COMMUNICATING MMS EVENTS IN A DISTRIBUTED MANUFACTURING SYSTEM USING CORBA

T. Ariza * F.R. Rubio *

* *Departamento de Ingeniería de Sistemas y Automática, Universidad de Sevilla, Escuela Superior de Ingenieros, Camino de Los Descubrimientos s/n. E-41092 Sevilla, Spain.*
Phone: +34 95 446 73 64. Fax: +34 95 448 73 85.
E-mail: matere@trajano.us.es

Abstract: Nowadays, distributed computing systems are widely used due to the great advances that have been achieved in computer networks in the past few years, and partly due to their low cost. Developing software for these systems turns out to be complicated, mainly 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 the access to them. This work proposes the use of the ORB specified in CORBA from the OMG to communicate MMS events.

Keywords: Distributed models, Object-Oriented programming, Communication protocols, Events, Manufacturing Systems.

1. INTRODUCTION

Distributed systems have obtained great importance in the past few years. They have been introduced into a great diversity of environments and explicitly in manufacturing systems. So that, where there were isolated components, there are now integrated components in the distributed computing system, where the communication and cooperation between them is allowed.

This advantage is decreased due to the heterogeneity of the devices that can be found in the manufacturing environments. Because of this, the necessity of using a common protocol in order to communicate these interconnected elements arises. In addition to this, using services provided by the protocol must be easy from the point of view of the user application.

MMS is a communication protocol widely used in manufacturing distributed systems. It is an application layer protocol that homogenises the use of the devices that compound the system. MMS makes use of the object oriented 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.

Taking this into account, research has been undertaken, but they do not deal with the event communication, an important aspect of the MMS protocol that allows to send event notifications to and from manufacturing devices.

The objetive of this work is to propose CORBA as the methodology used to communicate MMS events in manufacturing environments. In the following sections, the aim of this paper will be presented, as well as a summary on the background of it and the way to use CORBA in order to communicate events.

2. AIM OF THE WORK

Taking into account the CORBA architecture, that allows the split of the system in objects and communication of these objects in an easy way and taking into account the VMD event management services, communicating MMS events and carrying out service requests by means of CORBA is proposed.

In this way, structuring the distributed system in a set of objects that are communicated by means of service requests is allowed. But this distribution is transparent for the user because it is the Object Bus that undertakes the responsibility to find the several objects. And furthermore, it is used to communicate MMS events to the several VMD's.

This work proposes to make available the MMS services for the event management by means of the CORBA Object Bus. These services are the services specified in the MMS standard. The translation scheme of these services to CORBA IDL are going to be studied.

The services for the event management in MMS are divided in two groups: The services carried out by the MMS server and the services carried out by the MMS client.

This approach is used to communicate clients to the several Virtual Manufacturing Devices by means of the ORB, using the MMS interface for events.

3. BACKGROUNDS

In order to understand the work carried out, MMS, the event model and CORBA are going to be briefly introduced.

3.1 *MMS*

In distributed heterogeneous systems, where each device has different features, carries out different tasks and is probably owned by a different manufacturer, the need to interconnect all devices that compound the system rises in order to achieve the integration of each one in the whole system.

It is possible to use two different focuses to allow the devices to communicate:

- Using the tools and the languages provided by the manufacturer for each device.
- Using a common language for all the devices. This language must be independent to the particular low level features.

The disadvantages of this first approach are evident:

- The specialised staff must learn the particular features of each device.
- The application depends on the device.
- It is difficult to debug and maintain applications for these systems.

The second approach does not have these problems, because of this, the engineer in charge of developing applications that run in heterogeneous systems only has to learn one language. Moreover, the application will be independent to the device and therefore easier to debug and maintain.

Manufacturing Message Specification (MMS) can be used in this second approach. MMS is a communication language to aid the interconnection of devices in a heterogeneous environment. It is a protocol that falls in the application level standardised by ISO (International Standard Organisation).

Although MMS is not a complete Object-Oriented language, as it does not support all the features of the Object-Oriented programming, it splits the system in object and presents a well defined interface for each one. The most important object in the system is the Virtual Manufacturing Device (VMD). The VMD hides the real manufacturing device to the programmer. The system is structured in a set of VMD.

MMS is also based on the client/server model. The VMD acts as a server, and so carries out a set of services that are described in the MMS protocol. The clients are the requesters of the services provided by the VMD.

A particular implementation of the MMS server must provide the mapping between the VMD model, which is an abstraction, and the functionality of the real manufacturing device.

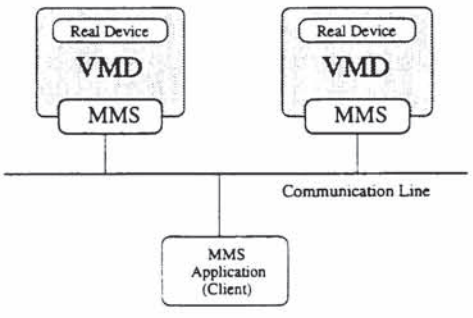

Fig. 1. System based on MMS.

The scheme of a system based on the communication protocol MMS is showed in figure 1.

Each MMS server manages a set of objects associated to the VMD. Some of the most important are the following:

- *The Domain Object* : It represents a subset of the capabilities of the VMD which is used for a specific purpose.

- *The Program Invocation Object* : It is a dynamic element which most closely corresponds to an execution thread in a multitasking environment.
- *The variable Object* : It is used to model real variables of the VMD.

Other important objects that are managed by the VMD are events. These objects are explained in the next section.

In order to handle these objects, the VMD provides a set of services for them.

A revision of MMS can be found in (Pimentel, 1990; CCE-CNMA, 1995) and a complete description of all these services and the protocol specification in (ISO/IEC. 9506-1, 1990; ISO/IEC. 9506-2, 1990).

3.2 *MMS EVENTS*

The event management services provide facilities which allow a client MMS-user to define and manage event objects at a VMD and to obtain notifications of event occurrences.

The event management model extends the VMD with three additional objects and services that work in these objects. Each one of these objects models a specific aspect of the state information associated to the MMS event management. They are briefly commented on in the following:

- *The Event Condition Object* :
  The event management model defines two classes of Event Condition object, Network-triggered event condition object and Monitored Event Condition object. The first one models an event which occurs due to the explicit request of a client using the TriggerEvent service. Although it also models events occurring internally, as a result of autonomous VMD action. The second one is a virtual representation of an aspect of the activity of the VMD which represents a significant occurrence in the processing of the VMD. Either class of Event Condition object includes the potential for VMD initiation of EventNotification service executions.
- *The Event Action Object* :
  It models the portion of the state information which is concerned with the execution of MMS services due to the occurrence of an event. An Event Action is a confirmed MMS service which will be executed when a specific transition of a state attribute of the Event Condition object is detected.
- *The Event Enrollment Object* :
  It is used optionally in order to bind an Event Condition object to an Event Action object and to bind the notifications that are results of an event transition to a client. It represents the request of a client to be notified of the occurrence of one or more specific transitions of the state attribute of the Event Condition object, or to delay the execution of a confirmed MMS service until the occurrence of one or more specific transitions of the state attribute of the Event Condition status.

The services that have been added to the VMD to manage events are the following:

- In order to define and delete objects: DefineEventCondition, DeleteEventCondition, DefineEventAction, DeleteEventAction, DefineEventEnrollment, DeleteEventEnrollment.
- In order to obtain and modify attributes: GetEventConditionAttributes, AlterEventConditionMonitoring, GetEventActionAttributes, GetEventEnrollmentAttributes, AlterEventEnrollment.
- In order to obtain alarm summary: GetAlarmSummary, GetAlarmEnrollmentSummary.
- In order to inform about the status objects: ReportEventConditionStatus, ReportEventActionStatus, ReportEventEnrollmentStatus.
- In order to notify events: TriggerEvent, EventNotification, AcknowledgeEventNotification.

3.3 *CORBA*

The Common Object Request Broker Architecture (CORBA) is a distributed object architecture that allows software objects to interact across networks. CORBA was first introduced in 1991 by the Object Management Group (OMG). It is an international consortium of over 800 software vendors, developers and end users.

The aim of this group is to specify an open software bus on which object components written by different vendors can interoperate regardless of the implementation language, location or host platform.

The object bus provides an Object Request Broker (ORB) that lets clients invoke methods on remote objects either statically or dinamically.

As a result of their work, OMG approved a set of specifications -called CORBA 2.0- in late 1994.

The Object Management Architecture (OMA) specified by OMG, is shown in figure 2 and consists of the following elements:

- *Object Request Broker (ORB)* : It provides the mechanisms by which objects transparently make and receive requests and responses. In doing this, the ORB provides interoperability between applications on different machines in heterogeneous distributed

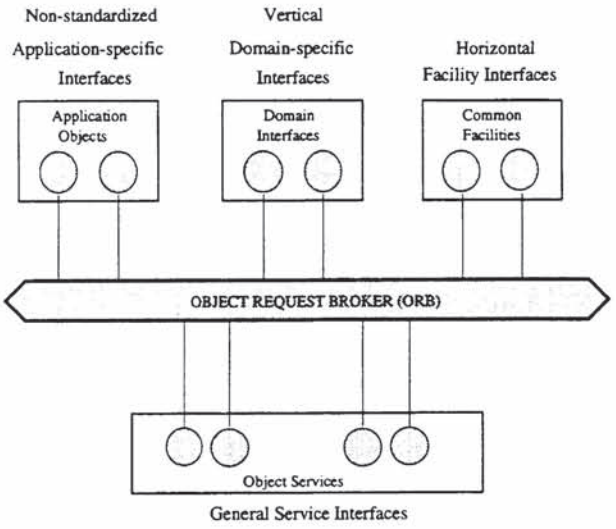

Fig. 2. The Object Management Architecture (OMA) specified by OMG.

environments and seamlessly connects multiple object systems.

- *Object Services* : They are services that complement the functionality of the ORB. These components standardize the life-cycle management of objects.
- *Common facilities* : Services of direct use to application objects. It provide a set of generic application functions that can be configured to the specific requirements of a particular configuration.
- *Domain Interfaces* : They represent vertical areas that provide functionality of direct interest to end-users in particular application domains.
- *Application Objects* : They are specific components to end-user applications. An application is typically built from a large number of basic objects - some specific to the application at hand, some domain specific, some from object services and some built from a set of common facilities.

The object specification is carried out using the Interface Definition Language (IDL). It is a descriptive language used to define the interface through which a client may access a server. IDL provides operating system and programming language independent interfaces to all the services and components that reside on a CORBA bus.

The static method invocation used by the CORBA's implementation is shown in figure 3 (CORBA also define a dynamic method invocation, but it is not explained). The following components are necessary to carry out the invocation:

- *Client IDL stub* : Client code used by an object to encode invocations in a form which can be handled by the ORB, and to decode replies received via the ORB.

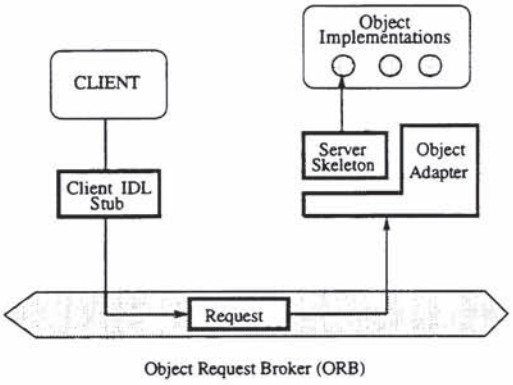

Fig. 3. Object's method's static invocation.

- *Skeleton* : Server code up-called by the ORB, capable of decoding requests transmitted by the ORB, converting it into an invocation of the implementation object, and encoding the results to be sent back to the client via the ORB.
- *Object adapter*: It defines how an object is activated. It can do this by creating a new process, creating a new thread within an existing process, or by reusing an existing thread or process.

More information regarding CORBA can be found in (Object Management Group, 1995*a*; Object Management Group, 1995*b*).

4. SYSTEM DESCRIPTION

In order to structure the system and to adapt MMS to CORBA, the approach proposed in (G. Guyonnet, 1997) has been followed. This CORBA approach for ISO-MMS is known as COOL-MMS, because it has been thought of for the CHORUS-COOL system (CHORUS Systems, 1996).

The features of the system 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 a IDL specification. The event services have been added so that events can be communicated.
- 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 unsolicited information.

The scheme of the system is shown in figure 4.

Although the association management is not implemented due to the fact that they are managed by the ORB, some considerations must be made.

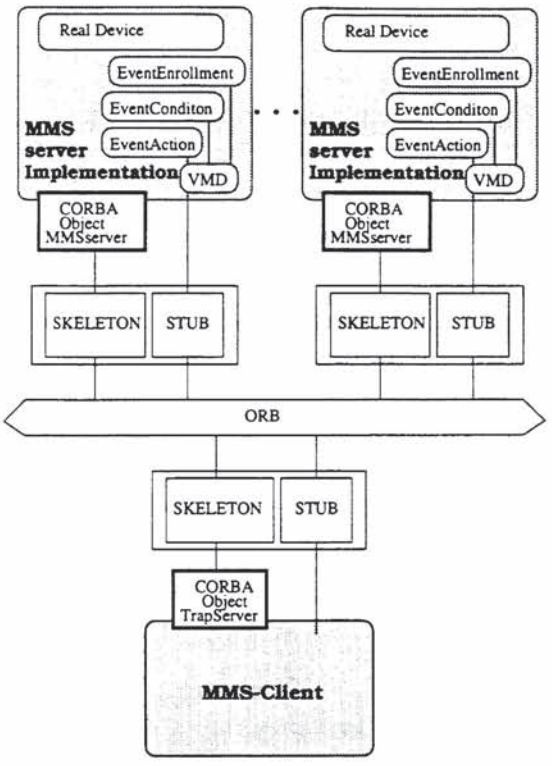

Fig. 4. Event Management in CORBA.

In some cases, in the objects and services of the event model, being able to indicate a specific association is important. For example, the Event Enrollment object has an attribute that is a reference to the application that must be advised with an Event Notification service when the state of a variable changes. Then, the MMS server must be capable of indicating this to a client. In the standard MMS, the application reference is defined as specified below:

```
ApplicationReference ::= SEQUENCE
{
  ap-title
      [0] AP-title                    OPTIONAL,
  ap-invocation-id
      [1] AP-invocation-identifier OPTIONAL,
  ae-qualifier
      [2] AE-qualifier                OPTIONAL,
  ae-invocation-id
      [3] AE-invocation-identifier OPTIONAL
}
```

But this information is only meaningful in an OSI environment. In CORBA, the following application reference has been used:

```
enum ApplicationReferenceChoice
{
   name,
   reference
};
union ApplicationReference
      switch (ApplicationReferenceChoice)
{
  case name:
      Identifier ap-name;
  case reference:
      TrapServer ap-reference;
};
```

This reference is the application name or the reference to a TrapServer object. Having the reference, the MMS server can communicate with the client directly by means of the ORB. Otherwise, having the name, it can obtain this reference asking to the Naming Services.

5. EVENT SERVICES IN CORBA

In order to carry out the translation from ASN.1 to idl the specification Inter-domain Management: Specification Translation of The Open Group (Open Group, 1997) is followed.

Next, the example of the TriggerEvent service in idl and the way to translate the necessary data structures from ASN.1 to idl is shown.

The data specified in the MMS protocol to realize a request and return the response for the TriggerEvent services are the following.

```
Identifier ::= VisibleString
ObjectName ::= CHOICE {
   vmd-specific
         [0] IMPLICIT Identifier,
   domain-specific
         [1] IMPLICIT SEQUENCE {
                domainId          Identifier,
                itemId            Identifier
         },
   aa-specific
         [2] IMPLICIT Identifier
}
Unsigned8 ::= INTEGER
Priority ::= Unsigned8
TriggerEvent-Request ::= SEQUENCE {
   eventConditionName
          [0] ObjectName,
   priority
          [1] IMPLICIT Priority OPTIONAL
}
TriggerEvent-Response ::= NULL
```

The corresponding structure in idl obtained by means of the translation following the specification of the Open Group are the following:

```
typedef char ASN1_Null;
const ASN1_Null ASN1_NullValue='\x00';
typedef string ASN1_VisibleString;
typedef ASN1_VisibleString Identifier;
typedef long ASN1_Integer;
typedef ASN1_Integer Unsigned8;
typedef Unsigned8 Priority;
typedef ASN1_Null TriggerEvent_Response;
```

```
enum ObjectNameChoice {
   vmd_specificChoice,
   domain_specificChoice,
   aa_specificChoice
};
union ObjectName switch (ObjectNameChoice) {
   case vmd_specificChoice:
      Identifier vmd_specific;
   case domain_specificChoice:
      struct {
         Identifier domainId;
         Identifier itemId;
      } domain_specific;
   case aa_specificChoice:
      Identifier aa_specific;
};
union PriorityTypeOpt switch (boolean) {
   case TRUE: Priority value;
};
struct TriggerEvent_Request {
   ObjectName eventConditionName;
   PriorityTypeOpt priority;
};
```

The method in the MMSserver interface specified in idl is:

```
TriggerEvent_Response
      TriggerEvent(TriggerEvent_Request)
      raises(ServiceError);
```

Once the VMD interface to manage MMS events is defined in idl, the MMSserver object services for the event management can be accessed through the ORB from any place in the manufacturing distributed system.

6. IMPLEMENTATION

A prototype has been implemented. The function of it is to accept requests from the clients and send unconfirmed requests to them. It only deals with the communication of events. The event management has not been implemented, but some information regarding this can be found in (Swiss Federal Institude of Technology, Lausanne, 1992).

The ORB used is JavaIDL. JavaIDL is an Object Request Broker provided with the JDK 1.2. Together with the idltojava compiler, it can be used to define, and access CORBA objects from the Java programming language. JavaIDL is compliant with the CORBA 2.0 Specification and the IDL-to-Java Language Mapping.

JavaIDL also provides a transient nameserver to organize objects into a tree-directory structure. The nameserver is compliant with the Naming Service Specification described in CORBAServices: Common Object Services Specification (Object Management Group, 1995*b*).

7. CONCLUSION

The aim of developing this work is to advance in the integration of the Virtual Manufacturing Devices (VMD) in distributed systems based on CORBA.

More explicitly, the MMS event management model has been dealt with. By means of the Object Bus defined in CORBA, making available the MMS event management services have been achieved.

In order to accomplish this, the interface for these services have been built in idl. Starting from the data specified in ASN.1 used by the protocol, and following the Specification Translation of The Open Group, the specification of these data in idl have been obtained.

In this paper, the CORBA bases, the MMS and services bases are also explained.

8. ACKNOLEDGEMENT

This work is supported in part by the CICYT under grant num. TAP-95-0307

9. REFERENCES

CCE-CNMA, ESPRIT Consortium (1995). *MMS: A Communication Language for Manufacturing.* Springer.

CHORUS Systems (1996). *CHORUS/COOL ORB Programmer's Guide.* CS/TR-96-2.2.

G. Guyonnet, E. Gressier-Soudan, F. Weis (1997). *COOL-MMS: a CORBA approach for ISO-MMS.* ECOOP'97 WorkShop.

ISO/IEC. 9506-1 (1990). *Industrial Automation Systems - Manufacturing Message Specification", Part 1: Service Definition.* International Standards Organization.

ISO/IEC. 9506-2 (1990). *Industrial Automation Systems - Manufacturing Message Specification", Part 2: Protocol Specification.* International Standards Organization.

Object Management Group (1995*a*). *CORBA: Common Object Request Broker Architecture and Specification.* OMG, Framingham, MA.

Object Management Group (1995*b*). *CORBAservices: Common Object Services Specification.* OMG, Framingham, MA.

Open Group (1997). *Inter-domain Management: Specification Translation.* X/Open Document Number: P509.

Pimentel, Juan R. (1990). *Communication Networks for Manufacturing.* Prentice-Hall.

Swiss Federal Institude of Technology, Lausanne (1992). *MMS Event Management.* Computer Engineering Dept.