Running Distributed Applications Using MMS-CORBA

T.Ariza and F.R.Rubio
matere@trajano.us.es, rubio@cartuja.us.es
Dept. Ingeniería de Sistemas y Automática.
Escuela Superior de Ingenieros. Univ. de Sevilla. Spain

Abstract **- Nowadays, distributed computing systems are widely used. Developing software for these systems has proven to be a complicated task, due to the heterogeneity of the distributed system components and the underground communication layers. MMS is a suitable language to carry out the communication between devices in a heterogeneous manufacturing system because it provides uniformity in accessing them. CORBA architecture has been used in recent work in order to implement this protocol in an easier way. The most important objects defined in MMS have been adapted to this new architecture (VMD, Domain, Program Invocation, Event, ...). This paper proposes the use of these objects in order to build distributed applications in a flexible manufacturing system.**

I. INTRODUCTION

Distributed systems have obtained great importance in the past few years. They have been introduced into a great diversity of environments but more explicitly in manufacturing systems. Where there were isolated components, there are now integrated components where communication and cooperation between them is permitted, although this advantage is reduced due to the heterogeneity of the devices that can be found in the manufacturing environments. Because of this, the necessity to use a common protocol in order to communicate these interconnected elements arises.

MMS is an application layer protocol that homogenises the use of the devices that compound the system. MMS makes use of this approach to specify the services that a user can invoke to communicate with these devices. On the other hand, distributed object methodologies, where CORBA is framed, provide all the object oriented methodology advantages in distributed systems. CORBA can make the MMS service user application to request the service easier, as it allows to structure the system in objects and at the same time to have these objects distributed along the several components of the system, making the distribution transparent to the user.

The objective of this work is to propose CORBA and MMS as the framework to run distributed applications in manufacturing environments. In the following sections, a summary of the MMS and CORBA features are going to be given, as well as the way to run applications using these facilities.

In the previous work [3,4,5] the proposal carried out in [1] has been extended in order to include the event management and the specification Inter-domain Management: Specification Translation of The Open Group [2] is followed in order to carry out the translation of MMS services from ASN.1 to IDL.

By doing this, access to the most important MMS objects by means of the ORB of CORBA has been achieved. The features of the model that has been followed can be summarised in this way:

- The MMSserver is a CORBA object and its interface corresponds to the MMS services provided by the VMD, it is defined in an IDL specification.
- The confirmed MMS services are declared as synchronous invocation methods.
- The request to send unsolicited information is implemented by means of one-way invocation methods; the MMSserver uses the TrapServer object of the client in order to send this.
- The association management is not implemented due to the fact that the ORB manages them.

This approach is used to communicate clients to the several virtual manufacturing devices by means of the ORB, using the MMS interface.

II. AIM OF THE WORK

With this work the run of distributed applications in manufacturing systems is intended to be made easy. The distributed application is considered as compounded for several tasks carried out in different components of the distributed system. The concurrent execution of various tasks at the same time must be allowed but synchronisation points between them must also be considered.

These various components are modelled as Virtual Manufacturing Devices (VMD) specified in MMS. The task activation is accomplished by using the services

associated to the Program Invocation object contemplated in this standard.

The task activation is carried out as a result of event notifications received from the various VMDs.

A component that is responsible of supervising the distributed application execution is introduced and which has to undertake the responsibility for:

- Building the Program Invocation Objects in the various VMDs. These Program Invocations are monitored. This is a possibility that MMS allows, and which means that an event notification will be sent to the application (specified in the Program Invocation object) when the program execution ends.
- Starting the different tasks in the several VMDs when it is necessary.
- Receiving the event notifications sent by the VMDs.
- Controlling the system state, updating this with the notifications received from the VMDs and ordering the execution of tasks in accordance with the evolution of the system state.

All the communication between this supervisor component and the VMDs is carried out using the ORB of CORBA. The services that the supervisor is able to request from the VMDs are ones that are specified in MMS.

In order to specify the system behaviour, Petri Nets have been used, as they are a specification tool that allow model concurrent actions and synchronisation. Binary Petri Nets have been used due to their simplicity and easy implementation.

On one hand, the start of the execution of a task is modelled as a place in the Petri Net, on the other hand, the event notification is modelled as a transition that is able to be fired when the notification arrives if the input places are marked.

The supervisor component must carry out the suitable upgrading in the Petri Net when an event notification arrives. This means that it must upgrade the state of the transition corresponding to this event notification indicating that the event has happened and so the transition can now be fired. Next, it has to check if the Petri Net is able to evolve after the change in the state of the transition. In order to do this, it verifies if the transition is enabled. If this is the case, the firing of it is carried out, and as a result, the task associated to the output places are started.

In short, the supervisor controls the evolution of the Petri Net, firing transitions when the events happen and starting the tasks associated to the places.

III. SYSTEM DESCRIPTION

The scheme of the system architecture is shown in figure 1. It shows the several components that are necessary on both the client and server side.

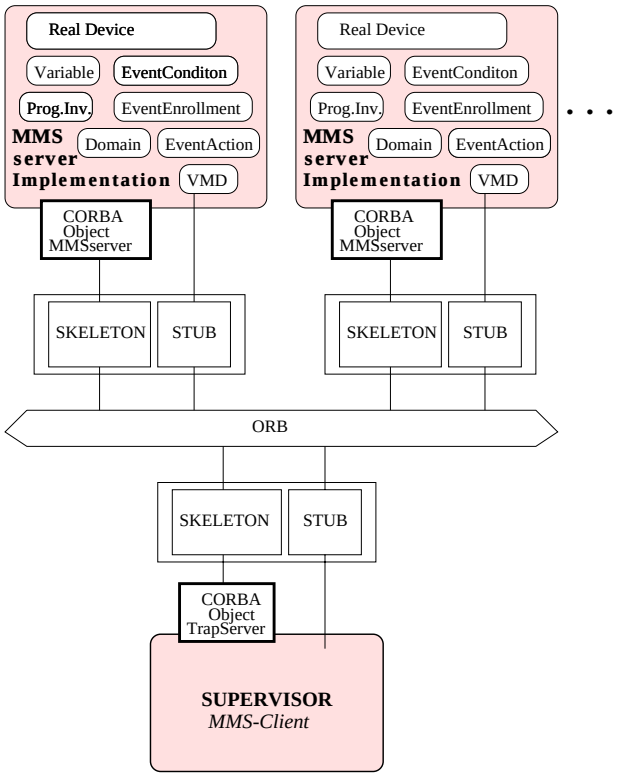

Fig. 1 - System Architecture.

On the server side the components are the followings:

- MMS Server: CORBA object that can be accessed through the object bus. The services provided by this object are the services specified in the MMS standard for the server. The description of this object is specified using the Interface Definition Language (IDL).
- MMS Server Implementation: It is the method implementation of the CORBA object MMS Server. In order to carry out the methods of this object specified in IDL, the objects defined in MMS are used.
- Skeleton: It allows the clients to call the MMS Server methods using the ORB.
- Stub: It is used in order to carry out the remote invocation to the methods of the TrapServer object so that the server can notify the events to the client when they happen.
- Real Device: It is the physical device that is hidden by the MMS Server object.

On the client side, the following components can be identified:

- Supervisor: It is the object that is in charge of controlling the execution of the distributed

application. In order to achieve this, it must activate the several tasks in the different manufacturing devices and receive the event notifications sent by them. It is a MMS client.

- TrapServer: It is the CORBA object that allows the reception of events on the client side. The specification has been carried out using IDL.
- Skeleton: It allows the TrapServer object methods to be called through the ORB.
- Stub: It undertakes the responsibility for calling the remote methods of the MMS Server object.

All the communication between the supervisor and the several Virtual Manufacturing Devices that compound the manufacturing system, and have to carry out the different tasks in order to accomplish the execution of the distributed application, is fulfilled through the CORBA object bus, the ORB (Object Request Broker).

IV. AN APPLICATION

In this section, the system described in the previous section will be illustrated by an example. The manufacturing system, where the application is run, is shown in figure 2 (this is the cell which is in our laboratory). It is compounded by several manufacturing devices (robots, conveyor, warehouse, numerical control machine, and so on).

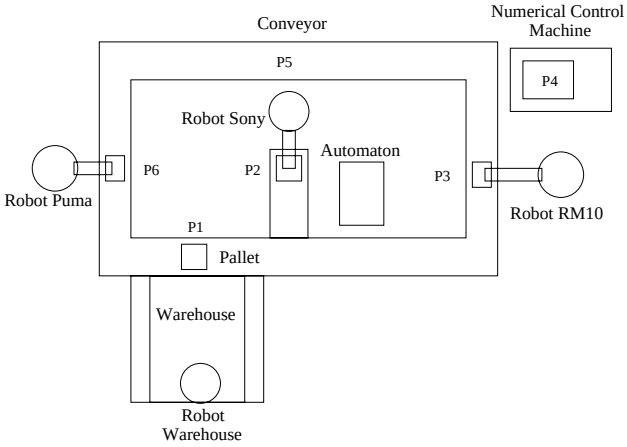

Fig. 2 - Manufacturing System.

A distributed application is going to be executed in this manufacturing system. The application consists of a set of tasks, carried out by several devices which allows two types of parts to be taken from the warehouse, processed, assembled and them to be returned once again.

There are tasks that can be run in a concurrent way but taking into account certain synchronisation points. The application has been as simplified as possible in order to be clearer. Anomalous situations are not considered, but the system functionality may be extended to deal with some of these system malfuctionings.

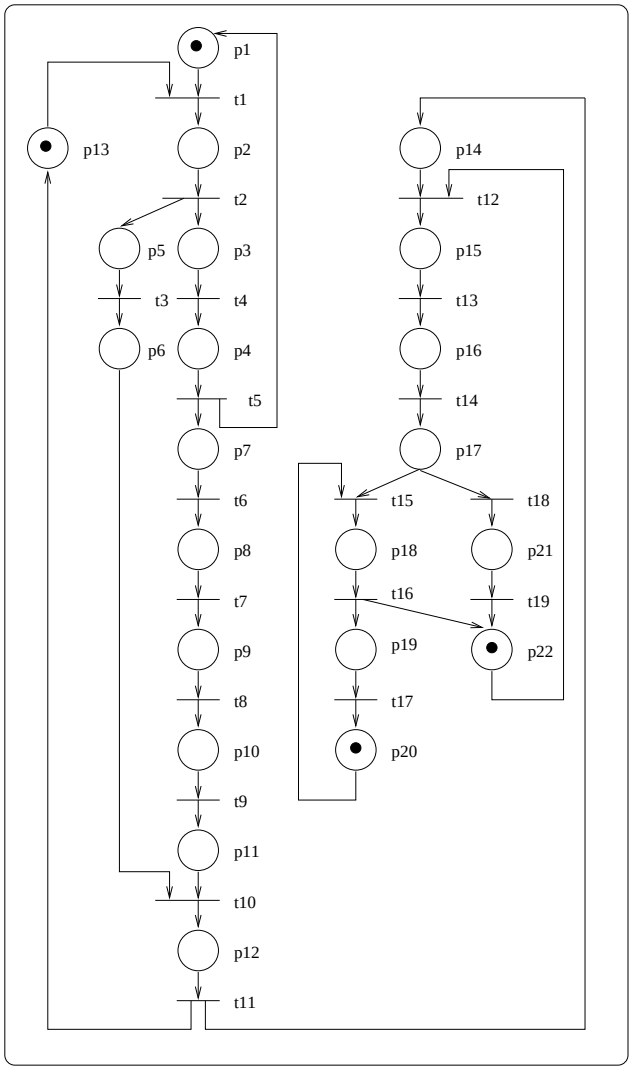

Fig. 3 - Behaviour model using Petri nets.

The Petri net is shown in figure 3 and the different tasks that compound the application are the following:

- P1WAREHOUSE: Move part type 1 from the warehouse to the pallet.
- P2WAREHOUSE: Move part type 2 from the warehouse to the pallet.
- P3WAREHOUSE: Move the part from the pallet to the warehouse.
- PijCONVEYOR: Move the pallet from position Pi to position Pj.
- P1SONY: Process part type 2.
- P2SONY: Assembly part type 1 and type 2.
- P1RM10: Move part type 2 from position P3 to position P4.
- P2RM10: Move part type 2 from position P4 to position P3.
- P1NCM: Process part type 2.
- P1PUMA: Check the new part.
- P2PUMA: Throw out the part.

Event notifications happen when the tasks end and when the PUMA detects that the assembling of the parts have or have not been suitably completed.

The event communication carried out by the manufacturing devices has been associated with the transitions in the net. Moreover, transitions have been introduced in order to mean synchronisation. The transitions are shown in figure 4. The execution of tasks in the several components and the wait have been associated to the places. They are shown in figure 5.

t1: Notification: P1WAREHOUSE ended
t2: Notification: P12CONVEYOR ended
t3: Notification: P1SONY ended
t4: Notification: P2WAREHOUSE ended
t5: Notification: P13CONVEYOR ended
t6: Notification: P1RM10 ended
t7: Notification: PNCM ended
t8: Notification: P2RM10 ended
t9: Notification: P35CONVEYOR ended
t10: Synchronization
t11: Notification: P2SONY ended
t12: Synchronization
t13: Notification: P56CONVEYOR ended
t14: Notification: P1PUMA ended
t15: Notification: CORRECTPIECE
t16: Notification: P61CONVEYOR ended
t17: Notification: P3WAREHOUSE ended
t18: Notification: NOCORRECTPIECE
t19: Notification: P2PUMA

Fig. 4 - Transitions.

p1: Start P1WAREHOUSE	p12: Start P2SONY
p2: Start P12CONVEYOR	p13: Wait
p3: Start P2WAREHOUSE	p14: Wait
p4: Start P13CONVEYOR	p15: Start P56CONVEYOR
p5: Start P1SONY	p16: Start P1PUMA
p6: Wait	p17: Wait
p7: Start P1RM10	p18: Start P61CONVEYOR
p8: Start PNCM	p19: Start P3WAREHOUSE
p9: Start P2RM10	p20: Wait
p10: Start P35CONVEYOR	p21: Start P2PUMA
p11: Wait	p22: Wait

Fig. 5 – Places.

The supervisor receives the Petri net as the input. The Petri net is specified as a set of input places, output places and the initial marks. The programs that are run in the different devices are also given as input to the supervisor. With this information it must create the Program Invocation objects in each Virtual Manufacturing Device.

In order to begin the application it starts the tasks associated to the marked places on the initial mark. When it receives an event notification it must upgrade the state of the corresponding transition and fire it if possible. When the transition is fired, the supervisor will activate the tasks associated to the output places.

V. IMPLEMENTATION

A prototype for this work has been built, where the object-oriented programming language JAVA has been used. This prototype can be run in any computer and operating system where the Java interpreter can be run. The VMD used is a CORBA object developed in [3] that attend to MMS requirements.

CORBA has been used as the communication architecture and the ORB chosen is JAVAIDL because previous work had been done with it. Threads have had to be used in order to divide the supervisor into client and TrapServer.

VI. CONCLUSION

The aim of developing this work is to run distributed applications in a manufacturing system using the VMD specified in MMS and the CORBA architecture as the support of the communications.

More explicitly, a supervisor component has been proposed. This supervisor starts tasks and receives event notifications of the different VMDs by means of the Object Bus defined in CORBA and is responsible for running the distributed application, taking into account the concurrent work and the synchronisation carried out between the several devices.

VII. ACKNOWLEDGEMENT

This work is partially supported by the CICYT under grant num. TAP-98-0541.

REFERENCES

[1] G. Guyonnet, E. Gressier-Soudan, F. Weis, "COOL-MMS: a CORBA approach for ISO-MMS", ECOOP'97 WorkShop,1997.

[2] Open Group, "Inter-domain Management: Specification Translation", X/Open Document Number: P509,1997.

[3] T. Ariza, F.R. Rubio, "Communicating MMS Events in a Distributed Manufacturing System using CORBA", Preprints DCCS'98,1998.

[4] T. Ariza, F.R. Rubio, "MMS-Manager: Device Management in Heterogeneous Environment Based on CORBA", Preprints Controlo'98,1998.

[5] T. Ariza, F.R. Rubio, "Notifying MMS Events to CORBA Objects", Preprints MIC'99,1999.