Abstract:
Application services, in particular MMS (Manufacturing Message Specification), are now available in local area networks used in flexible manufacturing systems. The time performance of these services is however not specified. This is a gap for the systems design and management. A behavior model for the services is proposed in this paper. This model may be implemented on top of the application layer to improve the service provided to the user. The parameters adjustment, and the use of the model in a research project, to create a new flexible cell, are briefly presented.

1. INTRODUCTION
In FMSs (Flexible Manufacturing System) local area networks are essential components for the interoperating of different devices. MAP (Manufacturing Automation Protocol) is a specification for standards which will allow this different devices to interoperate with one another. One of the application Services defined is MMS (Manufacturing Message Specification).
This is an application service for messaging functions in MAP networks. The service and protocol are well specified in the ISO Draft International Standard 9506. However, the service time performance and means of monitoring function execution are not defined, so that there is no possibility to manage them. The behavior model that we propose to separate the application process into two parts: Application Module, Control Module.
The two modules are based on a behavior model of the network entity. The organization of the model is shown in the Figure 1.

3. THE BEHAVIOR MODEL
3.1. The model
The network management applied to an application process at the high level imposes a well adapted structure. We propose to separate the application process into two parts:
- Application Module,
- Control Module.
The two modules are based on a behavior model of the network entity. The organization of the model is shown in the Figure 1.
An advantage of this model is that it provides an application layer image of the network entity that will be used by the application process. Note that the use of the model is not a requirement. It is to be perceived as an optional improvement for network entity supervision and control in a client-server structure.

3.2. The external behavior of the model

3.2.1. Introduction

The concept of the behavior model is to promote the structuring of the application process, according to the external behavior description of a service or a function (service of the application layer).

3.2.2. The client-server structure

Although the network views two cooperating users via MMS Services as being equal, the nature of the Services is inherently asymmetric in its behavior. One user plays the "client" role, requesting another device, the "server", to perform some application-specific operation. This is illustrated in the Figure 2, in which the four basic kinds of primitives are shown:
- request,
- indication,
- response,
- confirm.

Not all the primitives are needed for certain "unconfirmed" services, where there is no response information and therefore no response or confirm primitives.

Figure 2: The client-server structure

3.2.3. Modeling

In a client-server structure, the model we propose is entirely incorporated in the client. It takes charge of the primitive request and primitive confirm that concern the client, and includes the timing aspect of the operation.

The user views the external behavior of the model, which returns information upon an invoked order.

There is a difference between the confirmation of a service and the return of the behavior model. The confirmation is only part of the return.

The return also indicates a bad functioning of the primitive exchanges (for example, when the device associated with the server is out of service, in which case, no response is sent to the client, and no confirmation is received).

Figure 3 shows the interactions between the model and the application layer using the (request, confirmation primitives) and between the model and control module of the application process.

3.3. The internal description of the model

To describe our model we use a state graph (GRAFCET). The stages (schematized by numbered squares) represent the states of the model and the transitions (schematized by cross lines +) show the evolution conditions of the model.

The state graph that describes the internal functioning of the model is shown in Figure 4. The model begins with an order from the application process, causing it to invoke a request to a service of the application layer which depends on the nature of the order (order ---> request). The next stage depends on whether the application layer accepts or rejects the request. If the request is rejected (example, due for a bad parameter) the model delivers a return with an appropriate error code. If the request is accepted, the model goes into a waiting state.

Figure 4: The internal description of the model

At the end of the wait state, one of the following may occur:
1. A confirmation with a positive response or a negative response, depending on whether the requested operation was successfully performed or not.
2. A time-out alarm. This indicates that of the time allocated to the in-process request has been exceeded. The advantage of this time-out alarm is to prevent a malfunctioned server or a lengthy process from holding up the client process.

In either case, a code is delivered. The use of time-out in this model is very important, especially when the variety services provided by a LAN requires critical time control.

3.4. The computation methods for the time-out

3.4.1. The parameter of the model

For each type of request (service) we associate a time-out which depends on the execution time of the request and its confirmation. The first value of this time-out is determined by a series of tests [3], and is modified from time to time as the network characteristics (Lan loads, device loads, etc...) change.

A dynamic value that changes with the environment appears to be better.

3.4.2. Learning and adaptation

We can take the average of the recorded time for each request as the average time of a request and its confirmation. The first value of this time-out is determined by a series of tests [3], and is modified from time to time as the network characteristics (Lan loads, device loads, etc...) change.

The above formula has a disadvantage: the more the number of the measured values increases, the less is the variation on the new measures. To overcome this disadvantage, we have to define the average and the standard deviation with reference to the L last sets. The L sets of variation on the new measures. To overcome this

In the precedent formula, we will take:

\[ T_{\text{average}} = \frac{1}{N} \sum_{i=1}^{n} T_i \]  
\[ T_{\text{time-out}} = a \cdot T_{\text{average}} + b \cdot \text{Standard deviation} \quad (1) \]

where: \( a > 1 \) and \( b > 0 \).

In this formula, \( T_{\text{average}} \) represent the internal characteristics of the station (speed, programm version, etc...). The Standard deviation represent the internal context variation (the position of a request in the queue) or the external context variation (number of station on the network, the load, type of devices, etc...).

The above formula has a disadvantage: the more the number of the measured values increases, the less is the variation on the new measures. To overcome this disadvantage, we have to define the average and the standard deviation with reference to the L last sets. The L sets of measured values are used in the evaluation.

In the precedent formula, we will take:

\[ T_{\text{average}} = \frac{1}{N} \sum_{i=1}^{n} \frac{T_i}{L} \]

\[ T_{\text{time-out}} = a \cdot T_{\text{average}} + b \cdot \text{Standard deviation} \quad (1) \]

\[ \text{Standard deviation}^2 = \frac{1}{L-1} \sum_{i=1}^{n} (T_i - \bar{T})^2 \]  

\[ T_{\text{time-out}} = a \cdot T_{\text{average}} + b \cdot \text{Standard deviation} \quad (1) \]

\[ \text{Standard deviation}^2 = \frac{1}{L-1} \sum_{i=1}^{n} (T_i - \bar{T})^2 \]

In order not to have unacceptable values, it is advisable to saturate the time-out value. An example of the time-out computation is shown in Figure 5.

These values were recorded on a platform which is composed of two PCs connected with a CONCORD MAP network. The first twenty requests, shown in Figure 5 (1 to 20), were invoked after the reception of the precedent confirmation. The rest of the requests (21 to 40) were invoked with the queuing possibility of MMS-EASE (configured with a maximum of 5 outstanding requests).

### Table 1

<table>
<thead>
<tr>
<th>N°</th>
<th>Time</th>
<th>Time out</th>
<th>Maximum + 50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101</td>
<td>179</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>116</td>
<td>184</td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>139</td>
<td>197</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>137</td>
<td>207</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>137</td>
<td>217</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>129</td>
<td>226</td>
<td>26</td>
</tr>
<tr>
<td>7</td>
<td>138</td>
<td>234</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>138</td>
<td>195</td>
<td>28</td>
</tr>
<tr>
<td>9</td>
<td>137</td>
<td>190</td>
<td>29</td>
</tr>
<tr>
<td>10</td>
<td>122</td>
<td>187</td>
<td>30</td>
</tr>
<tr>
<td>11</td>
<td>118</td>
<td>188</td>
<td>31</td>
</tr>
<tr>
<td>12</td>
<td>117</td>
<td>188</td>
<td>32</td>
</tr>
<tr>
<td>13</td>
<td>119</td>
<td>186</td>
<td>33</td>
</tr>
<tr>
<td>14</td>
<td>117</td>
<td>181</td>
<td>34</td>
</tr>
<tr>
<td>15</td>
<td>117</td>
<td>176</td>
<td>35</td>
</tr>
<tr>
<td>16</td>
<td>118</td>
<td>169</td>
<td>36</td>
</tr>
<tr>
<td>17</td>
<td>118</td>
<td>157</td>
<td>37</td>
</tr>
<tr>
<td>18</td>
<td>118</td>
<td>154</td>
<td>38</td>
</tr>
<tr>
<td>19</td>
<td>123</td>
<td>157</td>
<td>39</td>
</tr>
<tr>
<td>20</td>
<td>117</td>
<td>157</td>
<td>40</td>
</tr>
</tbody>
</table>

![Figure 5: Recording times and the computed time-out](image)

3.5. Application to MAP and MMS

3.5.1. Introduction

Our development was implemented on the Sisco's MMS-EASE [4] (Manufacturing Message Specification - Embedded Application Service Element) in C language. The proposed programme structure is in relation with this implementation. Nevertheless it can be modified to suit other implementations.
3.5.2. The application protocol MMS-EASE

MMS-EASE is a protocol that meets the requirement for MAP communication between programmable devices on factory floor. It consists of a library of C language function calls and data structures that provide application programmes access to all the services needed for communication utilizing the MMS Protocols.

Each MMS-EASE Request consists of a function call and an associated data structure.

3.5.3. Queue data structures

When the user issues a request, by calling an MMS-EASE mp_xxxx or mv_xxxx function, MMS-EASE provides a control structure for tracking the outstanding request. A pointer to this structure is returned to the MMS-EASE user's program, which can then modify some of the control information, if desired. When a confirmation is received MMS-EASE looks for the pending queue data structure that was used to initiate the request and for which the confirmation is to respond to. An active element of type MMSREQ_PEND is freed up after the confirmation is popped off the queue and the corresponding user function u_xxxx_conf is executed and returned from.

The MMSREQ_PEND type is defined as follows:

typedef struct rqdat
{
  struct
  {
    SHORT pri;
    struct rqdat *prev;
    struct rqdat *next;
  } q;
  SHORT chan;
  SHORT llp;
  USHORT context;
  USHORT op;
  ULONG id;
  SHORT resp_err;
  SHORT cancel_state;
  LONG req_time;
  BYTE *req_info_ptr;
  LONG resp_time;
  BYTE *req_info_ptr;
  BYTE *reqbuf_ptr;
  BYTE *rxbuf_ptr;
  BYTE *rxbuf;
  VOID (*resp_rcvd_fun)();
} cmd-service
;

This variable contains the SUC channel number over which the request was sent and over which the confirmation should be received.

3.5.4. Command service data structure

The following structure, cmd_service, is a member of MMSREQ_PEND structures and contains user specific information regarding the service of request/confirmation. The information in this structure is strictly for the user program.

struct cmd_service
{
  SHORT (*serve_fun)(struct*);
  BYTE *cmd_info;
  LONG cmd_stat;
  SHORT aux_flags[6];
};

cmd_service.(*serve_fun)(); : this is a pointer to the function you wish to be called when the confirmation PDU to a request is first parsed and paired with the MMSREQ_PEND structure.

cmd_info, cmd_stat, aux_flags : these variables can contain any information we wish, and will be the variables needed for the model.

Figure 6 shows an example of queue data structure with its associated structures.

In our case, we use the structure to drive the model. The function pointed by cmd_service.(*serve_fun)() will be called either by the confirmation reception or the time-out occurrence. The variables needed for the model are:

- stage number,
- current time-out value,
- initial time-out value,
- error code.

The initial value of the timeout is adjusted according to the method described in section 3.4. The third and fourth stage of the model provide the readjustment of this value (alive model).

This method applied to transfer file requests cause a problem. The execution time of the request is in relation with
the file size. This problem can be resolved by the identification of this relation:

\[ \text{Time} = f(\text{file size}) \]

the model has then to ask first for the file size (service file directory) before invoking the request.

Queue data structure
MMSREQ_PEND

Information Present?
Pointer to this information

Operation-Specific Data
User Structure
Pointer to a Function
Pointer to a data
Long Integer
Array of short integer

Figure 6: Data Structure.

4. APPLICATION AND CONCLUSION

This study will be implemented in a self-controlled flexible cell, which is a contract with the French Ministry of Research and Technology [5]. This cell consists of several numerical control machine tools, robots for loading and unloading the parts and tools, and several sensors that give information about the states of the machines and the quality of the produced parts.

Two types of networks are used: one is a fieldbus, which interconnects each controller to sensors and actuators. The other is MAP, which interconnects the controllers.

On the application layer of each station of either networks, the behavior model for fault detection is used. Mechanical problems on the machines cause faults which are detected by the fieldbus. There are also faults on MAP because of the computers, programs, or the network itself...

When a fault appears, the system is either stopped in a particular situation, or a lower function mode is entered. The fault is always displayed on a graphic screen that shows an animated image of the cell.

The fault detection by means of a behavior model is so simple and fast that it facilitates management of the user process and the network.

At present, the model is not able to detect all the faults, especially those which produce positive responses but with erroneous data. The model may be completed by testing on the validity domain which must give response to a given service. Work on these is in progress.

5. REFERENCES

[1] MMS
Manufacturing Message Specification
- Services definition ISO DIS 9506/1
- Protocol specification ISO DIS 9506/2

Thèse de l'Université de NANCY I
P. LHOSTE - Decembre 1985

[3] EVALUATION ET MODELISATION DES PERFORMANCES TEMPORELLES DE MESSAGERIES INDUSTRIELLES.
Thèse de l'Université de NANCY I
F. AFILAL - Juillet 1989

[4] MMS-EASE
Manufacturing Message Specification - Embedded Application Service Element
Reference Manual Revision 8 - SISCO 1988

[5] AUTO-CONTROLE DE CELLULES FLEXIBLES D'USINAGE, INTEGRATION DANS UN ENSEMBLE CFAO.
Contrat N° 88 P 0718
Ministère de la Recherche et de la Technologie - Laboratoire d'Automatique et de Commande Numérique de l'Université de NANCY I.