Proceedings of the 1998 IEEE
International Conference on Robotics & Automation
Leuven, Belgium • May 1998

Development of a Distributed Object-Oriented System Framework for the Computer-Integrated Manufacturing Execution System

Fan-Tien Cheng Eric Shen Jun-Yan Deng
Institute of Manufacturing Engineering
National Cheng Kung University
Tainan, Taiwan, R.O.C.

Kevin Nguyen
Mitta Technology Group Inc.
710 Lakeway Drive, Suite 100
Sunnyvale, CA 94086 U.S.A.

ABSTRACT

Today, most of the Manufacturing Execution Systems (MES) are large, monolithic, insufficiently configurable and difficult to modify. Using a distributed object-oriented technique, we will present a systematic approach to develop a computer-integrated MES Framework which is open, modularized, distributed, configurable, interoperable, collaborative, and maintainable. The CORBA infrastructure is adopted to develop this integratable MES. We also use OLE Automation and COM objects to construct sample applications. An example is shown to demonstrate the fruit of this systematic approach.

1 Introduction

A semiconductor manufacturing entity includes integration of the processing equipment with all of the supporting systems for product and process specification, production planning and scheduling, and material handling and tracking. IC fabrication is a very complex and capital-intensive manufacturing process [1].

In this paper, we will present a distributed object-oriented technique, to develop a computer-integrated MES Framework [2,3,5] which is open, modularized, distributed, configurable, interoperable, collaborative, and easy to maintain.

Our systematic approach is started with system analysis by collecting system requirements and analyzing domain knowledge. Our MES Framework is designed by the process of constructing abstract object model based on system requirements, partitioning application domain into components, identifying generic parts among components, defining framework inter-communication and messaging, and developing design patterns [11] for generic parts. Following the MES Framework design, various functional components can be designed by inheriting appropriate design patterns of the MES Framework's common components. An individual Application can be constructed by invoking corresponding methods of related components. At the final stage, the proposed MES can be integrated and tested.

The CORBA infrastructure [8,7] (see Fig. 1) is adopted to develop this integratable MES. Also, OLE Automation technique is applied to construct Applications (see Fig. 2). An example is shown to demonstrate the fruit of this systematic approach.

2 MES Development Procedure

As shown in Fig. 3, the MES development procedure consists of 5 steps: 1) system analysis; 2) MES Framework design; 3) component design; 4) Application construction; and 5) system integration and testing.

This work was supported by the National Science Council, Republic of China, under Contract NSC-87-2218-E-006-001.

The major purpose of the first step: system analysis is to collect all the related domain requirements and fully analyze these requirements such that the fullscale domain knowledge is obtained. According to this domain knowledge, system requirements, functions, constraints, and performance are studied and analyzed.

With the domain knowledge, the second step: MES Framework design can be commenced. The procedure of MES Framework design includes 1) construct abstract object model according to the domain knowledge; 2) partition application domain into components; 3) identify generic parts among components; 4) define framework messages; and finally 5) develop design patterns for generic parts.

After accomplishing the MES Framework design, the third step is the component design based on a design pattern; the fourth step is the Application construction which involves a variety of components; and the final fifth step is the system integration and testing.

In this paper, an integratable MES for the IC packaging factory is adopted as an example to demonstrate the MES Framework development procedure shown in Fig. 3. The detailed design will be explained in the following sections.

3 System Analysis

System analysis, the first step of the MES development procedure, is concerned with devising a precise, concise, understandable, and correct model of the real system. Before developing an MES, the developer must collect the domain requirements and the real-world environment in which it will operate. Then, he must examine and analyze these domain requirements such that all the important features and domain knowledge can be obtained.

Figure 4 shows a typical distributed architecture for an IC packaging manufacturing entity. As depicted in Fig. 4, system manager monitors and controls the status of the whole factory. Scheduler is in charge of scheduling and dispatching job assignments. Common database stores customer orders, job assignments, equipment status, recipes, bills of materials and other engineering data. Equipment manager controls and monitors equipment. Material manager handles the movement of AGV's, AS/RS, robots, and material.

In fact, a complete MES will also have the capabilities for work-in-process (WIP) tracking, statistical process control (SPC), etc. For lack of space and without loss of generality, these capabilities are not covered in this paper. After obtaining the result of system analysis as stated above, we are ready to commence the MES Framework design.

4 MES Framework Design

Our goal is to design an integratable MES by using the distributed object-oriented approach. Those basic foundations for distributed objects and framework [7] as well as the methodology of the Object Modeling Technique (OMT) [6] will be applied to achieve this goal.

The procedure for MES Framework design is composed of 5 steps (see Fig. 3):

1. Constructing abstract object model;
2. Partitioning application domain into components;
3. Identifying generic parts among components;
4. Defining framework messages; and
5. Developing design patterns for generic parts.

The details of these 5 steps are described below.

4.1 Constructing Abstract Object Model

As shown in Fig. 3, the abstract object model is constructed according to the domain knowledge obtained from system analysis. Using the system shown in Fig. 4 as an example, and considering the fact that an MES is composed of several functional modules that handle specifics like material, equipment, labor and planning [4], the abstract object model of an MES is constructed as in Fig. 5. The three key elements of a factory are Equipment, Material, and Labor. Each element is managed by its specific manager. All these three specific managers are controlled by System Manager. System Manager also dispatches orders to Scheduler. Scheduler dispatches jobs to Equipment, Material, and Labor Managers. As for the Common Database, it supports all of the objects to access data.

4.2 Partitioning Application Domain into Components

Our goal is to design an integratable MES which is highly distributed. Therefore, its application domain shall be partitioned systematically and methodologically.

The system shown in Fig. 5 may be partitioned into 6 components as depicted in Fig. 6. They are System Management, Scheduler, Common Database, Equipment Management, Material Management, and Labor Management Components.

System Management Component (which includes System Manager) is in charge of system-level management and services such as life-cycle services, collection services, and query services. Scheduler Component (which includes Scheduler) accepts orders from System Manager and conducts scheduling tasks, then dispatches jobs to Equipment Manager, Material Manager, and Labor Manager.

Equipment Management Component (which includes Equipment Manager and Equipment) manages process equipment; Material Management Component (which includes Material Manager and Material) controls the movements of materials, AGV's, AS/RS, and robots; Labor Management Component (which includes Labor Manager and Labor) handles labor. We define Equipment, Material, and Labor as system Resources. Therefore, these three Management Components are also considered as Resource Management Components.

4.3 Identifying Generic Parts among Components

Among the 6 components shown in Fig. 6, the Common Database Component represents a common facility of the CORBA infrastructure. As for other components indicated in Fig. 6, the structure of Equipment Management Component, Material Management Component, and Labor Management Component are quite similar. The structure of System Manager to control the other managers is similar to that of the Equipment Management Component. According to these observations, we identified generic parts among these four components and, further, we are able to propose a design pattern. As for the Scheduler Component, because its structure is different from those of the other components, a different design pattern is required.

4.4 Defining Framework Messages

The messages which enable interoperability and collaboration among all the components are termed framework messages. In order to maintain system uniformity, only framework messages are allowed to pass into and out of the MES Framework components.

According to the partition shown in Fig. 6, the framework messages are defined as in Fig. 7. System Manager invokes *Initiate*, *StartUp*, *ShutDown*, and *StandBy* methods of Resource Management Components, and Resource Management Components will reply *EventReport* and *AlarmErrorReport* messages.

Scheduler will accept *DispatchOrder* and *CancelOrder* from System Manager, and reply *OrderDoneReport* to it. Also, Scheduler will invoke *DispatchJob* and *CancelJob* methods of Resource Management Components and accept *JobDoneReport* from them. Among Resource Management Components, various demand-service messages are sent and their corresponding reply messages are received.

In addition to those messages described above, objects within each component also define several function-related-service messages in their IDL's to serve the other components. For example, System Manager provides a *CreateOrder* method in its IDL for the outside world to invoke such that an order can be created; and the Common Database Component provides *StoreData*, *RetrieveData*, *DeleteData*, and *ModifyData* methods to serve the other components.

Based upon the partition shown in Fig. 6 and the framework messages defined in Fig. 7, design patterns within the MES Framework can be developed. They are described in the next subsection.

4.5 Developing Design Patterns for Generic Parts

For lack of space, only the development of the Generic Component Design Pattern (GCDP) for Resource Management Components and System Management Component is explained here. Using the OMT methodology [6] this GCDP consists of object model, dynamic model, and functional model. Only the object model is described here.

In order to maximize the generality and reuse, we pay special attention to the common characteristics and behaviors among the components. By observing the object model shown in Fig. 6, we conclude that, the fundamental structure of a component is a System Manager controls several Component Managers and a Component Manager manages several Resources. Therefore, we propose the structure of the GCDP as in Fig. 8.

Base class is the superclass for Manager and Resource. It specifies the common attributes and operations for Manager and Resource to inherit. Among the attributes, *Owner* specifies its upper management class, *Name* and *Status* are self-descriptive, and *Capability* is optional and shows the functional abilities of this class. All of the operations in Base are designed for the purpose of system-management services.

Manager inherits all the attributes and operations of Base. Also, with the help of Resource Collection, Manager manages and serves several Resources. The operations in Manager are for the purposes of life-cycle services (*RegisterResource* and *RemoveResource*), resource-query service (*CheckResource*), and event-report services (*EventReport* and *AlarmErrorReport*). As for the operation of *GetResourceCollection*, it will get the object reference of Resource Collection for Manager. The purpose of Resource Collection is to help Manager accomplish the resource-collection services (*AddResource* and *DeleteResource*), resource-query services (*QueryResourceStatus*, *ListAllResource*, and *Verify*), and system-management service (*InitiateResource*).

Resource also inherits all the attributes and operations of Base. In addition, it adds an attribute: *Parameter* (to

specify its own parameters) and 3 operations: *StartRunning*, *AbortRunning*, and *Reset*. These 3 operations are mainly for the purpose of handling job processes which will be described below.

Component Manager inherits all the attributes and operations of Manager. Besides, since Component Manager needs to handle job assignments and dispatches them to Resources for manufacturing processes, it adds three operations (*DispatchJob*, *CancelJob*, and *CompleteJob*) and includes the help of Job Handler to achieve those tasks. The relationship between Component Manager and Job Handler is similar to that between Manager and Resource Collection. Therefore, Component Manager needs the *GetJobHandler* operation to get the object reference of Job Handler. In order to help Component Manager handles job assignments, Job Handler is designed to have job-collection services (*AddJob* and *DeleteJob*), job-query service (*ListAllJob*), and parameter-setting service (*SetParameter*).

After all of the design patterns have been developed, the backbone of the MES Framework is established. The MES Framework architecture will be described next.

4.6 The MES Framework Architecture

The MES Framework architecture is shown in Fig. 9. The MES Framework is built on top of CORBA infrastructure which includes ORB, object services, and common facilities [8]. In this research, we treat the Common Database Component as one of the common facilities.

Since the fundamental framework messages and interfaces for the framework components have been considered in the various design patterns, each specific component shall select a proper design pattern to inherit and then include its own designated properties into the component. As such, the component can be integrated into the MES Framework easily and in a plug-and-play fashion.

The MES Framework provides suitable design patterns for the application component that is highly pluggable.

As shown on top of Fig. 9, an Application is constructed by involving several related components. The method for constructing an Application will be explained in Section 6. After finishing the design of the MES Framework, the next step is component design and implementation which will be described next.

5 Component Design and Implementation

The step for component design and implementation determines the full definitions of the classes and associations used in the implementation, as well as the interfaces and algorithms of the methods used to implement operations. As described in Section 4.6, a component shall inherit a proper design pattern and then include its own designated functions into the component.

By way of illustration, the procedures for designing and implementing System Management Component and Equipment Management Component will be selected and demonstrated in this section. To begin with, the GCDP developed in Section 4.5 will be implemented. Then, System Management Component and Equipment Management Component will be designed and implemented.

5.1 Implementation of Generic Component Design Pattern

After the processes of analysis and design as explained in Sections 3 and 4, we will have the object, dynamic, and functional models [6] of the GCDP, but the object model is the main framework around which the design and implementation is constructed. In fact, for this research the object model in Fig. 8 has already considered the necessities for converting the actions and activities of the dynamic model and the processes of the functional model into operations attached to classes in the object model itself.

With the operations in the object model being defined, their corresponding algorithms shall be designed for implementation. For lack of space, the details of the algorithms for all the operations are not illustrated here. As for the interfaces of the object model, they will be expressed in IDL.

Note that, only those attributes/operations of sever objects which are allowed to be accessed/invoked by the outside clients need to be defined in IDL. Therefore, the operations: *InitiateResource*, *AddResource*, and *DeleteResource* of Resource Collection and the operations: *AddJob*, *DeleteJob*, and *ListAllJob* of Job Handler are not shown in the IDL. As such, the IDL of GCDP is listed below:

```
interface Base {
    readonly attribute string Owner;
    readonly attribute string Name;
    readonly attribute short Status;
    readonly attribute long Capability;
    exception Reject { long ErrorNum; };
    void StartUp()          raises(Reject);
    void ShutDown()         raises(Reject);
    void StandBy()          raises(Reject);
    void Initiate (in string owner, in string name, in long capability)
                            raises(Reject);
    void StartOperation()   raises(Reject);
};
interface Manager : Base {
    exception Reject { long ErrorNum; };
    exception Alarm { string Message; };
    void RegisterResource(in string name)      raises(Alarm);
    void RemoveResource(in string name)        raises(Alarm);
    boolean CheckResource(in short state);
    ResCollection GetResourceCollection();
    void EventReport(in string msg);
    void AlarmErrorReport(in string msg);
};
interface ResCollection {
    string ListAllResources();
    short QueryResStatus(in string name);
    boolean Verify(in string name);
};
interface ComMag : Manager {
    exception Reject { long ErrorNum; };
    void DispatchJob(in JobAssignment aJob)
    raises(Reject);
    void CancelJob(in string JobID)
    raises(Reject);
    void CompleteJob(in string JobID);
    JobHandler GetJobHandler();
};
interface JobHandler {
    string ListAllJobs();
};
interface Res : Base {
    readonly attribute long parameter;
    void StartRunning();
    void AbortRunning();
    void Reset();
};
```

For brevity, the concise model of the GCDP may be depicted as in Fig. 10.

5.2 Design and Implementation of System Management Component

The role of System Management is to provide system-management, life-cycle, component-manager-collection, component-manager-query, and event-report services to Component Managers. Besides, System Manager will create orders and dispatch them to Scheduler. Also, System Manager will accept order-done report from Scheduler and provide order status for other components to query. And, an order may be removed by System Manager.

Based on the functional requirements stated above, the System Management Component may be constructed as in Fig. 11 which uses GCDP as the foundation. Because the relationship between System Manager and Component Managers is similar with the relationship between Manager and Resources of GCDP, System Manager can be designed by inheriting the properties from Manager (of GCDP) and adding its own unique properties. As such, the IDL file for the System Management Component is shown below:

```
interface SysMag : Manager {
    string CreateOrder(in mfg_order_attribute aOrder);
    void OrderDoneReport(in string OrderNum, in short status);
    short QueryOrderStatus(in string OrderNum);
    void RemoveOrder(in string OrderNum);
};
```

5.3 Design and Implementation of Equipment Management Component

By the same token, as depicted in Fig. 12, the Equipment Management Component is designed by inheriting GCDP. The GCDP provides the basic behavior and fundamentals of a framework component. Therefore, by inheriting GCDP, the designer may not worry too much about the framework rule and can concentrate on the design of the specific properties belonging to the component itself. In the following section, Application construction with system integration and testing will be introduced.

6 Application Construction with System Integration and Testing

An Application is constructed by invoking corresponding methods of related components as shown at top of Fig. 9. In this research, the technique for CORBA integration with OLE/COM as shown in Fig. 2 is applied. In other words, Applications (client sides) are constructed by Microsoft's Visual Basic 4.0 [12] using the technique of OLE Automation and the components (server sides) are built by Microsoft's Visual C++ 4.0 [13] using Iona's Orbix 2.0.2 [14] which implements CORBA objects.

For the purpose of allowing Automation controllers [9,12] to view CORBA objects [14] as Automation objects [9,12] the correspondence (a gateway as shown in Fig. 2) between CORBA IDL and OLE Automation are defined [10]. For each IDL type, one or more corresponding OLE Automation type definitions as generated by the Orbix-OLE Wizard [10]. For example, an OLE Automation dual interface containing property and method definitions is generated for each IDL interface containing attribute and operation definitions. To invoke an operation on a remote CORBA object, an Automation controller simply invokes a method on the corresponding OLE Automation interface [10].

An example (called Application 1) for an IC wire bonding manufacturing process is adopted. For the purpose of demonstration and brevity, with this process, only a System Manager, a Scheduler, a Common Database, an Equipment Manger, and an Equipment of the IC packaging manufacturing entity shown in Fig. 4 are involved. For system bootstrap and resources monitoring purposes, an Application Console which serves as an OLE Automation controller is also included for the demonstration. Fig. 13 shows the message flow for the manufacturing process of Application 1. To begin with, all the components of the MES shall be bootstrapped, then an order will be created, and then the manufacturing process will be proceeded automatically following the message flow shown in Fig. 13.

The MES bootstrap procedure via Application Console is depicted in Fig. 14. For lack of space, the detailed explanation for this procedure is not shown here.

After the bootstrap procedure, all of the components in the MES will be in the *Operation* state and ready to receive an order.

Now, we may create an order on the Application Console, then the manufacturing process of Application 1 will be proceeded automatically according to the message flow shown in Fig. 13. Correspondingly, the framework messages for Application 1 are depicted in Fig. 15. The interaction of these framework messages have also been omitted here.

We may also use Application Console to construct generic Graphic User Interface (GUI) for querying and monitoring the status of all the Managers and Resources as well as all the jobs and orders. The approach is again to simply invoke the associated methods such as *QueryResourceStatus*, *ListAllResources*, *ListAllJobs*, etc.

Currently, this MES Framework Architecture and the demonstration programs of Application 1 have been successfully set up and running at the Factory Automation Laboratory of the Institute of Manufacturing, National Cheng Kung University, Tainan, Taiwan, Republic of China.

7 Summary and Conclusions

Applying the distributed object-oriented technique, a systematic approach for developing a computer-integrated MES Framework was proposed in this paper. Based on this MES Framework, an integratable MES which is open, distributed, interoperable and collaborative is achievable. Each component of the MES Framework was developed by inheriting a proper design pattern which is considered as the basic designs for architecture, framework messages, and interfaces of this component to interoperate and collaborate with the other components. The specific properties and implementation of the component can then be added into the component in a systematic approach. The component is integratable into the MES Framework in a plug-and-play fashion. Applications were built by invoking corresponding methods of related components with the techniques of OLE Automation and CORBA integration with OLE/COM. A design example was included in this paper. From this example, it is believed that the proposed systematic approach for developing an integratable MES is indeed a viable and efficient method.

References

[1] W. Maly, "Computer-Aided Design for VLSI Circuit Manufacturability," in *IEEE Proc. Design for Manufacturability*, 1993.

[2] D. Scott, "Comparative Advantage through Manufacturing Execution Systems," in *SEMICON Taiwan 96 IC Seminar*, pp. 227-236, Taipei, Taiwan, R.O.C., September 1996.

[3] K. Nguyen, "Flexible Computer Integrated Manufacturing Systems," in *SEMICON Taiwan 96 IC Seminar*, pp. 241-247, Taipei, Taiwan, R.O.C., September 1996.

[4] A. MacDonald, "MESs Help Drive Competitive Gains in Discrete Industries," *I&CS*, pp. 69-72, September 1993.

[5] J. McGehee, J. Hebley, and J. Mahaffey, "The MMST Computer-Integrated Manufacturing System Framework," *IEEE Transactions on Semiconductor Manufacturing*, Vol. 7, No. 2, pp. 107-115, May 1994.

[6] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1991.

[7] R. Orfali, D. Harkey, and T. Edwards, *The Essential Distributed Objects Survival Guide*, New York: John Wiley and Sons, Inc., 1996.

[8] Object Management Group, *Common Object Request Broker: Architecture and Specification, Revision 2.0*, Framingham, MA: Object Management Group, 1995.

[9] K. Brockschmidt, *Inside OLE*, Redmond, Washington: Microsoft Press, 1995.

[10] *Orbix 2: Orbix Desktop for Windows User's Manual*, Cambridge, MA: IONA Technologies Ltd, October 1996.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, Massachusetts: Addison-Wesley, 1995.

[12] *Microsoft Visual Basic Tutorials Version 4.0*, Redmond, Washington: Microsoft Corporation, 1995.

[13] *Microsoft Visual C++ Tutorials Version 4.0*, Redmond, Washington: Microsoft Corporation, 1995.

[14] *Orbix 2: Programming Guide*, Cambridge, MA: IONA Technologies Ltd., October 1996.

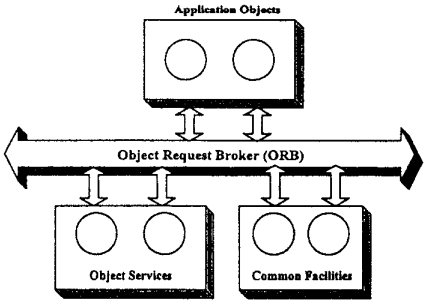

Figure 1: The Object Management Architecture

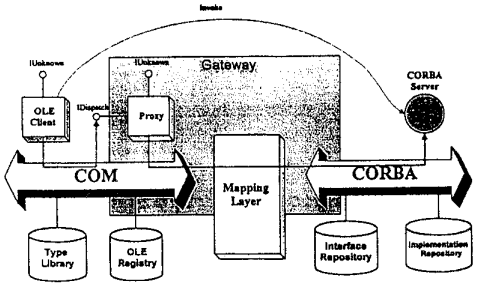

Figure 2: OLE-to-CORBA via a Dynamic Invocation Gateway

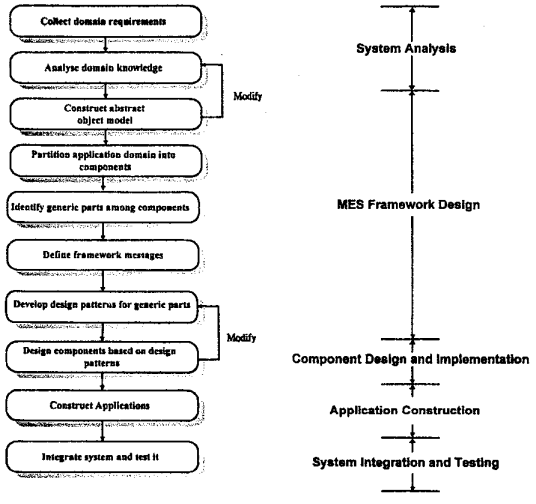

Figure 3: MES Development Procedure

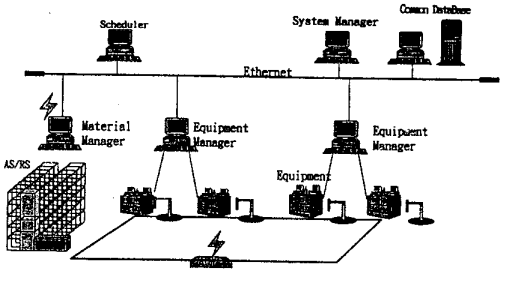

Figure 4: Typical Distributed Architecture for an IC Packaging Manufacturing Entity

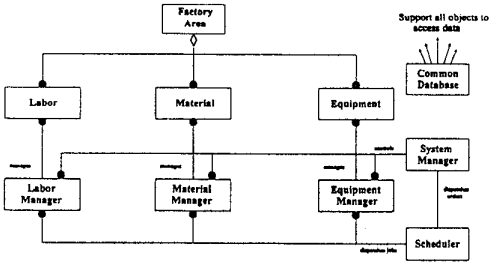

Figure 5: Abstract Object Model of an MES

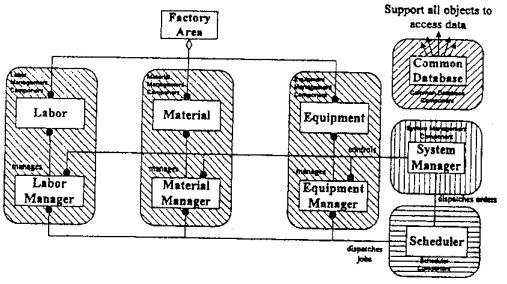

Figure 6: Partitioning Application Domain into Components and Identifying Generic Parts among Components

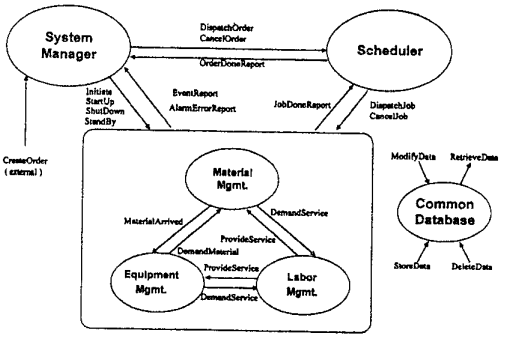

Figure 7: Defining Framework Messages

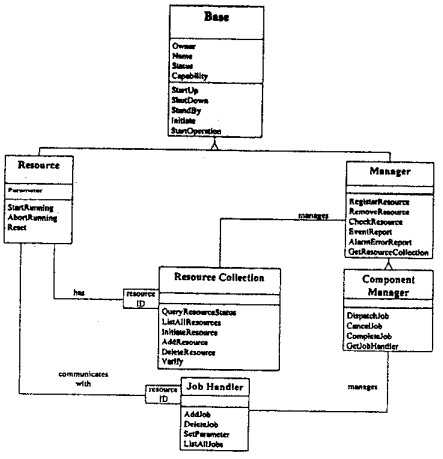

Figure 8: Object Model for the Generic Component Design Pattern

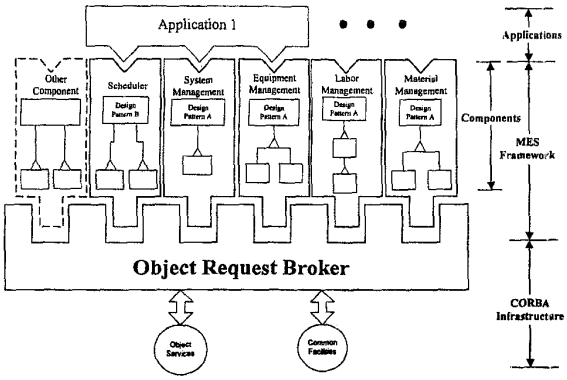

Figure 9: The MES Framework Architecture

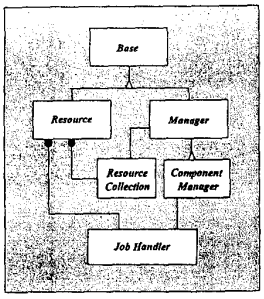

Figure 10: Concise Model of the Generic Component Design Pattern

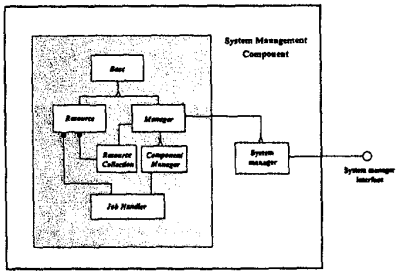

Figure 11: System Management Component

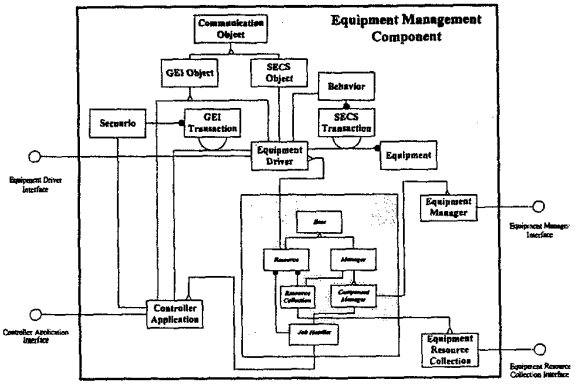

Figure 12: Equipment Management Component

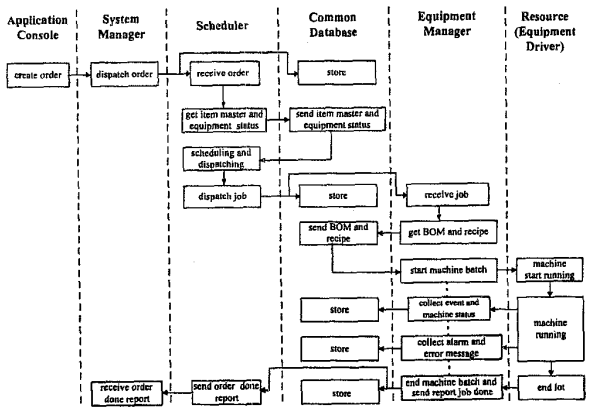

Figure 13: Application 1 Message Flow

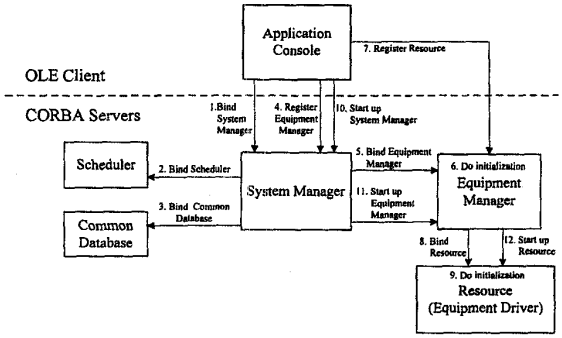

Figure 14: MES Bootstrap Procedure

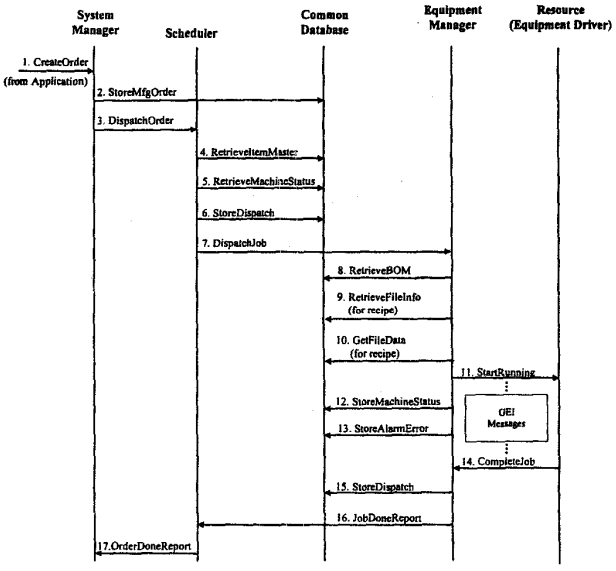

Figure 15: Application 1 Framework Messages Interaction Diagram