology, 3(5), 81-98, 1988.

7. D.D'Amore & G. Coleman, A Modular Approach to FMS control Systems, Advanced Manufacturing systems, Exposition and conference 24-26 June 1986.

8. D. Wolfson, The Entity-Server model for Generic Cell Control, Siemens Corporate Research, inc (RTL-88-TR-176) 1988.

9. Andrew P. Black & Yeshagahv Artsy, Implementing Location independent invocation, Distributed Systems Advanced Development, Digital Equipment Corporation, Distributed Systems Conference,TX, IEEE 1989.

10. Peter A. Murray, A Binding View of Distributed Systems, NJIT CIS 651, 1990. Report

11. Dan Wolfson, Machine Scheduler Specification Version 2.2, SCR, Inc.,SCR-89-TM-232, March 1990, Internal Document.

12. Frank J. Maslar, Cell Alarm Server Specification Version 1.0, SCR, Inc., SCR-89-TM-226, March 1990, Internal Document.

13 Dan Wolfson, Donna Scharkss, Paul J. Bruschi,Equipment Recipe Server Specification Version 1.1, SCR, Inc., SCR-89-TM-227, March 1990, Internal Document.

14. Michael J. Shaw, Dynamic Scheduling in Cellular Manufacturing Systems: A Framework for Networked Decision Making, University of Illinois at Urbana-Champaign, Champaign, Illinois, Journal of Manufacturing Systems Volume 7 No. 2.

15. ØYVIND BJØRKE, Towards Integrated Manufacturing Systems Manufacturing Cells and Their Subsystems, Robotics & Computer-Integrated Manufacturing, Vol. 1. No. 1. pp 3-9, 1984.

16. Alexander D. Stoyenko, Rich Meyer, Peter Murray, Functional Requirements for a CMS Generic Workcell, Department of computer science and information science. NJIT University Heights Newark NJ, 1991.

17. Wolfgang A. Halang, Alexander D. Stoyenko, Constructing Predictable Realtime Systems, Addison-Wesley Publishing Company © 1991.

11.0 Figure/Program Listings Directory

10.1 Figure 1

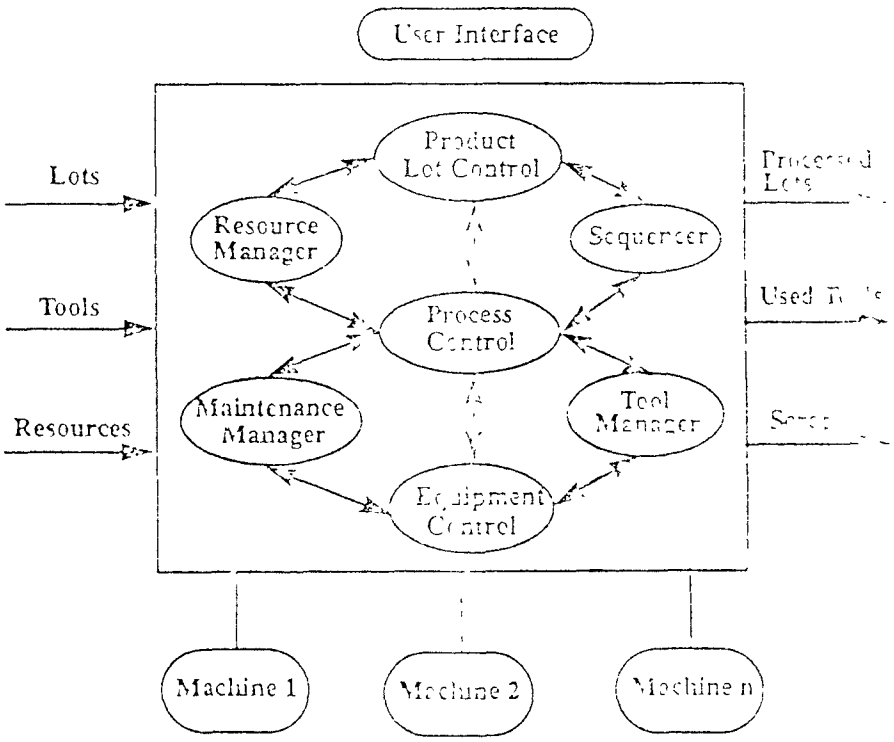
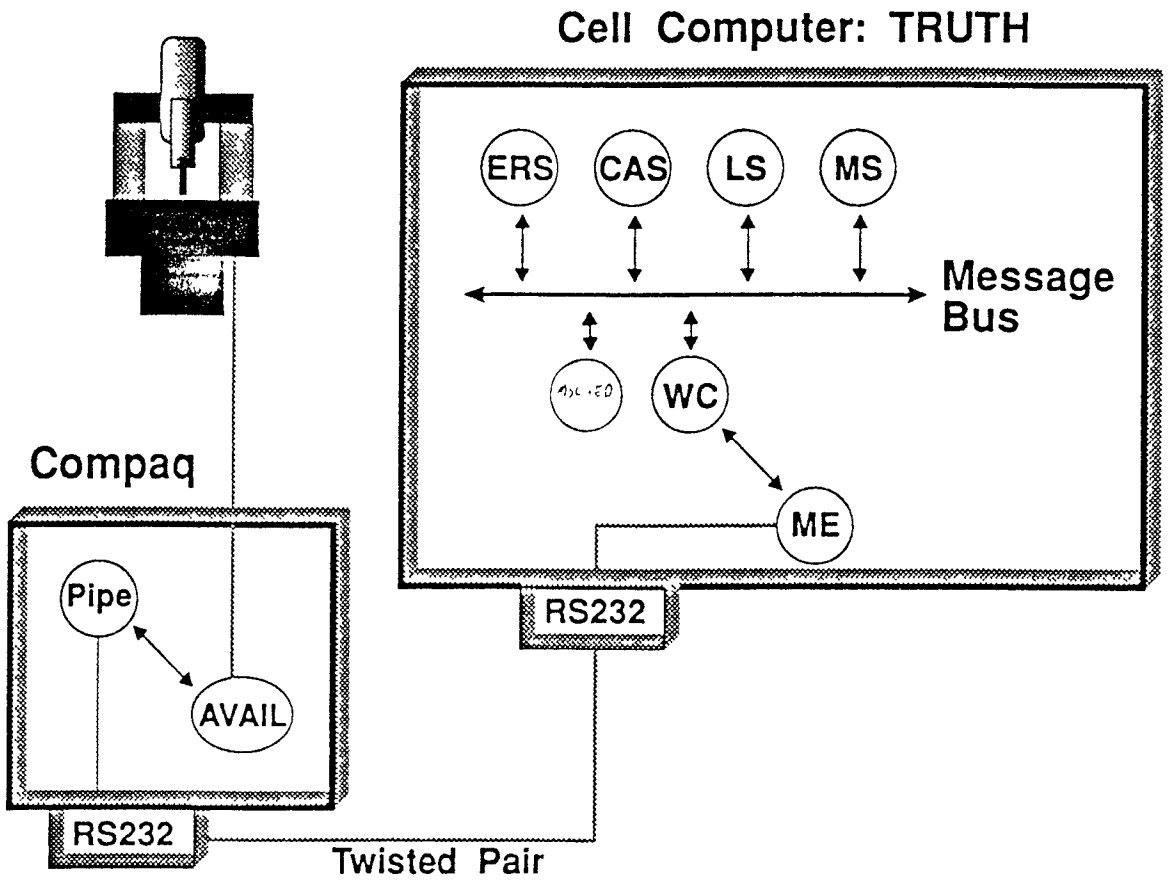


Figure 1. Conceptual Workcell Environment

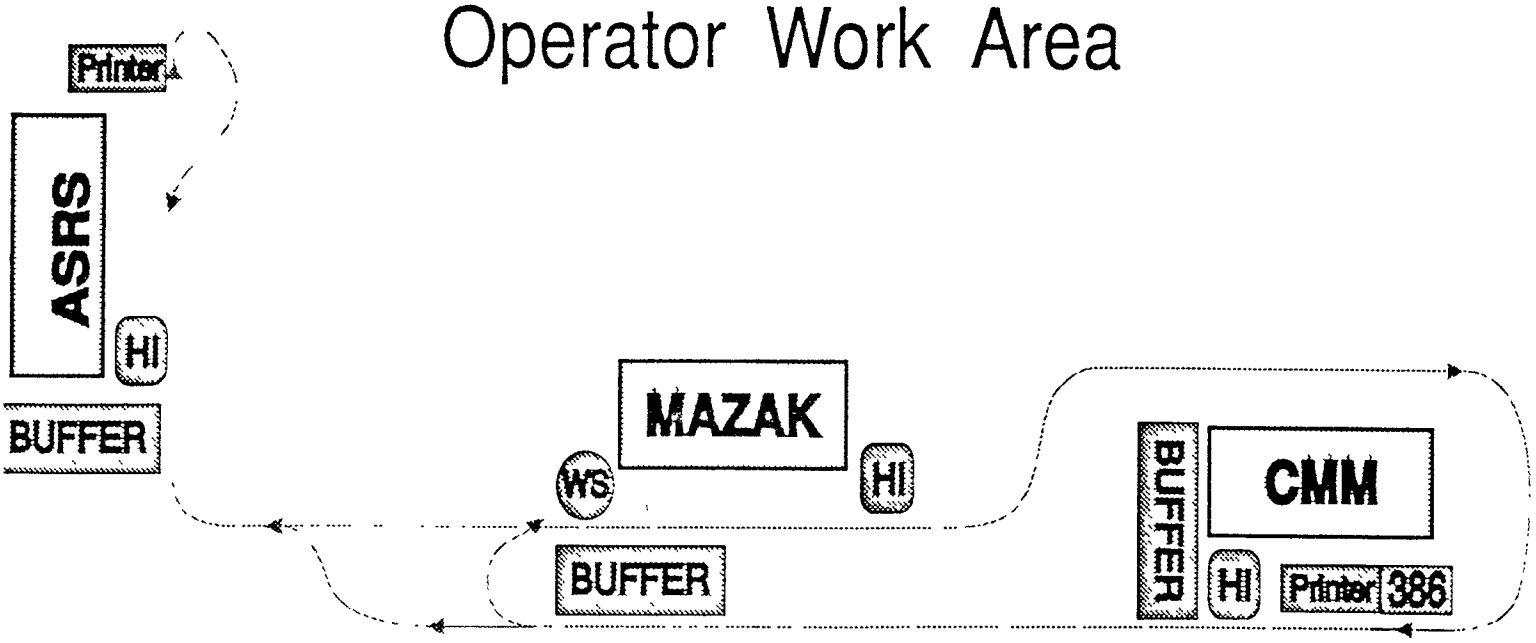
10.2 Figure 2, Log File entry.

```
====================================
From    init_mq
Date    Fri Dec 13 11:09:25 1991
User    bruschi
Term    /dev/tty
PID     2899
--------
Note:
 initialize message queue
--------
====================================
From    init_mq
Date    Fri Dec 13 11:09:26 1991
User    bruschi
Term    /dev/tty
PID     2898
--------
Note:
 initialize message queue
--------
====================================
From    main
Date    Fri Dec 13 11:09:28 1991
User    bruschi
Term    /dev/tty
PID     2898
--------
Note:
 Starting
--------
```

10.3 Figure 3, GWC Diagram

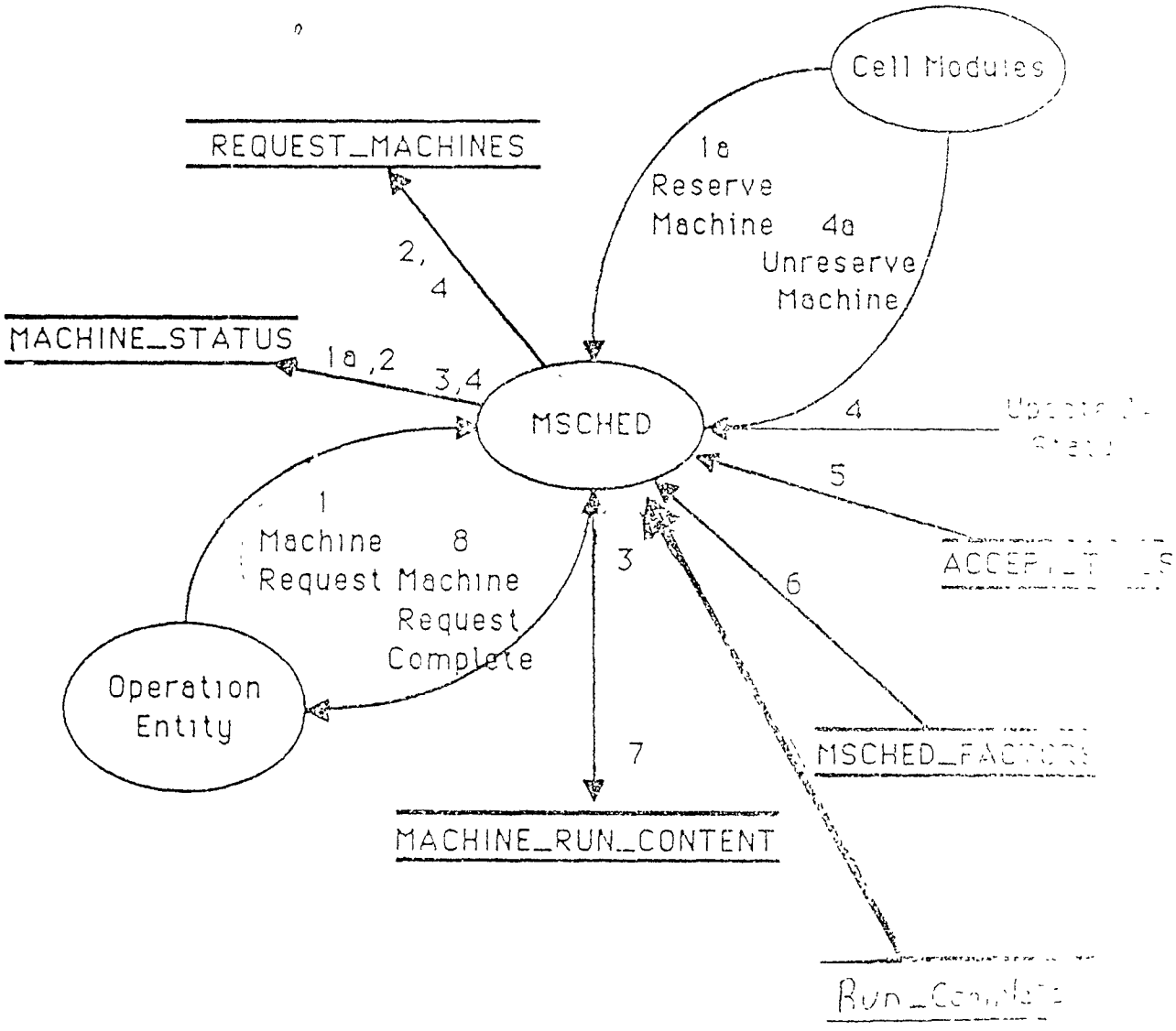


10.4 Figure 4,SIEMENS-NJIT GWC Floor lay out.

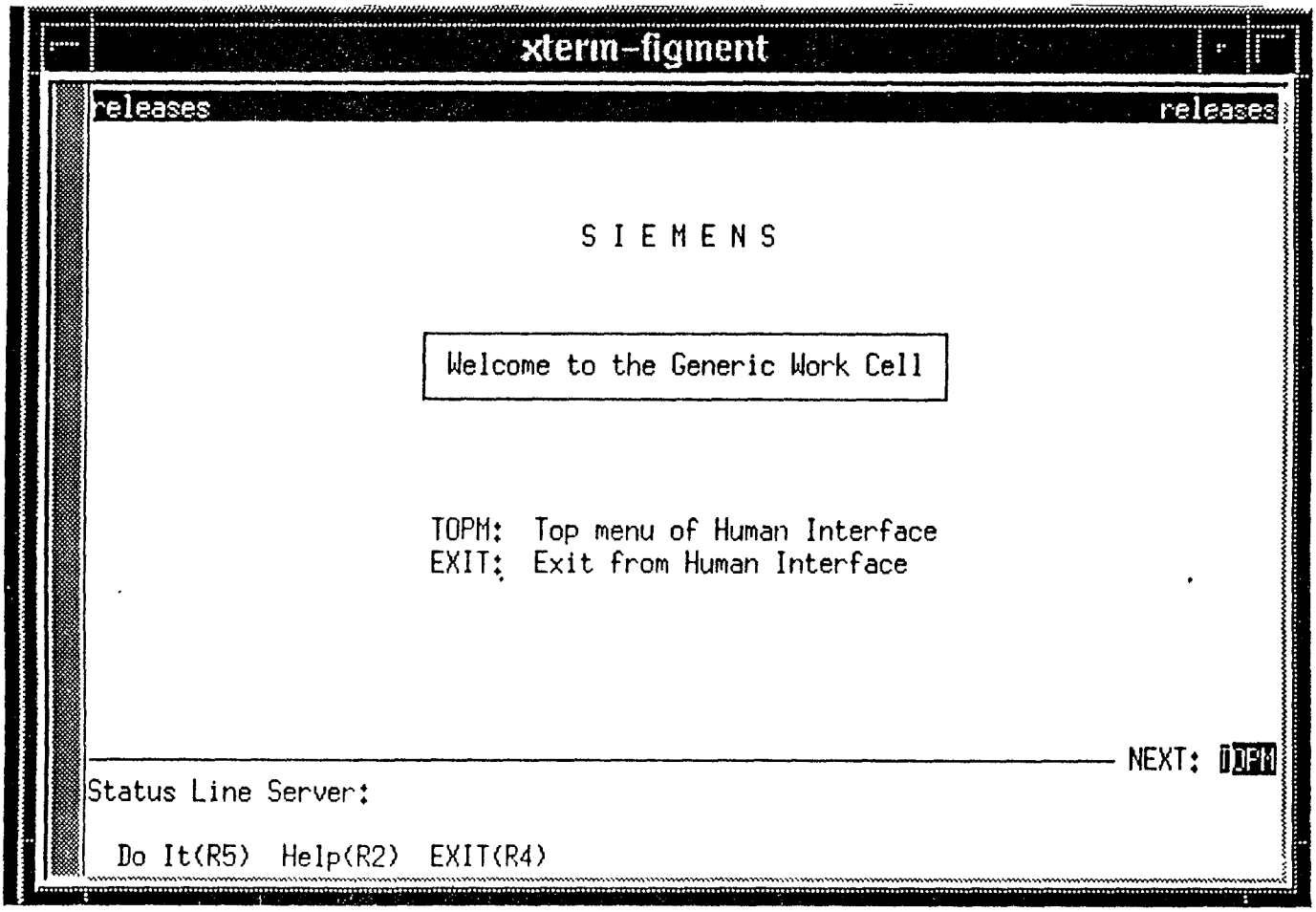


Reference 16.

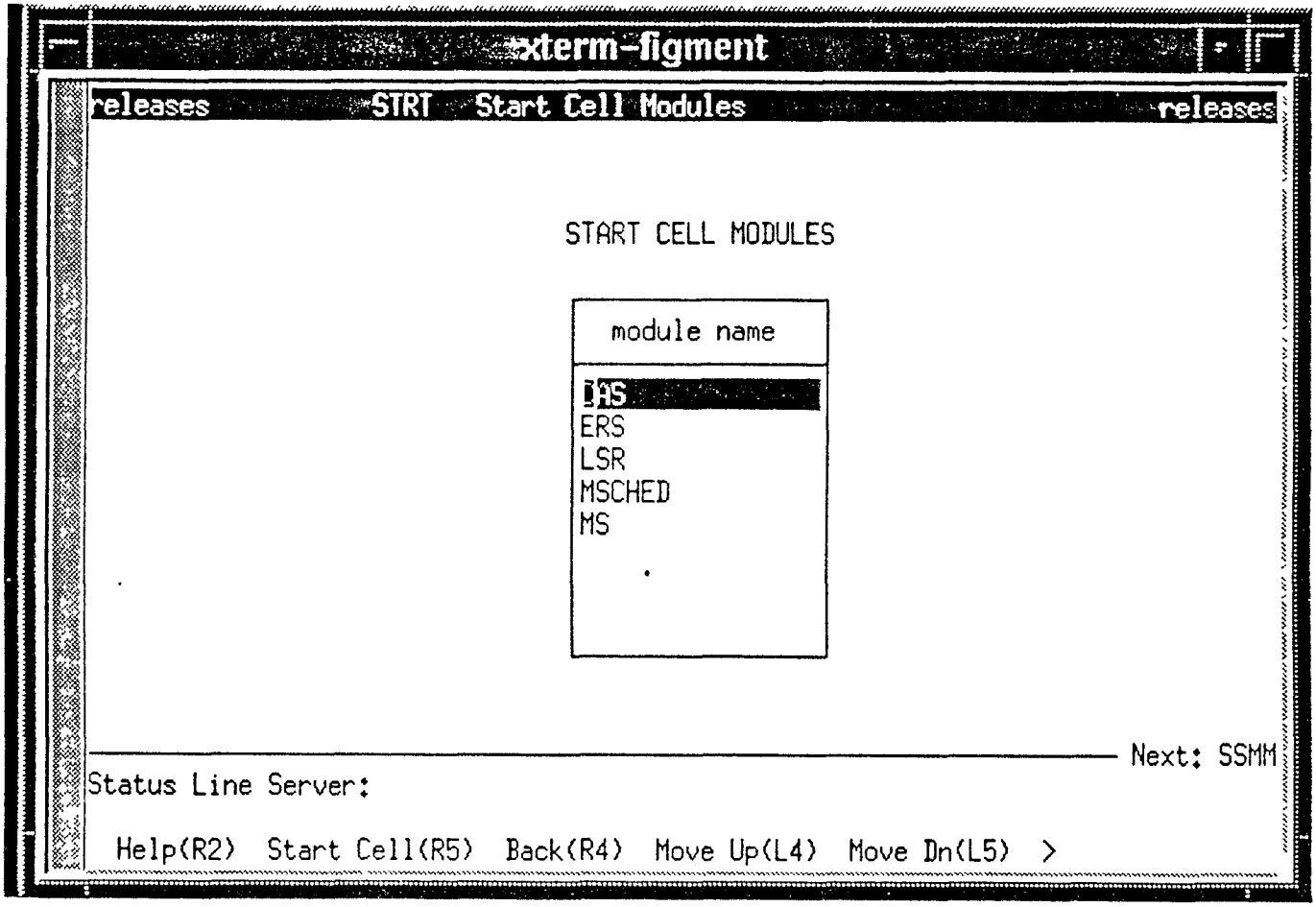
10.5 Figure 5, MSCHED



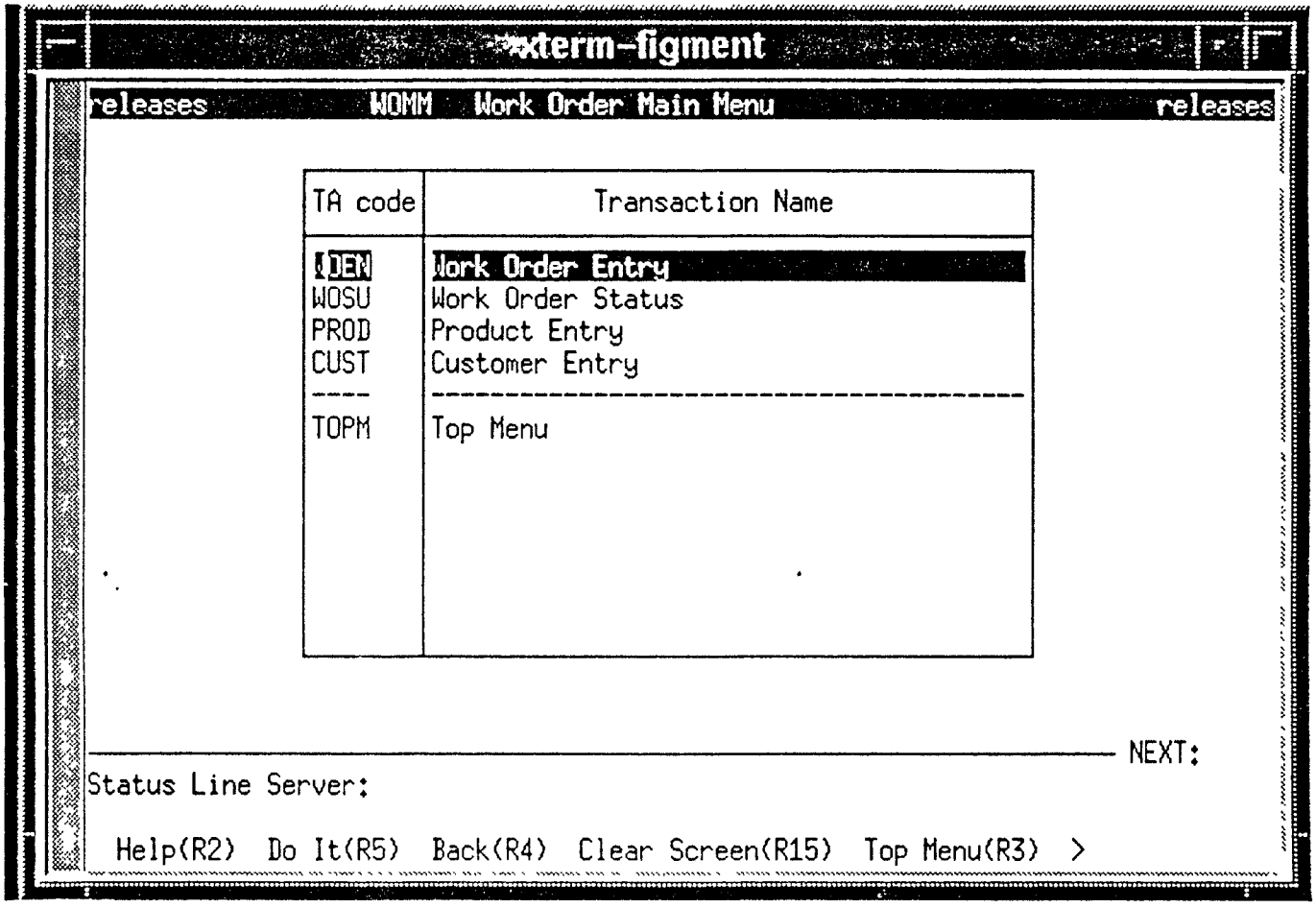
10.60 Figure 6.0



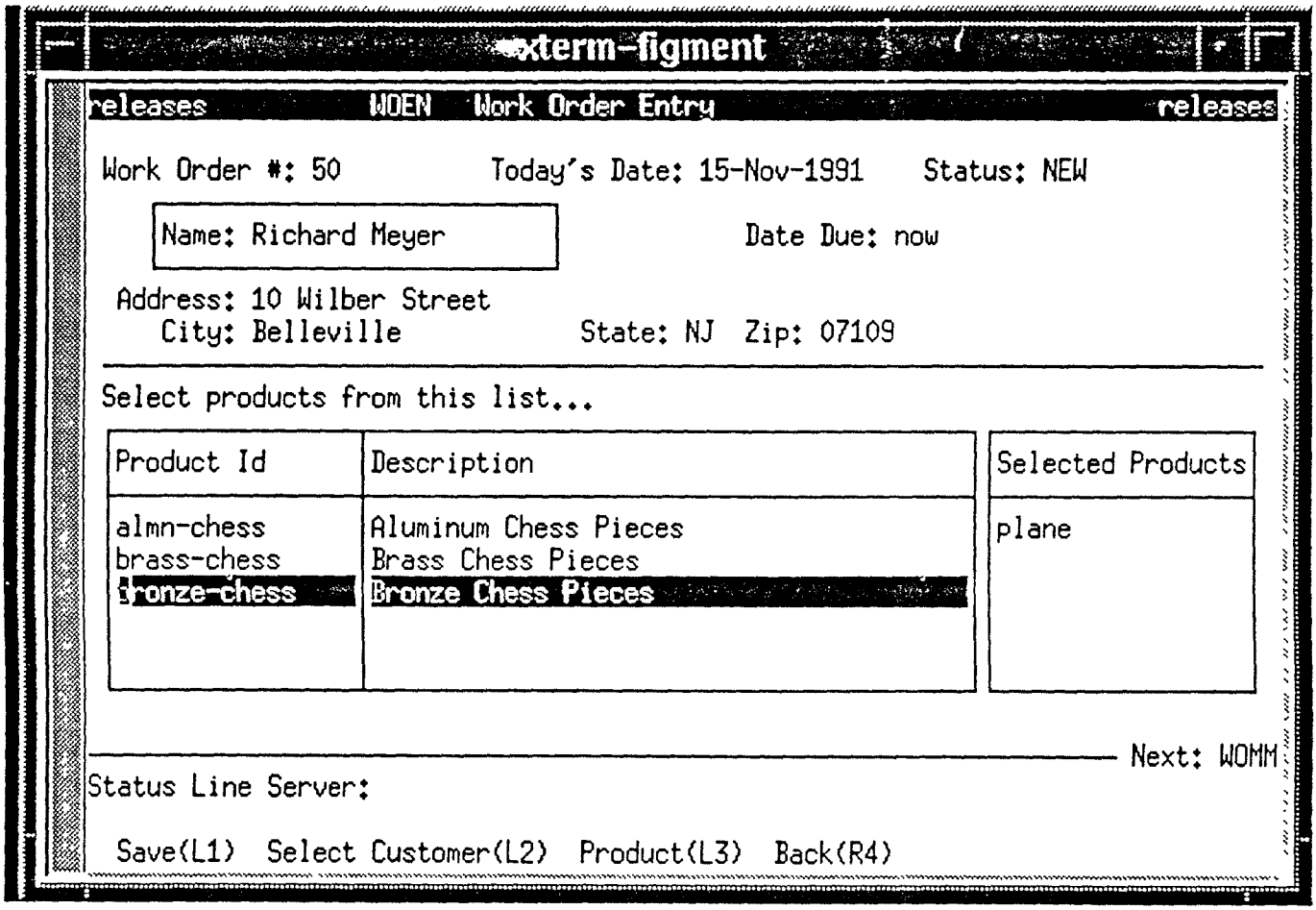
10.61 Figure 6.1



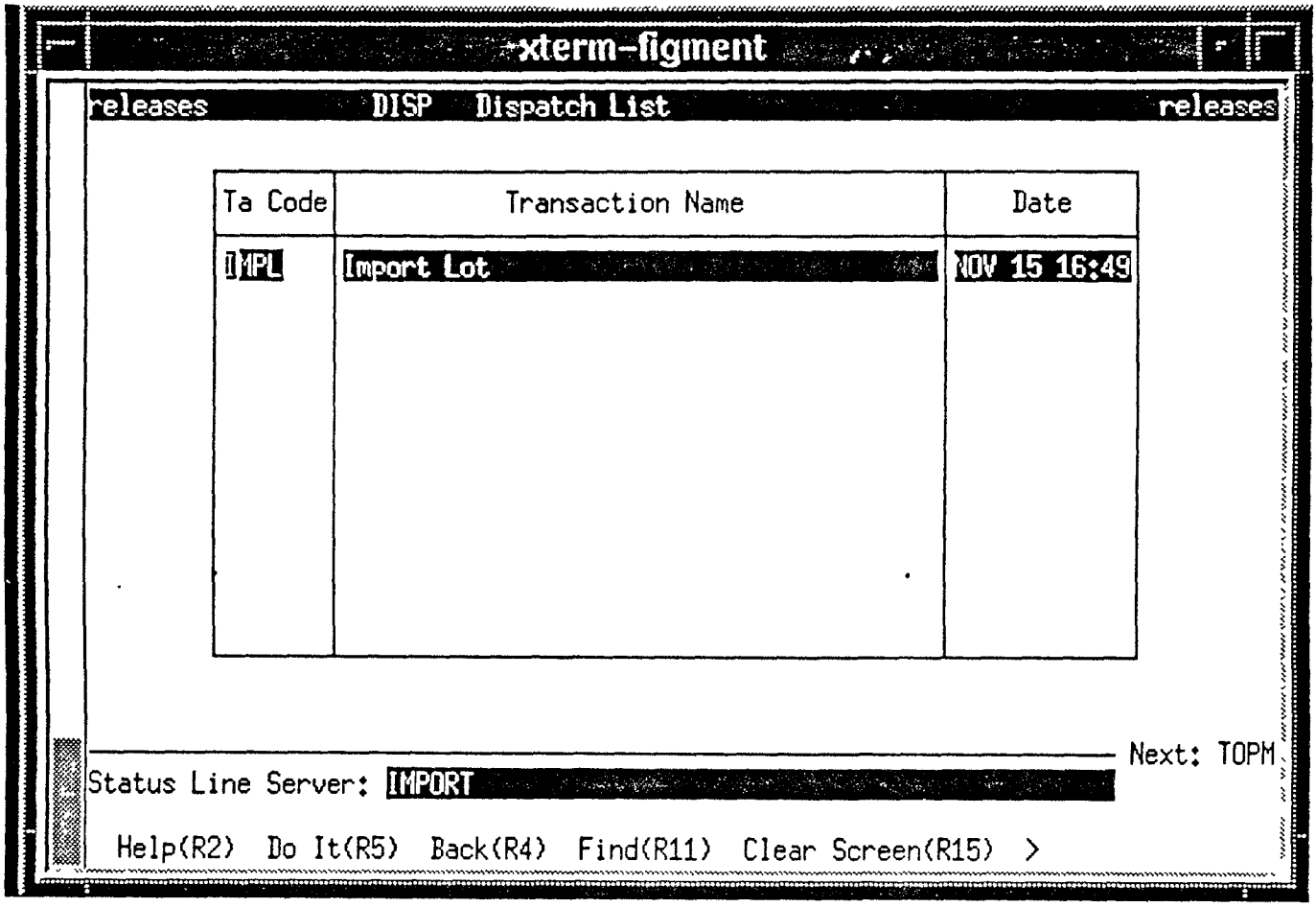
10.62 Figure 6.2



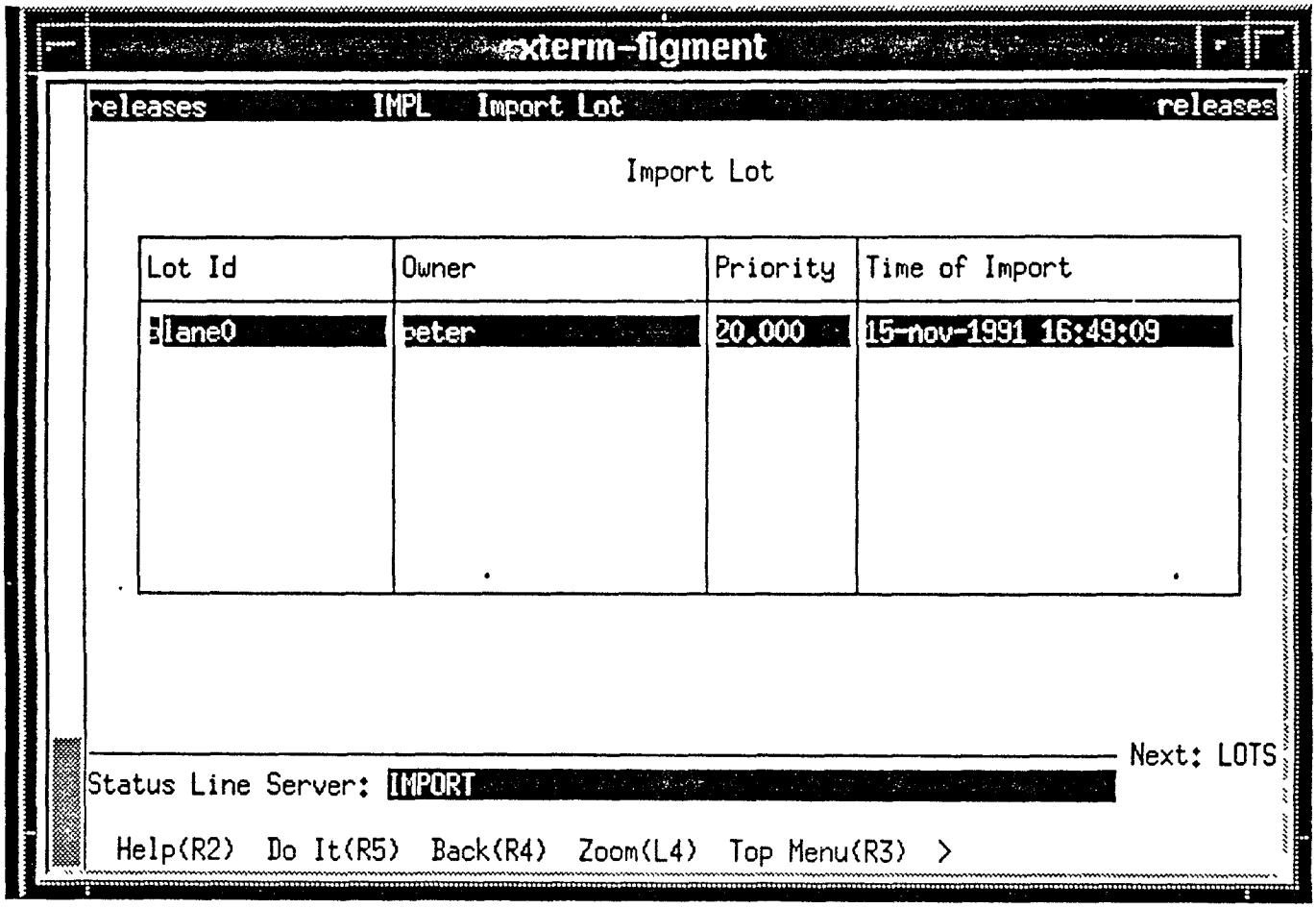
10.63 Figure 6.3



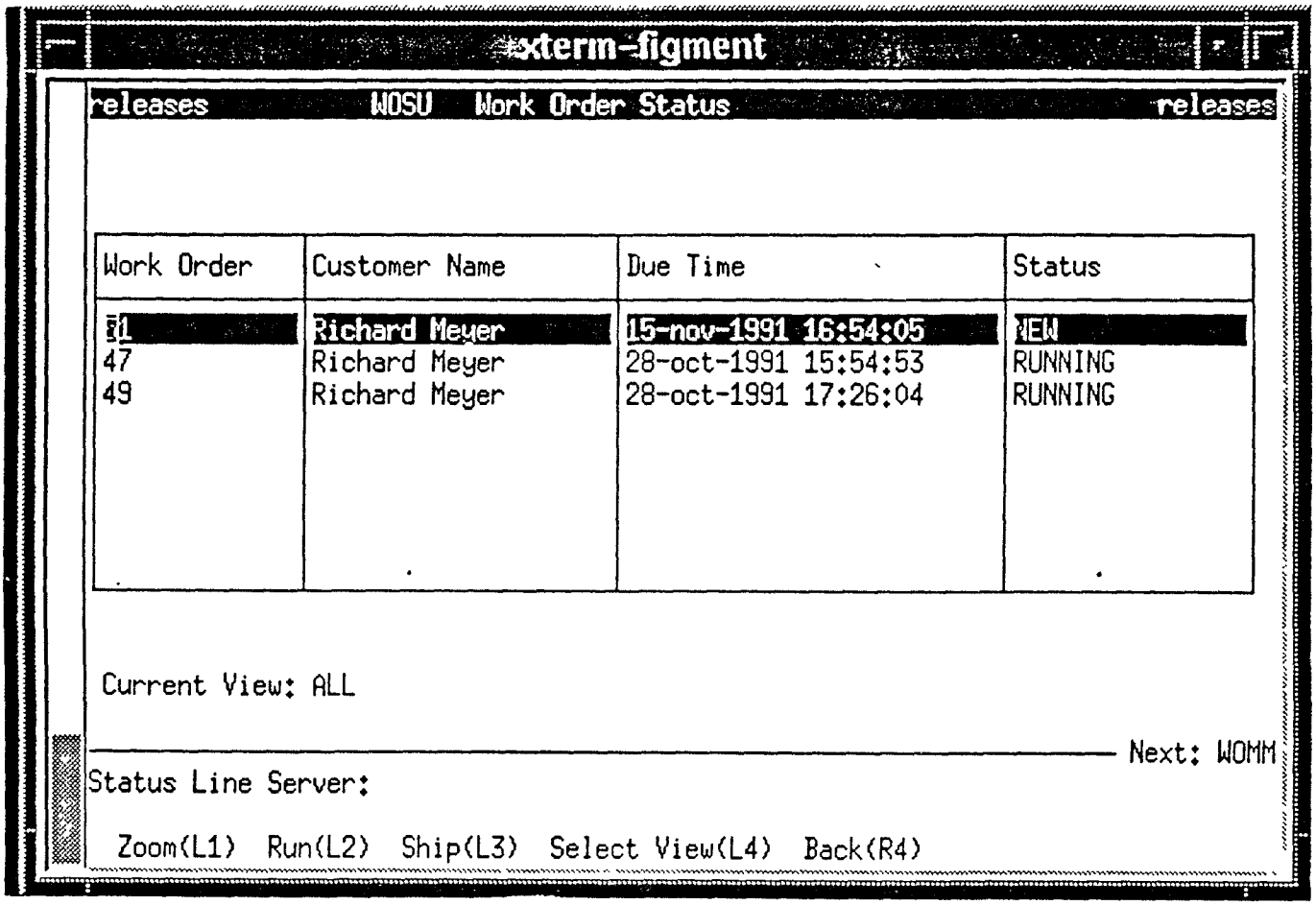
10.64 Figure 6.4



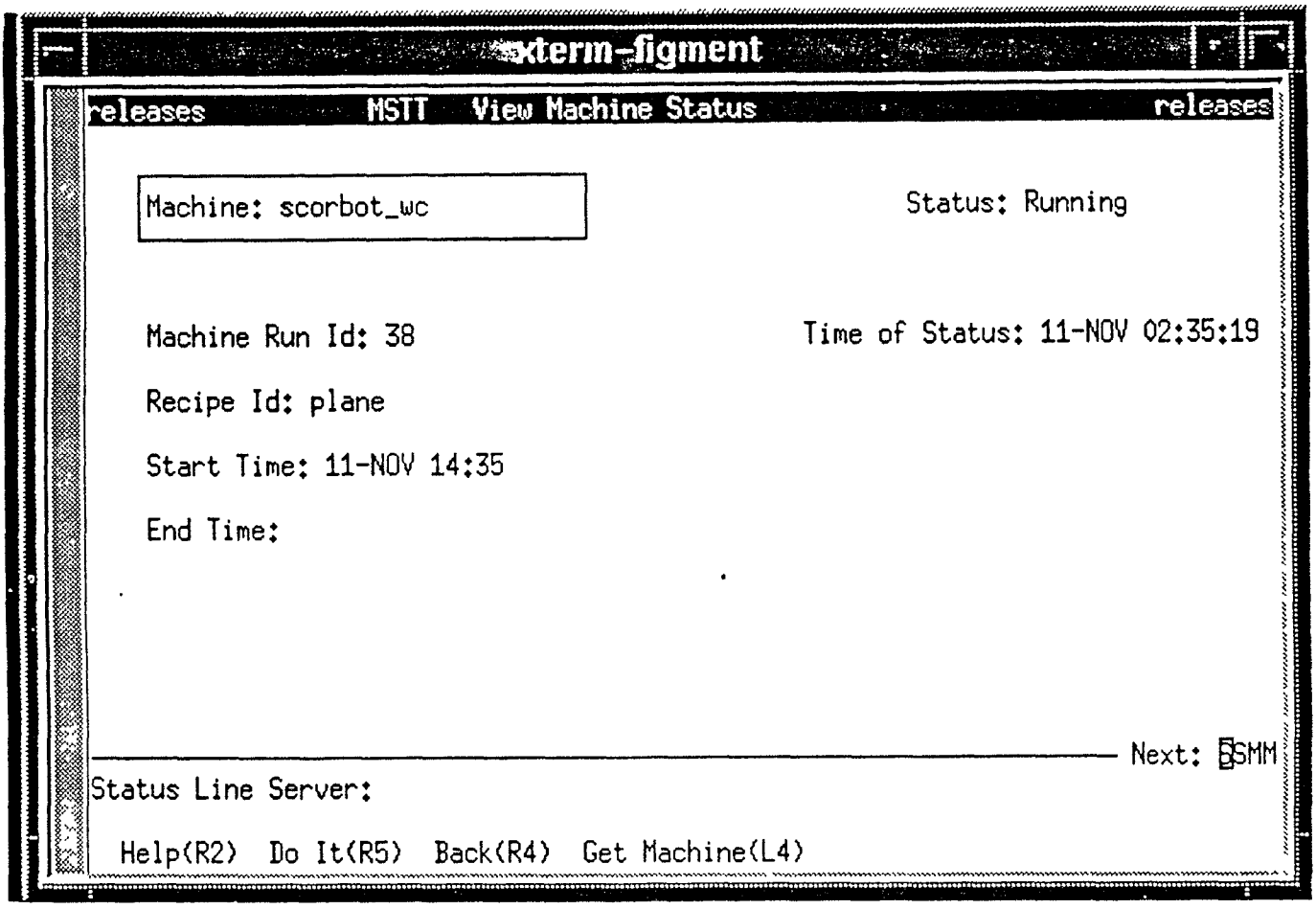
10.65 Figure 6.5



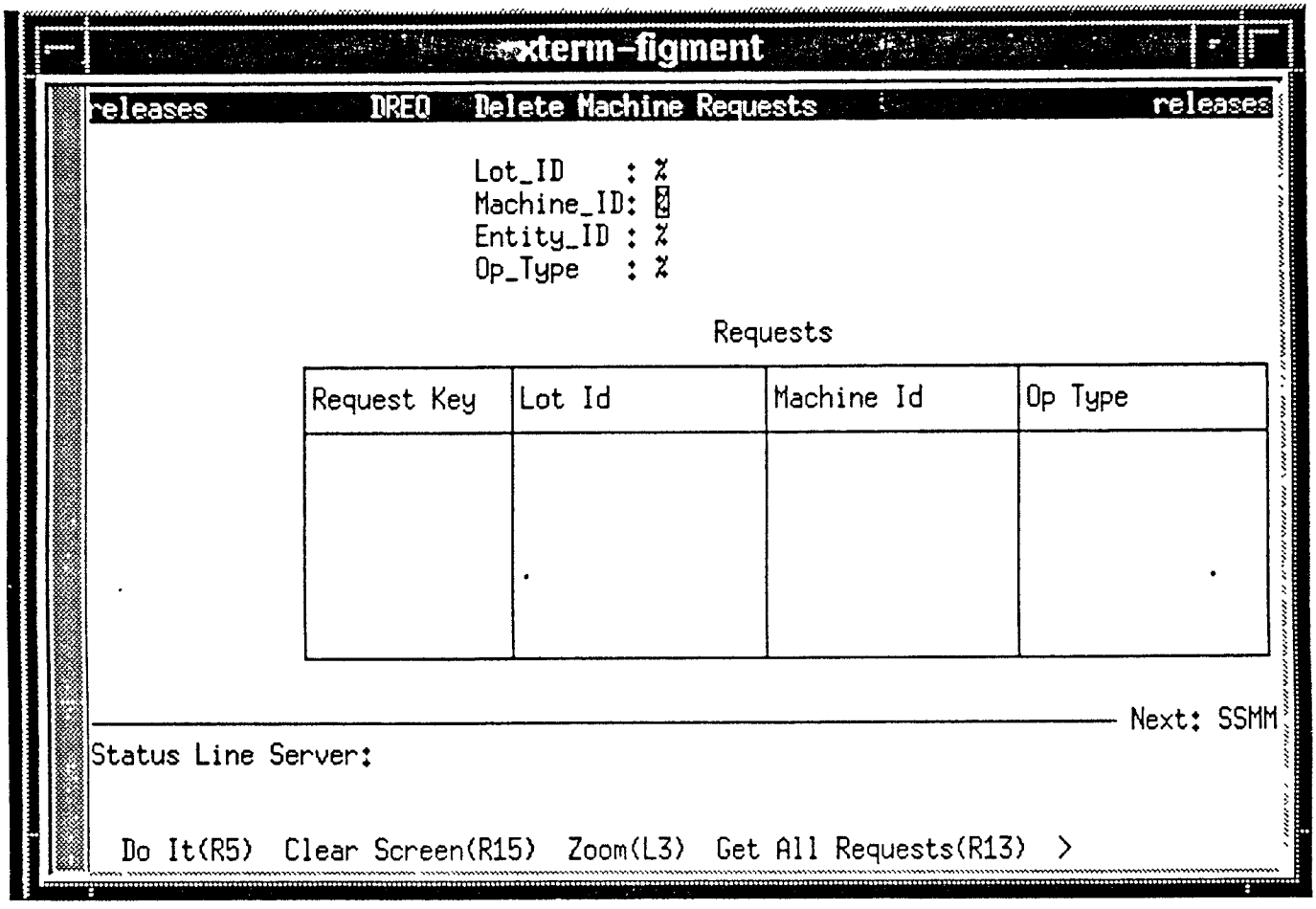
10.66 Figure 6.6



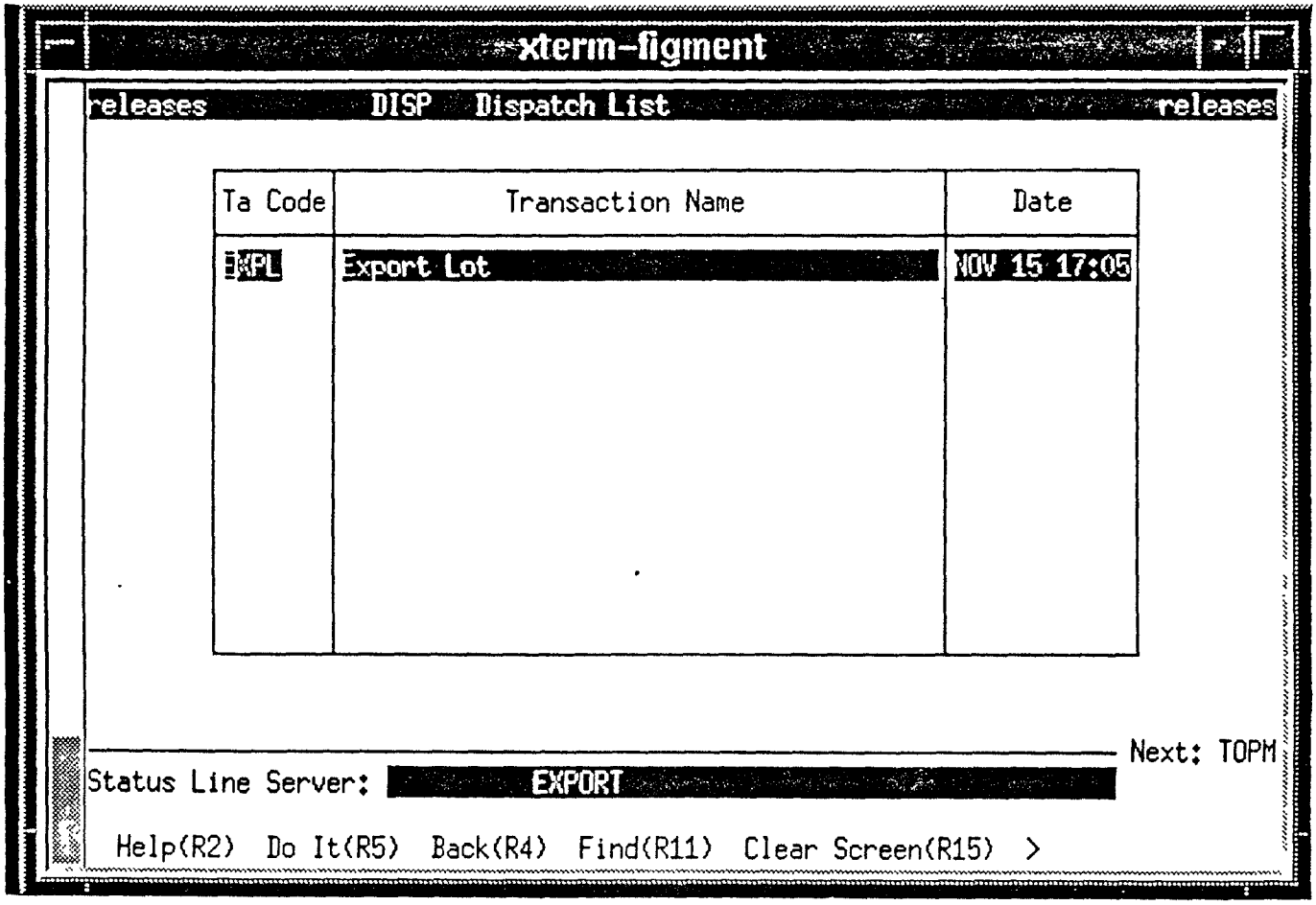
10.67 Figure 6.7



10.68 Figure 6.8



10.69 Figure 6.9



11.0 Listing 1, Operation Entity/Lot Script

```
#include <stdio.h>
#include <strings.h>
#include <osfcn.h>
#include "ack_codes.h"
#include "logs.h"
 exec sql include sqlca;'
#include "esql.h"                          /* contains function prototypes  */
                                           /* for ingres functions */
include(handlers.inc)                      /* M4 macro's */

extern int ingres_error(char*,char*,int);
extern int isis_sign_on();
extern int startcell();
extern int stopcell();
extern int ers_download(char*,char*);
extern int ers_upload(char*,char*);
extern int setup_run(char*,char*,int,char*,char*);
extern int start_run(char*,int,char*,char*);
extern int run_complete(int);
extern int create_lot_entity(char*,char*,int,float,char*,char*,char*,char*,char*
extern int wait_for_import(char*);
extern int lot_completed(char*);
extern int wait_for_export(char*);
extern int wait_for_machine(char*,int&,int&,int&,char*[][17]);
extern int send_request(char*,char*,int&,int&,char*[][17]);
extern int unreserve_machine(char*);

static char send_array[0][17] = { "cmm_wc" };

main (int argc, char **argv)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return break;

  if (argc < 2)
  {
     puts("Number of prameters passed was incorrect");
     exit(0);
  }
/*
   prams passed in order....
   lot_id      - char 17,
*/
  char lot_id[17];
  strcpy(lot_id, argv[1]);
// should be able to get this from lot_it, but not implemented
// this way yet
  char op_type[13];
  strcpy(op_type,"ALUM_CHESS");

char machine_list[10][17];      // list of machines to schedule
strcpy(machine_list[0],"cmm_wc");
int machine_run_id = 26551;     // one for all machines on list, bug....
int num_machine = 1;            // number of machines to sched
char recipe_name[10][20];       // Name of the recipes to run
strcpy(recipe_name[0],"gwc1");
char file_name[10][81];         // of recipes on disk, w/full path..
strcpy(file_name[0],"gwc1");
```

```
char lot_type[17];              // but really used yet...
strcpy(lot_type,"CMM_LOT");
float priority = 20.0;          // not really used yet...
char param1[17];                // Result parms...
strcpy(param1,"RCP_ACTIONS");
char param2[17];                // what a running of this script should
strcpy(param2,"RCP_STATUS");
char param3[17];                // return.
strcpy(param3,"OP_TYPES");
char script_id[17];             // Id for this script non-changing
strcpy(script_id,"Syd_Barrett");
char entity_id[17];             // Created by the log server.
int request_key;                // used for msched

int qty = 1;                    // number of entyities to create.
char owner[17];                 // owner of the entity
strcpy(owner,"B. B. King");
int count = 0;
exec sql begin declare section;
   char *database_name = DATABASE_NAME;
exec sql end declare section;

dbconnect:
exec sql connect :database_name;
check_and_recover_to("main",dbconnect,
   "An ingres error occurred during connect",DONT_CHECK)


int req_num_machine = num_machine;
  if (0 != isis_sign_on())
  {
    printf("isis sign_on failed\n");
    exit(0);
  }
  printf("create lot entity lot_id = %s...\n",lot_id);
  if (0 != create_lot_entity(lot_type,lot_id,qty,priority,owner,
                             param1,param2,param3,script_id,entity_id))
  {
    printf("create lot entity failed.\n ");
    exit(0);
  }
  printf("lot entity_id %s created\n",entity_id);
  printf("wait for lot to be imported...\n");
  if (0 != wait_for_import(entity_id))
  {
    printf("wait for import failed. \n"),
    exit(0);
  }

  printf("Request machines...\n");
  printf("entity_id: %s, op_type: %s, request_key: %d, req_num_machine: %d\n",
         entity_id, op_type, request_key, req_num_machine);
  if (0 != send_request(entity_id,
                        op_type,
                        request_key,
                        req_num_machine,
                        (char *[][17])&send_array[0][0]))
  {
    printf("send_request failed.\n");
    exit(0);
```

```
}
printf("got request_key %d, wait for machine...\n",request_key);
printf("entity_id: %s, op_type: %s, request_key: %d, req_num_machine: %d machi
        entity_id, op_type, request_key, req_num_machine, machine_list[0]);
if (0 != wait_for_machine(entity_id,
                          machine_run_id,
                          request_key,
                          req_num_machine,
                          (char *[][17])&machine_list[0][0]))
{
  printf("wait_for_machine failed.\n");
  exit(0);
}
if (num_machine < req_num_machine )
{
  printf("Not enough machines to satisfy request. bye\n");
  exit(0);
}
printf("got machine_run_id = %d request_key %d for %d machines satisfied.\n",
       machine_run_id,request_key, num_machine);

int list = num_machine-1;
for (;list >= 0;list--)
{
  printf("setup run...\n");
  if (0 != setup_run((char *)&machine_list[list][0],
                     op_type,machine_run_id,
                     (char *)&recipe_name[list][0],
                     (char *)&file_name[list][0]))
  {
    printf("setup run failed. machine_run_id = %d\n",machine_run_id);
    exit(0);
  }
  printf("start run on all machine...\n");
  if (0 != start_run((char *)&machine_list[list][0],
                      machine_run_id,
                     (char *)&recipe_name[list][0],
                     (char *)&file_name[list][0]))
  {
    printf("start run failed. machine_run_id = %d\n",machine_run_id);
    exit(0);
  }
}
printf("wait for run complete...%d\n",machine_run_id);
if (0 != run_complete(machine_run_id))
{
  printf("run complete failed. machine_run_id = %d\n",machine_run_id);
  exit(0);
}

int un_res_count = 0;
while (un_res_count < num_machine)
{
  sleep(10);
  printf("unreserve machine %s\n",(char*)&machine_list[un_res_count][0]);
  if (0 != unreserve_machine((char*)&machine_list[un_res_count][0]))
  {
    printf("unreserve machine failed.\n");
    exit(0);
  }
```

```
    un_res_count++;
  }

  printf("lot completed...\n");
  if (0 != lot_completed(entity_id))
  {
    printf("lot_completed failed. count = %d\n",count);
    exit(0);
  }
  printf("wait for lot exported...\n");
  if (0 != wait_for_export(entity_id))
  {
    printf("wait for export failed. \n");
    exit(0);
  }
}
```

11.1 Listing 2, MSCHED

```
/*
 *         MSCHED.M4
 *
 */
#include <stdio.h>
#include <strstream.h>
#include "Conversation.h"
#include "cell_messages.h"
#include "nullitems.h"
#include "ack_codes.h"
#include "sls.h"
#include "logs.h"
exec sql include sqlca;'
#include "esql.h"
include(handlers.inc)
/*
 * The following code(class) is for storing the list of machines to be
 * scheduled.  It is used internally to MSCHED only....
 * it was writtent by Rich A. Taft and slightly modified by
 * Peter A. Murray.
 */

#include <iostream.h>
#include "list.h"
#include <stddef.h>

const int FFALSE = 0;
const int TTRUE = !FFALSE;

charelt :: charelt (const char * n)
{
  strcpy(value,n);
  _next = (charelt *)NULL;
}
charelt :: ~charelt ()
{
  if (_next != NULL)
    delete _next;
}
charelt*
charelt :: next()
{
  return _next;
}

const char *
charelt :: val()
{
  return value;
}

charlist :: charlist () : _head((charelt*)NULL), _tail((charelt*)NULL)
{
  ;
}

charlist :: ~charlist ()
{
  charelt *p = _head;
  while (p != NULL) {
```

```
    p = p -> _next;
    _head -> _next = (charelt *) NULL;
    delete _head;
    _head = p;
  }
}

charelt *
charlist :: head() { return _head; }

charelt *
charlist :: tail() { return _tail; }

void
charlist :: add (const char * n)
{
  if (_head == NULL) {          // empty list
    _head = new charelt (n);
    _tail = _head;
    _tail -> _next = (charelt *)NULL;   // just to be safe
    return;
  }

  else if (_head == _tail) {    // single elt
    if (_head -> val () == n)
      return;
    else if (_head -> val() < n) {
      _head -> _next = new charelt (n); // insert after existing val
      _tail = _head -> _next;
      _tail -> _next = (charelt *)NULL; // just to be safe
      return;
    }
    else {                      // head > n
      _head = new charelt (n);  // insert before existing elt
      _head -> _next = _tail;
      return;
    }
  }

  else {                        // find where to put n
    charelt *p = _head;
    charelt *f = (charelt *)NULL;               // f follows p
    do {
      if (p -> val () == n)
        return;
      else if (p -> val () > n) {
        charelt *nxt = p;
        p = new charelt (n);    // insert before existing elt
        p -> _next = nxt;
        if (f == NULL)          // p points to head
          _head = p;
        else
          f -> _next = p;

        return;
      }
      else {                    // head < n
        f = p;
        p = p -> _next;
      }
```

```
    } while (p != NULL);
    // p has reached end without inserting
    _tail -> _next = new charelt (n);
    _tail = _tail -> _next;
    _tail -> _next = (charelt *)NULL;
  }

}

void
charlist :: remove (const char * n)
{
  charelt *head = _head;
  charelt *trail = (charelt *)NULL;
  int found = FFALSE;

  while(!found && head != NULL) {
    if (strcmp(n, head->value) == 0) {
      found = TTRUE;
    } else {
      trail = head;
      head = head->_next;
    }
  }
  if(found) {
    if(head == _head) {
      _head = _head->_next;
      head->_next = (charelt *)NULL;
      delete head;
    } else {
      if(head == _tail) {
        _tail = trail;
        _tail->_next = (charelt *)NULL;
        head->_next = (charelt *)NULL;
        delete head;
      } else {
        trail->_next = head->_next;
        head->_next = (charelt *)NULL;
        delete head;
      }
    }
    if(_head == NULL) _tail = (charelt *)NULL;
  }
}


int
charlist :: operator == (charlist& il)
{
  for (charelt *p1 = _head, *p2 = il._head;
       p1 != NULL && p2 != NULL && *p1 == *p2;
       p1 = p1 -> _next, p2 = p2 -> _next) ;

  if (p1 == NULL && p2 == NULL)
    return TTRUE;
  else
    return FFALSE;
}

ostream& operator << (ostream& os, const charelt& i)
```

```
{
  os << i.value;
  return os;
}

ostream& operator << (ostream& os , const charlist& i)
{
  charelt *p = i._head;

  while(p != NULL) {
    os << *p;
    if (p -> next() != NULL)
      os << " ";
    p = p -> next ();
  }
  os << flush;
  return os;
}

/* End of the class stuff....  */

#define IS_NULL      -1       /* used by inquire_ingres */
#define TRUE          1
#define FALSE         0
#define OK            0
#define NOT_OK       -1
#undef clean_return

string12 module_name = "MSCHED";
string12 busname = "CELL";

extern "C" IISQLCA sqlca;
exec sql whenever sqlerror continue;
exec sql whenever sqlwarning continue;
exec sql whenever sqlmessage continue;

static char* startup[] = { "STARTUP", (char*) NULL};
static char* clist[] = { "SHUTDOWN", "MACH_REQ", "CANCEL_RE",
            "UPD_ME_ST", "ACC_TYP_C", "UNRESERVE", (char*) NULL };

extern void error_lookup(int,char*);
extern int ingres_error(char*,char*,int);
extern void LOGS(char*, char*, char* = "%s", LOG_MSG_TYPE = note);
extern void kill_mq_sm(int = 0);

void msched_startup(Conversation *C,Startup_Request& M);
void msched_shutdown(Conversation *C,Shutdown_Request& M);
int msched_shutdown_scope();
void machine_request(Conversation *C,Machine_Request& M);
int request_machine_scope(int,char*,char*,int&);
void sched_algorithm();
void update_me_status(Conversation *C,Update_ME_Status& M);
int update_me_status_scope(char*,int&,char*);
void cancel_request(Conversation *C,Cancel_Request& M);
int cancel_request_scope(int,char*);
void unreserve_machine(Conversation *C,Unreserve_Me& M);

int err = OK;
int ack;
charlist me_list;
```

```
main()
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return break;

ifdef(`on_error',`undefine(`on_error')')
define(on_error,`kill_mq_sm();')

ifdef(`on_badrows',`undefine(`on_badrows')')
define(on_badrows,`kill_mq_sm();')

exec sql begin declare section;
   char database_name[64];
exec sql end declare section;
strcpy(database_name, DATABASE_NAME);
sprintf(line,"Starting %s",module_name);
LOGS("main",line);

dbconnect:
exec sql connect :database_name;
check_and_recover_to("main",dbconnect,
   "An ingres error occurred during connect",DONT_CHECK)

err = sign_on(module_name, conv_timeout, TRUE);
if (err != OK)
  {
  sprintf(line,"ISIS returned error #%d while %s was tring to sign on",
               err,module_name);
  LOGS("main",line,"%s",error);
  kill_mq_sm();
  }

err = interest(busname,startup);
if (err != OK)
  {
  sprintf(line,"ISIS returned error #%d while setting interest filters for %s",
               err,module_name);
  LOGS("main",line,"%s",error);
  sign_off();
  kill_mq_sm();
  }

sprintf(line,"Signed on to %s bus and waiting for startup",busname);
LOGS("main",line);

int bad_message = FALSE;
while (TRUE)
  {
  bad_message = FALSE;

  Message msg;
  Conversation *C;
  C = new Conversation;
  C->set_group(busname);

  first_message("main",C,msg)
```

```
switch( C->iclass() )
  {
  case C_STARTUP:
    if (msg.msg_class_id() == M_STARTUP_REQUEST)
    {
       Startup_Request sur(msg);
       msched_startup(C,sur);
    }
    else
       bad_message = TRUE;
    break;
  case C_SHUTDOWN:
    if (msg.msg_class_id() == M_SHUTDOWN_REQUEST)
    {
       Shutdown_Request sdr(msg);
       msched_shutdown(C,sdr);
    }
    else
       bad_message = TRUE;
    break;
  case C_MACH_REQ:
    if (msg.msg_class_id() == M_MACHINE_REQUEST)
    {
       Machine_Request mrr(msg);
       machine_request(C,mrr);
    }
    else
       bad_message = TRUE;
    break;
  case C_CANCEL_RE:
    if (msg.msg_class_id() == M_CANCEL_REQUEST)
    {
       Cancel_Request creq(msg);
       cancel_request(C,creq);
    }
    else
       bad_message = TRUE,
    break;
  case C_UPD_ME_ST:
    if (msg.msg_class_id() == M_UPDATE_ME_STATUS)
    {
       Update_ME_Status ums(msg);
       update_me_status(C,ums);
    }
    else
       bad_message = TRUE;
    break;
  case C_UNRESERVE:
    if (msg.msg_class_id() == M_UNRESERVE_ME)
    {
       Unreserve_Me unresme(msg);
       unreserve_machine(C,unresme);
    }
    else
       bad_message = TRUE;
    break;
  case C_ACC_TYP_C:
    if (msg.msg_class_id() == M_CHG_ACCEPT_TYPE)
      {
      C->ignore();
```

```
          { /* C++ is strange. I need to make this a block */
          Chg_Accept_Type cat(msg);
          /*
             Check if the sched algorithm must be run.  If
             no accept types are being added, then do not run it
          */
          if ((int)cat.num_messages().ifnull(INTNULL) > 0)
            sched_algorithm();
          }
        }
      else
        bad_message = TRUE;
      break;
    default:
      C->sprint_conversation(line);
      strcat(line,"\nUnexpected Conversation received");
      LOGS("main",line,"%s",warning);
      C->ignore();
      break;
    } /* end of switch */

  if (bad_message == TRUE)
    {
    sprintf(line,"Unexpected message received '%s' for conversation: '%s'.",
            msg.msg_name(),C->cclass() );
    LOGS("main",line,"%s",warning);
    C->ignore();
    }
  exec sql commit; /* make sure that the DB is not locked */
  delete C;
  } /* end of while */
} /* end of main */


/*****************************************************************************
 *
 *                  MSCHED_STARTUP
 *
 *****************************************************************************/
void msched_startup(Conversation *C,Startup_Request& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   return; \
   }

ifdef(`on_error',undefine(`on_error'))
define(on_error,
`Startup_Ack startack1(module_name,int_with_null(DB_Badupd));
 err = C->send(startack1);
 if (err != OK)
 {
   sprintf(line,"Error #%d sending Startup_Ack",err);
   LOGS("msched_startup",line,"%s",error);
 }')

if (strcmp((char *)M.module_id().ifnull(CHARNULL),module_name) == 0)
{
```

```
  LOGS("msched_startup","MSCHED startup received");

  /* make sure REQUEST_MACHINES and RESERVED_MACHINES tables are empty */
  saveto(sp_wipe_clean)
  exec sql delete from REQUEST_MACHINES;
  check_and_recover_to("msched_startup",sp_wipe_clean,
  "An ingres error occured while deleting REQUEST_MACHINES",DONT_CHECK)

  exec sql delete from RESERVED_MACHINES;
  check_and_recover_to("msched_startup",sp_wipe_clean,
  "An ingres error occured while deleting RESERVED_MACHINES",DONT_CHECK)
  exec sql commit;

  Startup_Ack startack(module_name,int_with_null(ACK_OK));
  err = C->send(startack);
  if (err != OK)
  {
    sprintf(line,"Error #%d sending Startup_Ack",err);
    LOGS("msched_startup",line,"%s",error);
    clean_return
  }
  err = no_interest(busname);
  if (err != OK)
  {
    sprintf(line,
    "Error #%d while setting conversation filter to no_intrest",err);
    LOGS("msched_startup",line,"%s",error);
    ack = Start_Fai;
  }
  else
  {
    err = interest(busname,clist);
    if (err != OK)
    {
      sprintf(line,"Error #%d while setting conversation filter",err);
      LOGS("msched_startup",line,"%s",error);
      ack = Start_Fai;
    }
  }
  Conversation *comp,COMP;
  comp = &COMP;
  comp->set_group(busname);

  Startup_Complete startcomp(module_name,ack);
  err = comp->send(startcomp);
  if (err != OK)
  {
    sprintf(line,"Error #%d while sending Startup_Complete",err);
    LOGS("msched_startup",line,"%s",error);
  }
}
else
{
  /* the startup is NOT for MSCHED */
  C->ignore();
}
clean_return
} /* end of msched_startup */
```

```
/*****************************************************************
 *
 *                    MSCHED_SHUTDOWN
 *
 * Change receive filters to not accept any conversation.
 * Complete actions for all pending Cell conversations.
 * Finally send a Shutdown Complete conversation and wait for a STARTUP.
 *
 *****************************************************************/
void msched_shutdown (Conversation *C, Shutdown_Request& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   return; \
   }

ack = ACK_OK;

if (strcmp((char *)M.module_id().ifnull(CHARNULL),module_name) == 0)
{
  LOGS("msched_shutdown","MSCHED shutdown received");

  Shutdown_Ack shutack(module_name,int_with_null(ACK_OK));
  err = C->send(shutack);
  if (err != OK)
  {
    sprintf(line,"Error #%d sending Shutdown_Ack",err);
    LOGS("msched_shutdown",line,"%s",error);
    clean_return
  }

  /* no longer interested in "all" conversations */
  err = no_interest(busname);
  if (err != OK)
  {
    ack = Shut_Fail;
    sprintf(line,
    "Error #%d while setting conversation filter to no_interest",err);
    LOGS("msched_shutdown",line,"%s",error);
  }
  /* interested in only the startup conversation */
  err = interest(busname,startup);
  if (err != OK)
  {
    ack = Shut_Fail;
    sprintf(line,"Error #%d while setting conversation filter",err);
    LOGS("msched_shutdown",line,"%s",error);
  }

  ack = msched_shutdown_scope();

  Conversation *comp, COMP;
  comp = &COMP;
  comp->set_group(busname);

  Shutdown_Complet shutcomp(module_name,ack);
  err = comp->send(shutcomp);
  if (err != OK)
```

```
  {
    sprintf(line,"Error #%d while sending Shutdown_Complet",err);
    LOGS("msched_shutdown",line,"%s",error);
  }
}
else
{
  /* the Shutdown is NOT for MSCHED */
  C->ignore();
}
clean_return
} /* end of msched_shutdown */

/*****************************************************************
 *
 *                  MSCHED_SHUTDOWN_SCOPE
 *
 * For each distinct request key in request machine send machine
 * request complete saying it cannot be satisfied
 *
 *****************************************************************/
int msched_shutdown_scope()
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   return(ack); \
   }

ifdef(`on_error',undefine(`on_error'))
define(on_error,
`ack = DB_Badrea;')

ifdef(`on_badrows',undefine(`on_badrows'))
define(on_badrows,
`ack = ACK_OK;')

exec sql begin declare section;
   char entity_id[17];
   int request_key;
   char errortext[512];
exec sql end declare section;


  while(TRUE)
  {
    saveto(sp_unsatisfied)
    {
      /*
        Get a request_key from REQUEST_MACHINES. It does not
        matter which one.  When there are none left, then return.
      */
      exec sql select distinct request_key, entity_id
          into :request_key, :entity_id
          from REQUEST_MACHINES
          where request_key = (select min(request_key)
                               from REQUEST_MACHINES);
      check_and_recover_to("msched_shutdown_scope",sp_unsatisfied,
      "An ingres error occured while reading REQUEST_MACHINES",
```

```
    CHECK_ONEROW_NOLOG)
    /*
      Remove all occurences of request_key from REQUEST_MACHINES
    */
    exec sql delete
         from REQUEST_MACHINES
         where request_key = :request_key;
    check_and_recover_to("msched_shutdown_scope",sp_unsatisfied,
    "An ingres error occured deleting from REQUEST_MACHINES",DONT_CHECK)
    exec sql commit;

    /* These will be initialized to null */
    int_with_null machine_run_id;
    Conversation CONV, *conv;
    conv = &CONV;
    conv->set_group(busname);
    Message req_msg;

    Machine_Req_Comp mrcnomach(NO_MACHIN,entity_id,machine_run_id,
                               request_key,0);
    err = conv->send(mrcnomach);
    if (err != OK)
    {
      sprintf(line,"Error #%d sending Machine_Req_Comp",err);
      LOGS("msched_shutdown_scope",line,"%s",error);
    }
    /* wait for ack */
    next_message("msched_shutdown_scope",conv,req_msg)

    /* make sure its the right ack message */
    if (req_msg.msg_class_id() == M_REQUEST_COMP_ACK)
    {
      Request_Comp_Ack reqack(req_msg);
      int ack_code = (int)reqack.ack_code().ifnull(INTNULL);
      if (ack_code != ACK_OK)
      {
        conv->end();
        error_lookup(ack_code,errortext);
        sprintf(line,"Entity returned ack_code #%d.\n%s",ack_code,errortext);
        LOGS("msched_shutdown_scope",line,"%s",warning);
        clean_return
      }
      conv->end();
    }
    else
    {
      conv->ignore();
      sprintf(line,"Unexpected message received: %s.\nThe conversation is: %s.
              req_msg.msg_class_id(),conv->cclass());
      LOGS("msched_shutdown_scope",line,"%s",warning),
      clean_return
    }
   } /* scope of lable...may work with out this...? */
  } /* end of while */
} /* end of msched_shutdown_scope */


/*****************************************************************
        Name: machine_request
```

```
    Function: request a machine for msched.

      Inputs: Conversation *, Machine_req&

     Returns:       None.

    Modifies:       me_list

    Dependancies: charelt (object)

    Language: C++

    Copyright: Siemens Corporate Research, INC 1991
           All rights reserved.

    By:
        Peter A. Murray

    Date:
        2/02/1991

*************************************************************************/
void machine_request(Conversation *C,Machine_Request &M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      delete o_type;   \
      delete e_id;     \
      delete m_id;     \
      return;          \
      }

Message temp_msg;
charelt *pt;
int  counter = 0; /* keep track of the number of machine_ids recieved */
int  request_key;
int  n_machine = (int)M.num_machine().ifnull(INTNULL);
char *o_type = (char*)M.op_type().ifnull(CHARNULL);
char *e_id   = (char*)M.entity_id().ifnull(CHARNULL);
char *m_id = NULL;

LOGS("machine_request","received machine_request");

pt = me_list.head();
while(pt != NULL)
{
      me_list.remove(pt->val());
      pt = pt->next();
}
/*
   Multiple second messages provide the machine_id(s).
   How many second messages == num_messages in first message.
   Now, get messages.
*/
while (counter < n_machine)
{

ifdef(`on_badreceive',undefine(`on_badreceive'))
```

```
define(on_badreceive,
`Machine_Req_Ack mreqa(request_key,int_with_null(Bad_messa));
 err = C->send(mreqa);
 if (err != OK)
 {
   sprintf(line,"Error sending Machine_Req_Ack err = %d",err);
   LOGS("machine_request",line,"%s",error);
   clean_return
 }')

  next_message("machine_request",C,temp_msg)

ifdef(`on_badreceive',undefine(`on_badreceive'))
define(on_badreceive,`');

  /* make sure its the right message */
  if (temp_msg.msg_class_id() == M_MACHINE_ENTRY)
  {
    /* store machine_id in the link list */
    Machine_Entry ME1 (temp_msg); /* cast msg */
    m_id = (char *)ME1.machine_id().ifnull(CHARNULL);
    me_list.add(m_id);
    delete m_id;
    /* set m_id = NULL because if the next message is bad,
       clean_return which will try to delete m_id.
       The non-NULL ptr which will core dump.
    */
    m_id = NULL;
    counter++; /* increase counter -- received next machine_id */
  }
  else
  {
    /* not a machine_entry msg,ignore it */
    sprintf(line,"Unexpected message received: %s",temp_msg.msg_class_id());
    LOGS("machine_request",line,"%s",warning);
    C->ignore();
  } /* end of else */
}/* end of while */

ack = request_machine_scope(n_machine,o_type,e_id,request_key);

Machine_Req_Ack mreqack(request_key,int_with_null(ack));
err = C->send(mreqack);
if (err != OK)
{
  sprintf(line,"Error sending Machine_Req_Ack err = %d",err);
  LOGS("machine_request",line,"%s",error);
  clean_return
}
if (ack == ACK_OK)
  sched_algorithm();
clean_return
} /* end of machine_request */

/*****************************************************************
 *
 *                    REQUEST_MACHINE_SCOPE
 *
 *****************************************************************/
int request_machine_scope(int num_machine,char *op_t,
```

```
                        char *e_id,int &r_key)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      r_key = request_key; \
      return(ack); \
      }

int counter = 0;
charelt *pt;
exec sql begin declare section;
      char machine_id[17];
      char *op_type = op_t;
      char *entity_id = e_id;
      char me_status[17];
      int row_count = 0;
      int request_key = 0;
exec sql end declare section;

saveto(sp_check)
{
ifdef(`on_error',`undefine(`on_error')')
define(on_error,`ack = DB_Badrea;')

ifdef(`on_badrows',`undefine(`on_badrows')')
define(on_badrows,`')

  if (num_machine < 1)
  {
    sprintf(line,"No machines were scheduled");
    LOGS("request_machine_scope",line,"%s",warning);
    ack = NO_MACHIN;
    clean_return
  }
  /*
    Now lets check if the machines are NOT shutdown.
    If one is, take it out of the link list.
  */
  pt = me_list.head();
  while (pt != NULL)
  {
    strcpy(machine_id,pt->val());
    sprintf(line,"Check if machine %s exists and is not shutdown",machine_id);
    LOGS("request_machine_scope",line);
    /*
      Lets get a count of how many there are....
    */
    exec sql select count(*)
         into :row_count
         from ME_CONFIG
         where machine_id = :machine_id;
    check_and_recover_to("request_machine_scope",sp_check,
    "An ingres error occured reading ME_CONFIG",DONT_CHECK)
    if (row_count != 1)
    { /* The machine does not exist  */
      sprintf(line,"Machine %s does not exist.",machine_id),
      LOGS("request_machine_scope",line,"%s",warning);
      ack = NO_MACHIN;
```

```
    clean_return
  }
  exec sql select count(*)
       into :row_count
       from MACHINE_STATUS
       where machine_id = :machine_id and
             me_status = 'Shutdown';
  check_and_recover_to("request_machine_scope",sp_check,
  "An ingres error occured reading MACHINE_STATUS",DONT_CHECK)
  if (row_count == 1)
  { /* The machine is shutdown  */
    sprintf(line,
     "Machine %s is shutdown. Cannot satisfy request",machine_id);
    LOGS("request_machine_scope",line,"%s",warning);
    ack = NO_MACHIN;
    clean_return
  }
  pt = pt->next();
} /* end of while */

/* generate the new request_key number */
exec sql execute procedure get_new_max_value (type = 'MACH_R_KEY')
     into :request_key;
if (1 > request_key)
{ /* an error occurred */
  LOGS("request_machine_scope",
       "An error occured when getting the request_key_number",
       "%s",error);
  ack = DB_Badupd;
  exec sql rollback;
  clean_return
} /* end of if */

/*
   We have now found all the machines and checked that they are running
   now add them to the REQUEST_MACHINES table.
*/
pt = me_list.head();  // go to the begining of the list..
//
// The machines in the list are all valid requests so lets put
// them into the REQUEST_MACHINES table....
//
while (pt != NULL)
{
  strcpy(machine_id,pt->val());
  sprintf(line,"Insert entity_id %s,request_key %d,machine_id %s,op_type %s",
          entity_id, request_key, machine_id, op_type);
  LOGS("request_machine_scope",line);
  exec sql insert
       into REQUEST_MACHINES
            (entity_id, request_key, machine_id, op_type)
       values (:entity_id, :request_key, :machine_id, :op_type);
  check_and_recover_to("request_machine_scope",sp_check,
  "An ingres error while inserting into REQUEST_MACHINES",DONT_CHECK)
  pt = pt->next();
} /* end of while */
exec sql commit;
ack = ACK_OK;
clean_return
}
```

```
} /* end of request_machine_scope */


/*****************************************************************************
      Name: sched_algorithm

   Function: To service the oldest request FIFO for a machine first.

     Inputs:        None.

    Returns:        None.

   Modifies:        request_machines

   Dependancies:    None.

   Language: C++/esql

   By:
      Peter A. Murray

   Date:
      07/07/1991

   Copyrite 1991 Siemens Corporate Reseaarch, INC

*****************************************************************************/
void sched_algorithm()
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   goto bye; \
   }

ifdef(`on_badrows',`undefine(`on_badrows')')
define(on_badrows,`')

ifdef(`on_error',`undefine(`on_error')')
define(on_error,`/* should send an alarm. */')

LOGS("sched_algorithm","run the sched algorithm");

exec sql begin declare section;
   int  request_key;
   char machine_id[17];
   char entity_id[17];
   char op_type[17];
   int  machine_run_id;
   int row_count;
   int num_machines;
   char errortext[512];
exec sql end declare section;

int done = FALSE;

saveto(sp_sched)
{
  exec sql declare cursor1 cursor for
```

```
     select distinct request_key, entity_id, op_type
     from REQUEST_MACHINES
     order by request_key;

exec sql open cursor1;
exec sql whenever not found goto bye;

while (!done)
{
/*
   go through the REQUEST_MACHINES table for each distinct
   request key - starting with the oldest.
*/
   exec sql fetch cursor1 into :request_key, :entity_id, :op_type;
   check_and_recover_to("sched_algorithm",sp_sched,
   "An ingres error occured when cursor1 was reading REQUEST_MACHINES",
   DONT_CHECK)

/*
   count the number of rows of the request key.  this will give the
   number of machines needed to satisfy the request.
*/
   exec sql select count(*)
       into :row_count
       from REQUEST_MACHINES
       where request_key = :request_key;
   check_and_recover_to("sched_algorithm",sp_sched,
   "An ingres error occured counting the rows of a request in REQUEST_MACHINES
   DONT_CHECK)

   sprintf(line,"There are %d machines for request_key %d",row_count,request_k
   LOGS("sched_algorithm",line);
/*
   Check if all of the machines in the request are available.
   The op_type and the machine must be in ACCEPT_TYPES and
   the machine must NOT be currently reserved
*/
   exec sql select count(*)
        into :num_machines
        from REQUEST_MACHINES r, ACCEPT_TYPES a
        where r.machine_id = a.machine_id and
              a.op_type    = :op_type and
              r.request_key = :request_key and
              r.machine_id not in
              (select machine_id
                 from RESERVED_MACHINES);
   check_and_recover_to("sched_algorithm",sp_sched,
   "An ingres error occured while reading REQUEST_MACHINES and ACCEPT_TYPES",
   DONT_CHECK)
   sprintf(line,"There are %d machines available for request_key %d",
           num_machines,request_key);
   LOGS("sched_algorithm",line);
/* if the two counts match, then the request is satisfied */
   if (num_machines == row_count)
      done = TRUE;
}
/* found a satisfied request key */
exec sql close cursor1;
/* reserve the machines */
exec sql insert into RESERVED_MACHINES
```

```
      (machine_id,entity_id)
      select machine_id,entity_id
         from REQUEST_MACHINES
         where request_key = :request_key;
  check_and_recover_to("sched_algorithm",sp_sched,
  "An ingres error while inserting into RESERVED_MACHINES",DONT_CHECK)

  /* generate the new machine_run_id number */
  exec sql execute procedure get_new_max_value (type = 'MACH_R_ID')
               into :machine_run_id;
  if (1 > machine_run_id)
  { /* an error occurred */
    check_and_recover_to("sched_algorithm",sp_sched,
    "An ingres error occured while inserting into MAX_VALUES",DONT_CHECK)
  }
  /*
    read the machine_ids into memory. Use a link list
  */
  charelt *pt;
  int counter = 0;
  pt = me_list.head();
  while(pt != NULL)
  { /* clear link list */
    me_list.remove(pt->val());
    pt = pt->next();
  }
  exec sql declare cursor2 cursor for
    select distinct machine_id
    from REQUEST_MACHINES
    where request_key = :request_key;
  exec sql open cursor2;

  while (counter < num_machines)
  {
    exec sql fetch cursor2 into :machine_id;
    check_and_recover_to("sched_algorithm",sp_sched,
    "An ingres error occured while reading REQUEST_MACHINES",DONT_CHECK)
    me_list.add(machine_id);
    counter++; /* increase counter -- get next machine_id */
  } /* end of while */

  exec sql close cursor2;
  sprintf(line,"Delete all requests for entity_id %s from REQUEST_MACHINES",
               entity_id);
  LOGS("sched_algorithm",line);
  exec sql delete
       from REQUEST_MACHINES
       where entity_id = :entity_id;
  check_and_recover_to("sched_algorithm",sp_sched,
  "An ingres error while deleting from REQUEST_MACHINES",DONT_CHECK)
  exec sql commit;

#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   return; \
   }
  sprintf(line,"satisfied request_key %d for entity %s of op_type %s",
               request_key,entity_id,op_type);
```

```
LOGS("sched_algorithm",line);

/*
  send out a Machine Request Complete conversation to the lot entity
  telling it that the machines are ready
*/
Conversation CONV, *conv;
conv = &CONV;
conv->set_group(busname);
Message req_msg;

Machine_Req_Comp mrcok((int_with_null)ACK_OK,entity_id,
                       machine_run_id,request_key,num_machines);
err = conv->send(mrcok);
if (err != OK)
{
  sprintf(line,"Error sending Machine_Req_Comp err: %d",err);
  LOGS("sched_algorithm",line,"%s",error);
  clean_return
}
//
// Start at the begining of the list of requests and send out the
// message that these machines are reserved...
//
pt = me_list.head();
while (pt != NULL)
{
  strcpy(machine_id,pt->val());
  Reserved_Machine resmach(machine_id);
  err = conv->send(resmach);
  if (err != OK)
  {
    sprintf(line,"Error sending Reserved_Machine err: %d",err);
    LOGS("sched_algorithm",line,"%s",error);
    clean_return
  }
  pt = pt->next();
}

/* wait for ack */
next_message("sched_algorithm",conv,req_msg)

/* make sure its the right message */
if (req_msg.msg_class_id() == M_REQUEST_COMP_ACK)
{
  Request_Comp_Ack reqack(req_msg);
  int ack_code = (int)reqack.ack_code().ifnull(INTNULL);
  if (ack_code != ACK_OK)
  {
    conv->end();
    error_lookup(ack_code,errortext);
    sprintf(line,"Entity returned ack_code #%d.\n%s",ack_code,errortext);
    LOGS("sched_algorithm",line,"%s",warning);
    clean_return
  }
  conv->end();
}
else
{
  conv->ignore();
```

```
    sprintf(line,"Unexpected message received: %s.\nThe conversation is: %s.",
                 req_msg.msg_class_id(),conv->cclass());
    LOGS("sched_algorithm",line,"%s",warning);
    clean_return
  }
 }
bye: exec sql commit;

clean_return

exec sql whenever not found continue;

} /* end of sched_algorithm */


/*************************************************************************
       Name: update_me_status

   Function: If the update_me_status changes the status of a machine to
             'Shutdown', this function deletes the machine from
             the REQUEST_MACHINES table and sendsthe machine_request_complete
             to all lot entities that requested a machine such that now
             the request cannot be filled.

     Inputs:         Conversation *C - the current conversation.
                     Update_ME_Status M - the message we look for

    Returns:         None.

   Modifies:         None.

   Dependancies:     handlers.inc M4 macro

   Language: C++/esql

   By:
      Peter A. Murray

   Date:
      07/07/1991

   Copyrite 1991 Siemens Corporate Reseaarch, INC

*************************************************************************/
void update_me_status(Conversation *C, Update_ME_Status& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   delete me_status;\
   delete machine_id; \
   return; \
   }

int_with_null machine_run_id;  // should be init to null
char *me_status  = (char *)(M.me_status().ifnull(CHARNULL));
char *machine_id = (char *)(M.machine_id().ifnull(CHARNULL));
int  request_key;
char entity_id[17];
```

```
char errortext[512];

/*
  There is no response for MSCHED. Tell the conversation
  tool to end the conversation.
*/
C->end();

/* if the machine is NOT Shutdown, then ignore the conversation */
if (strcmp(me_status,"Shutdown") == 0)
{
  sprintf(line,"machine %s has shutdown. Should any requests be canceled",
          machine_id);
  LOGS("update_me_status",line);
  /*
     go through the machine request table and find all of the requests
     which will be eliminated because there is no machine
     left (not shutdown) to satify the request. Call the
     update_me_status_scope routine until ack = ACK_OK.
  */
  while (NO_MACHIN == update_me_status_scope(machine_id,
                                             request_key,&entity_id[0]))
  {
    /* no machine is left, tell the waiting entity */
    Conversation CONV, *conv;
    conv = &CONV;
    conv->set_group(busname);
    Message req_msg;

    Machine_Req_Comp mrcnomach(NO_MACHIN,entity_id,machine_run_id,
                               request_key,0);
    err = conv->send(mrcnomach);
    if (err != OK)
    {
      sprintf(line,"Error #%d sending Machine_Req_Comp",err);
      LOGS("update_me_status",line,"%s",error);
    }
    /* wait for ack */
    next_message("update_me_status",conv,req_msg)

    /* make sure its the right message */
    if (req_msg.msg_class_id() == M_REQUEST_COMP_ACK)
    {
      Request_Comp_Ack reqack(req_msg);
      int ack_code = (int)reqack.ack_code().ifnull(INTNULL);
      if (ack_code != ACK_OK)
      {
        conv->end();
        error_lookup(ack_code,errortext);
        sprintf(line,"Entity returned ack_code #%d.\n%s",ack_code,errortext);
        LOGS("update_me_status",line,"%s",warning);
        clean_return
      }
      conv->end();
    }
    else
    {
      conv->ignore();
      sprintf(line,"Unexpected message received: %s.\nThe conversation is: %s.",
              req_msg.msg_class_id(),conv->cclass());
```

```
      LOGS("update_me_status",line,"%s",warning);
      clean_return
    }
  } /* end of while */
} /* end of if */
clean_return
} /* end of update_me_status */

/*****************************************************************
 *
 *                  UPDATE_ME_STATUS_SCOPE
 *
 *****************************************************************/
int update_me_status_scope(char *m_id,int &r_key,char *e_id)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      r_key = request_key; \
      return(ack); \
      }
ifdef(`on_error',undefine(`on_error'))
define(on_error,`request_key = INTNULL;')

ifdef(`on_badrows',undefine(`on_badrows'))
define(on_badrows,
`exec sql commit;
 request_key = INTNULL;')

exec sql begin declare section;
   char *entity_id = e_id;
   char *machine_id = m_id;
   int  request_key;
exec sql end declare section;

ack = ACK_OK;

saveto(sp_update)
{
  /*
     see if an entities are waiting for the machine which has just
     shutdown
  */
  exec sql select distinct request_key, entity_id
       into :request_key, :entity_id
       from REQUEST_MACHINES
       where machine_id = :machine_id;
  check_and_recover_to("update_me_status_scope",sp_update,
  "An ingres error occured reading from REQUEST_MACHINES",CHECK_ONEROW_NOLOG)

  /*
     delete the request from the table for the machine which shutdown
  */
  exec sql delete from REQUEST_MACHINES
       where request_key = :request_key;
  check_and_recover_to("update_me_status_scope",sp_update,
  "An ingres error occured deleting from REQUEST_MACHINES",DONT_CHECK)
  exec sql commit;
 /*
```

```
    not all machines are left to satisfy the request
    a Machine Request Complete is sent to the waiting
    entity indicating that the request has been removed
 */
  ack = NO_MACHIN;
  clean_return
}
} /* end of update_me_status_scope */

/*************************************************************************
       Name: cancel_request

   Function: to cancel a request for a machine.

     Inputs: Cancel_Request M // the message with the information in it.

    Returns:        None.

   Modifies: Request_machines table ....

   Dependancies:    None.

   Language: C++/esql

   By:
      Peter A. Murray
      Paul J. Bruschi

   Date:
      08/07/1991

   Copyrite 1991 Siemens Corporate Reseaarch, INC
*************************************************************************/
void cancel_request(Conversation *C,Cancel_Request& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   return; \
   }

int err = OK;
int_with_null machine_run_id;  // should be init to null
string_with_null machine_id;   // should be init to null
char entity_id[17];
char errortext[512];

int request_key = (int)M.request_key().ifnull(INTNULL);

ack = cancel_request_scope(request_key,&entity_id[0]);
if (ack == ACK_OK)
{
  Conversation CONV, *conv;
  conv = &CONV;
  conv->set_group(busname);
  Message req_msg;

  Machine_Req_Comp mrcreqcan(REQ_CANCE,entity_id,machine_run_id,
                             request_key, 0);
```

```
  err = conv->send(mrcreqcan);
  if (err != OK)
  {
    sprintf(line,"Error #%d sending Machine_Req_Comp",err);
    LOGS("cancel_request",line,"%s",error);
  }
  /* wait for ack */
  next_message("cancel_request",conv,req_msg)

  /* make sure its the right message */
  if (req_msg.msg_class_id() == M_REQUEST_COMP_ACK)
  {
    Request_Comp_Ack reqack(req_msg);
    int ack_code = (int)reqack.ack_code().ifnull(INTNULL);
    if (ack_code != ACK_OK)
    {
      conv->end();
      error_lookup(ack_code,errortext);
      sprintf(line,"Entity returned ack_code #%d.\n%s",ack_code,errortext);
      LOGS("cancel_request",line,"%s",warning);
      clean_return
    }
    conv->end();
  }
  else
  {
    conv->ignore();
    sprintf(line,"Unexpected message received: %s.\nThe conversation is: %s.",
              req_msg.msg_class_id(),conv->cclass());
    LOGS("cancel_request",line,"%s",warning);
    clean_return
  }
}
Cancel_Req_Ack cra1(request_key,ack);
err = C->send(cra1);
if (err != OK)
{
  sprintf(line,"Error #%d sending Cancel_Req_Ack",err);
  LOGS("cancel_request",line,"%s",error);
  clean_return
}
clean_return
} /* end of cancel_request */

/*****************************************************************************
 *
 *                        CANCEL_REQUEST_SCOPE
 *
 *****************************************************************************/
int cancel_request_scope(int r_key,char *e_id)
{
//
// Remove all occurences of a given request_key from the REQUEST_MACHINES
// table...
//
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
       return(ack); \
```

```
      }

ifdef(`on_error',undefine(`on_error'))
define(on_error, `ack = DB_Badrea;')

ifdef(`on_badrows',undefine(`on_badrows'))
define(on_badrows,
`ack = REQ_KEY_U;
 exec sql commit;')

exec sql begin declare section;
   char *entity_id = e_id;
   int request_key = r_key;
exec sql end declare section;

ack = ACK_OK;

saveto(sp_cancel_req)
{
  exec sql select distinct entity_id
       into :entity_id
       from REQUEST_MACHINES
       where request_key = :request_key;
  check_and_recover_to("cancel_request_scope",sp_cancel_req,
  "An ingres error occured reading REQUEST_MACHINES",CHECK_MANYROWS)

  exec sql repeated delete
       from REQUEST_MACHINES
       where request_key = :request_key;
  check_and_recover_to("cancel_request_scope",sp_cancel_req,
  "An ingres error occured deleting from REQUEST_MACHINES",DONT_CHECK)
  exec sql commit;
  clean_return
}
} /* end of cancel_request_scope */

/*****************************************************************************

      Name: UNRESERVE_MACHINE

   Function: To unreserve a machine wich was reserved earlier by
             msched.

     Inputs:         Conversation *C
                     Unreserve_Me &M

    Returns:         None.

   Modifies:         RESERVED_MACHINES

   Dependancies:     None.

   Language: C++/esql

   By:
      Paul J. Bruschi

   Date:
      08/07/1991
```

```
   Copyrite 1991 Siemens Corporate Reseaarch, INC

*****************************************************************************/
void unreserve_machine(Conversation *C, Unreserve_Me& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   delete machine_id; \
   return; \
   }

ifdef(`on_error',undefine(`on_error'))
define(on_error,
`Unreserve_Me_Ack unresack1(int_with_null(DB_Badupd));
 err = C->send(unresack1);
 if (err != OK)
 {
   sprintf(line,"Error #%d sending Unreserve_Me_Ack",err);
   LOGS("unreserve_machine",line,"%s",error);
 }')

ifdef(`on_badrows',undefine(`on_badrows'))
define(on_badrows,`')

saveto(sp_unres)
{
  exec sql begin declare section;
    char *machine_id = (char *)(M.machine_id().ifnull(CHARNULL));
  exec sql end declare section;

  exec sql delete from RESERVED_MACHINES
       where machine_id = :machine_id;
  check_and_recover_to("unreserve_machine",sp_unres,
  "An ingres error occured deleteing from RESERVED_MACHINES",DONT_CHECK)

  Unreserve_Me_Ack unresack(int_with_null(ACK_OK));
  err = C->send(unresack);
  if (err != OK)
  {
    sprintf(line,"Error #%d sending Unreserve_Me_Ack",err);
    LOGS("unreserve_machine",line,"%s",error);
  }

sched_algorithm();
clean_return
}
} /* end of unreserve_machine */
```

11.2 Listing 3, CMM_WC

```
/*
 * wc_interface
 *
 */
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "Conversation.h"
#include "cell_messages.h"
#include "ack_codes.h"
#include "logs.h"
#include "nullitems.h"
#include "machine_status.h"
#include "me_messages.h"
#include "me_defs.h"
#include "shared_mem_struct.h"
 exec sql include sqlca;'
#include "esql.h"

include(handlers.inc)

#undef clean_return

string12 machine_id = "cmm_wc";
string12 busname = "CELL";

extern "C" IISQLCA sqlca;

exec sql whenever sqlerror continue;
exec sql whenever sqlwarning continue;
exec sql whenever sqlmessage continue;

static char* startup[] = { "ME_START", (char*) NULL};
static char* start2[] = {"ALARM_CLE", (char *) NULL};
static char* clist[] = { "ME_SHUT", "START_RUN", "SETUP_RUN", "DNLD_ME_",
                         "UPLD_ME_", (char*) NULL };

MACHINE_STATUS status;

extern int ingres_error(char*,char*,int);
extern void LOGS(char*,char*,char* = "%s",LOG_MSG_TYPE = note);
extern int get_status(char*,MACHINE_STATUS*);
extern int change_status(char*,MACHINE_STATUS*);
extern int send_alarm(char*,char*,char*,char*,char*,char*,char*);
extern int convert_cname(char*);
extern int send_accept_types(char*,char*,int);
extern int mq_send(int msgid, long msg_type, char *msg_text);
int mq_read_wait(int msgid,long msg_type,char *msg_text);

void wc_startup(Conversation *C, ME_Startup& M);
void wc_shutdown(Conversation *C, ME_Shutdown& M);
void wc_setup_run(Conversation *C, Setup_Run& M);
void wc_start_run(Conversation *C, Start_Run& M);
void wc_download(Conversation *C, Download_Me& M);
void wc_upload(Conversation *C, Upload_Me& M);
void run_comp_action();
void alarm_action();
void alarm_action_sub();
void shutdown_machine();
void listen_me(int msgid);
```

```
extern void kill_mq_sm(int = 0);
extern int init_mq(char *);
extern int send_listen(int msgid,int conv,char *file_name,char *recp_id),
extern int mq_read_nowait(int msgid,long msg_type,char *msg_text);
WC_ME_BUFFER wc_me_message;
int msgid;

main()
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return break;

ifdef(`on_error',`undefine(`on_error')')
define(on_error,`kill_mq_sm();')

ifdef(`on_badrows',`undefine(`on_badrows')')
define(on_badrows,`kill_mq_sm();')

int err = 0;
exec sql begin declare section;
   char *database_name = DATABASE_NAME;
exec sql end declare section;

char process[20];
strcpy(process,"cmm_wc");
msgid = init_mq(process);
printf("WC:MSG_ID: %d\n", msgid);

LOGS("main","Starting ");

dbconnect:
exec sql connect :database_name;
check_and_recover_to("main",dbconnect,
   "An ingres error occurred during connect",DONT_CHECK)

/* Have this module sign on to ISIS  */
err = sign_on(machine_id,conv_timeout,TRUE);
if (err != 0)
  {
  sprintf(line,
  "ISIS returned error #%d while tring to sign on",err),
  LOGS("main",line,"%s",error);
  kill_mq_sm();
  }

/* have this module listen to the bus 'busname' and set the interest
 * to the conversation startup
 */
err = interest(busname,startup);
if (err != 0)
  {
  sprintf(line,
  "ISIS returned error #%d while setting interest filters. ",err);
  LOGS("main",line,"%s",error);
  sign_off();
  kill_mq_sm();
  }
```

```
sprintf(line,"Signed on to %s bus and waiting for startup",busname);
LOGS("main",line);

int bad_message = FALSE;
while (TRUE)
  {
  bad_message = FALSE;

  Message msg;
  Conversation *C;
  C = new Conversation;
  C->set_group(busname);

ifdef(`on_badreceive',undefine(`on_badreceive'))
define(on_badreceive, `listen_me(msgid);')

ifdef(`conv_timeout',undefine(`conv_timeout'))
define(conv_timeout, `3')

  first_message("main",C,msg)

ifdef(`on_badreceive',undefine(`on_badreceive'))
define(on_badreceive, `')

ifdef(`conv_timeout',undefine(`conv_timeout'))
define(conv_timeout, `60')

  switch( C->iclass() )
    {
    case C_ME_START :
      if (msg.msg_class_id() == M_ME_STARTUP)
        {
        ME_Startup sr12(msg);
        wc_startup(C,sr12);
        }
      else
        bad_message = TRUE;
      break;
    case C_ME_SHUT :
      if (msg.msg_class_id() == M_ME_SHUTDOWN)
        {
        ME_Shutdown sr11(msg);
        wc_shutdown(C,sr11);
        }
      else
        bad_message = TRUE;
      break;
    case C_SETUP_RUN :
      if (msg.msg_class_id() == M_SETUP_RUN)
        {
        Setup_Run sr13(msg);
        wc_setup_run(C,sr13);
        }
      else
        bad_message = TRUE;
      break;
    case C_START_RUN :
      if (msg.msg_class_id() == M_START_RUN)
        {
```

```
      Start_Run sr14(msg);
      wc_start_run(C,sr14);
      }
    else
      bad_message = TRUE;
    break;
  case C_DNLD_ME_ :
    if (msg.msg_class_id() == M_DOWNLOAD_ME)
      {
      Download_Me dm11(msg);
      wc_download(C,dm11);
      }
    else
      bad_message = TRUE;
    break;
  case C_UPLD_ME_ :
    if (msg.msg_class_id() == M_UPLOAD_ME)
      {
      Upload_Me um11(msg);
      wc_upload(C,um11);
      }
    else
      bad_message = TRUE;
    break;
  default:
    C->sprint_conversation(line);
    strcat(line,"\nUnexpected Conversation received");
    LOGS("main",line,"%s",warning);
    C->ignore();
    break;
  } /* end of switch */

 if ( bad_message == TRUE )
   {
   sprintf(line,"Unexpected message received '%s' for conversation: '%s'.",
           msg.msg_name(), C->cclass() );
   LOGS("main",line,"%s",warning);
   C->ignore();
   }
 exec sql commit;
 delete C;
 } /* end of while */
} /* end of main */


/*****************************************************************************
 *
 *                WC_STARTUP
 *
 *****************************************************************************/
void wc_startup(Conversation *C, ME_Startup& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   delete mach_id; \
   delete CONV1; \
   delete CONV2; \
   return; \
```

```
   }

int ack = ACK_OK;
int err = 0;

Message msg1;
Conversation *CONV1;
CONV1 = new Conversation;
CONV1->set_group(busname);

Message msg2,
Conversation *CONV2;
CONV2 = new Conversation;
CONV2->set_group(busname);


ifdef(`err_return',undefine(`err_return'))
define(err_return,
`send_alarm(busname,
           "MESTRT_FAIL",
           "MACHINE",
           "",
            machine_id,
           "See Error Log",
           "now");
 clean_return')

char *mach_id = (char *)M.machine_id().ifnull(CHARNULL);

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0)
{
   /*
     A startup message was received for the wc_interface
     wc checks for its status, if stauts is not Shutdown then
     sends back Notnow acknowledgement
   */

   LOGS("wc_startup","WC startup received");

   if (ACK != get_status(machine_id,&status))
   {
      ME_Startup_Ack mesa1(machine_id,DB_Badrea);
      err = C->send(mesa1);
      if (err != 0)
      {
         sprintf(line,"Error #%d sending ME_Startup_Ack",err);
         LOGS("wc_startup",line,"%s",error);
      }
      clean_return
   }
   if (strncmp(status.me_status,"Shutdown",strlen("Shutdown")) != 0)
   {
      LOGS("wc_startup","Sending Notnow. ME not Shutdown.","%s",warning);
      ME_Startup_Ack mesa2(machine_id,Notnow),
      err = C->send(mesa2);
      if (err != 0)
      {
         sprintf(line,"Error #%d sending ME_Startup_Ack",err);
         LOGS("wc_startup",line,"%s",error),
      }
```

```
   clean_return
}

/*
   status is Shutdown so wc sends back the acknowledgement OK
*/
ME_Startup_Ack mesa3(machine_id,int_with_null(ACK_OK));
err = C->send(mesa3);
if (err != 0)
{
   sprintf(line,"Error #%d sending ME_Startup_Ack",err);
   LOGS("wc_startup",line,"%s",error);
   clean_return
}

/*
   Status of WC is Shutdown. Send ME_Startup
   message to the machine and wait for ack.
*/

strcpy(wc_me_message.file_name, "NULL");
strcpy(wc_me_message.recipe_id, "NULL");
wc_me_message.message_id = ME_START;
wc_me_message.ack_code = ACK;
strcpy(wc_me_message.alarm_text,"NULL");

if (IS_MESSAGE != mq_send(msgid,TO_ME,(char *)&wc_me_message))
{
   sprintf(line, "Error sending to ME startup msg");
   LOGS("wc_startup",line,"%s",warning);
   err_return
}
if (IS_MESSAGE != mq_read_wait(msgid,FROM_ME,(char *)&wc_me_message))
{
   sprintf(line, "Got no responce from the ME");
   LOGS("wc_startup",line,"%s",warning);
   err_return
}

sprintf(line, "Message_id = %d \n",wc_me_message.message_id);
LOGS("wc_startup",line,"%s",warning);
if (wc_me_message.message_id == CELL_ALAR)
{
   sprintf(line, "Got an alarm for cmm home");
   LOGS("wc_startup",line,"%s",note);
   /* WARNING ----
   **this is a cluge to get the demo working....
   **alarm_action listens for an ack, any old ack, and it dosent care
   **who or what the ack is...
   **IT will then send an ack back to the ME half...
   **The ack should be confirmed that it is the one expected...
   **You want a demo well here is your demo.....
   */

   /* also listen for ALARMs begin cleared */
   err = interest(busname,start2);
   if (err != 0){
      sprintf(line,
      "Error #%d while setting conversation filter",err);
      LOGS("wc_startup",line,"%s",error),
```

```
        err = interest(busname,startup);
        if (err != 0){
           sprintf(line,"Error #%d while setting conversation filter",err);
           LOGS("wc_startup",line,"%s",error);
        }
       err_return
     }
     alarm_action();
     listen_me(msgid);   /* listen for an alarm or ack */
  }

     err = no_interest(busname);
     if (err != 0){
        sprintf(line,
        "Error #%d while setting conversation filter to no_intrest",err);
        LOGS("wc_startup",line,"%s",error);
        err_return
     }
     err = interest(busname,clist);
     if (err != 0){
        sprintf(line,
        "Error #%d while setting conversation filter",err);
        LOGS("wc_startup",line,"%s",error);
        err = interest(busname,startup);
        if (err != 0){
           sprintf(line,"Error #%d while setting conversation filter",err);
           LOGS("wc_startup",line,"%s",error);
        }
       err_return
     }


ifdef(`err_return',undefine(`err_return'))
define(err_return,
`err = send_alarm(busname,
                  "MESTRT_FAIL",
                  "MACHINE",
                  "",
                   machine_id,
                  "See Error Log",
                  "now");
 if (err == 0){
   clean_return
 }
 err = no_interest(busname);
 if (err != 0){
    sprintf(line,
    "Error #%d while setting conversation filter to no_intrest",err);
    LOGS("wc_startup",line,"%s",error);
    clean_return
 }
 err = interest(busname,startup);
 if (err != 0){
    sprintf(line,
    "Error #%d while setting conversation filter",err);
    LOGS("wc_startup",line,"%s",error);
 }
 clean_return')

   if (ACK != send_accept_types(machine_id,busname,ALLOPS)){
```

```
      err_return
   }
   strcpy(status.recipe_id,CHARNULL);
   strcpy(status.start_time,TIMENULL);
   strcpy(status.end_time,TIMENULL);
   strcpy(status.me_status,"Idle");
   status.machine_run_id = INTNULL;
   status.note_id = INTNULL;
   if (ACK != change_status(busname,&status)){
      sprintf(line, "Cannot change the status to %s",status.me_status);
      LOGS("wc_startup",line,"%s",warning);
      err_return
   }
   ME_Ready merd(machine_id);
   err = CONV1->send(merd);
   if (err != 0){
      sprintf(line,"Error #%d sending ME_Ready",err);
      LOGS("wc_startup",line,"%s",error);
      clean_return
   }
}
else
   C->ignore();
clean_return
} /* end of wc_startup */


/*****************************************************************************
 *
 *                 WC_SHUTDOWN
 *
 *****************************************************************************/
void wc_shutdown(Conversation *C, ME_Shutdown& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      delete mach_id; \
      return; \
      }
int ack = ACK_OK;
int err = 0;

char *mach_id = (char *)M.machine_id().ifnull(CHARNULL);

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0)
{
   /*
      A shutdown message was received for the wc_interface.
      First WC will set the accept types to accept no more operations.
      Then it will check its status: if it is "Idle" it will
      shutdown immediately. if it is "Setup" it will set its state to
      "Setup_Shut" and will shutdown when the run is complete.  If
      it is "Running" it will set its state to "Run_Shut" and
      will shutdown when the run is complete.
   */
   LOGS("wc_shutdown","WC shutdown received");

   ME_Shutdown_Ack mesh(machine_id,int_with_null(ACK_OK));
```

```
err = C->send(mesh),
if (err != 0)
{
   sprintf(line,"Error #%d sending ME_Shutdown_Ack",err);
   LOGS("wc_shutdown",line,"%s",error);
   clean_return
}
if (ACK != send_accept_types(machine_id, busname, NONE))
{
   clean_return
}
if (ACK != get_status(machine_id,&status))
{
   ack = DB_Badrea;
   sprintf(line,"Cannot read the current status");
   LOGS("wc_shutdown",line,"%s",warning);
   clean_return
}
if (strncmp(status.me_status,"Idle",strlen("Idle")) == 0)
{
   shutdown_machine();
   clean_return
}
if (strncmp(status.me_status,"Setup",strlen("Setup")) == 0)
{
   strcpy(status.me_status,"Setup_Shut");
   if (ACK != change_status(busname,&status)){
      sprintf(line, "Cannot change the status to %s",status.me_status);
      LOGS("wc_shutdown",line,"%s",warning);
   }
   clean_return
}
if (strncmp(status.me_status,"Running",strlen("Running")) == 0)
{
   strcpy(status.me_status,"Run_Shut");
   if (ACK != change_status(busname,&status))
   {
      sprintf(line, "Cannot change the status to %s",status.me_status);
      LOGS("wc_shutdown",line,"%s",warning);
   }
   clean_return
}

/*
   Status of WC is Shutdown. Send ME_Shutdown
   message to the machine and wait for ack.
*/

strcpy(wc_me_message.file_name, "NULL");
strcpy(wc_me_message.recipe_id, "NULL");
wc_me_message.message_id = ME_SHUT;
wc_me_message.ack_code = ACK;
strcpy(wc_me_message.alarm_text,"NULL");

if (IS_MESSAGE != mq_send(msgid,TO_ME,(char *)&wc_me_message))
{
   sprintf(line, "Error sending to ME startup msg");
   LOGS("wc_startup",line,"%s",warning);
   err_return
}
```

```
   if (IS_MESSAGE != mq_read_wait(msgid,FROM_ME,(char *)&wc_me_message))
   {
      sprintf(line, "Got no responce from the ME");
      LOGS("wc_startup",line,"%s",warning);
      err_return
   }

   sprintf(line, "Message_id = %d \n",wc_me_message.message_id);
   LOGS("wc_startup",line,"%s",warning);
   if (wc_me_message.message_id != ME_SHUT_C)
   {
      sprintf(line, "Got the wrong responce from the ME");
      LOGS("wc_startup",line,"%s",warning);
   }
}
else
   C->ignore();
clean_return
} /* end of wc_shutdown */

/*************************************************************************
 *
 *                  SHUTDOWN_MACHINE
 *
 *  This will change the status to "Shutdown", change the conversation
 *  filter to accept only a startup and then send out the ME_SHUTDOWN_COMPLETE
 *  conversation.
 *************************************************************************/
void shutdown_machine()
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      delete CONV1; \
      return; \
      }

int err = 0;

Conversation *CONV1;
CONV1 = new Conversation;
CONV1->set_group(busname);

LOGS("shutdown_machine","WC will shutdown the machine");

strcpy(status.recipe_id, CHARNULL);
strcpy(status.start_time,TIMENULL);
strcpy(status.end_time,TIMENULL);
strcpy(status.me_status,"Shutdown");
status.machine_run_id = INTNULL;
status.note_id = INTNULL;
if (ACK != change_status(busname,&status)){
   sprintf(line, "Cannot change the status to %s",status.me_status);
   LOGS("shutdown_machine",line,"%s",warning);
   clean_return
}
err = no_interest(busname);
if (err != 0){
   sprintf(line,
```

```
   "Error #%d while setting conversation filter to no_interest",err);
   LOGS("shutdown_machine",line,"%s",error);
   clean_return
}
err = interest(busname,startup);
if (err != 0){
   sprintf(line,"Error #%d while setting conversation filter to startup",err);
   LOGS("shutdown_machine",line,"%s",error);
   clean_return
}
ME_Stopped mest(machine_id);
err = CONV1->send(mest);
if (err != 0){
   sprintf(line,"Error #%d sending ME_Stopped",err);
   LOGS("shutdown_machine",line,"%s",error);
   clean_return
}
clean_return
}/* end of shutdown_machine */

/*************************************************************************
 *
 *                 WC_SETUP_RUN
 *
 *************************************************************************/
void wc_setup_run(Conversation *C, Setup_Run& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      delete mach_id; \
      delete op_type; \
      delete recipe_id; \
      delete filename; \
      return; \
      }

int ack = ACK_OK;
int err = 0;

exec sql begin declare section;
   char *mach_id;
   char *op_type;
   char *recipe_id;
   char *filename;
   int  row_count;
exec sql end declare section;

mach_id   = (char *)M.machine_id().ifnull(CHARNULL);
op_type   = (char *)M.op_type().ifnull(CHARNULL);
recipe_id = (char *)M.recipe_id().ifnull(CHARNULL);
filename  = (char *)M.filename().ifnull(CHARNULL);


ifdef(`on_badrows',undefine(`on_badrows'))
define(on_badrows,
`Setup_Run_Ack srnal(machine_id, Rec_Inval);
 err = C->send(srnal);
 if (err != 0){
```

```
    sprintf(line,"Error #%d sending Setup_Run_Ack",err);
    LOGS("wc_setup_run",line,"%s",error);
    }')

ifdef(`on_error',undefine(`on_error'))
define(on_error,
`Setup_Run_Ack srna2(machine_id, DB_Badrea);
 err = C->send(srna2);
 if (err != 0){
    sprintf(line,"Error #%d sending Setup_Run_Ack",err);
    LOGS("wc_setup_run",line,"%s",error);
    }')

ifdef(`err_return',undefine(`err_return'))
define(err_return,
`Setup_Run_Ack srna3(machine_id, ack);
 err = C->send(srna3);
 if (err != 0){
    sprintf(line,"Error #%d sending Setup_Run_Ack",err);
    LOGS("wc_setup_run",line,"%s",error);
    }
 clean_return')

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0){
   /*
      A setup_run message was received for the machine.
      WC will check if its status is "Idle", download the recipe,
      then send back the acknowledgement
   */
   LOGS("wc_setup_run","setup_run received");

   if (ACK != get_status(machine_id,&status)){
      ack = DB_Badrea;
      sprintf(line,"Cannot read the status");
      LOGS("wc_setup_run",line,"%s",warning);
      err_return
   }
   if (strncmp(status.me_status,"Idle",strlen("Idle")) != 0){
      ack = Notnow;
      sprintf(line,"Machine not Idle, try later.");
      LOGS("wc_setup_run",line,"%s",warning);
      err_return
   }
   /*
      Check if the recipe_id and op_type are valid for the machine
   */

   saveto(sp_oprec);
   exec sql select count(*)
          into :row_count
          from OP_MACHINE_RECIPE
         where machine_id = :mach_id and
               recipe_id = :recipe_id and
                 op_type = :op_type;
   check_and_recover_to("wc_setup_run",sp_oprec,
  "An ingres error occured while reading OP_MACHINE_RECIPE",CHECK_ONEROW_NOLOG)
   exec sql commit;

   /*
     Now setup the run by telling the me_interface to download the recipe
```

```
    to the machine
   */
   ack = Start_Fai;
   if (ACK != send_listen(msgid,SETUP_RUN,filename,recipe_id)){
      sprintf(line,"Cannot setup the run on the machine.");
      LOGS("wc_setup_run",line,"%s",warning);
      err_return
   }

   /*
      Change the accept types to none
   */
   ack = DB_Badupd;
   if (ACK != send_accept_types(machine_id,busname,NONE)){
      err_return
   }

   /*
     change the status to "Setup"
   */
   strcpy(status.recipe_id, recipe_id);
   strcpy(status.start_time,TIMENULL);
   strcpy(status.end_time,TIMENULL);
   strcpy(status.me_status,"Setup");
   status.machine_run_id = M.machine_run_id().ifnull(INTNULL);

   if (ACK != change_status(busname,&status)){
      sprintf(line, "Cannot change the status to %s",status.me_status);
      LOGS("wc_setup_run",line,"%s",warning);
      err_return
   }
   /*
      machine is setup for run so send setup_run_ack
   */
   Setup_Run_Ack srna4(machine_id,int_with_null(ACK_OK));
   err = C->send(srna4);
   if (err != 0) {
      sprintf(line,"Error #%d sending Setup_Run_Ack",err);
      LOGS("wc_setup_run",line,"%s",error);
      clean_return
   }
}
else
   C->ignore();
clean_return
} /* end of wc_setup_run */

/*****************************************************************************
 *
 *                WC_START_RUN
 *
 *****************************************************************************/
void wc_start_run(Conversation *C, Start_Run& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   delete mach_id; \
   delete recipe_id; \
```

```
   delete filename; \
   return; \
   }

int ack = ACK_OK;
int err = 0;

char *mach_id   = (char *)M.machine_id().ifnull(CHARNULL);
char *recipe_id = (char *)M.recipe_id().ifnull(CHARNULL);
char *filename  = (char *)M.filename().ifnull(CHARNULL);

ifdef(`err_return',undefine(`err_return'))
define(err_return,
`Start_Run_Ack strnal(machine_id,Start_Fai);
 err = C->send(strnal);
 if (err != 0){
    sprintf(line,"Error #%d sending Start_Run_Ack",err);
    LOGS("wc_start_run",line,"%s",error);
 }
 clean_return')

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0){
   /*
     A start_run message was received for the wc_interface
     wc send back the acknowledgement and sets the filters to
     receive all of the messages.
   */
   LOGS("wc_start_run","start_run received");

   if (ACK != get_status(machine_id,&status)){
      ack = DB_Badrea;
      sprintf(line,"Cannot read the status");
      LOGS("wc_start_run",line,"%s",warning);
      err_return
   }
   if ((strncmp(status.me_status,"Setup",strlen("Setup")) != 0) &&
      (strncmp(status.me_status,"Setup_Shut",strlen("Setup_Shut")) !=0)){
      ack = Notnow;
      sprintf(line,"Machine not setup for run, try later.");
      LOGS("wc_start_run",line,"%s",warning);
      err_return
   }

   /*
         Status of WC is setup or setup_shut so send start_run
         message to me and wait for ack.
   */
   if (ACK != send_listen(msgid,START_RUN, filename, recipe_id)){
      ack = Start_Fai;
      sprintf(line, "ME could not start machine");
      LOGS("wc_start_run",line,"%s",warning);
      err_return
   }
   /*
        change the status to "Running" or "Run_Shut"
   */
   strcpy(status.start_time,"now");
   strcpy(status.end_time,TIMENULL);
   if (strncmp(status.me_status,"Setup",strlen("Setup")) == 0)
       strcpy(status.me_status,"Running");
```

```
   else
       strcpy(status.me_status,"Run_Shut");
   if (ACK != change_status(busname,&status)){
      sprintf(line, "Cannot change the status to %s",status.me_status);
      LOGS("wc_start_run",line,"%s",warning);
      err_return
   }
   Start_Run_Ack strna2(machine_id,int_with_null(ACK_OK));
   err = C->send(strna2);
   if (err != 0) {
      sprintf(line,"Error #%d sending Start_Run_Ack",err);
      LOGS("wc_start_run",line,"%s",error);
      clean_return
   }
}
else
   C->ignore();
clean_return
} /* end of wc_start_run */

/*****************************************************************************
 *
 *                 WC_UPLOAD
 *
 *****************************************************************************/
void wc_upload(Conversation *C, Upload_Me& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      delete mach_id; \
      delete recipe_id; \
      delete filename; \
      return; \
      }

int ack = ACK_OK;
int err = 0;

char *mach_id   = (char *)M.machine_id().ifnull(CHARNULL);
char *recipe_id = (char *)M.recipe_id().ifnull(CHARNULL);
char *filename  = (char *)M.filename().ifnull(CHARNULL);

ifdef(`err_return',undefine(`err_return'))
define(err_return,
`Upload_Ack uplda1(ack);
 err = C->send(uplda1);
 if (err != 0){
    sprintf(line,"Error #%d sending Upload_Ack",err);
    LOGS("wc_upload",line,"%s",error);
 }
 clean_return')

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0){
   /*
      An upload_me message was received for the wc_interface
      wc will try to upload the recipe if the me_status is "Idle"
      then send back the acknowledgement.
   */
```

```
   LOGS("wc_upload","Upload received");

   if (ACK != get_status(machine_id,&status)){
      ack = DB_Badrea;
      sprintf(line,"Cannot read the status");
      LOGS("wc_upload",line,"%s",warning);
      err_return
   }
   if (strncmp(status.me_status,"Idle",strlen("Idle")) != 0){
      ack = Notnow;
      sprintf(line,"Machine not Idle, try later.");
      LOGS("wc_upload",line,"%s",warning);
      err_return
   }
   /*
         Status of the machine is "Idle", so it can upload the recipe.
         WC will send the upload message to ME and wait for ack.
   */
   if (ACK != send_listen(msgid,UPLD_ME_,filename,recipe_id)){
      ack = Notnow;
      sprintf(line,"Cannot upload recipe from the machine");
      LOGS("wc_upload",line,"%s",warning);
      err_return
   }
   Upload_Ack uplda2(int_with_null(ACK_OK));
   err = C->send(uplda2);
   if (err != 0) {
      sprintf(line,"Error #%d sending Upload_Me_Ack",err);
      LOGS("wc_upload",line,"%s",error);
      clean_return
   }
}
else
   C->ignore();
clean_return
} /* end of wc_upload */

/*************************************************************************
 *
 *                 WC_DOWNLOAD
 *
 *************************************************************************/
void wc_download(Conversation *C, Download_Me& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   delete mach_id; \
   delete recipe_id; \
   delete filename; \
   return; \
   }

int ack = ACK_OK;
int err = 0;

char *mach_id   = (char *)M.machine_id().ifnull(CHARNULL);
char *recipe_id = (char *)M.recipe_id().ifnull(CHARNULL);
char *filename  = (char *)M.filename().ifnull(CHARNULL),
```

```
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   delete COMP; \
   return; \
   }

int ack = ACK_OK;
int err = 0;

FILE  *result_fp;

exec sql begin declare section;
   char parm[13];
   char value[26];
   int  machine_run_id;
exec sql end declare section;

Conversation *COMP;
COMP = new Conversation;
COMP->set_group(busname);

ifdef(`on_error',undefine(`on_error'))
define(on_error,
`sprintf(line,"Error inserting into MACHINE_RUN_RESULTS");
 LOGS("run_comp_action",line);
 fclose(result_fp);
 Run_Complete rncom(machine_run_id);
 err = COMP->send(rncom);
 if (err != 0){
    sprintf(line,"Error #%d sending Run_Complete",err);
    LOGS("run_comp_action",line,"%s",error);
 }')

ifdef(`on_badrows',undefine(`on_badrows'))
define(on_badrows,`')

LOGS("run_comp_action","WC sees the end of a run.");

/*
     First get the status and update the end time
*/
if (ACK != get_status(machine_id,&status))
   {
   ack = DB_Badrea;
   sprintf(line,"Cannot read the status");
   LOGS("run_comp_action",line,"%s",warning);
   clean_return
   }

strcpy(status.end_time,"now");

if (ACK != change_status(busname,&status))
   {
   sprintf(line,"Cannot update the end time to %s",status.end_time);
   LOGS("run_complete_action",line,"%s",warning);
   clean_return
   }
```

```
machine_run_id = status.machine_run_id,

/* now get the results and put them into the database */
if ((result_fp = fopen(wc_me_message.file_name,"r")) == NULL)
   {
   sprintf(line,"Error in opening result file %s.",wc_me_message.file_name);
   LOGS("run_comp_action",line,"%s",error);
   }
else
   { /* get the results from the file and put into the database */
   saveto(sp_results)
   while (fscanf(result_fp,"%s",parm) != EOF)
      {
      fscanf(result_fp,"%s",value);
      sprintf(line,"machine result read - value = %s. parm = %s",value,parm);
      LOGS("run_comp_action",line);
      exec sql repeated insert
          into MACHINE_RUN_RESULT
                 (machine_run_id, param_id, value)
           values(:machine_run_id, :parm, :value);
      check_and_recover_to("run_comp_action",sp_results,
         "An ingres error occured while inserting MACHINE_RUN_RESULT",
          DONT_CHECK)
      }
   exec sql commit;
   /* close and delete result file */
   fclose(result_fp);
   if (unlink(wc_me_message.file_name) < 0)
     perror("unlink");
   }

Run_Complete rncom1(machine_run_id);
err = COMP->send(rncom1);
if (err != 0)
   {
   sprintf(line,"Error #%d sending Run_Complete",err);
   LOGS("run_comp_action",line,"%s",error);
   clean_return
   }
if ((strncmp(status.me_status,"Setup_Shut",strlen("Setup_Shut")) == 0) ||
   (strncmp(status.me_status,"Run_Shut",strlen("Run_Shut")) ==0))
   {
   shutdown_machine();
   clean_return
   }
strcpy(status.recipe_id,CHARNULL);
strcpy(status.start_time,TIMENULL);
strcpy(status.end_time,TIMENULL);
strcpy(status.me_status,"Idle");
status.machine_run_id = INTNULL;
status.note_id = INTNULL;

if (ACK != change_status(busname,&status)){
   sprintf(line,"Cannot change status to %s",status.me_status);
   LOGS("run_complete_action",line,"%s",warning);
   clean_return
}
if (ACK != send_accept_types(machine_id,busname,ALLOPS)){
   clean_return
}
```

```
clean_return
}/* end of run_comp_action */

/*********************************************************

                    ALARM_ACTION

*********************************************************/
void alarm_action()
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
    return; \
}
int err = 0;
int sa_ack = 0;

exec sql begin declare section;
   char alarm_id[13];
   char alarm[81];
   char run_id[16];
   char parameters[129];
exec sql end declare section;


ifdef(`on_error',undefine(`on_error'))
define(on_error, ')

ifdef(`on_badrows',undefine(`on_badrows'))
define(on_badrows,
`sprintf(line, "Alarm from machine not recognized.\n%s.",alarm);
 LOGS("alarm_action",line);
 exec sql commit;')

LOGS("alarm_action","WC sees an alarm");

if (status.machine_run_id == INTNULL)
   strcpy(run_id,CHARNULL);
else
   sprintf(run_id,"%d",status.machine_run_id);

strncpy(alarm,wc_me_message.alarm_text,sizeof(alarm));

saveto(sp_alarm);
exec sql select alarm_id, description
              into :alarm_id, :parameters
              from ALARM_DESCRIPTION
              where robot_alarm = :alarm;
check_and_recover_to("alarm_action",sp_alarm,
   "An ingres error occured while reading ALARM_DESCRIPTION",CHECK_ONEROW)
exec sql commit;
/*
  if an error occurred send_alarm returns a 0
  otherwise it returns a serial_number....
*/
sa_ack = send_alarm(busname,
           alarm_id,
           "MACHINE",
```

```
            run_id,
            machine_id,
            parameters,
            "now");
 alarm_action_sub();
}
void alarm_action_sub()
{
  int bad_message = FALSE;
  int DONE = FALSE;

  Message msg;
  Conversation *C,CONV;
  C = &CONV;
  C->set_group(busname);


  while(DONE == FALSE)
  {
ifdef(`on_badreceive',undefine(`on_badreceive'))
define(on_badreceive, `listen_me(msgid);')
ifdef(`conv_timeout',undefine(`conv_timeout'))
define(conv_timeout, `60')

   first_message("alarm_action",C,msg)

ifdef(`on_badreceive',undefine(`on_badreceive'))
define(on_badreceive, `')
ifdef(`conv_timeout',undefine(`conv_timeout'))
define(conv_timeout, `60')

  switch( C->iclass() )
    {
    case C_ALARM_CLE:
      if (msg.msg_class_id() == M_CELL_ALARM_CLEAR)
      {
        DONE = TRUE;
        sprintf(line, "Got CELL_ALARM_CLEAR");
        LOGS("alarm_action",line);
        wc_me_message.message_id = ME_START_WAIT;
        strcpy(wc_me_message.file_name, "NULL");
        strcpy(wc_me_message.recipe_id, "NULL");
        wc_me_message.ack_code = ACK;
        strcpy(wc_me_message.alarm_text, "NULL");
        if (mq_send(msgid, TO_ME, (char *)&wc_me_message) == ERROR_MSG)
        {
          sprintf(line, "alarm_action mq_send failed..");
          LOGS("alarm_action",line,"%s",error);
        }
      }
      else
        bad_message = TRUE;
      break;
    default:
      C->sprint_conversation(line);
      strcat(line,"\nUnexpected Conversation received");
      LOGS("main",line,"%s",warning);
      C->ignore();
      break;
    } /* end of switch */
```

```
    if ( bad_message == TRUE )
    {
    sprintf(line,"Unexpected message received '%s' for conversation: '%s'.",
            msg.msg_name(), C->cclass() );
    LOGS("main",line,"%s",warning);
    C->ignore();
    }
 } /* eo while */
clean_return
}

void listen_me(int msgid)
{
ifdef(`err_return',undefine(`err_return'))
define(err_return,
`send_alarm(busname,
           "MESTRT_FAIL",
           "MACHINE",
           "",
            machine_id,
           "See Error Log",
           "now");
 clean_return')

#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
    return; \
}
/* look for message from the me_interface */

if (mq_read_nowait(msgid,FROM_ME,(char *)&wc_me_message) == IS_MESSAGE)
{
  if (wc_me_message.message_id == NO_MESSAGE)
    return;
  if (wc_me_message.message_id == RUN_COMPL)
  {
    run_comp_action();
    wc_me_message.message_id = NO_MESSAGE;
    return;
  }
  if (wc_me_message.message_id == CELL_ALAR)
  {
     alarm_action();
    /* everything is ok send ack back to me... to continue */
    wc_me_message.message_id = ME_START_WAIT;
    if (IS_MESSAGE != mq_send(msgid, TO_ME,(char *)&wc_me_message))
    {
         sprintf(line, "WC had error sending ME_START_WAIT msg");
         LOGS("wc_startup",line,"%s",warning);
         err_return
    }
    wc_me_message.message_id = NO_MESSAGE;
    return;
  }
  if (wc_me_message.message_id == ME_START)
  {
    wc_me_message.message_id = NO_MESSAGE;
```

```
    if (wc_me_message.ack_code != ACK)
    {
         sprintf(line, "ME could not start machine");
         LOGS("wc_startup",line,"%s",warning);
         err_return
    }
    return;
  }
}
}
```

11.3 Listing 4, Start.csh

```
#
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/hi.csh
```

11.4 Listing 5, Stop.csh

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

11.5 Listing 6, Dummy WC_INTERFACE

```
/*
 * wc_interface
 *
 */
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "Conversation.h"
#include "cell_messages.h"
#include "ack_codes.h"
#include "logs.h"
#include "nullitems.h"
#include "machine_status.h"
#include "me_messages.h"
#include "me_defs.h"
#include "shared_mem_struct.h"
exec sql include sqlca;'
#include "esql.h"

include(handlers.inc)

#undef clean_return

string12 machine_id = "wcn";
string12 busname = "CELL";

extern "C" IISQLCA sqlca;

exec sql whenever sqlerror continue;
exec sql whenever sqlwarning continue;
exec sql whenever sqlmessage continue;

static char* startup[] = { "ME_START", (char*) NULL};
static char* clist[] = { "ME_SHUT", "START_RUN", "SETUP_RUN", "DNLD_ME_",
                         "UPLD_ME_", (char*) NULL };

MACHINE_STATUS status;

extern int ingres_error(char*,char*,int);
extern void LOGS(char*,char*,char* = "%s",LOG_MSG_TYPE = note);
extern int get_status(char*,MACHINE_STATUS*);
extern int change_status(char*,MACHINE_STATUS*);
extern int send_alarm(char*,char*,char*,char*,char*,char*,char*);
extern int convert_cname(char*);
extern int send_accept_types(char*,char*,int);

void wc_startup(Conversation *C, ME_Startup& M);
void wc_shutdown(Conversation *C, ME_Shutdown& M);
void wc_setup_run(Conversation *C, Setup_Run& M);
void wc_start_run(Conversation *C, Start_Run& M);
void wc_download(Conversation *C, Download_Me& M);
void wc_upload(Conversation *C, Upload_Me& M);
void run_comp_action();
void alarm_action();
void shutdown_machine();
void listen_me(int msgid);

extern void kill_mq_sm(int = 0);
extern int init_mq(char *);
extern int send_listen(int msgid,int conv,char *file_name,char *recp_id);
```

```
extern int mq_read_nowait(int msgid,long msg_type,char *msg_text);
WC_ME_BUFFER wc_me_message;
int msgid;

main(int argc, char *argv[])
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return break;

ifdef(`on_error',`undefine(`on_error')')
define(on_error,`kill_mq_sm();')

ifdef(`on_badrows',`undefine(`on_badrows')')
define(on_badrows,`kill_mq_sm();')

int err = 0;
exec sql begin declare section;
   char *database_name = DATABASE_NAME;
exec sql end declare section;

char process[20];
strcpy(process,"wcn");
msgid = init_mq(process);

LOGS("main","Starting ");

dbconnect:
exec sql connect :database_name;
check_and_recover_to("main",dbconnect,
   "An ingres error occurred during connect",DONT_CHECK)

/* Have this module sign on to ISIS  */
err = sign_on(machine_id,conv_timeout,TRUE);
if (err != 0)
  {
  sprintf(line,
  "ISIS returned error #%d while tring to sign on",err);
  LOGS("main",line,"%s",error);
  kill_mq_sm();
  }

/* have this module listen to the bus 'busname' and set the interest
 * to the conversation startup
 */

/* if there is a command line argument, the bus name will be set to it,
   otherwise the default "hard coded" value will be used */

if (argc == 2)
    strcpy(busname, argv[1]);

sprintf(line, "ISIS bus: '%s'", busname);
LOGS("main", line, "%s", note);

err = interest(busname,startup);
if (err != 0)
  {
  sprintf(line,
```

```
  "ISIS returned error #%d while setting interest filters. ",err);
  LOGS("main",line,"%s",error);
  sign_off();
  kill_mq_sm();
  }

sprintf(line,"Signed on to %s bus and waiting for startup",busname);
LOGS("main",line);

int bad_message = FALSE;
while (TRUE)
  {
  bad_message = FALSE;

  Message msg;
  Conversation *C;
  C = new Conversation;
  C->set_group(busname);

ifdef(`on_badreceive',undefine(`on_badreceive'))
define(on_badreceive, `listen_me(msgid);')

ifdef(`conv_timeout',undefine(`conv_timeout'))
define(conv_timeout, `3')

  first_message("main",C,msg)

ifdef(`on_badreceive',undefine(`on_badreceive'))
define(on_badreceive, `')

ifdef(`conv_timeout',undefine(`conv_timeout'))
define(conv_timeout, `30')

  switch( C->iclass() )
    {
    case C_ME_START :
      if (msg.msg_class_id() == M_ME_STARTUP)
        {
        ME_Startup sr12(msg);
        wc_startup(C,sr12);
        }
      else
        bad_message = TRUE;
      break;
    case C_ME_SHUT :
      if (msg.msg_class_id() == M_ME_SHUTDOWN)
        {
        ME_Shutdown sr11(msg);
        wc_shutdown(C,sr11);
        }
      else
        bad_message = TRUE;
      break;
    case C_SETUP_RUN :
      if (msg.msg_class_id() == M_SETUP_RUN)
        {
        Setup_Run sr13(msg);
        wc_setup_run(C,sr13);
        }
      else
```

```
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);
}
```

```
        bad_message = TRUE;
      break;
    case C_START_RUN :
      if (msg.msg_class_id() == M_START_RUN)
        {
        Start_Run sr14(msg);
        wc_start_run(C,sr14);
        }
      else
        bad_message = TRUE;
      break;
    case C_DNLD_ME_ :
      if (msg.msg_class_id() == M_DOWNLOAD_ME)
        {
        Download_Me dm11(msg);
        wc_download(C,dm11);
        }
      else
        bad_message = TRUE;
      break;
    case C_UPLD_ME_ :
      if (msg.msg_class_id() == M_UPLOAD_ME)
        {
        Upload_Me um11(msg);
        wc_upload(C,um11);
        }
      else
        bad_message = TRUE;
      break;
    default:
      C->sprint_conversation(line);
      strcat(line,"\nUnexpected Conversation received");
      LOGS("main",line,"%s",warning);
      C->ignore();
      break;
    } /* end of switch */

  if ( bad_message == TRUE )
    {
    sprintf(line,"Unexpected message received '%s' for conversation: '%s'.",
            msg.msg_name(), C->cclass() );
    LOGS("main",line,"%s",warning);
    C->ignore();
    }
  exec sql commit;
  delete C;
  } /* end of while */
} /* end of main */


/*****************************************************************************
 *
 *                 WC_STARTUP
 *
 *****************************************************************************/
void wc_startup(Conversation *C, ME_Startup& M)
{
#ifdef clean_return
#undef clean_return
#endif
```

```
#define clean_return { \
   delete mach_id; \
   delete CONV1; \
   delete CONV2; \
   return; \
   }

int ack = ACK_OK;
int err = 0;

Message msg1;
Conversation *CONV1;
CONV1 = new Conversation;
CONV1->set_group(busname);

Message msg2;
Conversation *CONV2;
CONV2 = new Conversation;
CONV2->set_group(busname);


ifdef(`err_return',undefine(`err_return'))
define(err_return,
`send_alarm(busname,
           "MESTRT_FAIL",
           "MACHINE",
           "",
            machine_id,
           "See Error Log",
           "now");
 clean_return')

char *mach_id = (char *)M.machine_id().ifnull(CHARNULL);

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0){
   /*
     A startup message was received for the wc_interface
     wc checks for its status, if stauts is not Shutdown then
     sends back Notnow acknowledgement
   */

   LOGS("wc_startup","WC startup received");

   if (ACK != get_status(machine_id,&status)){
      ME_Startup_Ack mesa1(machine_id,DB_Badrea);
      err = C->send(mesa1);
      if (err != 0){
         sprintf(line,"Error #%d sending ME_Startup_Ack",err);
         LOGS("wc_startup",line,"%s",error);
      }
      clean_return
   }
   if (strncmp(status.me_status,"Shutdown",strlen("Shutdown")) != 0){
      LOGS("wc_startup","Sending Notnow. ME not Shutdown.","%s",warning);
      ME_Startup_Ack mesa2(machine_id,Notnow);
      err = C->send(mesa2);
      if (err != 0){
         sprintf(line,"Error #%d sending ME_Startup_Ack",err);
         LOGS("wc_startup",line,"%s",error);
      }
```

```
      clean_return
   }

   /*
      status is Shutdown so wc sends back the acknowledgement OK
   */
   ME_Startup_Ack mesa3(machine_id,int_with_null(ACK_OK));
   err = C->send(mesa3);
   if (err != 0){
      sprintf(line,"Error #%d sending ME_Startup_Ack",err);
      LOGS("wc_startup",line,"%s",error);
      clean_return
   }

   /*
      Status of WC is Shutdown. Send ME_Startup
      message to the machine and wait for ack.
   */
   if (ACK != send_listen(msgid,ME_START," "," ")){
      sprintf(line, "ME could not start machine");
      LOGS("wc_startup",line,"%s",warning);
      err_return
   }
   err = no_interest(busname);
   if (err != 0){
      sprintf(line,
      "Error #%d while setting conversation filter to no_intrest",err);
      LOGS("wc_startup",line,"%s",error);
      err_return
   }
   err = interest(busname,clist);
   if (err != 0){
      sprintf(line,
       "Error #%d while setting conversation filter",err);
      LOGS("wc_startup",line,"%s",error);
      err = interest(busname,startup);
      if (err != 0){
         sprintf(line,"Error #%d while setting conversation filter",err);
         LOGS("wc_startup",line,"%s",error);
         }
      err_return
   }

ifdef(`err_return',undefine(`err_return'))
define(err_return,
`err = send_alarm(busname,
                  "MESTRT_FAIL",
                  "MACHINE",
                  "",
                   machine_id,
                  "See Error Log",
                  "now");
 if (err == 0){
   clean_return
 }
 err = no_interest(busname);
 if (err != 0){
    sprintf(line,
    "Error #%d while setting conversation filter to no_intrest",err);
    LOGS("wc_startup",line,"%s",error);
```

```
    clean_return
 }
 err = interest(busname,startup);
 if (err != 0){
    sprintf(line,
    "Error #%d while setting conversation filter",err);
    LOGS("wc_startup",line,"%s",error);
 }
 clean_return')

   if (ACK != send_accept_types(machine_id,busname,ALLOPS)){
      err_return
   }
   strcpy(status.recipe_id,CHARNULL);
   strcpy(status.start_time,TIMENULL);
   strcpy(status.end_time,TIMENULL);
   strcpy(status.me_status,"Idle");
   status.machine_run_id = INTNULL;
   status.note_id = INTNULL;

   if (ACK != change_status(busname,&status)){
      sprintf(line, "Cannot change the status to %s",status.me_status);
      LOGS("wc_startup",line,"%s",warning);
      err_return
   }
   ME_Ready merd(machine_id);
   err = CONV1->send(merd);
   if (err != 0){
      sprintf(line,"Error #%d sending ME_Ready",err);
      LOGS("wc_startup",line,"%s",error);
      clean_return
   }
}
else
   C->ignore();
clean_return
} /* end of wc_startup */


/*************************************************************************
 *
 *                WC_SHUTDOWN
 *
 *************************************************************************/
void wc_shutdown(Conversation *C, ME_Shutdown& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      delete mach_id; \
      return; \
      }
int ack = ACK_OK;
int err = 0;

char *mach_id = (char *)M.machine_id().ifnull(CHARNULL);

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0){
   /*
```

```
      A shutdown message was received for the wc_interface.
      First WC will set the accept types to accept no more operations.
      Then it will check its status: if it is "Idle" it will
      shutdown immediately. if it is "Setup" it will set its state to
      "Setup_Shut" and will shutdown when the run is complete.  If
      it is "Running" it will set its state to "Run_Shut" and
      will shutdown when the run is complete.
   */
   LOGS("wc_shutdown","WC shutdown received");

   ME_Shutdown_Ack mesh(machine_id,int_with_null(ACK_OK));
   err = C->send(mesh);
   if (err != 0) {
      sprintf(line,"Error #%d sending ME_Shutdown_Ack",err);
      LOGS("wc_shutdown",line,"%s",error);
      clean_return
   }
   if (ACK != send_accept_types(machine_id, busname, NONE)){
      clean_return
   }
   if (ACK != get_status(machine_id,&status)){
      ack = DB_Badrea;
      sprintf(line,"Cannot read the current status");
      LOGS("wc_shutdown",line,"%s",warning);
      clean_return
   }
   if (strncmp(status.me_status,"Idle",strlen("Idle")) == 0){
      shutdown_machine();
      clean_return
   }
   if (strncmp(status.me_status,"Setup",strlen("Setup")) == 0){
      strcpy(status.me_status,"Setup_Shut");
      if (ACK != change_status(busname,&status)){
         sprintf(line, "Cannot change the status to %s",status.me_status);
         LOGS("wc_shutdown",line,"%s",warning);
      }
      clean_return
   }
   if (strncmp(status.me_status,"Running",strlen("Running")) == 0){
      strcpy(status.me_status,"Run_Shut");
      if (ACK != change_status(busname,&status)){
         sprintf(line, "Cannot change the status to %s",status.me_status);
         LOGS("wc_shutdown",line,"%s",warning);
      }
      clean_return
   }
}
else
   C->ignore();
clean_return
} /* end of wc_shutdown */

/*****************************************************************
 *
 *                  SHUTDOWN_MACHINE
 *
 *  This will change the status to "Shutdown", change the conversation
 *  filter to accept only a startup and then send out the ME_SHUTDOWN_COMPLETE
 *  conversation.
 *****************************************************************/
```

```
void shutdown_machine()
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      delete CONV1; \
      return; \
      }

int err = 0;

Conversation *CONV1;
CONV1 = new Conversation;
CONV1->set_group(busname);

LOGS("shutdown_machine","WC will shutdown the machine");

strcpy(status.recipe_id, CHARNULL);
strcpy(status.start_time,TIMENULL);
strcpy(status.end_time,TIMENULL);
strcpy(status.me_status,"Shutdown");
status.machine_run_id = INTNULL;
status.note_id = INTNULL;

if (ACK != change_status(busname,&status)){
   sprintf(line, "Cannot change the status to %s",status.me_status);
   LOGS("shutdown_machine",line,"%s",warning);
   clean_return
}
err = no_interest(busname);
if (err != 0){
   sprintf(line,
   "Error #%d while setting conversation filter to no_interest",err);
   LOGS("shutdown_machine",line,"%s",error);
   clean_return
}
err = interest(busname,startup);
if (err != 0){
   sprintf(line,"Error #%d while setting conversation filter to startup",err);
   LOGS("shutdown_machine",line,"%s",error);
   clean_return
}
ME_Stopped mest(machine_id);
err = CONV1->send(mest);
if (err != 0){
   sprintf(line,"Error #%d sending ME_Stopped",err);
   LOGS("shutdown_machine",line,"%s",error);
   clean_return
}
clean_return
}/* end of shutdown_machine */

/*****************************************************************
 *
 *                WC_SETUP_RUN
 *
 *****************************************************************/
void wc_setup_run(Conversation *C, Setup_Run& M)
{
```

```
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      delete mach_id; \
      delete op_type; \
      delete recipe_id; \
      delete filename; \
      return; \
      }

int ack = ACK_OK;
int err = 0;

exec sql begin declare section;
   char *mach_id;
   char *op_type;
   char *recipe_id;
   char *filename;
   int  row_count;
exec sql end declare section;

mach_id   = (char *)M.machine_id().ifnull(CHARNULL);
op_type   = (char *)M.op_type().ifnull(CHARNULL);
recipe_id = (char *)M.recipe_id().ifnull(CHARNULL);
filename  = (char *)M.filename().ifnull(CHARNULL);


ifdef(`on_badrows',undefine(`on_badrows'))
define(on_badrows,
`Setup_Run_Ack srna1(machine_id, Rec_Inval);
 err = C->send(srna1);
 if (err != 0){
    sprintf(line,"Error #%d sending Setup_Run_Ack",err);
    LOGS("wc_setup_run",line,"%s",error);
    }')

ifdef(`on_error',undefine(`on_error'))
define(on_error,
`Setup_Run_Ack srna2(machine_id, DB_Badrea);
 err = C->send(srna2);
 if (err != 0){
    sprintf(line,"Error #%d sending Setup_Run_Ack",err);
    LOGS("wc_setup_run",line,"%s",error);
    }')

ifdef(`err_return',undefine(`err_return'))
define(err_return,
`Setup_Run_Ack srna3(machine_id, ack);
 err = C->send(srna3);
 if (err != 0){
    sprintf(line,"Error #%d sending Setup_Run_Ack",err);
    LOGS("wc_setup_run",line,"%s",error);
    }
 clean_return')

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0){
   /*
      A setup_run message was received for the machine.
      WC will check if its status is "Idle", download the recipe,
```

```
   then send back the acknowledgement
*/
LOGS("wc_setup_run","setup_run received");

if (ACK != get_status(machine_id,&status)){
   ack = DB_Badrea;
   sprintf(line,"Cannot read the status");
   LOGS("wc_setup_run",line,"%s",warning);
   err_return
}
if (strncmp(status.me_status,"Idle",strlen("Idle")) != 0){
   ack = Notnow;
   sprintf(line,"Machine not Idle, try later.");
   LOGS("wc_setup_run",line,"%s",warning);
   err_return
}
/*
   Check if the recipe_id and op_type are valid for the machine
*/

saveto(sp_oprec);
exec sql select count(*)
       into :row_count
       from OP_MACHINE_RECIPE
      where machine_id = :mach_id and
            recipe_id = :recipe_id and
              op_type = :op_type;
check_and_recover_to("wc_setup_run",sp_oprec,
"An ingres error occured while reading OP_MACHINE_RECIPE",CHECK_ONEROW_NOLOG)
exec sql commit;

/*
  Now setup the run by telling the me_interface to download the recipe
  to the machine
*/
ack = Start_Fai;
if (ACK != send_listen(msgid,SETUP_RUN,filename,recipe_id)){
   sprintf(line,"Cannot setup the run on the machine.");
   LOGS("wc_setup_run",line,"%s",warning);
   err_return
}

/*
   Change the accept types to none
*/
ack = DB_Badupd;
if (ACK != send_accept_types(machine_id,busname,NONE)){
   err_return
}

/*
  change the status to "Setup"
*/
strcpy(status.recipe_id, recipe_id);
strcpy(status.start_time,TIMENULL);
strcpy(status.end_time,TIMENULL);
strcpy(status.me_status,"Setup");
status.machine_run_id = M.machine_run_id().ifnull(INTNULL);

if (ACK != change_status(busname,&status)){
```

```
      sprintf(line, "Cannot change the status to %s",status.me_status);
      LOGS("wc_setup_run",line,"%s",warning),
      err_return
   }
   /*
      machine is setup for run so send setup_run_ack
   */
   Setup_Run_Ack srna4(machine_id,int_with_null(ACK_OK));
   err = C->send(srna4);
   if (err != 0) {
      sprintf(line,"Error #%d sending Setup_Run_Ack",err);
      LOGS("wc_setup_run",line,"%s",error);
      clean_return
   }
}
else
   C->ignore();
clean_return
} /* end of wc_setup_run */

/*****************************************************************************
 *
 *                WC_START_RUN
 *
 *****************************************************************************/
void wc_start_run(Conversation *C, Start_Run& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   delete mach_id; \
   delete recipe_id; \
   delete filename; \
   return; \
   }

int ack = ACK_OK;
int err = 0;

char *mach_id   = (char *)M.machine_id().ifnull(CHARNULL);
char *recipe_id = (char *)M.recipe_id().ifnull(CHARNULL);
char *filename  = (char *)M.filename().ifnull(CHARNULL);

ifdef(`err_return',undefine(`err_return'))
define(err_return,
`Start_Run_Ack strna1(machine_id,Start_Fai);
 err = C->send(strna1);
 if (err != 0){
    sprintf(line,"Error #%d sending Start_Run_Ack",err);
    LOGS("wc_start_run",line,"%s",error);
 }
 clean_return')

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0){
   /*
     A start_run message was received for the wc_interface
     wc send back the acknowledgement and sets the filters to
     receive all of the messages.
   */
```

```
   LOGS("wc_start_run","start_run received");

   if (ACK != get_status(machine_id,&status)){
      ack = DB_Badrea;
      sprintf(line,"Cannot read the status");
      LOGS("wc_start_run",line,"%s",warning);
      err_return
   }
   if ((strncmp(status.me_status,"Setup",strlen("Setup")) != 0) &&
      (strncmp(status.me_status,"Setup_Shut",strlen("Setup_Shut")) !=0)){
      ack = Notnow;
      sprintf(line,"Machine not setup for run, try later.");
      LOGS("wc_start_run",line,"%s",warning);
      err_return
   }

   /*
         Status of WC is setup or setup_shut so send start_run
         message to me and wait for ack.
   */
   if (ACK != send_listen(msgid,START_RUN, filename, recipe_id)){
      ack = Start_Fai;
      sprintf(line, "ME could not start machine");
      LOGS("wc_start_run",line,"%s",warning);
      err_return
   }
   /*
        change the status to "Running" or "Run_Shut"
   */
   strcpy(status.start_time,"now");
   strcpy(status.end_time,TIMENULL);
   if (strncmp(status.me_status,"Setup",strlen("Setup")) == 0)
       strcpy(status.me_status,"Running");
   else
       strcpy(status.me_status,"Run_Shut");
   if (ACK != change_status(busname,&status)){
      sprintf(line, "Cannot change the status to %s",status.me_status);
      LOGS("wc_start_run",line,"%s",warning);
      err_return
   }
   Start_Run_Ack strna2(machine_id,int_with_null(ACK_OK));
   err = C->send(strna2);
   if (err != 0) {
      sprintf(line,"Error #%d sending Start_Run_Ack",err);
      LOGS("wc_start_run",line,"%s",error);
      clean_return
   }
}
else
   C->ignore();
clean_return
} /* end of wc_start_run */

/*****************************************************************************
 *
 *                WC_UPLOAD
 *
 *****************************************************************************/
void wc_upload(Conversation *C, Upload_Me& M)
{
```

```
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
      delete mach_id; \
      delete recipe_id; \
      delete filename; \
      return; \
      }

int ack = ACK_OK;
int err = 0;

char *mach_id   = (char *)M.machine_id().ifnull(CHARNULL);
char *recipe_id = (char *)M.recipe_id().ifnull(CHARNULL);
char *filename  = (char *)M.filename().ifnull(CHARNULL);

ifdef(`err_return',undefine(`err_return'))
define(err_return,
`Upload_Ack uplda1(ack);
 err = C->send(uplda1);
 if (err != 0){
    sprintf(line,"Error #%d sending Upload_Ack",err);
    LOGS("wc_upload",line,"%s",error);
 }
 clean_return')

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0){
   /*
      An upload_me message was received for the wc_interface
      wc will try to upload the recipe if the me_status is "Idle"
      then send back the acknowledgement.
   */
   LOGS("wc_upload","Upload received");

   if (ACK != get_status(machine_id,&status)){
      ack = DB_Badrea;
      sprintf(line,"Cannot read the status");
      LOGS("wc_upload",line,"%s",warning);
      err_return
   }
   if (strncmp(status.me_status,"Idle",strlen("Idle")) != 0){
      ack = Notnow;
      sprintf(line,"Machine not Idle, try later.");
      LOGS("wc_upload",line,"%s",warning);
      err_return
   }
   /*
         Status of the machine is "Idle", so it can upload the recipe.
         WC will send the upload message to ME and wait for ack.
   */
   if (ACK != send_listen(msgid,UPLD_ME_,filename,recipe_id)){
      ack = Notnow;
      sprintf(line,"Cannot upload recipe from the machine");
      LOGS("wc_upload",line,"%s",warning);
      err_return
   }
   Upload_Ack uplda2(int_with_null(ACK_OK));
   err = C->send(uplda2);
   if (err != 0) {
```

```
      sprintf(line,"Error #%d sending Upload_Me_Ack",err);
      LOGS("wc_upload",line,"%s",error);
      clean_return
   }
}
else
   C->ignore();
clean_return
} /* end of wc_upload */

/*************************************************************************
 *
 *                WC_DOWNLOAD
 *
 *************************************************************************/
void wc_download(Conversation *C, Download_Me& M)
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   delete mach_id; \
   delete recipe_id; \
   delete filename; \
   return; \
   }

int ack = ACK_OK;
int err = 0;

char *mach_id   = (char *)M.machine_id().ifnull(CHARNULL);
char *recipe_id = (char *)M.recipe_id().ifnull(CHARNULL);
char *filename  = (char *)M.filename().ifnull(CHARNULL);

ifdef(`err_return',undefine(`err_return'))
define(err_return,
`Download_Ack dnlda(ack);
 err = C->send(dnlda);
 if (err != 0){
    sprintf(line,"Error #%d sending Download_Ack",err);
    LOGS("wc_download",line,"%s",error);
 }
 clean_return')

if (strncmp(machine_id,mach_id,strlen(machine_id)) == 0){
   /*
     A download_me message was received for the wc_interface
     wc send download message to me
   */
   LOGS("wc_download","Download received");

   if (ACK != get_status(machine_id,&status)){
      ack = DB_Badrea;
      sprintf(line,"Cannot read the status");
      LOGS("wc_download",line,"%s",warning);
      err_return
   }
   if (strncmp(status.me_status,"Idle",strlen("Idle")) != 0){
      ack = Notnow,
      sprintf(line,"Machine not Idle, try later.");
```

```
      LOGS("wc_download",line,"%s",warning);
      err_return
   }
   /*
      Status of the machine is "Idle" so it can download the recipe.
      WC will send a message to ME and wait for ack.
   */
   ack = Notnow;
   if (ACK != send_listen(msgid,DNLD_ME_,filename,recipe_id)){
      sprintf(line,"Cannot download recipe to the machine");
      LOGS("wc_download",line,"%s",warning);
      err_return
   }
   Download_Ack dnlda2(int_with_null(ACK_OK));
   err = C->send(dnlda2);
   if (err != 0) {
      sprintf(line,"Error #%d sending Download_Ack",err);
      LOGS("wc_download",line,"%s",error);
      clean_return
   }
}
else
   C->ignore();
clean_return
} /* end of wc_download */

/*********************************************************

                     RUN_COMP_ACTION

*********************************************************/
void run_comp_action()
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
   delete COMP; \
   return; \
   }

int ack = ACK_OK;
int err = 0;

FILE  *result_fp;

exec sql begin declare section;
   char parm[13];
   char value[26];
   int  machine_run_id;
exec sql end declare section;

Conversation *COMP;
COMP = new Conversation;
COMP->set_group(busname);

ifdef(`on_error',undefine(`on_error'))
define(on_error,
`sprintf(line,"Error inserting into MACHINE_RUN_RESULTS");
 LOGS("run_comp_action",line);
```

```
 fclose(result_fp);
 Run_Complete rncom(machine_run_id);
 err = COMP->send(rncom);
 if (err != 0){
    sprintf(line,"Error #%d sending Run_Complete",err);
    LOGS("run_comp_action",line,"%s",error);
 }')

ifdef(`on_badrows',,undefine(`on_badrows'))
define(on_badrows, ')

LOGS("run_comp_action","WC sees the end of a run.");

/*
     First get the status and update the end time
*/
if (ACK != get_status(machine_id,&status))
   {
   ack = DB_Badrea;
   sprintf(line,"Cannot read the status");
   LOGS("run_comp_action",line,"%s",warning);
   clean_return
   }

strcpy(status.end_time,"now");

if (ACK != change_status(busname,&status))
   {
   sprintf(line,"Cannot update the end time to %s",status.end_time);
   LOGS("run_complete_action",line,"%s",warning);
   clean_return
   }

machine_run_id = status.machine_run_id;

/* now get the results and put them into the database */
if ((result_fp = fopen(wc_me_message.file_name,"r")) == NULL)
   {
   sprintf(line,"Error in opening result file %s.",wc_me_message.file_name);
   LOGS("run_comp_action",line,"%s",error);
   }
else
   { /* get the results from the file and put into the database */
   saveto(sp_results)
   while (fscanf(result_fp,"%s",parm) != EOF)
      {
      fscanf(result_fp,"%s",value);
      sprintf(line,"machine result read - value = %s. parm = %s",value,parm);
      LOGS("run_comp_action",line);
      exec sql repeated insert
          into MACHINE_RUN_RESULT
                 (machine_run_id, param_id, value)
           values(:machine_run_id, :parm, :value);
      check_and_recover_to("run_comp_action",sp_results,
         "An ingres error occured while inserting MACHINE_RUN_RESULT",
          DONT_CHECK)
      }
   exec sql commit,
   /* close and delete result file */
   fclose(result_fp);
```

```
    if (unlink(wc_me_message.file_name) < 0)
      perror("unlink");
    }

Run_Complete rncom1(machine_run_id);
err = COMP->send(rncom1);
if (err != 0)
    {
    sprintf(line,"Error #%d sending Run_Complete",err);
    LOGS("run_comp_action",line,"%s",error);
    clean_return
    }
if ((strncmp(status.me_status,"Setup_Shut",strlen("Setup_Shut")) == 0) ||
    (strncmp(status.me_status,"Run_Shut",strlen("Run_Shut")) ==0))
    {
    shutdown_machine();
    clean_return
    }
strcpy(status.recipe_id,CHARNULL);
strcpy(status.start_time,TIMENULL);
strcpy(status.end_time,TIMENULL);
strcpy(status.me_status,"Idle");
status.machine_run_id = INTNULL;
status.note_id = INTNULL;

if (ACK != change_status(busname,&status)){
    sprintf(line,"Cannot change status to %s",status.me_status);
    LOGS("run_complete_action",line,"%s",warning);
    clean_return
}
if (ACK != send_accept_types(machine_id,busname,ALLOPS)){
    clean_return
}
clean_return
}/* end of run_comp_action */

/*******************************************************

                    ALARM_ACTION

*******************************************************/
void alarm_action()
{
#ifdef clean_return
#undef clean_return
#endif
#define clean_return { \
    return; \
}

int err = 0;

exec sql begin declare section;
   char alarm_id[13];
   char alarm[81];
   char run_id[16],
   char parameters[129];
exec sql end declare section,
```

```
ifdef(`on_error',undefine(`on_error'))
define(on_error, ')

ifdef(`on_badrows',undefine(`on_badrows'))
define(on_badrows,
 `sprintf(line, "Alarm from machine not recognized.\n%s.",alarm);
 LOGS("alarm_action",line);
 exec sql commit;')

LOGS("alarm_action","WC sees an alarm");

if (status.machine_run_id == INTNULL)
   strcpy(run_id,CHARNULL);
else
   sprintf(run_id,"%d",status.machine_run_id);

strncpy(alarm,wc_me_message.alarm_text,sizeof(alarm));

saveto(sp_alarm);
exec sql select alarm_id, description
              into :alarm_id, :parameters
              from ALARM_DESCRIPTION
              where robot_alarm = :alarm;
check_and_recover_to("alarm_action",sp_alarm,
   "An ingres error occured while reading ALARM_DESCRIPTION",CHECK_ONEROW)
exec sql commit;

send_alarm(busname,
           alarm_id,
           "MACHINE",
           run_id,
           machine_id,
           parameters,
           "now");
clean_return
}

void listen_me(int msgid)
{
/* look for message from the me_interface */

if (mq_read_nowait(msgid,FROM_ME,(char *)&wc_me_message) == IS_MESSAGE)
  {
  if (wc_me_message.message_id == NO_MESSAGE)
    return;
  if (wc_me_message.message_id == RUN_COMPL)
    {
    run_comp_action();
    wc_me_message.message_id = NO_MESSAGE;
    return;
    }
  if (wc_me_message.message_id == CELL_ALAR)
    {
    alarm_action(),
    wc_me_message.message_id = NO_MESSAGE;
    return;
    }
  }
}
```

11.6 Listing 7, Dummy ME_INTERFACE

```
/*****************************************************
*                                                    *
*     MACHINE ENITITY PROGRAMM FOR A DUMMY MACHINE   *
*                      "me_dummy?"                   *
*****************************************************/

#include <stdio.h>
#include <unistd.h>
#include <strings.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "me_messages.h"
#include "shared_mem_struct.h"
#include "logs.h"

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

//
// prototypes
//
extern int init_mq(char*);
extern int mq_read_nowait(int msgid, long msg_type, char *msg_text);
extern void LOGS(char*,char*,char* = "%s",LOG_MSG_TYPE = note);
extern int mq_send(int msgid,long msg_type,char *msg_text);

extern "C" int fork();
// extern "C" void execl(const char*,const char*,DOTDOTDOT);

WC_ME_BUFFER wc_me_message;

main()
{
int msgid;
char process[20];
char msgidst[20];

strcpy(process,"wcn");

//
//  INITIALIZE MSG QUEUE
//
msgid = init_mq((char *)&process);

while (TRUE)
{
  while (mq_read_nowait(msgid,TO_ME,(char *)&wc_me_message) == IS_NOMESSAGE)
  {
      sleep(2);
  }

  switch(wc_me_message.message_id)
  {
    case ME_START :
      sleep(2);
      wc_me_message.ack_code = ACK;
      wc_me_message.message_id = ME_START;
      break;
```

```
    case SETUP_RUN :
      sleep(2);
      wc_me_message.ack_code = ACK;
      wc_me_message.message_id = SETUP_RUN;
      break;
    case START_RUN :
      signal(SIGCLD,SIG_IGN);  /* ignore the death of the child */
      switch(fork())
        {
        case -1:
          perror("Cant create a new process.\n");
          LOGS(process,"fork failed...","%s",error);
          wc_me_message.ack_code = NACK;
          wc_me_message.message_id = START_RUN;
          break;
        case 0:       // child process
          sprintf(msgidst,"%d",msgid);
          execl(wc_me_message.file_name,wc_me_message.file_name,msgidst,
                process,(char*)NULL);
          LOGS(process,"exec failed...","%s",error);
          break;
        default:
          wc_me_message.ack_code = ACK;
          wc_me_message.message_id = START_RUN;
          break;
        }
      break;
    case DNLD_ME_ :
      sleep(2);
      wc_me_message.ack_code = ACK;
      wc_me_message.message_id = DNLD_ME_;
      break;
    case UPLD_ME_ :
      sleep(2);
      wc_me_message.ack_code = ACK;
      wc_me_message.message_id = UPLD_ME_;
      break;
    default   :
      sprintf(line,"Unknown message from wc_interface #%d",
                   wc_me_message.message_id);
      LOGS(process,line,"%s",warning);
      break;
    } /* end of switch */
   if (mq_send(msgid,FROM_ME,(char *)&wc_me_message) == ERROR_MSG)
      LOGS(process,"mq_send failed...","%s",error);
  } /* end of while */
}
```

11.7 Listing 8, HI Cancel Request

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

   Purpose: Allows a user to update information about a
   machine request.
*/

initialize(
   name = varchar(16),
   err = integer,
   change = integer,
   reply = char(1),
   on_table = integer,
   m = vchar(80),                  /* message */
   rows = integer,
   p_machine_id = varchar(12),
   p_recipe_id = varchar(20),
   selected_table = char(20),
   datarows = integer,
   p_note_id = integer,
   op_to_move = varchar(12),
/* 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 mach_requests (reverse (sls_data) = 0,
                                    blink (sls_data) = 0);
     sls_data := null;
   elseif err = 1 then
     set_forms field mach_requests (reverse (sls_data) = 1,
                                    blink (sls_data) = 1);
     sls_data := tmp;
   endif;
/* End of Status Line Server */

   inittable key_list read;

   /* % is the ingres wild card i.e. * unix */
   machine_id = '%';
   entity_id  = '%';
   op_type    = '%';
   lot_id     = '%';

   err = callproc ing_err();
   if err != 0 then return NULL;endif;
   commit;
   set_forms frs (timeout = 10);
```

```
   resume field machine_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 mach_requests (reverse (sls_data) = 0,
                                    blink (sls_data) = 0);
     sls_data := null;
   elseif err = 1 then
     set_forms field mach_requests (reverse (sls_data) = 1,
                                    blink (sls_data) = 1);
     sls_data := tmp;
   endif;
   set_forms frs (timeout = 10);
   /* End of Status Line Server */
}

/*
        This section updates the fields on the screen according to
        where the cursor is....
*/

'Do It ', key frskey4 =  {
   set_forms frs(timeout = 0);
     inquire_forms field mach_requests(on_table = table);
     if :on_table = 1 then
        /* send msched a message telling it to cancel the */
        /* request with key request_key                   */
        err = callproc cancel_request(:key_list.request_key);
        mach_requests = select machine_id, entity_id, op_type
           from REQUEST_MACHINES
           where request_key = :key_list.request_key;
     else
      key_list = select lot_id, request_key, machine_id, op_type
      from REQUEST_MACHINES r, LOT_STATUS l
         where l.lot_entity_id = r.entity_id
         and r.machine_id like :machine_id
         and   r.entity_id  like :entity_id
         and   r.op_type    like :op_type
         and   l.lot_id     like :lot_id
         order by request_key;
     endif;
   commit;
   set_forms frs (timeout = 10);
}
/*
   this set the screen up like when you first entered the menu...
*/
'Clear Screen ', key frskey9 =  {
   set_forms frs(timeout = 0);
   /* % is the ingres wild card i.e. * unix */
   machine_id = '%';
   entity_id  = '%';
   op_type    = '%';
```

```
   lot_id     = '%';

   clear field key_list;

   err = callproc ing_err();
   if err != 0 then return NULL;endif;
   commit;
   set_forms frs (timeout = 10);
   resume field machine_id;
}


'Zoom ', key frskey10 =  {
   set_forms frs(timeout = 0);

   inquire_forms field mach_requests(name = name);
   if :name = 'machine_id' then
      machine_id := callframe choices
                (choices.name = name);
   elseif :name = 'op_type' then
      op_type := callframe choices
                (choices.name = name);
   elseif :name = 'entity_id' then
      entity_id := callframe choices
                (choices.name = name);
   elseif :name = 'lot_id' then
      lot_id := callframe choices
                (choices.name = name);
   endif;
   err = callproc ing_err();
   if err != 0 then return NULL;endif;
   set_forms frs (timeout = 10);

      key_list = select lot_id, request_key, machine_id, op_type
      from REQUEST_MACHINES r, LOT_STATUS l
         where l.lot_entity_id = r.entity_id
         and r.machine_id like :machine_id
         and   r.entity_id  like :entity_id
         and   r.op_type    like :op_type
         and   l.lot_id     like :lot_id
         order by request_key;

   resume field machine_id;
}

'Get All Requests ', key frskey8 =  {
   set_forms frs(timeout = 0);
   /* % is the ingres wild card i.e. * unix */
   machine_id = '%';
   entity_id  = '%';
   op_type    = '%';
   lot_id     = '%';

      key_list = select lot_id, request_key, machine_id, op_type
      from REQUEST_MACHINES r, LOT_STATUS l
         where l.lot_entity_id = r.entity_id
         order by request_key;

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

```
   commit;
   set_forms frs (timeout = 10);
   resume field machine_id;
}


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

11.8 Listing 9, RS232

```
/*
   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;
```