Application of manufacturing message specification for flexible manufacturing system control

J.J.L. Neve a,*, A.W.A. Plasschaert b

a Delft University of Technology, Laboratory for Flexible Production Automation, Landbergstraat 3, 2628 CE Delft, The Netherlands
b Eindhoven University of Technology, Pav. C10, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Received 22 September 1995; revised 8 February 1996; accepted 8 February 1996

Abstract

This paper describes the development of the internal communication in a Flexible Manufacturing System (FMS), using Manufacturing Message Specification (MMS) and based on a Manufacturing Automation Protocol (MAP) network. Because the machines, of which the FMS consists, had few possibilities for communication with external devices, effort had to be put into making machine-specific interfaces between the NC machines and the MAP network. The interfaces were implemented by developing applications that translate MMS messages into machine-specific data and vice versa. The results of the project show that MMS can be used for the control of manufacturing machines that were not designed for use in a network for flexible manufacturing. MMS is not yet widely used. Therefore this paper is meant to demonstrate that MMS can easily be put into practice using relatively simple and off-the-shelf equipment.

Keywords: MMS; FMS; MAP; Manufacturing; NC machine

1. Introduction

In order to create a standard for shop-floor communication, a standard for industrial networks called Manufacturing Automation Protocol (MAP) was developed early 1980’s. This standard was developed according to the ISO-OSI (Open Systems Interconnection) seven-layer reference model [1]. For the seventh layer, a special protocol was developed for communication between manufacturing devices, the Manufacturing Message Specification (MMS) [2]. This is a worldwide ISO standard since 1990 but the protocol is still far from widely used. Though a lot of study has been done on MMS (e.g. [4–9]), and quite a few manufacturers offer MMS products [3], very few working implementations exist. To demonstrate the applicability of MMS, this paper describes a project concerning the use of MMS in a working manufacturing environment. It has to be stressed that no new concepts on MMS are developed: the emphasis lies in showing the use of MMS in practice.

In the Laboratory for Flexible Production Automation at Delft University of Technology, a Flexible Manufacturing System (FMS) is being built with MMS used for the internal communication, based on a MAP network. The FMS consists of a lathe, a milling machine, a machining centre and one transport system, with a relatively simple gripper for cylindrical objects (Fig. 1) [10–13]. For more on FMSs in general see [14].
The machines in the FMS are controlled by one main PC (486-type), the supervisor. To connect the machines to the MAP network, interfaces had to be developed. Four 386-type PCs equipped with MAP boards by Concord [15] were used for this, creating a Token-bus carrier-band network. The supervisor PC, which is called the ‘Client’ in the MMS protocol, was also equipped with such a board. The boards use a standard coax cable for data transmission. For a schematic view of the control structure of the FMS, see Fig. 2.

In Section 2 of this paper, the development of the communication between each machine and its PC is described. In Section 3, the communication between the PC and the network, by using the MMS protocol, is explained. Section 4 describes how some of the MMS services were implemented in the interfaces. Finally, Section 5 states the results and in Section 6 the conclusions of this project are drawn. This paper is written assuming that the reader has some knowledge on MMS. For further reading on MMS, we recommend for example [1].

2. Communication between each machine and its PC

The FMS serves two purposes: one is the use for education at the university. With the machines, students are taught how to program NC machines and Programmable Logic Controllers (PLCs). Another purpose is research: the FMS is used to do research on MAP/MMS, production control and planning. Because the machines are being used to learn how to program, they also have to be used stand-alone. To combine the development of an interface to the network and the stand-alone control of each machine, PCs were used. With this PC, the machine can be controlled by an operator, which is called the ‘local mode’, and by the network, which is called the ‘remote mode’.

Operations that have to be performed with the PC are for example downloading a part program, starting and stopping the machine, requesting the available memory and requesting the status of the machine. Since for most machines, communication meant both parallel as well as serial data exchange, an application had to be developed that combined both means of communication, which will be referred to as the ‘communication module’.

2.1. Parallel and serial communication

Communicating with the machines is not easy because most of them were not designed for that
purpose. All four machines of the FMS have a serial (RS-232) interface, which is directly connected to the numerical controller. In most cases, however, this interface can only be used for the transfer of part programs and not for giving commands to the machine. In the FMS of the FPA laboratory only the lathe could be completely controlled by means of this serial interface. For the other three machines, another way to control the machine had to be found. These machines all use an internal PLC to have the machine controlled by the operator. This means that some control buttons (like the start button) are connected to the PLC. The PLC in its turn is connected to the numerical controller, to transfer these signals to the numerical controller itself.

By connecting several inputs of the PLC to the PC and changing the PLC program so that the PLC responds to these inputs, the pressing of control buttons can be emulated with the PC. Also by connecting several outputs of the PLC to the PC, the PLC can send information to the PC. Though usually the machine is not totally controllable from the PLC, most crucial operations can be started from the PLC and thus from the PC.

Now two ways of communicating with the machine are available in order to control the machine from a PC (Fig. 3). Also, by using a PC, the interface is very flexible and can be adapted to each machine, while the user interface can be similar for each machine.

2.2. Status polling

Besides controlling the machines, the supervisor PC must also be aware of the status of the machines. This means that each of the other four PCs must also be capable of receiving input signals from its machine indicating the machine status. This information has to be sent to the supervisor in the form of an appropriate signal. For example when a machine finishes a product, the supervisor should be notified. This can be done in two ways. One way is to use interrupts. This means that when a signal changes, a specific interrupt service routine is being processed. However the use of external interrupts was not possible due to the operating system used on the PCs and the closed character of a PC itself.

Another way is to have a program periodically read the status data, which is called “polling”. This is the only option left and therefore used for the status checking. The problem that occurs is that the communication module, which is in fact a standard communication program using a script language, is dedicated to serial communication only. Also, this script language has minimal system functions. Therefore the procedures for parallel communication can not be included in the mainprogram, but an extra program has to be developed, the parallel module. This program can read and write data from and to the parallel port and place the information in a file. This file is being read by the communication module.

When the communication module needs to retrieve or send data via the parallel port, the parallel module is started. The parallel module reads or writes the information from or to a file. Then the execution of the parallel module is terminated and the communication module continues and reads the data from that file. The parallel module just places data coming from the parallel port in a file and vice-versa. The program merely acts as an interface between the communication module and the parallel port (Fig. 4).

For most machines, a total of eight status signals is sufficient. These signals are:
- No emergency.
- NC program is running.
- The machine has been homed (brought in its initial position).
3. Communication between each PC and the MAP network

Besides the communication module, another application has to be developed. This application called the "MMS module", is required to establish the connection between the network and the communication module (Fig. 5).

Though it seems extra complex to create two applications, this has been done because after some experimenting it appeared to be too troublesome to integrate both the communication procedures and the MMS/MAP procedures in one MS-DOS application. During the development and testing, it seemed better to first create a working communication module and then connect it with the MMS module. Another reason to do this was that with just the communication module, the machine could be controlled stand-alone with the PC by an operator, which was a requirement of the project. In this way the communication module can be adapted to each machine, while the MMS module, a more complex application, can be identical for each machine. This reduces the number of different applications in the project. The consequences of using two modules are that a multi-tasking Operating System for PCs has to be used, and that a way has to be found to exchange data between the two. How this was done is described in Section 3.2 and Section 3.3.

3.1. Development of the MMS module

To create the MMS module, a software package by SISCO Inc., called "MMS-EASE" [16], was used. This package consists of C-procedures that can be used to develop a Server or Client application on a PC, using the MAP boards for transmission of MMS messages. A "demo application" is included which can be used for developing a specific application. This demo program calls the corresponding procedures when an incoming MMS message is received from the network (in MMS terms "Indication") and contains procedures to send a "Response" (a reply to an "Indication") (Fig. 6).

The Indication procedures simply put all the received data on the screen: the user-specific actions...
have to be programmed by the user itself. Also when a Confirmation (the Response that is received by the supervisor) is received, just the data is shown. What is done with that data by the receiving MMS party has to be included in the software by the user (Fig. 6).

What has to be done is to somehow link the communication module with these MMS procedures so that when an Indication is received the communication module can be used to send a message to the machine. First for each machine a list can be made of the MMS messages that can be implemented for that specific machine. Some can easily be "mapped" to the machine (e.g. the Program Invocation services): others cannot be implemented because they are not applicable to that specific machine. After mapping the MMS messages to the machines, a way has to be found to send messages to the machine.

3.2. Execution of an MMS instruction by the communication module

On the supervisor PC, OS/2 (from IBM) has been installed as Operating System. For the Operating System for the other PCs (Servers), the best choice at that moment was the "Desqview" multitasking Operating System (from Quarterdeck). With this Operating System, there is no possibility for exchanging data between two applications. Therefore data has to be exchanged by using additional files or memory so that one application could write the data while the other could read this data. For the execution of an MMS-instruction by the communication module, additional files were used, while for the status processing (see Section 3.3, the Random Access Memory (RAM) of the PC was used. In the future the method of executing instructions will be changed so that all communication between the two applications will take place using the RAM. This is to avoid the frequent use of the same files.

Whenever the MMS module needs to send an instruction to the communication module, it puts the instruction in an instruction-file and places the invoke-ID (every MMS message has a specific number) in a small file. This file is checked constantly by the communication module. When it contains a number higher than that of the last executed instruction, the instruction-file is read by the communication module and the instruction is executed. When the communication module has finished the execution, the invoke-ID is written to another file. When the MMS module reads this number, it knows that the communication module has executed the instruction.

3.3. Status processing in the MMS module

Besides giving instructions to the communication module, the MMS module should receive information from the communication module regarding the status of the machine. This information is more time-critical than the instructions, because any change in the status of the machine should be known to the FMS control almost immediately and therefore to the network and the MMS module. In the MMS protocol, "Variables" exist, that represent a certain condition of the machine. In MMS-EASE, Variables can be created by defining a memory address of the PCs memory in which the value of the Variable is stored. To exchange the data between the communication module and the MMS module, the communication module would have to write the status data to addresses that can be read by the MMS module.

In MMS-EASE the procedures for creating Variables were already available. To write the values of the status data (eight in total for most machines) the parallel module has been extended with a procedure that writes the data directly to the memory. An advantage of this method is that the information is directly known to MMS, instead of via the communication module, so that it travels the shortest route from the parallel port to the MMS module.

For this method a small amount of the Random Access Memory of the PC is used that is not used by any other program used on the PC. The addresses of this specific memory are "absolute" which means that they can be referred to by both modules. Now
the status of the machine can be obtained by simply requesting the value of the defined Variables at the Server. The status data is updated continuously by the parallel module.

4. Implementation of some MMS services

There are several groups of MMS services. Each group covers a different aspect of MMS. These groups are (for more on these services see [1]):
- Context Management services
- Virtual Manufacturing Device Management services
- Domain Management services
- Program Invocation Management services
- Variable Access and Management services
- Semaphore Management services
- Event and Alarm Management services
- Journal Access and Management services
- Operator Communication services
- File Access and Management services

In this section, the implementation of three groups of services is described: Domain Management, Event Management and Program Invocation Management. These services required most of the work on MMS during this project.

Because MMS can be used in rather different fields, for example for robots, NC devices and PLCs, several MMS standards exist. These standards, the "Companion Standards", are dedicated to one specific application area. For this project, the "Companion Standard for Numerical Control" (part 4 of [2]) was used. For the implementation of the services, this Companion Standard was used where possible.

4.1. Domain Management services

With Domain Management services, data can be stored on MMS Servers and Clients and data can be transferred from one machine to the other. For the implementation of the Domain Management services, first the definition of a "Domain" has to be examined. A Domain means a set of data, regardless of what the data is used for. This can be for example a part program, or data regarding tool wear. In the FMS at the Laboratory for Flexible Production Automation, a Domain will usually be a part program, that has to be loaded into the machine. Because a machine in the FMS has a PC attached to it, which

Option 1

Domain Download procedure is used. A Domain is considered "loaded" when its contents are stored on the Server's harddisk. The data (program) is loaded in the machine when a program is started.

Option 2

Domain Download procedure is used, but the contents are directly stored in the machine itself. Similar to the procedure when the a MAP-board is built into the machine.

Option 3

Obtain File and Load Domain Content procedures are being used. The Domain data is copied to the Server's harddisk, then the Domain is "loaded" by loading the data in the machine.

Fig. 7 Options for the Domain Download procedure.
can also serve to store data, the question is when a Domain can be regarded as "loaded". The data can be stored on the hard disk of the PC and in the memory of the NC machine. To implement a consistent procedure for downloading a Domain, there are three possibilities (see Fig. 7).

In option 1 the PC and the NC machine are seen as a whole. The hard disk of the PC is considered as the central storage of data. Only when a program is started the necessary data is loaded in the memory of the machine. An advantage of using the hard disk as central storage instead of the machine memory, is the relatively large capacity. A disadvantage is the slowing down of the "Start Program Invocation" service because of the extra loading operation.

Option 2 uses the same MMS service, but directly loads the data into the machine. Therefore the hard disk is not used, and all the Domains that are loaded are stored in the machine’s memory. Because the machine’s memory is limited compared to the PC’s memory, only a limited number of programs can be stored. This is not very efficient because at times a program will have to be erased when the memory is full and reloaded when it is needed again.

In option 3 the Download sequence is not used, instead two services are used for loading a Domain. First the data for a Domain is transferred from the Client to the Server with the service "Obtain File". Then a Domain is loaded in the machine with the service "Load Domain Content". An advantage of this method is that the data for the Domains is stored at the lowest possible level without using the smaller memory of the NC machine. Also the Domain data is loaded into the machine before the program is started. When a program is started, it is already present in the machine’s memory, and so the response time is short.

Of these options, option 3 is the one with the most advantages compared to the others. These are: fast response to the "Start" request (program is already in machine memory) and the use of the Server’s hard disk for storage of data, ensuring short loading-times combined with a large storage capacity. Therefore this procedure has been implemented in the FMS.

4.2. Event Management services

The Event Management services are a very important and complex part of the MMS protocol. Basically, they enable a Server to notify other MMS parties of the occurrence of a certain Event at that Server. Therefore with these services, the NC machine (Server) can initiate communication with the supervisor (Client) instead of just waiting for commands from it. MMS Events are sometimes also considered to be inadequate for handling time critical events [7,8]. Therefore proposals have been made to extend the MMS standard in order to address these problems. Castori has been working on MMS Events [4–6] and has developed an extension to the standard for detection of "non Boolean" Events [6].

In an MMS Server application, an Event Monitor should be installed that detects the change of Variables and takes the necessary actions. In MMS-EASE this was not yet the case: an Event Monitor had to be developed. For the implementation, either a flexible or non-flexible Event Monitor can be chosen. A flexible Event Monitor means that the Client can change the monitored Variables at a Server. This is done by the "Define Event Condition/Action/Enrollment" services. A non-flexible monitor means that at the Server only a predefined set of Variables is monitored and that this monitoring can’t be

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value change</th>
<th>Event Notification sent to Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC_running</td>
<td>1 -&gt; 0</td>
<td>&quot;PC_stopped&quot;</td>
</tr>
<tr>
<td>No_emergency</td>
<td>1 -&gt; 0</td>
<td>&quot;EMERGENCY&quot;</td>
</tr>
<tr>
<td></td>
<td>0 -&gt; 1</td>
<td>&quot;Emergency_no_longer_active&quot;</td>
</tr>
<tr>
<td>NC_error</td>
<td>0 -&gt; 1</td>
<td>&quot;NC_error_has_occurred&quot;</td>
</tr>
<tr>
<td>Par_comm_failed</td>
<td>0 -&gt; 1</td>
<td>&quot;Parallel_communication_error&quot;</td>
</tr>
<tr>
<td>PC_control</td>
<td>1 -&gt; 0</td>
<td>&quot;Connection_with_machine_lost&quot;</td>
</tr>
<tr>
<td>Machine_homed</td>
<td>0 -&gt; 1</td>
<td>&quot;The_machine_has_been_homed&quot;</td>
</tr>
<tr>
<td></td>
<td>1 -&gt; 0</td>
<td>&quot;The_machine_has_not_been_homed&quot;</td>
</tr>
</tbody>
</table>

Fig. 8. Implemented Event Conditions in the FMS.
changed. Since in this project, the machines are equipped with very specific interfaces, creating new Variables (i.e. status signals) is not possible because the number of Variables is limited by the hardware (the PC-PLC interface). Therefore the Event Monitor has been implemented as a procedure that monitors a fixed set of Variables. This is sufficient for the experimental stage, later a flexible Event Monitor can be developed.

In the Server application a procedure has been implemented that periodically scans the Variables and sends an Event Notification to the Client for the value changes listed in Fig. 8.

4.3. Program Invocation services

In short, Program Invocation services concern the management of programs that will be executed by NC machines. For example, with the "Start", "Stop" and "Resume" services, the execution of a part program by an NC machine can be started, stopped and resumed. Most of the Program Invocation services are fairly simple to implement. Services such as "Start", "Stop", "Resume" and "Reset" can be translated to machine messages almost directly. The more difficult part is to manage the state of a particular Program Invocation. In MMS, a Program Invocation can have eight different states [1]. Every possible transition, whether caused externally by a Request or internally, by a certain Event at the machine, has to be noticed and stored in the Server's database. For example, when a program has stopped, the state should change from "Running" to "Stopped". How can the Server, and after that the Client, be informed of this change? When the program stops, the status-signal "NC_running" drops from 1 to 0. The parallel module receives this signal and the value of the MMS variable also changes from 1 to 0. Then the Event Monitor notices this change and the state of the program is changed at the Server and simultaneously an Event Notification ("PI_stopped") is sent to the Client, containing the information on the Variable.

The problem is that with some machines, the signal "NC_running" changes to 0 when the program has reached its end. When a "Stop" Request has been received, the "NC_running" can not be used for checking whether the machine really stopped, since the signal could be 1 or 0, depending on the machine. Therefore the MMS module has to rely on the Program's state, according to the MMS database, rather than on the signal "NC_running".

5. Results

For all machines in the FMS, an MMS Server has been implemented. The lathe, milling machine, machining-centre as well as the transport system can be controlled by the supervisor (Client). For the NC machines, eight status signals are constantly monitored by an Event Monitor that has been implemented, which is not yet flexible (only predefined Variables are monitored). For the transport system, the same signals, plus an additional five for the gripper's status are being monitored. Event Notifications have been implemented. For Program Invocation Management, response times are quite satisfying and will be improved in the future by optimizing the communication between both modules. For the lathe, response times are best, because of the complete serial control of the machine. Average response times for starting and stopping are around 3 seconds for this machine, on the others (using also parallel communication) about 5 to 6 seconds. By optimizing the polling procedures, these times will be reduced in the future.

All Domain Management services have been implemented enabling remote loading and deleting of data. During start-up, Domains are automatically created at each Server, corresponding with the Programs that are stored in the machines memory, so that the Domain database at the Server PC reflects the data contained by the machine.

File Access and Management were already covered by the MMS-EASE source-code, so that files can be transferred from and to all machines. Finally, all machines can be controlled stand-alone with the PC, and all MMS Server applications (MMS-modules) are identical and machine-independent. This ensures complete MMS compatibility.
6. Conclusions

During the project described in this paper, a communication network has been developed for the communication in an FMS, consisting mostly of machines that were originally not designed for extensive communication with external devices. In spite of the few possibilities of the machines, it was possible to develop interfaces, so that each machine is now a fully operational MMS Server. The entire FMS can be controlled by one PC, the supervisor (in MMS terms the Client) which communicates with the machines through the use of an MMS application. The hardware is standard: only the applications of the PCs are developed specifically for this purpose. The effort spent on developing the applications was about one man-year. Half of this effort was put into developing the interfaces between the PCs and the NC machines, the other half into implementation of the MMS services with the use of MMS-EASE.

In this project, MMS proved to be a very useful and well-suited protocol for shop floor communication. This fact seems to be proven also by the fact that more and more fieldbus protocols are based on the MMS standard. Most MMS services were implemented such as Domain Management, Program Invocation Management, Variable Access and Management and Event Management. Other services were already available through the use of MMS-EASE.

Because the project is still in an experimental phase, the hard- and software can still be greatly improved. Further improvements can be: the use of faster PCs (now 386s are used), optimization of the communication between the two modules by using a different multi-tasking Operating System and integration of the communication and MMS module into one application.

Acknowledgements

The authors would like to acknowledge J.B.W. Hoebeek, T. Storm and H.R. Willemse for their support during this project.

References


Alain Plasschaert, born in 1970, received his M.Sc. degree in Mechanical Engineering from the Delft University of Technology in 1994. Currently he is engaged in a 2-year advanced postgraduate programme, “Computer Aided Design and Manufacture of Discrete Products” at the Stan Ackermans Institute, Centre for Technological Design, at Eindhoven University of Technology. His research interests include flexible production automation, production planning and control systems in general, and Shop Floor Control in particular.
Jan Neve received his M.Sc. degree in Mechanical Engineering from the Eindhoven University of Technology in 1976. After that he worked four years on the development of NC-software. In 1980 he joined the staff of the laboratory for Flexible Production Automation of the Delft University of Technology, department of Mechanical Engineering and Marine Technology. He specialises in software for production automation. Recently he has been appointed manager of the Subfaculty of Design and Production.